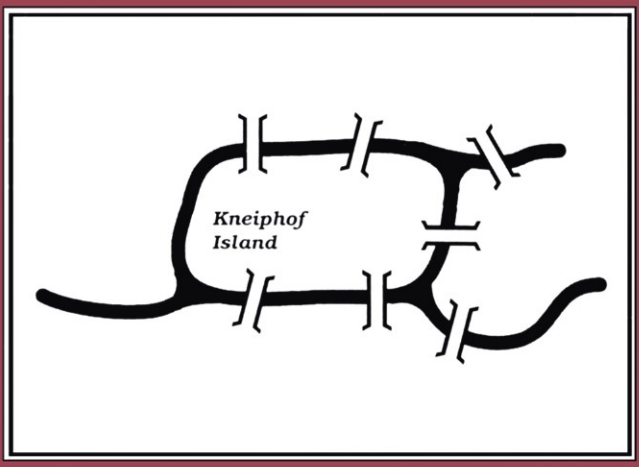
Contributors

E. Benavent
Lawrence Bodin
James F. Campbell
A. Corberán
Ulrich Derigs
Moshe Dror
Richard W. Eglese
H.A. Eiselt
Herbert Fleischner
Alain Hertz
Ellis L. Johnson
André Langevin
Gilbert Laporte
Adam N. Letchford
Janny M.Y. Leung
Laurence Levy
Michel Mittaz
Paul A. Mullaseril
J.M. Sanchis

ARC ROUTING: Theory, Solutions and Applications



Edited by **Moshe Dror**

ARC ROUTING

ARC ROUTING
Theory, Solutions and Applications

Edited by
MOSHE DROR
University of Arizona

Springer Science+Business Media, LLC

Library of Congress Cataloging-in-Publication Data

Arc routing : theory, solutions, and applications / edited by Moshe Dror.
p. cm.
Includes bibliographical references and index.
ISBN 978-1-4613-7026-0 ISBN 978-1-4615-4495-1 (eBook)
DOI 10.1007/978-1-4615-4495-1
1. Operations research. 2. Graph theory. I. Dror, Moshe.

T57.6 A73 2000
658.4'034--dc21

00-056135

Originally published by Kluwer Academic Publishers, New York in 2000
Softcover reprint of the hardcover 1st edition 2000

Printed on acid-free paper.

in memory of
Chaja & Asher
Frydman

Contents

Preface xvii

Contributing Authors xiii

1
A Historical Perspective on Arc Routing 1
H.A. Eiselt, Gilbert Laporte
1.1 Introduction 1
1.2 The Chinese Postman Problem 2
1.2.1 The Undirected CPP 3
1.2.2 The Directed CPP 6
1.2.3 The Mixed CPP 7
1.2.4 The Windy CPP 8
1.2.5 The Hierarchical CPP 9
1.3 The Rural Postman Problem 9
1.3.1 The Undirected RPP 10
1.3.2 The Directed RPP 10
1.3.3 The Mixed RPP 11
1.4 The Capacitated Arc Routing Problem 11
1.5 Research Outlooks 12

Part I THEORY

2
Traversing Graphs: The Eulerian and Hamiltonian Theme 19
Herbert Fleischner
2.1 Introductory Remarks 19
2.2 Basics of Graph Theory 20
2.2.1 Graphs and Their Parts 20
2.2.2 Walks, Trails, Paths, Cycles; Connectedness 23
2.2.3 Bipartite Graphs, Trees, Blocks, Mappings 27
2.3 Connectivity, Menger's Theorem, the Splitting Lemma, and Factors 32
2.4 Eulerian Graphs and Covering Walks, Cycle Decompositions and Cycle Covers 40
2.4.1 Algorithms for Constructing Eulerian Trails 47
2.4.2 Mazes 48
2.5 Hamiltonian Cycles and Vertex-Covering Walks 50
2.6 Elements of Matching Theory 58
2.7 The Chinese Postman Problem, The Traveling Salesman Problem, and Related Problems 69
2.8 Elements of Network Theory 77

3
Matching: Arc Routing and the Solution Connection 89
Ulrich Derigs
3.1 Introduction 89
3.2 Matching: Applications 91
3.2.1 Team Selection 91
3.2.2 Task Scheduling 92
3.2.3 Processor Scheduling 93
3.2.4 Route Connection 94
3.2.5 Arc Routing 95
3.2.6 Node Routing 98
3.2.7 General Routing 101
3.2.8 Set Partitioning 104
3.3 Matching: Combinatorial Aspects 107
3.4 Matching: Polyhedral Aspects 112
3.5 Matching Algorithms: Linking Combinatorial and Polyhedral Results 116
3.6 Matching Algorithms: Implementation Issues 119
3.6.1 Start Procedures: Constructing the Initial Extreme Matching 120
3.6.2 Organization of Dual Updates 122
3.6.3 Price and Reoptimize 123

4
Arc Routing: Complexity and Approximability 133
Moshe Dror
4.1 Introduction: Easy and Hard Problems 133
4.2 CPP as a Problem in $\mathcal{P}$ 141
4.3 $\mathcal{NP}$-Hard Generalizations of the CPP 143
4.3.1 The Mixed CPP 143
4.3.2 The MCPP $\mathcal{NP}$-Completeness 144
4.3.3 The Rural Postman Problem 147
4.3.4 The Windy Postman Problem 148
4.3.5 Non-intersecting Eulerian Circuits and A-trails in Eulerian Graphs 149
4.3.6 Dominating Trails 151
4.3.7 Precedence in Arc Routing 152
4.3.8 Capacitated Arc Routing 154
4.4 Approximation Algorithms 156
4.5 Approximation Results for Arc Routing 159
4.5.1 The Mixed CPP 159
4.5.2 The Windy CPP 161
4.5.3 The RPP and Other Variants 161
4.5.4 The CARP 162
4.6 Conclusions 164

5
Chinese Postman and Euler Tour Problems in Bi-directed Graphs 171
Ellis L. Johnson
5.1 Bi-directed Graphs, Euler Tours, and Postman Tours 171
5.2 Aircraft Routing in a Space-time Network 176
5.3 The Chinese Postman Problem in an Undirected Graph 181
5.4 Binary Group Problems and Blocking Problems 189
5.5 Ideal Binary Matrices 192
5.6 Four Problems on Planar Graphs 193

Part II Solutions

6
Polyhedral Theory for Arc Routing Problems 199
Richard W. Eglese, Adam N. Letchford
6.1 Introduction 199
6.2 The Basics of Polyhedral Theory 200
6.3 The Routing Problems Defined 203
6.4 Variants of the Chinese Postman Problem 205
6.4.1 The CPP 205
6.4.2 The DCPP 205
6.4.3 The MCPP 206
6.4.4 The WPP 208
6.5 Variants of the Rural Postman Problem 209
6.5.1 The RPP 209
6.5.2 The GRP 212
6.5.3 The DRPP 216
6.5.4 The MRPP 217
6.6 The Capacitated Arc Routing Problem 219
6.6.1 Preliminaries 219
6.6.2 Sparse Formulations of the CARP 220
6.6.3 The Dense and Supersparse Formulations of the CARP 224
6.7 Conclusions 226

7
Linear Programming Based Methods for Solving Arc Routing Problems 231
E. Benavent, A. Corberán, J.M. Sanchis
7.1 Introduction 232
7.2 Chinese Postman Problems 236
7.2.1 The Undirected CPP 236
7.2.1.1 Odd-Cut Separation 237
7.2.2 The Directed CPP 238
7.2.3 The Mixed CPP 239
7.2.3.1 A Cutting Plane Algorithm for the MCPP 240
7.2.3.2 Odd-Cut Separation 241
7.2.3.3 Balanced Set Separation 242
7.2.4 The Windy Postman Problem 243
7.3 Rural Postman Problems 244
7.3.1 The Undirected RPP 244
7.3.1.1 Connectivity Separation 246
7.3.1.2 R-odd cut Separation 246
7.3.1.3 K-C Inequalities Separation 247
7.3.1.4 Cutting Plane and Branch & Cut Algorithms for the RPP 249
7.3.2 The General Routing Problem 250
7.3.2.1 Honeycomb Separation 251
7.3.2.2 Path-Bridge Separation 253
7.3.2.3 Cutting Plane and Branch & Cut Algorithms for the GRP 255
7.3.3 The Directed RPP 256
7.3.4 The Mixed RPP 257
7.3.4.1 Connectivity Separation 258
7.3.4.2 R-odd Cut and Balanced Set Separation 259
7.3.4.3 K-C and Path-Bridge Separation 259
7.4 The Capacitated Arc Routing Problem 259
7.4.1 Sparse Formulations 260
7.4.1.1 Connectivity Separation 261
7.4.1.2 Parity Separation 262
7.4.1.3 Obligatory Cutset Separation 263
7.4.1.4 Separation of Constraints from the Knapsack Problem 263
7.4.2 Supersparse Formulation for the CARP 264

7.4.2.1 Capacity Constraints Separation 267
7.4.2.2 Disjoint Path Inequalities Separation 267
7.4.3 Exact Methods Based on the Sparse Formulation 268
7.4.4 A Cutting Plane Algorithm for the CARP Based on the Supersparse Formulation 269
7.5 Other Problems 269
7.6 Conclusions 270

8
Transformations and Exact Node Routing Solutions by Column Generation 277
Moshe Dror, André Langevin
8.1 Introduction 278
8.2 Transformations to Node Routing: Why? 279
8.2.1 The Capacitated Rural Postman Problem 279
8.2.2 Mathematical Formulation of the CARP 280
8.2.2.1 Time Window Constraints for Arc Routing 281
8.2.3 When to Transform to Node Routing 282
8.3 Arc Routing Transformations: How 283
8.3.1 Transformations of Uncapacitated Arc Routing Problems 284
8.3.2 Transformations of Capacitated Arc Routing Problems 285
8.3.3 Transformation of CARP with Time Windows to VRPTW 287
8.3.4 Split Delivery Arc Routing with Time Windows 289
8.4 Column Generation for Routing Problems with Non-split Delivery 290
8.4.1 Revised Simplex: Chvátal's Introduction to Column Generation 290
8.4.2 Set Covering, Vehicle Routing, and Column Generation 292
8.4.3 The Shortest Path Subproblem 294
8.4.4 An Algorithm for the Shortest Path with Resource Constraints 295
8.4.5 Solving SPRCP with only Elementary Paths 297
8.4.5.1 Overlooking an Optimal Elementary Path 299
8.4.5.2 An SPRCP – Improved Elementary Path Algorithm 300
8.4.5.3 Description of the Algorithm 302
8.5 Column Generation for Routing Problems with Split Delivery 305
8.5.1 Properties of Split Deliveries with Triangle Inequality 306
8.5.2 Formulation 308
8.5.3 Set Covering Approach for Split Deliveries 309
8.5.4 The Subproblem for Generating Feasible Columns for Split Delivery 311
8.5.4.1 Formulating the Subproblem 312
8.5.4.2 Alternating between the Master Problem and the Subproblem 313
8.5.5 The Computational Phase: A Mixed Integer Solver and a Dynamic Programming Algorithm 314
8.5.5.1 Discretizing the Split Deliveries in the SPPRC Subproblem 315
8.5.5.2 Discussion of Computational Experiments for the SDVRPTW 316
8.5.6 Another Set Covering Formulation for SDVRPTW 317
8.5.6.1 The (MP_3) Model and Column Generation Approach 318
8.5.6.2 The Branching Scheme for the SDVRPTW 321
8.6 Conclusion 322

9
Heuristic Algorithms 327
Alain Hertz, Michel Mittaz
9.1 Introduction 327
9.2 Heuristics for Uncapacitated Arc Routing Problems 329
9.2.1 The Chinese Postman Problem 329
9.2.1.1 The undirected chinese postman problem 329
9.2.1.2 The directed chinese postman problem 331
9.2.1.3 The mixed chinese postman problem 332
9.2.1.4 The windy postman problem 336
9.2.2 The Rural Postman Problem 338
9.2.2.1 The undirected rural postman problem 338
9.2.2.2 The directed rural postman problem 340
9.2.2.3 The mixed rural postman problem 344
9.2.2.4 The stacker crane problem 349
9.2.2.5 Additional algorithmic tools 355
9.3 Heuristics for Capacitated Arc Routing Problems 361
9.3.1 Simple Constructive Methods for the CARP 361
9.3.2 Two-Phase Constructive Methods for the CARP 373
9.3.2.1 Route first-cluster second algorithms 373
9.3.2.2 Cluster first-route second algorithms 377
9.3.3 Meta-Heuristics for the CARP 379
9.4 Conclusion 383

Part III Applications

10
Roadway Snow and Ice Control 389
James F. Campbell, André Langevin
10.1 Introduction 389
10.2 Brief History of RSIC 390
10.3 Characteristics of Arc Routing for RSIC 392
10.4 Solution Approaches 395
10.5 Early Work 396
10.6 Recent Work 400
10.6.1 CASPER 400
10.6.2 GeoRoute 406
10.6.2.1Ottawa, Canada 408
10.6.2.2Suffolk County, UK 410
10.6.3 Other Recent Work 411
10.7 Future 413

11
Scheduling of Local Delivery Carrier Routes for the United States Postal Service 419
Lawrence Bodin, Laurence Levy
11.1 Introduction 420
11.2 The Route Adjustment Process 422
11.2.1 Inspection 422
11.2.2 Route Adjustment 423
11.2.3 Auxiliary (or Remnant) Routes 424
11.3 Types of Delivery Routes 424
11.3.1 Park and Loop Routes 425
11.3.2 Curbline/ Dismount Routes 426
11.3.3 Combined Park and Loop and Curbline/ Dismount Routes 426

11.3.4 Relay Box Routes 427
11.4 Lines of Travel for the Four Types of Postal Delivery Routes 427
11.4.1 Walking Line of Travel 428
11.4.2 Driving Line of Travel 428
11.5 Examples of the Lines of Travel for the Four Types of Postal Delivery Routes 429
11.5.1 Park and Loop Routes 430
11.5.2 Curbline/ Dismount Routes 431
11.5.3 Combined Park and Loop and Curbline/ Dismount Routes 432
11.5.4 Relay Box Routes 433
11.5.5 Discussion 434
11.6 Algorithm for Route Adjustment 435
11.6.1 Estimate the Number of Routes to Form 436
11.6.2 Partition the AOI 437
11.6.3 Form the Line of Travel for each Partition 438
11.6.4 Is the Solution Balanced? 439
11.7 Manual Intervention 440
11.8 Conclusions 440

12
Livestock Feed Distribution and Arc Traversal Problems 443
Moshe Dror, Janny M.Y. Leung, Paul A. Mullaseril
12.1 Introduction 444
12.1.1 The Cattle Industry 444
12.1.2 The Cattle Yard Operations 444
12.2 Livestock Feed Distribution as Arc Traversals 447
12.2.1 Arc-Routing Models for Pen Inspection and Feed Delivery 448
12.2.1.1 Mathematical Formulation of the Capacitated Rural Postman Problem (CRPP) 448
12.2.2 Split Deliveries 450
12.2.3 Time Windows 451
12.3 A Heuristic Approach for Trip Generation 452
12.3.1 Generating a Non-split Feasible Solution 452
12.3.1.1 Extended Path Scanning Algorithm for Feeding-Time Feasibility 453
12.3.1.2 Modified Augment-Merge Algorithm 454
12.3.2 Arc Swapping 455
12.3.3 Generating Split Delivery Routes 456
12.3.4 Route Addition 456
12.4 Route-First Cluster-Second Generalized TSP Heuristic 457
12.4.1 The Generalized Traveling Salesman Problem 457
12.4.2 Transforming the CRPP to an Equivalent GTSP 457
12.4.3 Solving the Equivalent GTSP 458
12.5 Computational Results 458
12.6 Epilogue 461

Contributing Authors

E. Benavent
Department d'Estadística i Investigació Operativa
Universitat de València
Burjassot-València, Spain

Lawrence Bodin
R.H. Smith School of Business
University of Maryland
College Park, Maryland 20742 U.S.A.

James F. Campbell
School of Business Administration
University of Missouri - St. Louis
St. Louis, Missouri 63121-4499 U.S.A.

A. Corberán
Department d'Estadística i Investigació Operativa
Universitat de València
Burjassot-València, Spain

Ulrich Derigs
Seminar für Wirtschaftsinformatik und Operations Research
Universität zu Köln
Albertus-Magnus-Platz, D-50923, Cologne, Germany

Moshe Dror
Department of Management Information Systems
University of Arizona
Tucson, Arizona 85721 U.S.A.

Richard W. Eglese
Department of Management Science
Lancaster University
Bailrigg, Lancaster, LA1 4YW England

H.A. Eiselt
Faculty of Administration
University of New Brunswick
Fredericton, New Brunswick, E3B 5A3 Canada

Herbert Fleischner
Institute for Discrete Mathematics
Austrian Academy of Sciences
Vienna, Austria

Alain Hertz
Ecole Polytechnique Fédérale de Lausanne
Department de Mathematiques
MA-Ecubleus, Ch-1015 Lausanne, Switzerland

Ellis L. Johnson
School of Industrial and Systems Engineering
Georgia Institute of Technology
Atlanta, Georgia 30332 U.S.A.

André Langevin
GERAD and École Polytechnique
Montréal, Québec, H3C 3A7 Canada

Gilbert Laporte
Centre de recherche sure les transports
Université de Montréal
Montréal, Québec, H3C 3J7 Canada

Adam N. Letchford
Department of Management Science
Lancaster University
Bailrigg, Lancaster, LA1 4YW England

Janny M.Y. Leung
Department of Systems Engineering and Engineering Management
The Chinese University of Hong Kong
Shatin, Hong Kong

Laurence Levy
RouteSmart Technologies
Columbia, Maryland 21045 U.S.A.

Michel Mittaz
Ecole Polytechnique Fédérale de Lausanne
Department de Mathematiques
MA-Ecubleus, Ch-1015 Lausanne, Switzerland

Paul A. Mullaseril
Department of Management, College of Business
Mankato State University
Mankato, Minnesota 56002 U.S.A.

J.M. Sanchis
Department de Mathemática Aplicada
Universidad Politécnica de València
València, Spain

Preface

The title of this book is *ARC ROUTING: Theory, Solutions, and Applications.* The key word in this title is "*ROUTING*". The thesaurus listing for "route" reads **1** "itinerary, way, course, passage, circuit, beat, run, round, orbit, trajectory, artery, channel, detour, bypass, **2** road, street, roadway, thoroughfare, passage, passageway, artery, channel, corridor, track, trail, avenue, boulevard, highway, thruway, turnpike, **3** means, medium, agency, steps, instrumentality, way, system, *modus operandi*, method, procedure, practice, process, course". As a verb the thesaurus defines it as "channel, direct, send, point, aim, head, steer, drive, maneuver, and pilot." Different thesaurus listings may add to the correspondence of a one to many mapping. Most of these words implicitly or explicitly project the sense of moving along some well-understood (physical) entities or objects. "To route" is defined among other definitions as "to head", "to steer", "to drive", or "to maneuver" (about an object).

The other key word, "*ARC*" corresponds to "curve", "line", or "bend", connecting two points (or objects represented as points) with an indication of a direction from one point to the other. Thus, the title *ARC ROUTING* conveys that the main topic of this book is about ways of "maneuvering", "driving", "steering" along curved lines which connect different objects or points.

Another word associated with routing, especially arc routing, is "*TRAVERSING*" or "*TRAVERSAL*", which in the context of this book is to "pass through", "pass over", "travel across", "march over", "scan", "run through", and "inspect". This book could have been titled *ARC TRAVERSALS* and it would have been just as appropriate as the title *ARC ROUTING*, or perhaps even more so.

The first chapter in this book titled *A HISTORICAL PERSPECTIVE ON ARC ROUTING* by H.A. Eiselt and Gilbert Laporte, describes a problem posed to Leonhard Euler (a well known mathematician) around 1736. The question was whether there is a route (a marching band route) starting on an island in a city then called Königsberg, which would tra-

verse each of the city's seven bridges exactly once (with no repetitions) and end on the island from which the route started. This practical question regarding the planning of a marching band route is often sited as the beginning of modern graph theory (a branch of mathematics), and Leonhard Euler, who answered this question (proving that such a route does not exist for the seven bridges in Königsberg) is credited as the pioneer of graph theory as we know it today. By solving the seven bridges Königsberg problem, he can certainly be considered the "inventor" of arc routing. In honor of Euler, a whole family of graphs was classified as Eulerian graphs and the study of the various properties of such Eulerian graphs has been and still is of great interest for graph theory mathematicians, as is demonstrated in the chapter by Fleischner in this book.

The first time I encountered an arc traversal (arc routing) problem was in the context of examining work assignments for electric meter readers in the city of Beer Sheva in Israel (see Stern and Dror, 1979). The city's electric company periodically sent electric meter readers to record the consumption of electricity by the different households for billing purposes. The meter readers would traverse the city streets (to scan the households' meters) and this "work assignment" would have its routes planned in advance. It was very convenient and natural to represent the problem as a graph problem with the nodes of the graph as the street intersections and street segments between intersections as the edges (undirected arcs) of the graph. It was a "capacitated problem" in the sense that no meter reader was assigned a route which exceeded a specified number of work hours. It was clearly a problem classified as an arc routing problem. When looking for guidance in solving the meter reading problem, a "pandora's box" in the form of the extensive literature about routing and the variety of routing problems opened before me.

This book is primarily about arc traversal and the great variety of arc routing problems and the applications of arc routing theory for real-life problems with the key word *arc*. However, in contrast to arc routing, there are also node or vertex routing problems that focus on the vertices of graphs instead of the graphs' arcs. (See Toth and Vigo, 2000, for a recent collection of articles focused on node routing.) Just as Euler's name stands out for arc routing, Hamilton's name is emblematic for node routing, and subsequently for a name for a family of graphs: the Hamiltonian graphs. (For the historic account, see Hoffman and Wolfe, 1985.) In Hamiltonian graphs one might be concerned with the order in which the vertices of a graph are visited on a tour. Node routing is also covered in this book.

The idea for editing this book came to me while I was examining solutions to a capacitated arc routing problem with split deliveries and arc time windows which originated in the cattle yard industry. (See a detailed description in Dror, Leung, and Mullaseril, this book.) In the last 10 years or so, there has been extensive coverage of arc routing problems in the literature, especially from a graph theory perspective, in book format as well as in numerous articles. (See Fleischner, 1990, 1991, and a chapter in this book.) However, a collection of state of the art expositions of arc routing problems that explore their graph theoretical basis, as well as solution methodologies for arc routing problems, and a number of representative applications, has never (to my knowledge) been compiled in a single volume. This volume represents an attempt to provide just that. I believe that we have succeeded because of the elite group of individuals who have volunteered to contribute to the book, and I am very thankful for their contributions!

The book contains twelve chapters of various lengths organized into three main parts: Theory, Solutions, and Applications. The book begins with a chapter titled *A HISTORICAL PERSPECTIVE ON ARC ROUTING* by H.A. Eiselt and Gilbert Laporte. In this chapter the authors provide a historical road map for arc routing research. Eiselt and Laporte start with the definition and the description of the best known arc routing problem: the celebrated *Chinese Postman Problem.* They provide the motivation, and definitions, and outline the main ideas for problems with names such as the *Windy Chinese Postman Problem*, the *Hierarchical Chinese Postman Problem*, the *Rural Postman Problem*, the *Capacitated Arc Routing Problem*, and others. This discussion of a number of very important arc routing problems which form the backbone of arc routing research provides the reader with a perspective and vocabulary needed for the articles which follow.

The *Theory* part of the book starts with an excellent review of graph Theory for arc routing by Herbert Fleischner titled *TRAVERSING GRAPHS: THE EULERIAN AND HAMILTONIAN THEME.* It is a concise over- view of graph concepts, graph problems, and the state of the art of graph theory in relation to Eulerian graphs and their counterparts, the Hamiltonian graphs. It contains all the important graph theoretical "building blocks" such as *connectivity, Menger's Theorem, the Splitting Lemma, Factors, Eulerian and Hamiltonian graphs, Covering walks and Hamiltonian Cycles,Cycle Decomposition, Cycle Covers and Vertex-cover walks, Elements of Matching Theory*, and all the related problems such as the *Chinese Postman Problem*, the *Traveling Salesman Problem*, and many more. In addition, the chapter contains a bibliography listing all the proofs and a more detailed treatment of the chapter's

themes. The chapter truly represents a unique exposition for all readers interested in arc routing and in particular for the readers interested primarily in solutions and applications for arc routing.

The focus of the third chapter by Urlich Derigs, titled *MATCHING: ARC ROUTING AND THE SOLUTION CONNECTION*, is on the *Matching Problem* which is a central component in the solution of the Chinese Postman Problem and many other related and unrelated problems in combinatorial optimization. In its classical formulation, the problem is that of selecting the "best" pairs of nodes on a given graph. It is one of the most important "stepping stone" problems in the study of combinatorial optimization in general. The chapter by Urlich Derigs describes the state of the art for the matching problem in terms of both its combinatorial and polyhedral aspects. It provides the reader with a clear link between the two in the algorithmic description and discusses the implementational issues for matching algorithms. The chapter is written in a very clear fashion by the man who is arguably the most authoritative professional on theoretical and computational aspects of the matching problem. I still remember the competition in the early eighties in which the matching code of Urlich Derigs outperformed all the competition. This chapter adds to the discussion a very important component in arc routing research.

Chapter 4 is titled *ARC ROUTING: COMPLEXITY AND APPROXIMABILITY* by Moshe Dror. This chapter formally introduces the notion of complexity and attempts to provide a clear delineation between 'hard' arc routing problems and 'easy' (solvable in polynomial time) arc routing problems. In addition, it describes the state of the approximation schemes for the hard arc routing problems. This chapter enables the reader to appreciate the full scope of the different computational examinations of arc routing problems discussed in this book.

Chapter 5, by Ellis L. Johnson is titled *CHINESE POSTMAN AND EULER TOUR PROBLEMS IN BI-DIRECTED GRAPHS.* Written by one of the most prominent contributors to arc routing research (see Edmonds and Johnson, 1973), it describes the connection between important arc routing applications, especially for aircraft routing, and a number of mathematical constructs. The chapter examines a number of concepts such as binary group problems, blocking pairs of clutters, ideal binary matrices, and relates these concepts to the Chinese Postman Problem. In addition, the chapter provides important insights into the polyhedral and algebraic structures associated with the Chinese Postman Problem.

Chapter 6, *POLYHEDRAL THEORY FOR ARC ROUTING PROBLEMS* by Richard W. Eglese and Adam N. Letchford is the first in the *Solution Methodology* section of the book. Its focus (true to the title) is on the known facts related to the polyhedral description of arc routing problems. It provides a very clear overview of polyhedral theory and the state of the art of known valid inequalities and facets for a variety of arc routing problems. It is a very important chapter leading to exact solutions for arc routing problems and is similar in its role to chapters by Grotschel and Padberg (1985), and Padberg and Grotschel (1985), in the famous book *The traveling Salesman Problem* by Lawler et al. (1985). Three problems are examined in detail: the Chinese Postman Problem, the Rural Postman Problem, and the Capacitated Arc Routing Problem. The chapter represents an excellent summary of the status of polyhedral results for arc routing as we know it today.

The chapter by E. Benavent, A. Corberan, and J.M. Sanchis, Chapter 7, titled *LINEAR PROGRAMMING BASED METHODS FOR SOLVING ARC ROUTING PROBLEMS*, represents the logical continuation of the previous chapter by Eglese and Letchford. Whereas chapter 6 was about the descriptive (polyhedral) representation of arc routing problems, this chapter is about using such descriptive representation for constructing solutions to arc routing problems by linear programming in the best possible way. It is an excellent description of the cutting planes and Branch & Cut methodology and how they can be incorporated into the linear programming solution approach for the many different arc routing problem variants considered in this chapter. The chapter examines all the classical arc routing problem formulations in great detail and builds on the extensive cutting plane "machinery" developed in the node routing literature. The objective is to solve such problems exactly, which, for these $\mathcal{NP}$-hard problems, implies an implicit enumeration, which in this case are based on cutting planes and Branch & Cut procedures. The "best" and most successful ingredients for such Branch & Cut solution procedures for the different arc routing problems are superbly described in this chapter.

Chapter 8 in this book is titled *TRANSFORMATIONS AND EXACT NODE ROUTING SOLUTIONS BY COLUMN GENERATION.* This chapter is written by Moshe Dror and Andre Langevin. The chapter has two parts. The first part discusses a number of transformations for modeling arc routing problems as node routing context problems with a 1-1 mapping between the optimal solutions of the corresponding formulations. The second part of the chapter focuses on solutions using column generation methodology for capacitated node routing problems with time windows for the node deliveries. This is motivated by the fact

that a capacitated arc routing problem with time windows on arcs is difficult to model and thus justifies the transformation to a node routing setting. A special focus on the node routing analysis is given to the split delivery option where a node's demand can be delivered by any number of vehicles.

Chapter 9, by Alain Hertz and Michel Mittaz on *HEURISTIC ALGORITHMS* examines solution options for arc routing problems that do not guarantee optimal solutions. An important question regarding hard arc routing problems is that of selecting, from all possible heuristic tour constructions for arc routing described in the literature, the heuristic solutions with demonstrated computational or solution quality advantages. This chapter provides a very well organized and clear description of the different heuristic techniques available for solving a great variety of arc routing problems. In the first two sections after the Introduction, the focus is on more "traditional" heuristic approaches to arc routing problems such as simple constructive heuristics, two-phase heuristics with the flavor of cluster first – route second, or the opposite. In the second part of the chapter the authors present a very fine description of metaheuristics (Simulated Annealing and Tabu Search) and their role in solving arc routing problems.

Chapter 10, is the first of three chapters in the *Applications* section of the book. The chapter's title is *ROADWAY SNOW AND ICE CONTROL* written by James F. Campbell and Andre Langevin. As the authors of this chapter state "Roadway snow and ice control is one of the most complex and fascinating venues for arc routing applications". When trying to explain arc routing research to a novice, this is perhaps the clearest and best motivating application for arc routing. The chapter begins by detailing the background of "snow-plowing" characteristics and complexities. It also describes some early research results for roadway snow and ice control. However, the main focus is on two successful system developments and their applications. The first system (currently used in Indiana according to the authors) is called CASPER. The second system described in detail in the chapter has been implemented in a number of localities in Canada and the U.K.

The second chapter in the Applications section (Chapter 11) is titled *SCHEDULING OF LOCAL DELIVERY CARRIER ROUTES FOR THE UNITED STATES POSTAL SERVICE.* It is written by Lawrence Bodin and Laurence Levy, two experts in automating solutions in this complex arc routing environment. Both authors have been constantly involved in arc routing applications since the early papers by Lawrence Bodin (see Beltrami and Bodin, 1974, and Bodin and Kursh, 1978, 1979)

and the dissertation work of Laurence Levy (1986). This is the "classic" application of arc routing: the generation of postman routes. The book would not be complete without this chapter. The chapter presents a clear description of the many issues the post office faces when constructing and assigning postal delivery routes.

The last chapter in the book, Chapter 12, *LIVESTOCK FEED DISTRIBUTION AND ARC TRAVERSAL PROBLEMS* by Moshe Dror, Janny M.Y. Leung, and Paul, A. Mullaseril presents the third chapter on arc routing applications. It describes a large cattle yard operation near Yuma, Arizona, where over 100,000 head of cattle are fed daily. The feed distribution activity is modeled as a Rural Postman Problem with time windows and split delivery. This chapter provides a detailed description of the arc routing application and the heuristic methodology experimented with by the authors for solving the actual feed distribution problems in this cattle yard. From the arc routing perspective, the cattle yard setting provides an archetypical arc routing setting with many similar problem instances all over the world. It serves as a very appropriate closing for this book.

As the editor of this collection of chapters, I am very honored to have such an excellent group of contributing authors participate in this book-writing endeavor. Clearly, this book is the result of collective effort. I am very thankful to the participants and to the many reviewers who read the chapters and commented constructively. It took longer to complete the book than originally envisioned. I am thankful to many that our efforts have reached such a successful outcome. This book would have required a more difficult 'endgame' without Gregory R. Lousignont's dedication and painstaking attention for the production details. My heartfelt thanks to Greg.

References

[1] Beltrami, E. and L. Bodin (1974). "Networks and vehicle routing for municipal waste collection", *Networks* 4, 65-94.

[2] Bodin, L. and S. Kursh (1978). "A computer-assisted system for the routing and scheduling of street sweepers", *Operations Research* 26, 525-537.

[3] Bodin, L. and S. Kursh (1979). "A detailed description of street sweeper routing and scheduling system", *Comput. & Ops. Res.* 6, 181-198.

[4] Dror, M., J.M.Y. Leung, and P.A. Mullaseril (2000). "Livestock feed distribution and arc traversal problems", in Dror, M. ed. *Arc Routing: Theory, Solutions and Applications* (this book).

[5] Edmonds, J., and E.L. Johnson (1973). "Matching, Euler Tours and the Chinese postman", *Mathematical Programming* 5, 88-124.

[6] Fleischner, H. (1990). *Eulerian Graphs and Related Topics*, Part I, Volume 1. *Annals of Discrete Mathematics* 45. North-Holland, Amsterdam.

[7] Fleischner, H. (1991). *Eulerian Graphs and Related Topics*, Part I, Volume 2. *Annals of Discrete Mathematics* 50. North-Holland, Amsterdam.

[8] Fleischner, H. (2000). "Traversing graphs: The Eulerian and Hamiltonian theme", in Dror, M. ed. *Arc Routing: Theory, Solutions and Applications* (this book).

[9] Grötschel, M. and M.W. Padberg (1985). "Polyhedral theory", in E.L. Lawler, J.K. Lenstra, A.H.G. Rinnooy Kan, and D.B. Shmoys (eds.), John Wiley & Sons Ltd., 251-305.

[10] Hoffman, A.J. and P. Wolfe (1985). "History", in *The Traveling Salesman Problem*, E.L. Lawler, J.K. Lenstra, A.H.G. Rinnooy Kan, and D .B. Shmoys (eds.), John Wiley & Sons Ltd., 1-15.

[11] Lawler, E.L., J.K. Lenstra, A.H.G. Rinnooy Kan, and D.B. Shmoys (eds.), (1985). *The Traveling Salesman Problem*, John Wiley & Sons Ltd.

[12] Levy, L. (1987). *The Walking Line of Travel Problem: An Application of Arc Routing and Partitioning*, Ph.D. Dissertation, University of Maryland at College Park, Maryland.

[13] Padberg, M.W. and M. Grotschel (1985). "Polyhedral computations", E.L. Lawler, J.K. Lenstra, A.H.G. Rinnooy Kan, and D.B. Shmoys (eds.), John Wiley & Sons, Ltd., 307-360.

[14] Stern, H.I. and M. Dror (1979). "Routing electric meter readers", *Comput. & Ops. Res.* 6, 209-223.

[15] Toth, P. and D. Vigo (2000). *The Vehicle Routing Problem,* to appear in SIAM Monographs on Discrete Mathematics and Applications.

Chapter 1

A HISTORICAL PERSPECTIVE ON ARC ROUTING

H.A. Eiselt
University of New Brunswick

Gilbert Laporte
Universite de Montréal

1. Introduction 1
2. The Chinese Postman Problem 2
   2.1 The Undirected CPP 3
   2.2 The Directed CPP 6
   2.3 The Mixed CPP 7
   2.4 The Windy CPP 8
   2.5 The Hierarchical CPP 9
3. The Rural Postman Problem 9
   3.1 The Undirected RPP 10
   3.2 The Directed RPP 10
   3.3 The Mixed RPP 11
4. The Capacitated Arc Routing Problem 11
5. Research Outlooks 12

1. INTRODUCTION

Arc routing problems consist of determining a least cost traversal of some arcs or edges of a graph, subject to side constraints. Such problems are encountered in a variety of practical situations such as road or street maintenance, garbage collection, mail delivery, school bus routing, meter reading, etc. Details on these applications are provided in Eiselt, Gendreau and Laporte (1995b), in Assad and Golden (1995), and in some chapters of this book. Billions of dollars each year are spent on arc routing operations, mainly by public administrations, and there exists a sizeable potential for savings. In recent years, new advances in optimization techniques and in computer technology have contributed to the dissemination and adoption of sophisticated arc routing software.

Nowadays, commercial packages make heavy use of rich data bases, geographical information systems, and interactive graphic interfaces. It is not exaggerated to affirm that there now exists a thriving arc routing industry, mostly sustained by consultancy firms.

But the field of arc routing has more humble beginnings. It all started as a riddle more than two and a half centuries ago with the celebrated Königsberg bridges problem, which gave rise to the birth of modern graph theory and to the first known theorems on arc routing. A number of algorithmic results were discovered in the nineteenth century and in the first half of the twentieth century, but most of the field was developed after 1950. This historical account will therefore cover some truly classical results but it would be of little value if it did not include the most recent developments which constitute the core of current knowledge. Throughout this chapter, we will attempt to answer the two questions "Who was first?" and "What came next?". There exist several general references on arc routing, but the following three books are particularly recommended for their historical perspective: König (1936) and Fleischner (1990, 1991).

This chapter is organized around three main problem classes. In Section 2, we present various versions of the *Chinese Postman Problem* (CPP). These are problems in which it is required to traverse all edges or arcs of a graph. Then, in Section 3, we examine the *Rural Postman Problem* (RPP) in which only some edges or arcs must be traversed. Section 4, deals with the *Capacitated Arc Routing Problem* (CARP), a constrained version of the CPP or the RPP with multiple real-life applications. Some research outlooks are presented in Section 5.

2. THE CHINESE POSTMAN PROBLEM

The Chinese Postman Problem is defined as follows. Let $G = (V, E \cup A)$ be a graph where V is a set of vertices, E is a set of (undirected) edges, and is a set of (directed) arcs. It is generally assumed that G is strongly connected, i.e., it is always possible to reach any vertex from any other vertex. With each edge or arc (v_i, v_j) is associated a cost c_{ij}. The CPP consists of determining a least cost traversal of all edges and arcs of G. Several cases are of interest:

i) the *undirected* CPP, where $A = \emptyset$;

ii) the *directed* CPP, where $E = \emptyset$;

iii) the *mixed* CPP, where $A \neq \emptyset$ and $E \neq \emptyset$;

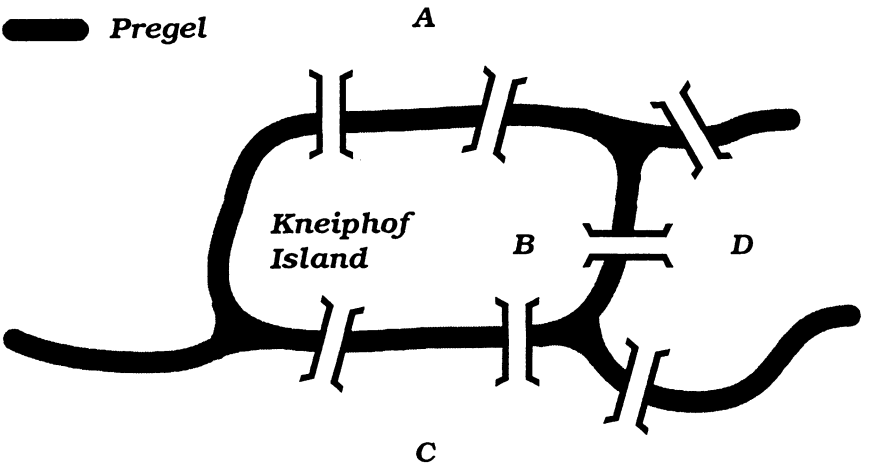


Figure 1.1 The seven bridges of Königsberg.

iv) the *windy* CPP , where $A \neq \emptyset$ but two values c'_{ij} and c''_{ij} are associated with each edge (v_i, v_j), equal to the cost of travel in each direction;

v) the *hierarchical* CPP , where $E \cup A$ is a partitioned into several classes and a precedence relation is established between these classes. If a class C_p of edges and arcs precedes another class C_q, then all edges and arcs of C_p must be serviced before those of C_q.

In this section, we will successively examine each of these cases.

2.1. THE UNDIRECTED CPP

In the early eighteenth century, the inhabitants of Königsberg (now Kaliningrad, Russia) debated whether there existed a closed walk traversing *exactly once* each of the seven bridges over the river Pregel (Figure 1.1). The question was put to the Swiss mathematician Leonhard Euler who showed there was none. He presented his results at the St. Petersburg Academy and in a short article (Euler, 1736).

The problem can be represented by an undirected graph (Figure 1.2), in which one vertex is used for each of the two shores of the river and for the two islands, and each edge corresponds to a bridge. Since a closed walk requires that each vertex be entered and left the same number of times, there is obviously no solution in this case since all vertices have

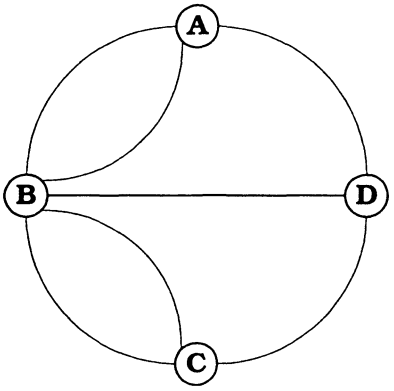


Figure 1.2 Graph representation of the Königsberg bridges problem.

odd degrees. In fact, Euler proved that for a closed walk in an undirected connected graph to exist, all vertices must have an even degree. (The sufficiency of this condition was proved much later by Hierholzer (1873) who was apparently unaware of Euler's work). Such graphs are called *unicursal* or *Eulerian.* Since duplicating each edge makes all vertex degrees even, there always exists, in an undirected connected graph, a closed walk covering each edge exactly twice. This has not actually been proved by Euler, but only illustrated by means of an example. Euler also showed that there exists such a walk using every edge in each direction. The original Latin version of Euler's paper is reproduced in Fleischner (1990), and an English translation has appeared in *Scientific American* (Euler, 1953).

While Euler provided conditions for unicursality, he was not apparently concerned with the problem of actually determining a closed walk in an Eulerian graph. This problem was addressed more than a century later by Hierholzer (1873). Even nowadays, Hierholzer's algorithm is probably the best available and has been rediscovered, sometimes with slight variations, by several other researchers (e.g., the End-Pairing algorithm described in Edmonds and Johnson, 1973). The original description does not follow the style we are accustomed to, but modern "translations" have been provided by some researchers (see, e.g., Even, 1979). It can be sketched as follows:

Step 1. Starting from an arbitrary vertex v, gradually trace a cycle by following untraversed edges, until this procedure cannot be continued; this can only happen at v.

Step 2. If all edges have been traversed, stop.

Step 3. Trace a second cycle starting from an unvisited edge incident to the cycle. Merge the two cycles into one. Go to Step 2.

Another algorithm, not quite as efficient, was proposed by Fleury (1885), a school headmaster in Marseille (Lucas, 1894).

Step 1. Starting from an arbitrary vertex v_i traverse an edge (v_i, v_j) that is not a bridge unless it is an end-edge (i.e., an edge whose removal leaves some edges unreachable), and erase (v_i, v_j).

Step 2. Stop if all edges have been deleted. Otherwise, set $v_i := v_j$ and go to Step 1.

The difficulty in this algorithm is to determine whether a candidate edge is a bridge. A thorough description of several other traversal algorithm for Eulerian graphs is provided in Fleischner (1991).

The Chinese Postman Problem was first stated by Meigu Guan (or Kwan Mei-Ko). The Königsberg bridge problem is exclusively concerned with the *existence* of a closed traversal in a graph, but the question of determining a least cost traversal is still highly relevant in non-Eulerian graphs. Guan (1962) addressed the question of *minimizing* the length of a walk passing through each edge of a graph *at least once.* His method proceeds from the observation that the graph always contains an even number of odd-degree vertices and that shortest chains must be added to link odd-degree vertices and thus make the graph Eulerian. It has been known, at least since the work of Edmonds and Johnson (1973), that this least cost reconnection can be determined by solving a matching problem. Guan does not state this property, but his result can be derived from it. He first proves the following necessary condition for a feasible solution to be optimal: (*i*) it has no redundancy (i.e., at most one edge is added for each edge of the original graph); (*ii*) the cost of added edges on every cycle does not exceed half the cost of the cycle. To establish the sufficiency of this result, the author shows that all feasible solutions satisfying (*i*) and (*ii*) must in fact have the same cost (in fact, Guan's paper only deals with unit costs, but the proof for the general case follows along the same lines). The procedure suggested by Guan to determine such a solution in effect solves the matching problem by ensuring that the chains added to the graph do not intersect. Note that property (*ii*) lies at the heart of the proof used by Christofides (1976) for his 3/2 - approximation of the *Traveling Salesman Problem* (TSP). Guan's Chinese postman article is important, not only because it introduces what has now become a standard network optimization problem, but because it addresses for the first time the "augmentation problem", i.e. the problem of determining the least cost way of making a graph Eulerian by introducing extra edges or arcs. The augmentation problem

is in fact the core problem of arc routing. Once a unicursal graph has been obtained, determining an actual traversal is relatively easy.

The article "Matching, Euler Tours and the Chinese Postman" by Edmonds and Johnson (1973) must be considered as a corner-stone in the field of arc routing. In it, the authors formulate the augmentation problem in an undirected graph using binary integer variables x_{ij} $(i < j)$ equal to the number of copies of edge (v_i, v_j) introduced into the graph in order to make it Eulerian. Letting $\delta(S) = (v_i, v_j) : \{v_i \in S, v_i \in V \setminus S$ or $v_i \in V \setminus S, v_j \in S\}$ for any non-empty subset S of V, the problem can be formulated as follows.

$$(UCPP) \qquad \text{Minimize} \sum_{(v_i,v_j)\in E} c_{ij}x_{ij} \tag{1.1}$$

subject to

$$\sum_{(v_i,v_j)\in\delta(S)} x_{ij} \geq 1, \qquad (S \subset V, S \text{ has an odd number of odd-degree verticies}) \tag{1.2}$$

$$x_{ij} \geq 0, \qquad ((v_i, v_j) \in E) \tag{1.3}$$

$$x_{ij} \; is \text{ integer}, \qquad ((v_i, v_j) \in E) \tag{1.4}$$

The authors prove that the polyhedron of solutions to (1.2) and (1.3) is equal to the convex hull of feasible solutions to (UCPP). The authors show that this program can be solved efficiently by first computing least cost chains between all pairs of odd degree vertices (there is always an even number of them), and by then solving a minimum cost matching problem over the set of odd degree vertices, using these costs. This is done by adapting a previous algorithm by Edmonds for minimum cost matching.

2.2. THE DIRECTED CPP

A strongly connected graph is Eulerian if and only if the in-degree of each vertex is equal to its out-degree. This condition is formally stated by Ford and Fulkerson (1962, p. 60) but it has been common knowledge for a long time (it was already treated in König's (1936) book).

The procedure for solving the augmentation problem in a directed graph was proposed almost simultaneously by Edmonds and Johnson (1973), Orloff (1974), and Beltrami and Bodin (1974). Instead of matching odd-degree vertices as is done in the undirected case, it uses the transportation algorithm in order to make the graph symmetric, i.e., the in- and out-degrees of each vertex are equal. In other words, let I be

the set of vertices v_i for which the number of incoming arcs exceeds the number of outgoing arcs by s_i and let J be the set of vertices v_j for which the number of outgoing arcs exceeds the number of incoming arcs by d_j. Thus, s_i can be interpreted as a supply, and d_j as a demand. In addition, let c_{ij} be the length of a shortest path from v_i to v_j. The problem is then as follows.

$$(DCPP) \qquad \text{Minimize} \sum_{v_i \in I} \sum_{v_j \in J} c_{ij} x_{ij} \tag{1.5}$$

subject to

$$\sum_{v_j \in J} x_{ij} = s_i, \quad (v_i \in I) \tag{1.6}$$

$$\sum_{v_i \in I} x_{ij} = d_j, \quad (v_j \in J) \tag{1.7}$$

$$x_{ij} \geq 0, \quad (v_i \in I, v_j \in J)\,. \tag{1.8}$$

The optimal values represent the number of extra times each arc of a shortest path has to be traversed. Once a unicursal graph has been obtained by this method, determining an actual traversal can be achieved by adapting, for example, Hierholzer's algorithm. An interesting alternative approach is provided in a rather early article by van Aardenne-Ehrenfest and de Bruijn (1951). Their $\mathcal{O}(|V| + |A|)$ time algorithm can be summarized as follows:

Step 1. Construct a spanning arborescence rooted at any vertex .

Step 2. Label all arcs as follows: Order and label the arcs outgoing from in an arbitrary fashion; order and label the arcs out of any other vertex consecutively in an arbitrary fashion, so long as the last arc is the arc used in the arborescence.

Step 3. Obtain an Euler tour by first following the lowest labeled arc emanating from an arbitrary vertex; whenever a vertex is entered, it is left through the arc not yet traversed having the lowest label. The procedure ends with an Euler circuit when all arcs have been covered.

2.3. THE MIXED CPP

Ford and Fulkerson (1962, p. 60) were the first to propose necessary and sufficient conditions for unicursality in a mixed graph: Every vertex must be incident to an even number of directed and undirected arcs; moreover, for every non-empty subset S of V, the absolute value of the

difference between the number of arcs from S to $V \setminus S$ and the number of arcs from $V \setminus S$ to S must be less than or equal to the number of edges between S and $V \setminus S$.

Unfortunately, verifying these conditions or solving the augmentation problem in a mixed graph is $\mathcal{NP}$-hard, as shown by Papadimitriou (1976). This is true even if the graph is planar or if all c_{ij} coefficients have the same value. The standard approach is to formulate the augmentation problem as an integer linear program in which variables represent the number of copies of each arc or edge that must be introduced into the graph in order to make it Eulerian. Three such formulations were proposed by Christofides et al. (1984). Grötschel and Win (1992) and Nobert and Picard (1996). The first of these formulations was solved by branch-and-bound and the last two by branch-and-cut. Nobert and Picard report the most extensive computational results. They solved 440 instances with $16 \leq |V| \leq 225$, $2 \leq |A| \leq 5569$ and $15 \leq |E| \leq 4455$. Out of these, 313 instances were solved to optimality at the root of the search tree and the number of constraints generated in the course of the algorithm was of the order of $|V|$.

Once a unicursal mixed graph has been obtained, it is easy to make it completely directed by using some simple network flow techniques (see Eiselt, Gendreau and Laporte (1995a) for a full description), and any algorithm valid for the directed case can then be applied to determine a traversal.

Heuristics for the mixed CPP were proposed by Edmonds and Johnson (1973), and later improved by Frederickson (1979) and Christofides et al. (1984). The best known heuristics are probably MIXED1 and MIXED2, due to Frederickson, each having a worst-case ratio of 2. If the two heuristics are applied in succession, the worst-case ratio goes down to 5/3.

2.4. THE WINDY CPP

The windy CPP is defined on an undirected graph where two costs are associated with each edge, as with and against the wind, but it is only required to traverse each edge at least once. The problem was introduced by Minieka (1979). Brucker (1981) and Guan (1984) have shown that the windy CPP is $\mathcal{NP}$-hard, but the problem can be solved in polynomial time if G is Eulerian (Win, 1989) or if the cost functions are cycle balancing (see, e.g., Fleischner, 1991). The polyhedral structure of this problem has been analyzed by Win (1987, 1989) and by Grötschel and Win (1988, 1992). These authors have also proposed an integer linear programming formulation and a branch-and-cut algorithm for this

problem. Using this approach, they have solved to optimality instances with $52 \leq |V| \leq 264$ and $78 \leq |E| \leq 489$. Out of 36 instances that were considered, 31 were solved at the root of the search tree.

2.5. THE HIERARCHICAL CPP

The Hierarchical CPP was introduced by Dror, Stern and Trudeau (1987). This problem arises naturally in snow plowing, where streets have different priority levels (Stricker, 1970; Lemieux and Campagna, 1984; Alfa and Liu, 1988; Haslam and Wright, 1991), in garbage collection (Bodin and Kursh, 1978), and in flame cutting (Manber and Israni, 1984). The hierarchical CPP is $\mathcal{NP}$-hard, but a polynomially solvable case has been identified by Dror et al. It occurs when (*i*) G is fully undirected or fully directed, (*ii*) the order relation between the classes C_p of arcs and edges is complete, and (*iii*) each class induces a connected graph. An $\mathcal{O}\left(k\,|V|^5\right)$ time algorithm was proposed by Dror et al. where k is the number of classes. Recently, Ghiani and Improta (2000) showed that this type of hierarchical CPP can be solved as a matching problem on an auxiliary graph with $\mathcal{O}(k\,|V|)$ vertices. A variant of this problem, introduced by Letchford and Eglese (1998), is to set a deadline for the completion of service in each class C_p and to determine whether a solution satisfying these deadlines exists. The authors have proposed a branch-and-cut algorithm capable of solving small to medium size instances.

3. THE RURAL POSTMAN PROBLEM

The Rural Postman Problem is also defined on a graph $G = (V, E \cup A)$, but this time only a subset of $E \cup A$ must be traversed. Such edges and arcs are said to be *required*. The remaining edges and arcs may be part of the solution. Denote by E' and A' the subsets of required edges and arcs. As for the CPP, several cases can be considered, but not all have effectively been studied. The RPP can be viewed as that of designing routes for a postman who has to deliver mail in several villages. Several streets may have to be serviced within each village, but there may be no mail to deliver on streets linking the villages. Hence only a subset of the streets would be required. The problem was introduced by Orloff (1974) and proved later to be $\mathcal{NP}$-hard by Lenstra and Rinnooy Kan (1976). However, the RPP on a completely undirected or directed graph can be solved in polynomial time if the subgraph induced by the required edges or arcs is strongly connected as it then reduces to a CPP.

3.1. THE UNDIRECTED RPP

As suggested by Frederickson (1979), a heuristic with a worst-case performance ratio of 3/2 can be constructed for the undirected RPP along the lines of the TSP heuristic proposed by Christofides (1976). The worst-case behavior of this method can only be guaranteed if the cost matrix satisfies the triangle inequality. The method works by constructing a shortest spanning tree between connected components of required edges, and by then matching the odd-degree vertices. Recently, Hertz, Laporte and Nanchen-Hugo (1999) produced a family of post-optimization heuristics for the undirected RPP. These typically produce optimal or near-optimal solutions on benchmark instances. The authors show that when adapted to an arc routing context, simple operations such as insertions, deletions and exchanges, so common in TSP heuristics, become intricate and hard to visualize.

The first integer linear programming formulation for the RPP was proposed by Christofides et al. (1981). Its variables represent the number of copies of each edge that must be introduced into the graph in order to make it Eulerian. It was solved by branch-and-bound, using Lagrangean relaxation to compute lower bounds, and applied to instances with $9 \leq |V| \leq 84$, $13 \leq |E| \leq 184$, and $4 \leq |E'| \leq 78$. An alternative formulation was later proposed by Sanchis (1990) and Corberán and Sanchis (1994). These authors conducted a polyhedral analysis of their formulation and developed a branch-and-cut code. They solved to optimality, at the root of the search tree, 23 of the 24 instances of Christofides et al. (1981), as well as two new instances. This line of research was pursued by Letchford (1996) and by Ghiani and Laporte (2000) who identified new classes of valid inequalities and integrated them within branch-and-cut algorithms. In the Ghiani and Laporte article, several classes of instances, involving up to 350 vertices, are solved to optimality.

3.2. THE DIRECTED RPP

As shown by Christofides et al. (1986), a heuristic for the directed RPP can be constructed as for the undirected case, by computing a shortest (directed) arborescence between the connected components, instead of a shortest spanning tree. Copies of some arcs are then introduced by solving a transportation problem, as for the directed CPP. No worst-case guarantee has been derived for this heuristic.

An integer linear programming formulation was developed by Christofides et al. (1986), along the lines of the undirected case. Again, the algorithm uses Lagrangean relaxation to compute lower bounds within a branch-and-bound scheme. The algorithm was used to solve to optimal-

ity 23 out of 24 instances with $13 \leq |V| \leq 80$, $24 \leq |E| \leq 180$, and $7 \leq |E'| \leq 74$.

3.3. THE MIXED RPP

Research on the mixed RPP is relatively new and scarce. We are aware of only one contribution, by Corberán, Marti and Romero (2000). It describes a tabu search heuristic which has been applied to instances with $20 \leq |V| \leq 220$, $20 \leq |E| \leq 220$, $5 \leq |E'| \leq 150$, $55 \leq |A| \leq 350$, and $5 \leq |A'| \leq 200$. The deviation of the solution value to that of a lower bound ranges between 0% and 1.35%. The lower bounds were obtained by means of a constraint generation algorithm.

4. THE CAPACITATED ARC ROUTING PROBLEM

In the Capacitated Arc Routing Problem, a nonnegative quantity q_{ij} is associated with each edge or arc (v_i, v_j). A fleet of m vehicles, each having a capacity Q, must traverse all edges or arcs of the graphs and collect (or deliver) the associated quantities, without ever exceeding Q. As in the standard Vehicle Routing Problem, the number of vehicles may be given a priori or can be a decision variable. The CARP was introduced by Golden and Wong (1981), but a variant in which all q_{ij} are strictly positive was investigated earlier by Christofides (1973). In other words, the CARP studied by Golden and Wong and the majority of subsequent researchers can be viewed as a capacity constrained RPP with m vehicles, whereas the problem defined by Christofides is a capacity constrained CPP with m vehicles.

Between 1973 and 1991, several researchers proposed heuristics for the CARP based on various edge or arc partitioning criteria and on tour construction methods. Perhaps the best known methods are the construct-strike algorithm (Christofides 1973), later improved by Pearn (1989), and the augment-insert algorithm (Pearn, 1991). The modified construct-strike algorithm is best adapted to dense graphs (70% to 100% edge or arc density). It gradually constructs feasible cycles and removes them from the graph. When feasible cycles can no longer be found, an Eulerian cycle is constructed on the remaining graph and the search for feasible cycles is repeated. Augment-insert first inserts edges or arcs in feasible cycles connected to the depot, as long as this is possible. Then the remaining links are inserted into cycles by using a savings criterion. This algorithms works best on sparse graphs (up to 30% edge or arc density). The main drawbacks of all heuristic developed until 1991 is that they contain little or no post-optimization. Improving a feasible solution in an arc routing context is indeed difficult as we mentioned

earlier. Recently, the post-optimization tools developed by Hertz, Laporte and Nanchen-Hugo (1999) for the undirected RPP were embedded within a tabu search algorithm for undirected CARPs (Hertz, Laporte and Mittaz, 2000). Tests performed on benchmark instances showed that running a truncated version of this algorithm, even for just one second, outperformed all previously known heuristics for the CARP. Further tests showed that running the full tabu search algorithm consistently yields optimal or near-optimal solutions on benchmark instances.

Assessing the quality of a heuristic is possible only if good lower bounds are available. The earlier lower bounds proposed by Golden and Wong (1981), Assad, Pearn and Golden (1987) and Benavent et al. (1992) are in general too weak to provide any meaningful information. It is only recently that stronger bounds have been derived. These are computed by formulating the CARP as an integer linear program, solving its linear relaxation, and generating strong valid inequalities. A fine example of this approach is provided by Belenguer and Benavent (1998). Comparing the lower bounds obtained from the linear relaxation with upper bounds given by tabu search heuristics shows gaps typically below 1%, which speaks highly for the quality of both the lower and upper bounding approaches.

5. RESEARCH OUTLOOKS

Arc routing has a long and rich history, but only in recent years have we started to witness the widespread use of software in the area of arc routing, and we may only have seen the tip of the iceberg. More and more municipalities, regional authorities, post office administrations, electricity and gas companies, school bus operators, etc. are adopting such systems. This phenomenon is driven in part by a number of technological factors, such as breakthroughs in the micro-computer industry and in data base processing, but also by the ever increasing need to be competitive and cost-efficient. This growth has been paralleled by the development of a number of powerful optimization techniques. The two most important are probably tabu search in the area of heuristics, and branch-and-cut for exact optimization. We expect much of the research growth, in the coming years, to be based on these two techniques. Not all recent scientific discoveries have yet found their way into commercial computer software, but it should be only a matter of time before this materializes.

Acknowledgment

This work was partially supported by the Canadian Natural Sciences and Engineering Research Council under grants OGP0009160 and OGP0039682 This support is gratefully acknowledged. Thanks are also due to a ref-

eree for making several valuable suggestions, and to Kenneth Rosing and Catherine Roucairol for providing some hard to find references.

References

[1] Alfa, A.S., and D.Q. Liu. 1988. Postman Routing Problem in a Hierarchical Network. *Engineering Optimization* 14, 127-138.

[2] Assad, A.A., and B.L. Golden. 1995. Arc Routing Methods and Applications. In Network Routing, M.O. Ball, T.L. Magnanti, C.L. Monma, and G.L. Nemhauser (eds.), Handbooks in Operations Research and Management Science. North-Holland, Amsterdam, 375-483.

[3] Assad, A.A., W.L. Pearn and B.L. Golden. 1987. The Capacitated Chinese Postman Problem: Lower Bounds and Solvable Cases. *American Journal of Mathematics and Management Science* 7, 63-88.

[4] Benavent, E., V. Campos, A. Corberán and E. Mota. 1992. The Capacitated Arc Routing Problem. Lower Bounds. *Networks* 22, 669-690.

[5] Belenguer, J.M. and E. Benavent. 1998. A Cutting Plane Algorithm for the Capacitated Arc Routing Problem.Submitted for publication.

[6] Beltrami, E.L., and L.D. Bodin. 1974. Networks and Vehicle Routing for Municipal Waste Collection. *Networks* 4, 65-94.

[7] Bodin, L.D., and S.J. Kursh. 1978. A Computer-Assisted System for the Routing and Scheduling of Street Sweepers. *Operations Research* 26, 525-537.

[8] Brucker, P. (1981) The Chinese postman problem for mixed graphs. *Proc. Int. Workshop, Lecture Notes in Computer Science* 100, 354-366.

[9] Christofides, N. 1973. The Optimum Traversal of a Graph. *Omega* 1, 719-732.

[10] Christofides, N. 1976. Worst-Case Analysis of a New Heuristic for the Traveling Salesman Problem. Report No 388, Graduate School of Industrial Administration, Carnegie Mellon University, Pittsburgh.

[11] Christofides, N., E. Benavent, V. Campos, A. Corberán and E. Mota. 1984. An Optimal Method for the Mixed Postman Problem. In System Modelling and Optimization, Lecture Notes in Control and Information Sciences 59, P. Thoft-Christensen (ed.). Springer, Berlin.

[12] Christofides, N., V. Campos, A. Corberán and E. Mota. 1981. An Algorithm for the Rural Postman Problem. Imperial College Report. I.C.O.R.81.5, London.

[13] Christofides, N., V. Campos, A. Corberán and E. Mota. 1986. An Algorithm for the Rural Postman Problem on a Directed Graph. *Mathematical Programming Study* 26, 155-166.

[14] Corberán, A., R. Marti, and A. Romero. 2000. A Tabu Search Algorithm for the Mixed Rural Postman Problem. *Computers & Operations Research* 27, 183-203.

[15] Corberán, A., and J.M. Sanchis. 1994. A Polyhedral Approach to the Rural Postman Problem. *European Journal of Operational Research* 79, 95-114.

[16] Dror, M., H. Stern and P. Trudeau. 1987. Postman Tour on a Graph With Precedence Relation on Arcs. *Networks* 17, 283-294.

[17] Edmonds, J., and E.L. Johnson. 1973. Matching, Euler Tours and the Chinese Postman Problem. *Mathematical Programming* 5, 88-124.

[18] Eiselt, H.A., M. Gendreau, and G. Laporte. 1995. Arc Routing Problems, Part I: The Chinese Postman Problem. *Operations Research* 43, 231-242.

[19] Eiselt, H.A., M. Gendreau, and G. Laporte. 1995. Arc Routing Problems, Part II: The Rural Postman Problem. *Operations Research* 43, 399-414.

[20] Euler, L. 1736. Solutio Problematis ad Geometrica Situs Pertinentis. *Commentarii academiae scientarum Petropolitanae* 8, 128-140.

[21] Euler, L. (J.R. Newman, Ed.). 1953. Leonhard Euler and the Koenigsberg Bridges. *Scientific American* 189, 66-70.

[22] Even, S. 1979. Graph Algorithms. *Computer Science Press,* Rockville.

[23] Fleischner, H. 1990. Eulerian Graphs and Related Topics (Part 1, Volume 1), *Annals of Discrete Mathematics* 45. North-Holland, Amsterdam.

[24] Fleischner, H. 1991. Eulerian Graphs and Related Topics (Part 1, Volume 2), *Annals of Discrete Mathematics* 45. North-Holland, Amsterdam.

[25] Fleury, M. 1885. Deux Problèmes de Géométrie de Situation. *Journal de Mathématiques Élémentaires*, 157.

[26] Ford, L.R. and D.R. Fulkerson. 1962. Flows in Networks. Princeton University Press, Princeton, N.J.

[27] Frederickson, G.N. 1979. Approximation Algorithms for Some Postman Problem. *Journal of the ACM* 26, 538-554.

[28] Ghiani, G., and G. Improta. 2000. An Algorithm for the Hierarchical Chinese Postman Problem. *Operations Research Letters.* Forthcoming.

[29] Ghiani, G., and G. Laporte. 2000. A Branch-and-Cut Algorithm for the Undirected Rural Postman Problem. *Mathematical Programming.* Forthoming.

[30] Golden, B.L., and R.T. Wong. 1981. Capacitated Arc Routing Problems. *Networks* 11, 305-315.

[31] Guan, M. 1962. Graphic Programming Using Odd and Even Points. *Chinese Mathematics* 1, 273-277.

[32] Haslam, E., and J.R. Wright. 1991. Application of Routing Technologies to Rural Snow and Ice Control. *Transportation Research Record* 1304, 202-211.

[33] Hertz, A., Laporte, G., and P. Nanchen-Hugo. 1999. Improvement Procedures for the Undirected Rural Postman Problem. *INFORMS Journal on Computing* 11, 53-62.

[34] Hertz, A., Laporte, G., and M. Mittaz. 2000. A Tabu Search Heuristic for the Capacitated Arc Routing Problem. *Operations Research.* Forthcoming.

[35] Hierholzer, C. 1873. Über die Möglichkeit, einen Linienzug ohne Wiederholung und ohne Unterbrechung zu umfahren. *Mathematische Annalen* VI, 30-32.

[36] König, D. 1936. Theorie der endlichen und unendlichen Graphen . Akademische Verlagsgesellschaft, Leipzig.

[37] Lemieux, P.F., and L. Campagna. 1984. The Snow Ploughing Problem Solved by a Graph Theory Algorithm. *Civil Engineering Systems* 1, 337-341.

[38] Lenstra, J.K., and A.H.G. Rinnooy Kan. 1976. On General Routing Problems. *Networks* 6, 273-280.

[39] Letchford, A.N., and R.W. Eglese. 1998. The Rural Postman Problem with Deadline Classes. *European Journal of Operational Research* 105, 390-400.

[40] Letchford, A.N. 1996. Polyhedral Results for Some Constrained Arc-Routing Problems. Ph.D. Thesis, Department of Management Science, Lancaster University.

[41] Lucas, M.É. 1894. Récréations Mathématiques IV . Gauthiers-Villars et fils, Paris.

[42] Manber, U., and S. Israni. 1984. Pierce Point Minimization and Optimal Torch Path Determination in Flame Cutting. *Journal of Manufacturing Systems* 3, 81-89.

[43] Minieka, E. 1979. The Chinese Postman Problem for Mixed Networks. *Management Science* 25, 643-648.

[44] Nobert, Y., and J.-C. Picard. 1996. An Optimal Algorithm for the Mixed Chinese Postman Problem. *Networks* 27, 95-108.

[45] Orloff, C.S. 1974. A Fundamental Problem in Vehicle Routing. *Networks* 4, 35-64.

[46] Papadimitriou, C.H. 1976. On the Complexity of Edge Traversing. *Journal of the ACM* 23, 544-554.

[47] Pearn, W.-L. 1989. Approximate Solutions for the Capacitated Arc Routing Problem. *Computers & Operations Research* 16, 589-600.

[48] Pearn, W.-L. 1991. Augment-Insert Algorithms for the Capacitated Arc Routing Problem. *Computers & Operations Research* 18, 189-198.

[49] Sanchis, J.M. 1990. El Poliedro del Problema del Cartero Rural. Ph.D. Thesis, Universidad de Valencia, Spain.

[50] Stricker, R. 1970. Public Sector Vehicle Routing: The Chinese Postman Problem. M.Sc. Dissertation, Department of Electrical Engineering, Massachusetts Institute of Technology, Cambridge, Mass.

[51] van Aardenne-Ehrenfest, T., and N.G. de Bruijn. 1951. Circuits and Trees in Oriented Linear Graphs. *Simon Stevin* 28, 203-217.

[52] Win, Z. 1987. Contributions to Routing Problems. Doctoral Dissertation, Universität Augsburg, Germany.

[53] Win, Z. 1989. On the Windy Postman Problem in Eulerian Graphs. *Mathematical Programming* 44, 97-112.

I

THEORY

Chapter 2

TRAVERSING GRAPHS: THE EULERIAN AND HAMILTONIAN THEME

Herbert Fleischner
Institute for Discrete Mathematics, Austrian Academy of Sciences

1. Introductory Remarks 19
2. Basics of Graph Theory 20
   2.1 Graphs and Their Parts 20
   2.2 Walks, Trails, Paths, Cycles; Connectedness 23
   2.3 Bipartite Graphs, Trees, Blocks, Mappings 27
3. Connectivity, Menger's Theorem, the Splitting Lemma, and Factors 32
4. Eulerian Graphs and Covering Walks, Cycle Decompositions and Cycle Covers 40
   4.1 Algorithms for Constructing Eulerian Trails 47
   4.2 Mazes 48
5. Hamiltonian Cycles and Vertex-Covering Walks 50
6. Elements of Matching Theory 58
7. The Chinese Postman Problem, The Traveling Salesman Problem, and Related Problems 69
8. Elements of Network Theory 77

1. INTRODUCTORY REMARKS

This chapter deals with basic graph theoretical concepts and then focuses on special topics which are – in their applications – of relevance to theoretical and practical problems in OR. Therefore, this chapter is structured as follows:

1 Basics of Graph Theory.

2 Connectivity, Menger's Theorem, the Splitting Lemma, and Factors.

3 Eulerian Graphs and Covering Walks, Cycle Decompositions and Cycle Covers.

4 Hamiltonian Cycles and Vertex-Covering Walks.

5 Elements of Matching Theory.

6 The Chinese Postman Problem, The Traveling Salesman Problem, and Related Problems.

7 Elements of Network Theory.

If treated in all details, these seven topics would cover several volumes. In fact, eulerian graphs and corresponding applications alone are subject of a monograph comprising three volumes two of which have already been published some years ago [16]. Therefore, in order to keep the size of this chapter in accordance with the overall size of this collection of survey articles, we will restrict ourselves in presenting proofs of theorems and justifications of algorithms to the cases where these results are of central importance for this chapter. Nonetheless we will mention in various instances developments in different directions, in order to give the reader an indication of the overall development of this area of graph theory.

2. BASICS OF GRAPH THEORY

2.1. GRAPHS AND THEIR PARTS

We write a graph G as $G = V \cup E$, where V is a finite set and E is a finite multiset. The elements of V are called *vertices* while the elements of E are called *edges*. Edges are unordered pairs of not necessarily distinct vertices from V. We call $V =: V(G)$ the vertex set of G, while $E =: E(G)$ is called the edge set of G. Vertices joined by an edge are called *adjacent*. They are also called the *ends* of the edge. An edge is said to be *incident* to its ends.

If $e \in E(G)$ is joining x and y, then we may denote e as xy. A *loop* of G is an edge of the form $vv, v \in V(G)$. The number of edges joining the pair u, v of vertices is the *multiplicity* $\lambda(uv)$ of the edges of the form uv. If $\lambda(uv) = k > 1$ then the edges $e_1, e_2, \ldots, e_k$ joining u, v are called *parallel edges*. A graph with no loops and no parallel edges is called a *simple* graph.

Likewise, a digraph D is the union of a finite vertex set V and a finite multiset A, called *arc set*, which consists of ordered pairs of vertices, called *arcs*. The vertices joined by an arc $a = (u, v)$ are also said to be adjacent, with u being the tail of a and v being the head of the arc.

A *mixed graph* has both edges and arcs. It is represented in the form $M = V \cup E \cup A$.

Let G be a graph. If we replace each $e = xy \in E(G)$ with an arc $a_e = (x, y)$ or (y, x) then we call $D_G = V(G) \cup \{a_e : e \in E(G)\}$ an *orientation* of G. Conversely, if D is a digraph and we replace every $a = (u, v) \in A(D)$ by the edge $e_a = uv$ then we call $G_D := V(D) \cup \{e_a : a \in A(D)\}$ the graph *underlying* D.

(a) (b)

Figure 2.1 (a) a graph G, (b) an orientation D_G of G.

The set E_v denotes the set of edges incident with v. $\lambda_v := \lambda(vv)$ is the number of loops at v. The sum $|E_v| + \lambda_v$ is called the *degree* $d(v)$ of v. Notice that this definition means that we count each loop incident with v twice. A vertex of odd degree is called an *odd vertex* while one of even degree is an *even vertex*. $y \in V(G)$ is a *neighbor* of $x \in V(G)$ if $xy \in E(G)$; the set of neighbors of x is denoted by $N(x)$, whereas $\overline{N}(x) := N(x) \cup \{x\}$. Observe that $N(x) = \overline{N}(x)$ if and only if $xx \in E(G)$. A simple counting argument leads to the following relationship, which implies that the number of odd vertices in a graph is even.

$$\sum_{v \in V(G)} d(v) = 2|E(G)|$$

(this equation is also known as the *Handshaking Lemma*).

Define a *k-valent* vertex as a vertex of degree k. A 1-valent vertex is called a *terminal vertex* or *endvertex* while a 0-valent vertex is called an *isolated vertex*. The *maximum degree* $\Delta(G)$ of a graph G is defined to be $\Delta(G) := \max_{v \in V(G)} d(v)$. Similarly we define the *minimum degree* of a graph G to be $\delta(G) := \min_{v \in V(G)} d(v)$. Finally, we call $p := |V(G)|$ and $q := |E(G)|$ the *order* and *size*, respectively, of G.

In the case of digraphs, we set A_v^- to be the set of arcs having v as head and A_v^+ to be the set of arcs having v as tail. Then the *in-degree* $d^-(v) = |A_v^-|$ and the out-degree $d^+(v) = |A_v^+|$. We have the following relationship:

$$\sum_{v \in V(D)} d^+(v) = \sum_{v \in V(D)} d^-(v) = |A(D)|$$

A vertex v of a digraph D is a *source* if its in-degree is zero. It is a *sink* if its out-degree is zero. Order and size of a digraph are defined as in the case of graphs.

We often deal with parts of a graph $G = V \cup E$. Given $V_1 \subseteq V$ and $E_1 \subseteq E$, we consider $G_1 = V_1 \cup E_1$; it is a graph if for every $e_1 \in E_1$ the ends of e_1 are in V_1. If so, then G_1 is a *subgraph* of G and G is a *supergraph* of G_1. If $V_1 = V$ then we say G_1 is a *spanning subgraph* of G. If a vertex set consists of a singleton v we may often just write v instead of $\{v\}$. Similarly for singleton edge-sets.

A subgraph $G_1 = V_1 \cup E_1 \subseteq G$ is called a *vertex-induced* subgraph of G if $E_1 = \{v_1v_2 \in E : v_1, v_2 \in V_1\}$. We write $G_1 = \langle V_1 \rangle_G$. We may delete the subscript G if it is clear from the context. So for instance, for $V_1 \subseteq V$ we define $G - V_1 = \langle V - V_1 \rangle$. If $V_1 = \{v\}$ then $G - v := \langle V - v \rangle$.

Given $E_1 \subseteq E$, we call a subgraph $G_1 = V_1 \cup E_1 \subseteq G$ *edge-induced* if $v \in V_1$ implies that at least one edge in E_1 is incident with v. We say $G_1 = \langle E_1 \rangle$. However, $G - E_1 = V \cup (E - E_1)$ may not be an edge-induced graph, unlike its vertex counterpart.

The difference between a graph G and any of its subgraphs G_1 is defined by

$$G - G_1 = G - E(G_1) - \{v \in V(G) : d_G(v) = d_{G_1}(v)\}$$

(that is, any isolated vertices caused by the removal of the edges of the subgraph are deleted). Given $V_0 \subseteq V(G)$ we can set $\bar{V_0} = V(G) - V_0$. Then the *coboundary* of V_0 (or just an *edge cut* of G) is the edge set $E(V_0, \bar{V_0}) = \{e = xy : x \in V_0, y \in \bar{V_0}\}$. If $\langle V_0 \rangle, \langle \bar{V_0} \rangle$ are both connected then the coboundary of V_0 is also called a *cocycle* of V_0. Note that the latter concept is often called cocircuit in which case the term cocycle is used to denote an edge cut.

The *complement* $\bar{G}$ of a graph G is defined by

$$V(\bar{G}) = V(G),$$

$$E(\bar{G}) = \{xy : x, y \in V(G), xy \notin E(G)\}.$$

In the case where G is of order n and size 0, we denote $K_n := \bar{G}$ and call K_n the *complete graph* on n vertices.

A vertex v of a graph G is called a *cutvertex* if G can be written in the form $G = G_1 \cup G_2$ where G_1, G_2 are subgraphs of G with $G_1 \cap G_2 = v$ such that $d_{G_1}(v) \neq 0, d_{G_2}(v) \neq 0$. This definition is equivalent to the classical definition of a cutvertex in the case of loopless graphs, but is different for graphs with loops. However, Bondy's definition of a cutvertex, [7, p. 10] coincides, basically, with the one given here.
Note that (considering Figure 2.2) cutvertex v would still be a cutvertex

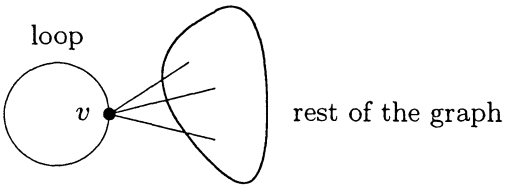


Figure 2.2 A cutvertex in a graph with a loop

if we placed another vertex on the loop.

An edge of G is a *bridge* if $G = G_1 \cup G_2 \cup e$, where G_1, G_2 are disjoint nonempty subgraphs of G such that one end of e is in G_1 and the other end is in G_2.

Just as a bridge is a special case of an edge cut, a cutvertex is a special case of a vertex cut $S \subseteq V(G)$ which is defined for loopless G by nonempty subgraphs $G_1, G_2 \subseteq G$ and

$$G = G_1 \cup G_2 \cup \langle S \rangle \cup E(S, \overline{S}), \quad G_1 \cap G_2 = \langle S \rangle \cap G_i = \emptyset,$$

$i = 1, 2$ (see [7, p. 34]). Observe, finally, that for any bridge $xy \in E(G)$, x is a cutvertex of G if and only if $d(x) > 1$.

2.2. WALKS, TRAILS, PATHS, CYCLES; CONNECTEDNESS

Given a graph $G = V \cup E$ or a digraph $D = V \cup A$, a *walk* $W = W(v_0, v_n) = v_0, e_1, v_1, \ldots, e_n, v_n$ from v_0 to v_n is an alternating sequence of vertices and edges of G, respectively arcs of D, so that v_{i-1} and v_i are the ends of $e_i \in E$, respectively v_{i-1} is the tail and v_i is the head of $e_i \in A$ for $1 \leq i \leq n$. The number of times an edge/arc e appears in a walk is termed its *multiplicity* $\lambda_W(e)$. If $v_0 = v_n$ then W is called a *closed walk*, otherwise it is an *open walk*.

If no edge/arc appears more than once in W then it is called a (open, closed respectively) *trail.* If no vertex appears more than once in an open trail then it is called a *path.* A closed trail without repetition of vertices (except $v_0 = v_n$) and with at least one edge/arc is called a *cycle.* The inverse W^{-1} of a walk W is a listing of W in reverse order.

In a walk $W(v_0, v_n)$ from v_0 to v_n in a digraph, the arcs are traversed according to their orientation. A sequence where arcs are traversed in any direction regardless of their orientation is called a *chain.* A chain is a *simple chain* if none of its vertices (except possibly the first and the last) are repeated. The inverse W^{-1} of a walk W in a digraph is not a walk but it is a chain.

The *length* of a walk/trail/path/cycle/chain W is the number of edges/arcs in the corresponding sequence and will be denoted by $l(W)$. Correspondingly we will call W even/odd if $l(W)$ is even/odd.

In what follows we will not distinguish between (the various types of) a walk W and the subgraph induced by the edges/arcs of W; this especially applies to the cases where W is a trail, path or cycle.

If a path $P(x, y)$ exists for $x, y \in V(G)$, then the *distance* $d(x, y) := \min_{P(x,y)}\{l(P(x, y))\}$. If such a path does not exist, then we set $d(x, y) = \infty$.

Theorem 1 *If $W(v_0, v_n)$ is an open walk from v_0 to v_n then there is a subsequence $P(v_0, v_n)$ of $W(v_0, v_n)$ such that $P(v_0, v_n)$ is a path from v_0 to v_n.*

It is not true however, for a closed walk W in a graph that there is a subsequence of W which is a cycle: just consider for $e \in E$ having ends $x, y \in V$, the closed sequence x, e, y, e, x; no subsequence of it is a cycle (note that a cycle contains at least one edge/arc). On the other hand, a closed walk in a digraph contains a subsequence which is a cycle containing any given vertex of the original closed walk.

A graph $G = V \cup E$ is called *connected* if for every $x, y \in V$, there is a $P(x, y)$ (or equivalently, a $W(x, y)$). A graph which is not connected is called *disconnected.* A subgraph G' of a graph G is called a *component* of G if G' is a maximal connected subgraph (maximal w.r.t. set inclusion). $c(G)$ denotes the number of components of G.

Theorem 2 *A graph $G = V \cup E$ is connected if and only if for any fixed $x \in V$ and every $y \in V$ there is a $P(x, y)$.*

Theorem 3 *Let $\sim$ be the equivalence relation defined on $V(G)$ by $x \sim y \iff$ there is a $P(x,y)$, and let $C_1, \ldots, C_k$ $(k \geq 1)$ be the equivalence classes of $\sim$. Then $\langle C_i \rangle, 1 \leq i \leq k$ is a component of G and $G = \bigcup_{i=1}^{k} \langle C_i \rangle$.*

Corollary 4 *A graph G is disconnected if and only if $G = G_1 \dot{\cup} G_2$ where $G_1 \neq \emptyset \neq G_2$, and G_1, G_2 are subgraphs of G.*

Note that the symbol $\dot{\cup}$ refers to the *disjoint union* of two sets.

Theorem 5 *Let G be a graph and $v \in V(G)$. Then v is a cutvertex if and only if either $vv \in E(G)$ and $d_G(v) \geq 3$ or $G - v$ has more components than G.*

Proof. ($\Rightarrow$) Suppose v is a cutvertex. Then $G = G_1 \cup G_2$ where $G_1 \cap G_2 = v$ and $d_{G_i}(v) > 0, i = 1, 2$. If $|V(G_i)| = 1$ for at least one $i = 1, 2$ then $vv \in E(G_i)$. Thus $d_{G_i}(v) \geq 2$ and since $d_{G_j}(v) \geq 1$ $(j = 3 - i)$ we have $d_{G_1}(v) + d_{G_2}(v) = d_{G(v)} \geq 3$. Whence suppose G contains no loop of the form vv.

Case 1: G is connected. Then $G - v = (G_1 - v) \dot{\cup} (G_2 - v)$ and both $G_1 - v$ and $G_2 - v$ are nonempty. By Corollary 4, $G - v$ is disconnected, i.e. $G - v$ has at least two components, which is more than connected G had (namely 1).

Case 2: G is disconnected, i.e. $G = \dot{\bigcup}_{i=1}^{k} G_i'$ $(k \geq 2)$ and each G_i' is a component of G. v is in exactly one component, without loss of generality $v \in V(G_1')$. Then $G - v = (G_1' - v) \dot{\cup} (\dot{\bigcup}_{i=2}^{k} G_i')$. G_1' is connected so $G_1' - v$ has more components than G_1' (by case 1). Thus $G - v$ has more components than G.

($\Leftarrow$) If $vv \in E(G)$ and $d_G(v) > 2$, set $G_1 = \{v\} \dot{\cup} \{e\}$ (where $e = vv$) and $G_2 = G - e$. It follows that $G_1 \cap G_2 = v$. Since $d_{G_1}(v) = 2$ and $d_G(v) \geq 3$ we must have $d_{G_2}(v) \geq 1$, meaning that v is a cutvertex. Suppose $G - v$ has more components than G. $G = \dot{\bigcup}_{i=1}^{l} G_i$, where each G_i $(1 \leq i \leq l)$ is a component of G. W.l.o.g. we suppose that $v \in V(G_1)$. So $G - v = (G_1 - v) \dot{\cup} (\dot{\bigcup}_{i=2}^{l} G_i)$. Setting $G_1^* = G_1 - v$, $G_i^* = G_i$ $(2 \leq i \leq l)$ we have $G - v = \dot{\bigcup}_{i=1}^{l} G_i^*$. Note that G_i^* is a connected subgraph of G for $2 \leq i \leq l$.

If there is a connected graph $G_i^{**} \supseteq G^*$ such that G_i^* is a proper subgraph of G_i^{**} (for some $2 \leq i \leq l$) then G_i^{**} contains an edge f not in G_i^*. This edge must be in some other G_j^*, meaning that $G_j^* \cap G_i^* \neq \emptyset$, a contradiction. Thus each G_i, $2 \leq i \leq l$ is maximal and thus a component

of $G - v$. So G_1^* has at least two components $G_{1,1}^*, \ldots, G_{1,s}^*$. Defining $G_1'' := \langle V(G_{1,1}^*) \cup \{v\}\rangle_G$ and $G_2'' := \bigcup_{i=2}^{l} G_i \cup (G_1 - V(G_{1,1}^*))$ we have $G_1'' \cap G_2'' = \{v\}$ and $G_1'' \cup G_2'' = G$. Since $d_{G_1''}(v) \neq 0 \neq d_{G_2''}(v)$, v is a cutvertex of G. ■

Corollary 6 *A vertex v of a loopless connected graph G is a cutvertex if and only if $V(G) - v$ can be written in the form $V_1 \dot\cup V_2$ where V_1, V_2 are nonempty sets such that for every $v_1 \in V_1, v_2 \in V_2$ it follows that every path in G from v_1 to v_2 contains v.*

Bridges of a graph can be characterized similarly.

Theorem 7 *For any graph G and $e \in E(G)$, e is a bridge of G if and only if $G - e$ has exactly one component more than G, i.e., $c(G - e) = c(G) + 1$.*

Proof. ($\Rightarrow$) e is a bridge of G so $G = G_1 \cup G_2 \cup e$ where $e = xy$, $x \in G_1$, $y \in G_2, G_1 \cap G_2 = \emptyset$, $G_1 \neq \emptyset \neq G_2$. Let G' be the component of G containing $e = xy$ and thus also x, y. Define $G_1' := (G' - e) \cap G_1$, $G_2' := (G' - e) \cap G_2$. So $x \in G_1', y \in G_2'$ and $G' - e = G_1' \cup G_2'$. If $G' - e$ is connected then it has a $P(x, y)$. This path must contain an edge $f = uv \neq e$ such that $u \in G_1', v \in G_2'$. So $f \notin G_1' \cup G_2' = G' - e$, a contradiction. So $c(G' - e) \geq 2$. If $G' - e$ has (at least) three components G_1^*, G_2^*, G_3^* then one of them, say G_1^*, contains neither x nor y. Adding e back to $G' - e$ will not affect G_1^*, which therefore remains disconnected from the rest of G', contradicting our assumption that G' was connected. So $c(G' - e) = 2$ and $c(G - e) = c((G - e) - (G' - e)) + c(G' - e) = c(G - G') + c(G' - e) = (c(G) - 1) + 2 = c(G) + 1$.

($\Leftarrow$) Let G' be the component of G containing e. Since all other components of G are unaffected by the removal of e and $c(G' - e) = c(G') + 1 = 2$, therefore $G' - e = G_1' \cup G_2'$, where G_1' and G_2' are the components of $G' - e$. W.l.o.g. $x \in G_1', y \in G_2'$ for $e = xy$. Thus $G - e = G_1 \cup G_2$ where $G_1 = G_1'$ and $G_2 = G_2' \cup (G - G')$ are both nonempty graphs with nothing in common since they are both unions of different components. Thus $G = G_1 \cup G_2 \cup e$, and so e is a bridge by definition. ■

Corollary 8 *Let G be a connected graph. Then $e \in E(G)$ is a bridge if and only if $V(G) = V_1 \dot\cup V_2$ such that every path joining any $v_1 \in V_1$ to any $v_2 \in V_2$ contains e.*

The next criterion characterizes bridges visavis cycles in graphs.

Theorem 9 *$e \in E(G)$ is a bridge if and only if no cycle of G contains e.*

However, in the case of digraphs D one distinguishes between three types of connectedness. Namely:

D is *strongly connected* if for all $x, y \in V(D)$ there exist paths $P(x,y)$ and $P(y,x)$.
D is *unilaterally connected* if for all $x, y \in V(D)$ either $P(x,y)$ or $P(y,x)$ exists.
D is *weakly connected* if its underlying graph G_D is connected.

A strongly/unilaterally/weakly connected component of D is a maximally strongly/unilaterally/weakly connected subdigraph of D. On the other hand, D is said to be *disconnected* if G_D is disconnected.

Evidently, a strongly connected digraph is also unilaterally connected, and a unilaterally connected digraph is also weakly connected. The converse is not true, in general. However, a weakly connected digraph is strongly connected if every arc belongs to a cycle.

Also, the weakly connected components of D correspond by definition bijectively to the components of G_D. The corresponding equivalence relation is defined by $x \sim y$ if and only if there is a $P(x,y)$ in G_D. In contrast, if we define w.r.t strong connectedness $x \sim y$ if and only if paths $P(x,y)$ and $P(y,x)$ exist in D, then this also defines a partition of $V(D)$. We observe that the latter equivalence relation is a refinement of the former. Unfortunately, unilateral connectedness does not give rise to an equivalenc relation analogous to the other two cases, reason being that the corresponding relation is not transitive, in general (while $P(x,y), P(z,y) \subseteq D$ may hold, D may not contain $P(x,z)$ nor $P(z,x)$.

We note in passing that a graph G has a strongly connected orientation D_G if and only if G is connected and bridgeless; this result has become known as Robbin's Theorem (see, e.g., [35, p. 8].)

2.3. BIPARTITE GRAPHS, TREES, BLOCKS, MAPPINGS

A bipartite graph is one whose vertex set can be partitioned into two sets such that edges join vertices of different sets only.

We present a useful characterization of such graphs.

Theorem 10 *A graph G is bipartite if and only if G contains no cycles of odd length.*

Proof. ($\Rightarrow$) Suppose G is bipartite with bipartition $V(G) = V_1 \dot{\cup} V_2$ and has a cycle $C = x_0, e_1, x_1, \ldots, e_n, x_n = x_0$. W.l.o.g. we assume $x_0 \in V_1$.

Then $x_1 \in V_2$, $x_2 \in V_1, \ldots, x_n = x_0 \in V_1$, hence n is even. Thus C is an even cycle.

($\Leftarrow$) W.l.o.g. assume G is connected (otherwise consider its components). Fix any vertex v in G and mark it red. Mark all vertices adjacent to v blue. Consider all unmarked vertices adjacent to a blue vertex (none are adjacent to red vertices since all neighbours of red vertices have been marked already) and mark them red. Continue this procedure with blue in place of red. Note that by this marking procedure a vertex marked red, say, cannot be adjacent to a previously marked vertex unless that vertex was marked in the immediately preceding step.

The marking stops after all elements of $V(G)$ have been marked. For suppose it stopped with $y \in V(G)$ unmarked; then, since G is connected a path $P(v,y) = P$ exists. Let $w \neq y$ be the last marked vertex in P. Then the vertex v', the successor of w in P, is marked, a contradiction.

We now set $A = \{\text{vertices marked red}\}$ and $B = \{\text{vertices marked blue}\}$. Say there is an edge $f = a_1a_2 \in E(G)$ with $a_1, a_2 \in A$. Then a_1, a_2 have been marked red in the same step of the marking procedure. Observe that the marking procedure produces at each step a path from v to each of the newly marked vertices. Each path is colored alternating red-blue-red-blue- ... This applies in particular to the respective paths $P(v,a_1), P(v,a_2)$. Let x be the last vertex (possibly v itself) that belongs to both paths as one walks from v to a_1 in $P(v,a_1)$. Since $P(x,a_1), P(x,a_2)$ are paths of equal length with singleton intersection $\{x\}$, $P(x,a_1), a_1a_2, P^{-1}(x,a_2)$ is a cycle of odd length. This contradiction finishes the proof of the theorem. ■

Following the above definition of the complement of a graph, we define for a bipartite graph G with bipartition $V(G) = V_1 \cup V_2$ the *bipartite complement* $\overline{G}^b$ by

$$V(\overline{G}^b) = V(G) = V_1 \dot{\cup} V_2$$

i.e., G and $(\overline{G}^b)$ have the same vertex bipartition,

$$E(\overline{G}^b) = \{\, v_1v_2 : v_1 \in V_1, v_2 \in V_2, v_1v_2 \notin E(G) \,\}.$$

Likewise, if $E(G) = \emptyset$, $V(G) = V_1 \dot{\cup} V_2$ such that $|V_1| = m, |V_2| = n$, then we call $K_{m,n} := \overline{G}^b$ the *complete bipartite graph* on m and n vertices.

A graph G is called *acyclic* or a *forest* if it has no cycles. A connected forest is a *tree*. Likewise, a digraph is called acyclic if it has no cycles.

A digraph D is a forest/tree if its underlying graph is a forest/tree. Since a digraph may be acyclic and yet contain a closed chain, the un-

derlying graph of an acyclic digraph may not be acyclic. In fact, any graph has an acyclic orientation.

Further, D is an *out-tree (in-tree)* if D is a tree and there exists $z \in V(D)$ such that for every $v \in V(D)$ a path $P(z,v)$ (a path $P(v,z)$) exists. z is called in both cases the *root* of D.

Theorem 11 *Every connected graph G contains a spanning tree T, i.e., a tree which is a spanning subgraph of G.*

A proof of Theorem 11 can be derived from the second part of the proof of Theorem 10, by modifying the marking procedure. We also note that a tree is a bipartite graph (since it has no cycles at all – see Theorem 10), and that every edge of a tree is a bridge (see below). The latter observation permits another proof of Theorem 11: one deletes – step by step – edges belonging to a cycle in the subgraph under consideration until one arrives at a (spanning) acyclic subgraph.

Next we present characterization theorems for trees and out-trees.

Theorem 12 *For any graph G of order p and size q, the following are equivalent.*

1 *G is a tree.*

2 *G is loopless and for all $x,y \in V(G)$ there is precisely one path $P(x,y)$.*

3 *G is connected and every edge is a bridge.*

4 *G is connected and $p = q + 1$.*

5 *G is acyclic and $p = q + 1$.*

6 *G is acyclic and for all $x,y \in V(G)$ satisfying $x \neq y$, $xy \notin E(G)$, the new graph $G \cup \{xy\}$ has precisely one cycle (which necessarily contains xy).*

Theorem 13 *Given a digraph D, the following are equivalent.*

1 *D is an out-tree with root z.*

2 *D is weakly connected, $d^-(v) = 1$ for every $v \in V(D) - z, d^-(z) = 0$.*

3 *D is acyclic and $d^-(v) = 1$ for every $v \in V(D) - z, d^-(z) = 0$.*

4 *D is acyclic and contains $z \in V(D)$ such that there is a unique path $P(z,v)$ for every $v \in V(D) - z$.*

In view of Theorems 11 and 13 one can prove the following on the existence of spanning out-trees.

Corollary 14 *If D is a weakly connected digraph with vertex z such that there is a path $P(z,v)$ for every $v \in V(D)$ then D contains a spanning outtree with root z.*

Results for in-trees analogous to Theorem 13 and Corollary 14 can be obtained by corresponding modification.

A graph is called *nonseparable* or simply a *block* if it has no cutvertices, whereas a *block of a graph* G is a maximal nonseparable subgraph B of G (see Figure 2.3). That is, B is a subgraph of G without cutvertices,

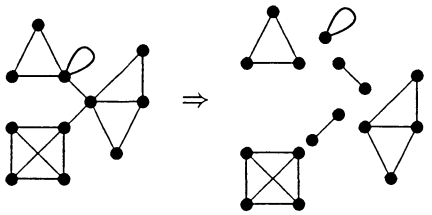


Figure 2.3 A graph and its blocks

and is maximal w.r.t. this property. However, B may contain many vertices which are cutvertices of G. – Correspondingly, a graph is separable if it has at least one cutvertex. Note also that for every bridge xy of G, $\{x, y, xy\}$ is a block of G and that two blocks of G have at most one vertex in common; such a common vertex is necessarily a cutvertex of G.

The concepts of cutvertex and block give rise to defining the *block-cutvertex graph* $bc(G)$ for any graph G; namely: the vertices of $bc(G)$ are in 1-1-correspondence to the blocks and cutvertices of G, and $e \in E(bc(G))$ if and only if one end of e corresponds to a cutvertex x of G and the other end corresponds to a block B of G such that $x \in V(B)$.

The next result describes the global structure of graphs.

Theorem 15 *For any graph G, $bc(G)$ is acyclic and $d(x,y)$ is even for any two endvertices x, y of the same component of $bc(G)$. Conversely, if H is an acyclic graph such that $d(x,y)$ is even for any two endvertices x, y of the same component of H, then there exists a graph G such that $bc(G)$ and H are isomorphic (see the next paragraph for isomorphy). Finally, there is a 1-1-correspondence between the components of G and those of $bc(G)$.*

Two graphs G_1, G_2 are said to be *isomorphic* if there exist bijections $\theta : V(G_1) \to V(G_2)$, $\phi : E(G_1) \to E(G_2)$ such that $\phi(xy) = \theta(x)\theta(y)$ for any edge xy of G_1. The pair (θ, ϕ) is called an *isomorphism*. Furthermore, if $G_1 = G_2$ then an isomorphism is called an *automorphism*.

Isomorphisms between digraphs D_1, D_2 are defined similarly, the main difference being that we replace the defining equation by $\phi((x,y)) = (\theta(x), \theta(y))$ for any arc (x,y) of D_1.

If $\theta : V(G_1) \to V(G_2)$, $\phi : E(G_1) \to E(G_2)$ are (not necessarily 1-1) mappings such that $\phi(xy) = \theta(x)\theta(y)$ then we call (θ, ϕ) a *homomorphism*, which in turn is called an *epimorphism* if $\phi(E(G_1)) = E(G_2)$.

The next concept is of a more topological nature. Suppose we are given a graph G with at least one edge $e = xy$. Let $s \notin V(G)$ be a vertex. Then $G' = (G - e) \cup \{s, xs, sy\}$ is the graph obtained from G by *subdividing* edge e. Two graphs G_1, G_2 are said to be *homeomorphic* if there is G_3 such that both G_1, G_2 can be obtained from G_3 by subdividing edges of G_3 by one or more vertices.

We are now in a position to describe the construction of all nonseparable graphs.

Theorem 16 *Every nonseparable graph G with at least 3 vertices can be obtained from K_3 by a sequence of two operations:*

1 *Adding an edge joining two given vertices.*

2 *Subdivision of an edge by one or more vertices.*

Various properties of nonseparable graphs are expressed by the next theorem which can be proved easily by the use of Menger's Theorem (see below).

Theorem 17 *Let G be a connected graph of order ≥ 3. Then the following statements are equivalent.*

1 *G is nonseparable.*

2 *G is loopless and every two vertices lie on a cycle.*

3 *For every $v \in V(G)$, $e \in E(G)$, there is a cycle containing v, e.*

4 *For every $e_1, e_2 \in E(G)$, there is a cycle containing e_1, e_2.*

5 *Given $x, y \in V(G)$, $e \in E(G)$, there is $P(x,y)$ containing e.*

6 *G is loopless and for every three distinct vertices x, y, z there is $P(x,y)$ containing z.*

7 *G is loopless and for every three distinct vertices* x, y, z *there is* $P(x,y)$ *not containing* z.

Further graph theoretical concepts will be explained in the subsequent sections of this chapter, in the context of various problems dealt with there.

3. CONNECTIVITY, MENGER'S THEOREM, THE SPLITTING LEMMA, AND FACTORS

Suppose G is a given graph with non-adjacent vertices x, y. The *local connectivity* $\kappa = \kappa(x,y)$ is the smallest number of vertices $v_1, \ldots, v_\kappa$ such that $G' = G - \{v_1, \ldots, v_\kappa\}$ is disconnected and x, y are in different components of G'. We say a loopless graph G has *connectivity* $\kappa(G) = n$ if G contains a spanning subgraph isomorphic to K_{n+1}, or $\kappa(G) = \min\{\, \kappa(x,y) : xy \notin E(G) \,\}$. For example, $\kappa(K_{m,n}) = \min\{m, n\}$.

A loopless graph G is called *n-connected* if $\kappa(G) \geq n$. If G is connected with loop vv and $d(v) > 2$ then $\kappa(G) := 1$. Thus, a connected graph G has a cutvertex if and only if $\kappa(G) = 1$.

In what follows call $r \geq 2$ paths *internally disjoint* if any two of these paths have at most their endvertices in common.

Theorem 18 (Menger's Theorem) *Given a graph* G *with non-adjacent vertices* x, y. *Then* $\kappa(x,y)$ *is the maximum number* $\rho(x,y)$ *of internally disjoint paths joining* x *and* y.

Proof. W.l.o.g. we can assume that G is a simple graph. We follow the classic proof of Dirac (see, e.g., [9, 27]).

If $\kappa(x,y) = 0$ then G must be disconnected with x, y in different components and the theorem is true. If $\kappa(x,y) = 1$ then G has a cutvertex separating x, y. Again the theorem holds. Observe that $\kappa(x,y) \geq \rho(x,y)$ in general (every $P(x,y)$ contains at least one element of any vertex cut separating x and y). So we must show that $\rho(x,y) \geq \kappa(x,y)$ holds when $\kappa(x,y) \geq 2$.

Suppose the theorem is false for some $\kappa(x,y) = \kappa \geq 2$. Among all graphs G having non-adjacent vertices x, y such that $\rho(x,y) < \kappa$ choose a graph for which $|E(G)|+|V(G)|$ is minimum. Set $\rho = \rho(x,y)$. Next, consider $G' := G-e$ for any $e \in E(G)$. Let $\rho' = \rho_{G'}(x,y)$ and $\kappa' = \kappa_{G'}(x,y)$. By the minimality of G, $\rho' = \kappa'$. Also, we have $\rho' \geq \rho - 1$. Clearly $\kappa' \leq \kappa$. Suppose $\kappa' = \kappa$. Since there are $\rho' = \kappa$ internally disjoint paths joining x, y in G', there must be at least κ internally disjoint paths joining x, y in G. So $\rho \geq \kappa$, which cannot be by the choice of G. So $\kappa' \leq \kappa - 1$.

Suppose $\kappa' \leq \kappa - 2$. Then let $S' = \{v_1, \ldots, v_{\kappa'}\}$ be a vertex cut separating x, y in G'. Let $x_e \notin \{x, y\}$ be a vertex incident to e. Then $S' \cup \{x_e\}$ is a vertex cut of $\kappa' + 1 \leq \kappa - 1$ elements separating x, y in G, which cannot be. So $\kappa' \geq \kappa - 1$ and hence $\kappa' = \kappa - 1$.

We now claim that there cannot be any vertex $z \in V(G) - \{x, y\}$ adjacent to both x, y in G. If there is, then let $G'' := G - z$. If we set $\rho'' = \rho_{G''}(x, y)$ and $\kappa'' = \kappa_{G''}(x, y)$ then $\rho'' = \kappa''$ by the choice of G. Also x, xz, z, zy, y is a path from x to y in G but not in G''. So $\rho - 1 \geq \rho''$. By the same argument as before, $\kappa'' + 1 \geq \kappa$. Then $\kappa - 1 \leq \kappa'' = \rho'' \leq \rho - 1$, so that $\kappa \leq \rho$. This is a contradiction again.

Let S be a vertex cut of size κ separating x, y in G. W.l.o.g. we may assume G to be a block. By the result of the previous paragraph, no vertex of S can be adjacent to both x and y.

We claim that either x is adjacent to every vertex of S or y is adjacent to every vertex of S. $(*)$

Define $P(x, S)$ to be a path starting in x and ending in some $s \in S$ such that it contains no other vertex of S. Let $P(S, y)$ be defined as a path starting in some $s \in S$, ending in y and containing no other vertex of S. It follows that

(a) every $P(x, y)$ contains a certain $P(x, S)$ as (initial) section and a certain $P(S, y)$ as (final) section (the elements of S contained in these latter two paths may be different);
(b) every $P(x, S)$ and every $P(S, y)$ have at most one vertex in common, and such vertex belongs to S; otherwise $P(x, S), P(S, y)$ contains a subsequence which is a $P(x, y)$ not containing any element of S;
(c) every $s \in S$ appears as endvertex (initial vertex) of some $P(x, S)$ $(P(S, y))$; otherwise, a proper subset of S would separate x and y in G.

Next, define two subgraphs $G_x, G_y \subseteq G$ by

$$G_x := \left\langle \bigcup V(P(x, S)) \right\rangle; \quad G_y := \left\langle \bigcup V(P(S, y)) \right\rangle$$

where the union is taken over all $P(x, S)$, $P(S, y)$ respectively. Note that

(i) $G_x \cap G_y = \langle S \rangle$ by (b) and (c) above;
(ii) S separates x and y in $G_x \cup G_y$, but no $S' \subseteq V(G)$ with $|S'| < |S|$ does so; otherwise $G - S'$ contains a $P(x, y)$ which in turn must contain some $s \in S$ (see the initial part of the proof). Whence $P(x, y)$ contains some $P(x, S)$ and some $P(S, y)$ which belong to G_x, G_y respectively. So $S' \cap P(x, S) \neq \emptyset$ or $S' \cap P(S, y) \neq \emptyset$ implying that every $P(x, y)$ in G

contains an element of S', i.e., S' separates x and y already in G, a contradiction.

By (ii), S is a minimum vertex cut in $G_x \cup G_y$ separating x and y, and by the choice of G, $G = G_x \cup G_y$ follows. Now construct new graphs

$$G_x^+ := G_x \cup \{ y, sy : s \in S \}, \quad G_y^+ := G_y \cup \{ x, xs : s \in S \}.$$

If the claim $(*)$ were false, the theorem would apply to both G_x^+ and G_y^+, thus yielding $|S|$ internally disjoint paths $P^+(x,S)$ in G_x and likewise $|S|$ such paths $P^+(S,y)$ in G_y (one just deletes the corresponding endvertex in the respective paths in G_x^+ and G_y^+). By (i) we can now form $|S| = \kappa$ pairs of paths $P^+(x,S), P^+(S,y)$, each of which yields a $P(x,y) \subset G$. This contradiction to the choice of G implies the validity of $(*)$, so $G = G_x^+$ or $G = G_y^+$ must hold (observe that S is also a minimum vertex cut in G_x^+ and G_y^+).

Now let $P = x, xu_1, u_1, u_1u_2, u_2, \ldots, y$ be a shortest path in G connecting x and y. Since $l(P) > 2$ (see above), $u_2 \neq y$ must hold, and $u_1y \notin E(G)$. Form $G' = G - \{u_1u_2\}$ which has a vertex cut $S' = \{v_1, \ldots, v_{\kappa-1}\}$ (see the first part of the proof) and both $S' \cup \{u_1\}$ and $S' \cup \{u_2\}$ are vertex cuts of size κ in G separating x and y. Since $u_1y \notin E(G)$, $xv_j \in E(G), 1 \leq j \leq \kappa - 1$, by $(*)$ applied to $S' \cup \{u_1\}$. Since $xu_2 \notin E(G)$ by the choice of P, we draw the same conclusion w.r.t. the edges $v_jy \in E(G)$ and $u_2y \in E(G)$ (considering $S' \cup \{u_2\}$). Since $\kappa \geq 2$, $S' \neq \emptyset$ and so $xv_1, v_1y \in E(G)$, i.e., there is a vertex $z = v_1$ adjacent to both x and y. This final contradiction proves the theorem. ■

Corollary 19 (Whitney's Theorem) *A simple graph G is n-connected if and only if for all $x, y \in V(G), x \neq y$, there are n internally disjoint paths joining x, y.*

Corollary 20 *Let G be a simple graph with $\kappa(G) \geq n$ and $|V(G)| \geq 2n$. Given two disjoint vertex sets $S = \{v_1, \ldots, v_n\}, S' = \{w_1, \ldots, w_n\}$, there are n totally disjoint paths between vertices of S and S'.*

Corollary 21 (Dirac) *Given a simple n-connected ($n \geq 2$) graph G there is a cycle C containing n specified vertices $v_1, \ldots, v_n$.*

We also observe that for $x, y \in V(G)$ and $xy \notin E(G)$, we always have

$$\kappa(G) \leq \kappa(x,y) \leq \Delta(G).$$

If G is any graph with at least two vertices, then given distinct $x, y \in V(G)$, the *local edge connectivity* of x and y is defined by

$$\lambda(x,y) := \min_{E_s \subseteq E(G)} \{ |E_s| : x, y \text{ are in different components of } G - E_s \}.$$

The edge connectivity $\lambda(G)$ of any graph G is taken to be either the minimum of $\lambda(x,y)$ for any pair of distinct vertices x,y of G (if $|V(G)| \geq 2$) or simply $\lambda(G) = 2$ if $|V(G)| = 1$, $E(G) \neq \emptyset$. This definition ensures that the edge connectivity of a graph is not affected by subdividing any of its loops. G is said to be *n-edge-connected* if $\lambda(G) \geq n$.

Corollary 22 *If G is not K_1 then we have the following:*

1 *G is connected $\Longleftrightarrow$ G is 1-connected and G is 1-edge-connected.*

2 *G is nonseparable and $|V(G)| \geq 3 \Longleftrightarrow$ G is 2-connected $\Rightarrow$ G is 2-edge-connected.*

3 *G is connected and bridgeless $\Longleftrightarrow$ G is 2-edge-connected.*

4 *G is connected and has a bridge $\Longleftrightarrow \lambda(G) = 1$.*

Theorem 23 *Given a graph G with $E(G) \neq \emptyset$ we have $\kappa(G) \leq \lambda(G) \leq \delta(G)$.*

The analogue to Menger's Theorem w.r.t. local edge-connectivity can be derived with the help of Menger's Theorem.

Proposition 24 *Let G be a loopless graph and $x,y \in V(G), x \neq y$, be given. Then $\lambda(x,y) = \rho_e(x,y)$, where $\rho_e(x,y)$ is the maximum number of edge-disjoint paths joining x and y.*

Corollary 25 *G is k-edge-connected if and only if for all $x,y \in V(G)$, $x \neq y$, there are k edge-disjoint paths from x to y.*

For the following discussion we need to consider *block chains* which are defined as graphs G such that $bc(G)$ is a path. Correspondingly, we call a block chain *trivial* (*non-trivial*) if $E(bc(G)) = \emptyset (\neq \emptyset)$. Analogously, call a block of any graph G an *end-block* if it corresponds to an endvertex of $bc(G)$.

Suppose G is a graph with some vertex v of degree at least 3. Let e_1, e_2 be distinct edges incident to v. Introduce $v_{1,2} \notin V(G)$ and replace $e_i = vu_i$ $(i = 1,2)$ by $u_i v_{1,2}$ (possibly $u_1 = u_2$, i.e. e_1, e_2 are parallel edges). Denote this graph by $G_{1,2}$. So $G_{1,2}$ has been obtained from G by splitting away e_1, e_2. The transition from G to $G_{1,2}$ is called the *splitting procedure.*

Lemma 26 *Let G be a nonseparable graph with $|V(G)| \geq 3$. Suppose $v \in V(G)$ with $d(v) \geq 3$ exists. Split away from v two edges e_1, e_2 to form $G_{1,2}$. Then $G_{1,2}$ is a block-chain. If it is a non-trivial block-chain then $v_{1,2}$ and v belong to different end-blocks of $G_{1,2}$ and $v_{1,2}, v$ are not cutvertices of $G_{1,2}$.*

Lemma 27 *Let G be a connected bridgeless graph with precisely two blocks B_1, B_2 with the unique cutvertex $v \in B_1 \cap B_2$. Choose $e_i \in E(B_i) \cap E_v$, $i = 1, 2$ and form $G_{1,2}$ (in the case where e_1 or e_2 is a loop, split away one of the 'half-edges' of e_i, i.e., replace the loop $e_i = vv$ by the edge $vv_{1,2}$). Then $G_{1,2}$ is nonseparable.*

Lemma 28 (Splitting Lemma) *Let G be a connected bridgeless graph with a vertex v of degree at least 4. Let $e_1, e_2, e_3 \in E_v$ be chosen arbitrarily, subject to the condition that e_1 and e_2 belong to different blocks if v is a cutvertex. Then at least one of $G_{1,2}$ and $G_{1,3}$ formed by splitting away e_1 and e_2, e_1 and e_3 respectively from v, is connected and bridgeless.*

Proof. We first observe that only the block(s) containing e_1, e_2, e_3 is/are changed, whereas all the other blocks of G are also blocks of $G_{1,2}, G_{1,3}$. This is so since the splitting procedure does not alter any equation of the form $B \cap B' = \{w\}$ or $= \emptyset$ where $w \neq v$ is a cutvertex of G, though it can happen (see Lemma 27) that B' is 'enlarged' by the vertices of another block.

Let e_i belong to the block $B_i \subset G$ where $i = 1, 2$ and $B_1 \neq B_2$ if v is a cutvertex, and $B := B_1 = B_2 = B_3$ otherwise. By the preceding consideration, $G - (B_1 \cup B_2)$, $G - B$ respectively, is bridgeless.

Case 1. v is a cutvertex of G. By Lemma 27, $(B_1 \cup B_2)_{1,2}$ is nonseparable. It follows that $G_{1,2} = (G - (B_1 \cup B_2)) \cup (B_1 \cup B_2)_{1,2}$ is bridgeless. Observe that v is a cutvertex of $G_{1,2}$ if and only if v is contained in some block B' of G, $B_1 \neq B' \neq B_2$. To see that $G_{1,2}$ is also connected, one only needs to realize how $bc(G_{1,2})$ arises from $bc(G)$. Let $b_i \in V(bc(G))$ correspond to $B_i, i = 1, 2$, and suppose v retains its name in $bc(G)$. Now, $bc(G_{1,2})$ is obtained from $bc(G)$ by first identifying b_1 and b_2; if v is 2-valent in $bc(G)$, delete it to obtain $bc(G_{1,2})$, otherwise delete one of the two parallel edges arising in identifying b_1 and b_2. In both cases, $bc(G_{1,2})$ is a tree since $bc(G)$ is a tree (cf. Theorem 15). Thus $G_{1.2}$ is connected as well.

Case 2. v is not a cutvertex of G. If B is of order 2, then v and w (the other vertex of B) are joined by at least four edges, and obviously $B_{1,2}$ is a bridgeless block chain with w as the only cutvertex.

Suppose $|V(B)| \geq 3$, then B is 2-connected. Applying Lemma 26 and assuming the Splitting Lemma to be false we conclude that

(a) both $B_{1,2}$ and $B_{1,3}$ are non-trivial block chains containing a bridge $f_{1,2}, f_{1,3}$ respectively;

(b) the end-blocks of $B_{1,j}, j = 2, 3$, are not bridges and they have at most a vertex in common in which case this is a cutvertex of $B_{1,j}$;

(c) each cycle of $B_{1,j}$ containing $v_{1,j}$ or v, $j = 2,3$, lies entirely in the corresponding end-block – thus a cycle containing $v_{1,j}$ and a cycle containing v in $B_{1,j}$ have at most one vertex in common;

(d) every path $P(v_{1,j}, v) \subset B_{1,j}$ contains all cutvertices and thus all bridges of $B_{1,j}$ – see Figure 2.4.

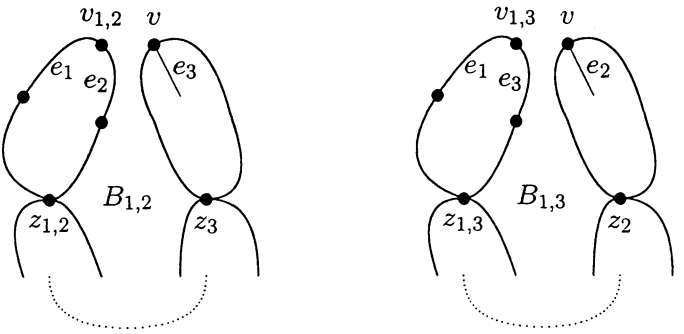


Figure 2.4

Now, B being 2-connected guarantees the existence of cycles $C_{1,3}$, $C_{2,4} \subseteq B$ such that $e_1, e_3 \in E(C_{1,3})$, $e_2, e_4 \in E(C_{2,3})$ (see Theorem 17). These two cycles lie in different endblocks of $B_{1,3}$ (see (c) above), so they have at most a vertex in common in $B_{1,3}$ and thus are edge-disjoint cycles (in $C_{1,3} \subset B_{1,3}$ we only change the name of v to $v_{1,3}$). On the other hand, $C_{1,3}$ and $C_{2,4}$ correspond in $B_{1,2}$ to paths $P_{1,3}$ and $P_{2,4}$ respectively, joining $v_{1,2}$ and v. By (a) and (d), both $P_{1,3}$ and $P_{2,4}$ contain all bridges of $B_{1,2}$, and there is at least one bridge. So,

$$E(P_{1,3}) \cap E(P_{2,4}) \neq \emptyset,$$

and thus

$$E(C_{1,3}) \cap E(C_{2,4}) \neq \emptyset,$$

contradicting what we just proved. Whence at least one of $B_{1,2}$, $B_{1,3}$ is bridgeless, and thus $G_{1,2}$ or $G_{1,3}$ is bridgeless. Assume w.l.o.g. that $B_{1,2}$ is connected and bridgeless, and thus $G_{1,2}$ is bridgeless. Suppose that $G_{1,2}$ is disconnected. Then v and $v_{1,2}$ lie in different components G'_1, G'_2 of $G_{1,2}$. Thus we can write

$$G = G_1 \cup G_2, G_1 \cap G_2 = \{v\}, \quad G_i = \langle E(G'_i) \rangle \neq \emptyset, \ i = 1,2.$$

Thus v is a cutvertex of G, contradicting the assumption of this case. This finishes the proof. ■

Corollary 29 *Let G be a nonseparable graph with a k-valent vertex v ($k \geq 4$) and let $e_1, e_2, e_3 \in E_v$ be chosen arbitrarily. If both $G_{1,2}$ and*

$G_{1,3}$ are nontrivial block-chains then $bc(G_{1,2}) = bc(G_{1,3}) = S(K_2)$ and $G_{1,2}, G_{1,3}$ have the same cutvertex.

Corollary 30 *Let $G \neq K_2$ be a nonseparable graph having a k-valent vertex v ($k \geq 4$). Let $e_1, e_2, e_3 \in E_v$ be chosen arbitrarily. If $G_{1,2}$ has a bridge then $G_{1,3}$ is nonseparable.*

The Splitting Lemma has a wide range of applications, particularly in the theory of eulerian graphs. However, it can also be used for the classic proof of Petersen's Theorem (see below) in that it yields a short proof of an intermediate result, Frink's Theorem (see [30, p. 251]). To state the former we need some more terminology.

A graph is called *k-regular* if $d(v) = k$ for every $v \in V(G)$. A digraph D is called k-regular if $d^+(v) = d^-(v) = k$ for all $v \in V(G)$.

Let G be a graph. A spanning k-regular subgraph of G is called *k-factor* of G. In particular, a 1-factor is also called a *linear factor* or *perfect matching*. A 2-factor is a *quadratic factor* and a connected 2-factor is called a *Hamiltonian cycle* of G. A k-factor in a digraph is often called a k-difactor since it is a $2k$-factor in the underlying graph. As a generalization of 1-factors, we define a *matching* to be a set of pairwise non-adjacent edges (we also speak of a set of *independent* edges). A *maximal* matching is one which cannot be extended to a larger one, whereas a *maximum* matching is one of largest size.

Theorem 31 (Petersen's Theorem) *If G is a 3-regular bridgeless graph and $e \in E(G)$ is arbitrary then $E(G) = L \dot{\cup} Q$ where $\langle L \rangle_G$ is a 1-factor and $\langle Q \rangle_G$ is a 2-factor containing e.*

Petersen's Theorem was key in the early studies of the *Four Color Problem* (see below). In this context, Tait had shown that solving this problem is equivalent to proving that every 2-connected 3-regular planar graph has a 1-factorization (a graph is *planar* if it can be drawn in the euclidian plane without edges crossing each other; a *1-factorization* of a k-regular graph G is a set of k pairwise edge-disjoint 1-factors of G). The Petersen graph (Figure 2.5) is the smallest 2-connected 3-regular graph which has no 1-factorization – it is not planar either.

However, $2k$-regular graphs can be written as the edge-disjoint union of k 2-factors (they have a *2-factorization*).

The Four Color Problem (4CP for short) stated that in any plane bridgeless graph G, one can color the faces of G with four colors such that any two faces having an edge in their respective boundary in common, are colored with different colors (a *face* of a plane graph G is – viewed in topological terms – an open arcwise connected point set F in

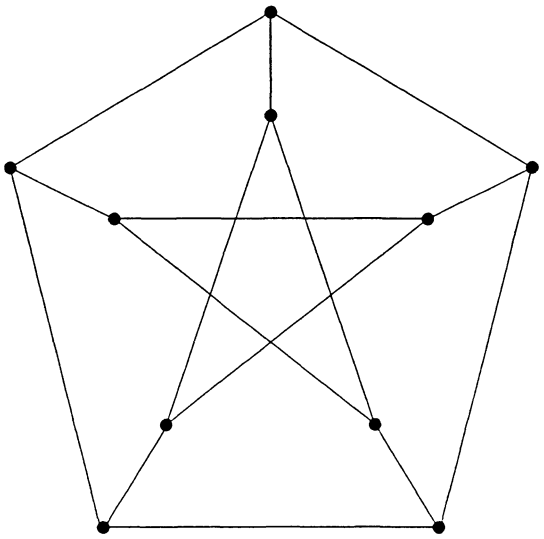


Figure 2.5 The Petersen Graph

the plane such that $\partial F = \overline{F} - F$ is a (not necessarily connected) subgraph of G, called the *boundary* of the face F). It is folklore to show that in solving the 4CP it suffices to consider 3-connected 3-regular planar graphs. This, in turn, had led Tait to formulate the 4CP in the equivalent form quoted above. This equivalent formulation then led Tait to conjecture that 3-connected 3-regular planar graphs are hamiltonian. Had this conjecture been true it would had yielded a simple proof of the 4CP (see Conjecture 54 below and the subsequent discussion of Section 5).

We note in passing that the dual formulation of the 4CP states that the vertices of any planar loopless graph can be colored with four colors such that adjacent vertices are colored differently; and that it suffices to prove the 4CP in its dual form for 4-connected triangulations of the plane (which are hamiltonian – see Theorem 63 below and the discussion related to it).

The 4CP was finally solved in its dual form by Appel, Haken, and Koch, [2]; theirs is a computer-aided proof, as is the more recent but shorter proof by Robertson, Sanders, Seymour, and Thomas, [37].

4. EULERIAN GRAPHS AND COVERING WALKS, CYCLE DECOMPOSITIONS AND CYCLE COVERS

A graph G is called *eulerian* if $d(v)$ is even for every $v \in V(G)$. A digraph D is called eulerian if $d^-(v) = d^+(v)$ for all $v \in V(D)$.

We observe that eulerian graphs G are bridgeless; otherwise, for a bridge e of G, the graph $G - e$ has two components each of which has precisely one odd vertex (cf. Theorem 7), contradicting the Handshaking Lemma.

A walk in a graph (digraph) H is a *covering walk* if $\lambda_W(h) > 0$ for every $h \in E(H)$ $(A(H))$. If a closed covering walk is a trail then it is called an *eulerian trail.* A closed covering walk W where $\lambda_W(h) = k$ for every $h \in E(G)$ $(A(D))$ is called a *k-tracing.* In particular, if $k = 2$, W is called a *double tracing.* A double tracing in a graph is called *bidirectional* if every edge $xy, y \neq x$ is passed in W once from x to y and once from y to x. A double tracing W is called *retracting-free* if for every $e = xy \in E(G)$, x, e, y, e, x is neither a section of W nor of W^{-1}.

Let H be a graph (digraph) and let $S = \{C_1, \ldots, C_r\}$, $r \geq 1$ be a collection of cycles of H. If every $e \in E(H)$ $(A(H))$ belongs to at least one cycle of S then S is called a *cycle cover.* If every $e \in E(H)$ $(A(H))$ belongs to exactly k cycles of S then S is called a *cycle k-cover.* A cycle 1-cover is also called a *cycle decomposition* of H as it corresponds to a partition of $E(H)$ $(A(H))$ into cycles. A cycle 2-cover is also called a *cycle double cover (CDC).*

Theorem 32 *Given a connected graph G, the following statements are equivalent.*

1 G is eulerian.

2 G has an eulerian trail.

3 G has a cycle decomposition.

Proof. (1) $\Rightarrow$ (2): G is eulerian and connected. If $E(G) = \emptyset$ then $G = K_1$ and $T = v$ with $V(G) = \{v\}$ is an eulerian trail. If $E(G) \neq \emptyset$ then choose any $v \in V(G)$ and start a trail by traversing an edge e incident with v. Reaching the other end w of e ($w = v$ if e is a loop) continue the trail T with any not yet traversed edge f incident with w, etc. So whenever we reach $x \in V(G)$ continue T at x with $h \in E_x - E(T)$.

Suppose this procedure stops at some $y \in V(G)$, i.e., $E_y - E(T) = \emptyset$. We claim that $y = v$. If not then every time we arrive at y we have

traversed an odd number of edges incident to y. So if T eventually stops at y the degree of y is odd, a contradiction. Thus $y = v$ and $E_v \subseteq E(T)$. If $E(T) = E(G)$ then T is an eulerian trail. So suppose this is not the case. Backtrack on T until $z \in V(G)$ is reached for which $E_z - E(T) \neq \emptyset$. Such z exists since G is connected. By assumption $E(G) \neq E(T)$, so define $G_1' := G - E(T)$ and produce a closed trail T_1' in G_1' starting and ending at z. Writing $T = v, e, w, \ldots, hz, z, zt, \ldots, v$ we conclude that by the construction of T, T_1' that $T_1 = v, e, w, \ldots, hz, T_1', zt, \ldots, v$ is a closed trail starting and ending at v with $E_v \cup E_z \subseteq E(T_1)$.

If $E(T_1) = E(G)$ then T_1 is an eulerian trail; otherwise backtrack on T_1 to find z_2 for which $E_{z_2} \cap E(T_1) \neq \emptyset$ but $E_{z_2} \not\subseteq E(T_1)$. Now continue to produce T_2' in $G_2 = G - E(T_1)$, starting and ending at z_2 as we did with respect to T_1' and construct T_2 by inserting T_2' into T_1 and so that $E_v \cup E_z \cup E_{z_2} \subseteq E(T_2)$. The process of producing closed trails $T, T_1, T_2, \ldots$ with $E(T) \subset E(T_1) \subset E(T_2) \ldots$ must come to an end with some T_r, $r \leq |V(G)| - 1$. Then $E(T_r) = E(G)$ so that T_r is an eulerian trail as required.

(*Alternate proof of* (1) $\Rightarrow$ (2) *using the Splitting Lemma.*) Since G is eulerian, it is bridgeless. G is also connected. If $d(v) = 2$ for every $v \in V(G)$ then G is a cycle and this cycle is an eulerian trail. Thus suppose $d(v) > 2$ for some $v \in V(G)$.Take $e_1, e_2, e_3 \in E_v : e_1, e_2$ belong to different blocks of G if v is a cutvertex. W.l.o.g. suppose $G_1 := G_{1,2}$ is connected and bridgeless. G_1 is also eulerian. If $d_{G_1}(v) > 2$ then we continue the splitting operation at v. If $d(v) = 2k$ then after $k - 1$ splitting operations at v, employing the Splitting Lemma, we have a connected eulerian bridgeless graph G_{k-1} where the original v has been replaced by k 2-valent vertices. If G_{k-1} is a cycle C then a run through C corresponds to an eulerian trail of G. If not then find $w \in V(G) : d_G(w) = d_{G_{k-1}}(w) > 2$ and apply the Splitting Lemma to w similarly. Consequently if $d(v) = 2k_v$, $k_v \in N$, then after applying the Splitting Lemma $\sum_{v \in V(G)}(k_{v-1}) = (\sum k_v) - p = q - p$ times, we obtain a graph which is a cycle. Running through this cycle corresponds to the traversal of G by an eulerian trail.

(2) $\Rightarrow$ (3): Since T is an eulerian trail it is a closed covering trail. Let C_1 be a shortest closed sequence in T starting and ending at x, say, with hx (xy) immediately preceding (following) C_1 in T. C_1 is a closed subtrail of T and since $l(C_1) = \min$, C_1 is a cycle indeed. Next, define $T_1 = v, e_1, w_1, \ldots, hx, x, xy, \ldots, v$. $E(T_1) = E(G) - E(C_1)$. So T_1 is an eulerian trail in $G_1 := G - C_1$. Repeat the above procedure to find a cycle C_2 in G_1. Define T_2 similarly as an eulerian trail in $G_2 = G_1 - C_2$. Then $E(C_2) \cap E(C_1) = \emptyset$, so we produce step-by-step a cycle C_i in G_{i-1}

such that $E(C_i) \cap E(C_j) = \emptyset$, $1 \leq j < i$ and $G_i := G_{i-1} - C_i \subset G_{i-1}$, implying that the procedure must end with some $G_k := G_{k-1} - C_k = \emptyset$ where $k \geq 1, G_0 = G$, i.e. $G_{k-1} = C_k$. Then $S = \{C_1, \ldots, C_k\}$ is a cycle cover with $E(C_i) \cap E(C_j) = \emptyset$ for $1 \leq j < i \leq k$. So S is even a cycle decomposition.

(3) $\Rightarrow$ (1): Since $d_{C_i}(v) \in \{0, 2\}$ for every $C_i \in S$, we have $d_G(v) = \sum_{i=1}^{k} d_{C_i}(v) \equiv 0 \bmod 2$. ■

Corollary 33 *If G is a connected eulerian graph then it has an eulerian trail starting at any prescribed $v \in V(G)$ and $e \in E_v$.*

Corollary 34 *Theorem 32 remains true if we replace "connected graph" by "weakly connected digraph" (note that a weakly connected eulerian digraph is strongly connected).*

Corollary 35 *Every connected graph has a bidirectional double tracing.*

While these corollaries can be obtained by applying Theorem 32, the next corollary results from a more intricate application of the Splitting Lemma (see also Kotzig's Theorem below).

Corollary 36 (Sabidussi) *If G is a connected graph without endvertices then it has a retracting-free double tracing.*

In view of part 3 of Theorem 32 and Corollary 35 the following has been conjectured.

Conjecture 37 (Cycle Double Cover Conjecture (CDCC)) *Every connected bridgeless graph has a cycle double cover.*

A generalization of this, namely that every connected bridgeless graph has an 'oriented' CDC S such that every edge is traversed in two different directions by the two cycles of S it belongs to, has become known as the *oriented cycle double cover conjecture*, whereas the *strong cycle double cover conjecture (SCDCC)* states that one might prescribe any given cycle of G to belong to some CDC.

The following results can also be proved quite easily.

Theorem 38 *Every double tracing in a tree T is bidirectional and has retractions if $E(T) \neq \emptyset$. Moreover, every closed covering walk in T is a double tracing.*

Corollary 39 *If G is a connected graph with precisely two odd vertices x, y then it has a covering trail starting at x and ending at y.*

Corollary 39 is also a consequence of the following result.

Corollary 40 *Let G be a graph without eulerian components, whose $2k$ odd vertices are denoted by $v_1, \ldots, v_{2k}$. There is a decomposition of $E(G)$ into k open trails $T_1, \ldots, T_k$ where each T_i starts at some v_i and ends at some v_j. Moreover, any decomposition of $E(G)$ into open trails must have at least k such trails.*

Let $S = \{W_1, \ldots, W_s\}$ be a decomposition of $E(G)$. S is called a *path/cycle decomposition* of $E(G)$ if for every $W_i \in S$, $\langle W_i \rangle$ is a path or a cycle.

Corollary 41 *Every graph G has a path/cycle decomposition S such that the number of paths in S is half the number of odd vertices in G.*

Let G be any graph. For every $v \in V(G)$ let $P(v)$ be a partition of E_v (for $d(v) = 0$ set $P(v) = \emptyset$). Define a multiset $P(G) := \bigcup_{v \in V(G)} P(v)$ (note that a class $C \in P(x)$ may also be a class in $P(y)$) and call $P(G)$ a *partition system* of G. Every C in $P(v)$ is called a *forbidden part* for every $v \in V(G)$. If $|C| \leq 2$ for every $C \in P(G)$ then $P(G)$ is called a *partial transition system.* If $|C| = 2$ for every $C \in P(G)$ then $P(G)$ is called a *transition system.*

For example, if G is a connected eulerian graph then it has an eulerian trail (written as an edge sequence) $T = e_1, e_2, \ldots, e_q$ (consider a loop to be two half-edges). This induces a transition system X_T of G, where

$$X_T = \{\{e_1, e_2\}, \{e_2, e_3\}, \ldots, \{e_{q-1}, e_q\}, \{e_q, e_1\}\}$$

Conversely, a transition system $X(G)$ defines a decomposition of $E(G)$ into closed trails.

Similarly, a cycle decomposition $S = \{C_1, \ldots, C_k\}$ induces a transition system X_S where two edges at any vertex are in the same element of X_S if and only if they are in the same cycle of S.

We call two partition systems $P_1(G), P_2(G)$ *compatible* if for every $v \in V(G)$ and $\{i, j\} = \{1, 2\}$, $C_i \not\subseteq C_j$ for every $C_i \in P_i(v) \subseteq P_i(G)$ and every $C_j \in P_j(v) \subseteq P_j(G)$. If $P_1(G) = X_T$ (or $= X_S$) then we say T (or S) is *compatible with $P_2(G)$* or *$P_2(G)$-compatible.*

Theorem 42 (Kotzig's Theorem) *Let G be a connected eulerian graph with a given partition system $P(G)$. Then there is a $P(G)$-compatible eulerian trail if and only if for every $v \in V(G)$ and every $C \in P(v)$ the inequality $|C| \leq \frac{1}{2}d(v)$ holds.*

The following is just a special case of Theorem 42; however, it can be proved directly by applying the Splitting Lemma.

Corollary 43 *Let G be an eulerian graph with $\delta(G) \geq 4$ and let S be a cycle decomposition of G. Then there is a X_S-compatible eulerian trail T in G.*

Conjecture 44 (Sabidussi's Compatibility Conjecture (SCC)) *Let G be an eulerian graph with $\delta(G) \geq 4$ and let T be an eulerian trail of G. Then there is a X_T-compatible cycle decomposition in G.*

However, the condition on the class sizes in Kotzig's Theorem does not allow to conclude the existence of a $P(G)$-compatible cycle decomposition. For example consider the eulerian graph K_5 with the cycle decomposition S shown in Figure 2.6, where $X_S =: X_5$ is characterized by the small arcs marking the transitions of the two cycles in S. We show that it does not have a X_5-compatible cycle decomposition.

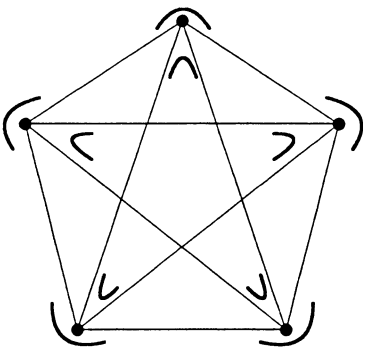


Figure 2.6

Suppose S' is a X_5-compatible cycle decomposition of K_5. Let $C \in S'$. Since S' is X_5-compatible, C contains alternately edges of the outer pentagon and the inner pentagram and thus must be even. Since $l(C) \leq 5$, we have $l(C) = 2$ or 4. But the former cannot be since K_5 is simple. Thus the cycles in S' are of length 4. So $|E(G)|$ must be a multiple of 4. But $|E(K_5)|=10$, a contradiction. So S' cannot exist.

This non-existence of a X_5-compatible cycle decomposition in K_5 is basically the reason why arbitrary 2-connected 4-regular graphs with given system of transition X may not have an X-compatible cycle decomposition. Also, this example of Figure 2.6 can be used to prove that the Petersen graph has no 1-factorization.

On the other hand, there is a closer relationship between the SCC and the CDCC than one might suspect at a first glance. To explain this

relationship we need some more terminology.

An edge cut E_0 of a graph G is called *essential* if $G - E_0$ has at least 2 nontrivial components. The *essential edge-connectivity* $\lambda_e(G)$ of G is the size of the smallest essential edge cut of G if such an edge cut exits, otherwise set $\lambda_e(G) = \lambda(G)$. G is said to be essentially k-edge-connected if $\lambda_e(G) \geq k$.

If G has two disjoint cycles then the edge cut E_1 is called a *cyclic edge cut* if $G - E_1$ has at least two components with each component having a cycle. The *cyclic edge-connectivity* $\lambda_c(G)$ of a graph G is defined to be $\lambda_e(G)$ if G has no two disjoint cycles, or the size of the smallest cyclic edge cut of G. And G is said to be cyclically k-edge-connected if $\lambda_c(G) \geq k$.

Corollary 45 *For any loopless graph G, $\lambda(G) \leq \lambda_e(G) \leq \lambda_c(G)$.*

Observe also that $\lambda_e(K_4) = \lambda_c(K_4) = 4$ and $\lambda_e(K_5) = \lambda_c(K_5) = 6$.

Given any graph G, a cycle C of G is called a *dominating cycle* if every edge of G is incident to a vertex of C.

Conjecture 46 (Dominating Cycle Conjecture (DCC)) *Any 3-regular graph G_3 with $\lambda_c(G_3) \geq 4$ has a dominating cycle.*

It is relatively easy to see, [13], that in proving the SCC it suffices to consider connected eulerian graphs having only 4- and 6-valent vertices. Such graphs together with an eulerian trail give rise – in a natural way ([13]) – to a 3-regular graph with dominating cycle. In fact, the SCC has been generalized by Bill Jackson and the author [14] (here we state the most important case of this generalization).

Conjecture 47 (General Compatibility Conjecture (GCC)) *Let $G \neq K_5$ be a 4-regular graph with $\lambda_c(G) \geq 6$ and an arbitrary transition system X. Then there is an X-compatible cycle decomposition S of G.*

In fact, the following is true.

Theorem 48

a. *The validity of the SCDCC implies the validity of the SCC.*

b. *The validity of the SCC and the DCC imply the validity of the CDCC.*

c. *The validity of the GCC implies the validity of the CDCC.*

Theorem 48 is a decisive reason why the SCC and GCC have attracted quite a few people's research. In fact, the GCC has been proved for large classes of graphs, even in a more generalized setting (see [43] for details

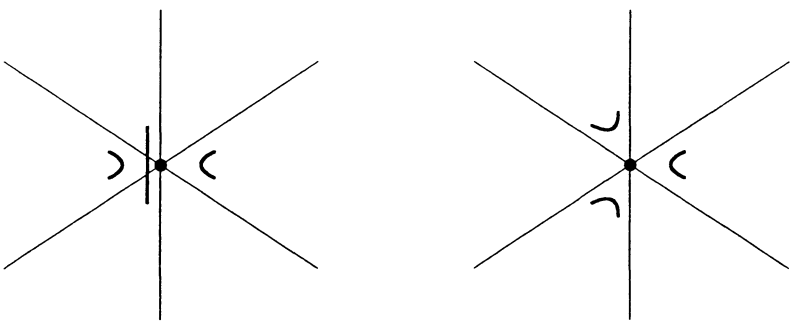
and more references). Thus the validity of the SCC is guaranteed for the same classes of graphs.

While (a) and (b) have been proved in [13], (c) is due to F. Jaeger, [29].

We note in passing that in [43], cycle covers in general, and cycle double covers in particular, are treated extensively.

Returning to the theme of eulerian trails, we now consider connected plane eulerian graphs G (note that a *plane* graph is a realization of a planar graph in the euclidean plane). Now, G being embedded in the plane allows one to speak of a *cyclic ordering of the edges* in E_v for every $v \in V(G)$ (w.l.o.g., we assume here and in what follows that G is loopless). Describe this cyclic ordering counterclockwise, say, by $(e_1, e_2, \ldots, e_{2k_v})$, $2k_v = d(v)$. Thus one can define a *non-intersecting eulerian trail* T as one where for any two transitions $\{e_i, e_j\}$, $\{e_k, e_l\}$ of T at any $v \in V(G)$ neither $i < k < j < l$ nor $i < l < j < k$ holds (w.l.o.g. $i = \min\{i, j, k, l\}$).

Figure 2.7(a) shows a 6-valent vertex with three pairwise non-intersecting transitions which are marked by little arcs.



(a) (b)

Figure 2.7

An *A-trail*, on the other hand, is one where for any transition $\{e_i, e_j\} \in X_T$ at the corresponding vertex $v \in V(G)$, it follows that $j = i+1$ or $j = i-1$ (read mod $d(v)$).

Figure 2.7(b) shows one of the two possible choices re. the behavior of an A-trail at the corresponding vertex. Note, however, that the tran-

sitions in Figure 2.7(a) *do not* correspond to an A-trail.

We observe that non-intersecting eulerian trails and A-trails are identical concepts in the cases where $\Delta(G) \leq 4$, but not in general. In fact, while the Splitting Lemma can be applied to show that *every* connected plane eulerian graph has a non-intersecting eulerian trail, there exist infinitely many 3-connected plane eulerian graphs which do not admit an A-trail. Moreover, the decision problem whether a given 3-connected plane eulerian graph has an A-trail is $\mathcal{NP}$-complete, [1], whereas non-intersecting eulerian trails can be found in polynomial time, as we shall see below. Regarding algorithmic complexity in general, we refer the reader to Chapter 4. Here we are only concerned with the question whether certain algorithms run in polynomial time, and which problems are $\mathcal{NP}$-complete.

4.1. ALGORITHMS FOR CONSTRUCTING EULERIAN TRAILS

In what follows $I(P)$ denotes the set of vertices which are 2-valent in the path P, i.e., the set of *inner vertices* of P.

Algorithm 49 (The Splitting Algorithm)
Input: *A connected eulerian graph G with $|E(G)| = q > 0$ and initial vertex $v_0 \in V(G)$.*
Step 0. $H := G, i := 0, T_0 = \{v_0\}$
Step 1. *Suppose $T_i = v_0, e_1, v_1, \ldots, e_i, v_i$ has been obtained by a (possibly empty) sequence of splitting operations such that T_i appears as a path P in H with $x \in I(P)$ implying $d_H(x) = 2$. If $i = 0$ then let $e_1 \in E_{v_0}$ be arbitrary. Go to Step 1.2. If $i \neq 0$ set $f_1 = e_i$.*
Step 1.1. *If $d_H(v_i) > 2$ choose $f_2, f_3 \in E_{v_i} \cap (E(H) - E(T_i))$. Form $H_{1,j}$ by splitting away f_1, f_j, $j = 2, 3$. Define $H := H_{1,2}$ if it is connected, otherwise set $H := H_{1,3}$. On the other hand, if $d_H(v_i) = 2$ then H remains unchanged.*
Step 1.2. *Set $e_{i+1} = v_i v_{i+1}$ for the edge not in T_i and incident with v_i in H. Set $T_{i+1} := T_i, e_{i+1}, v_{i+1}$ (possibly $v_{i+1} = v_i$).*
Step 1.3. *Set $i := i + 1$.*
Step 2. *If $i \neq q$ go to Step 1.*
Step 3. *T_q is an eulerian trail of G.*

Algorithm 50 (Fleury's Algorithm) *This only differs from the Splitting Algorithm in Step 1.*
Input: *A connected eulerian graph G with $|E(G)| = q > 0$ and initial vertex $v_0 \in V(G)$.*
Step 0. $H := G, i := 0, T_0 = \{v_0\}$.
Step 1. *Suppose $T_i = v_0, e_1, v_1, \ldots, e_i, v_i$ has been already constructed. Set $G_i := G - E(T_i)$.*

Step 1.1. *Choose* $e_{i+1} \in E_{v_i} \cap E(G_i)$ *such that* e_{i+1} *is a bridge of* G_i *only if* v_i *is an endvertex of* G_i.
Step 1.2. *Write* $e_{i+1} = v_i v_{i+1}$ *and set* $T_{i+1} = T_i, e_{i+1}, v_{i+1}$.
Step 1.3. *Set* $i := i + 1$.
Step 2. *If* $i \neq q$ *go to Step 1.*
Step 3. T_q *is an eulerian trail of* G.

Algorithm 51 (Hierholzer's Algorithm)
Input: *A connected eulerian graph* G *with* $|E(G)| > 0$ *and initial vertex* $v_0 \in V(G)$.
Step 1. *Construct a closed trail* T_0 *starting and ending at* v_0 *by traversing step-by-step an edge not yet traversed.* T_0 *ends at* v_0 *with* $E_{v_0} \subseteq T_0$. *Set* $i := 0$.
Step 2. *If* $E(T_i) = E(G)$ *then go to Step 4. If* $E(T_i) \neq E(G)$ *then choose* $v_{i+1} \in V(T_i)$: $E_{v_{i+1}} - E(T) \neq \emptyset$. *Construct a closed trail* T_i' *as in Step 1 starting and ending at* v_{i+1}, $T_i' \subseteq G - E(T_i)$.
Step 3. *Construct a closed trail* T_{i+1} *with* $E(T_{i+1}) = E(T_i) \cup E(T_i')$: *starting at* v_0 *and go in* T_i *to* v_{i+1}, *then traverse* T_i' *and then continue the run through* T_i *from* v_{i+1}. *Set* $i := i + 1$ *and go to Step 2.*
Step 4. T *is an eulerian trail of* G.

Looking back at the two independent proofs of the first part of Theorem 32, it becomes clear why Algorithms 49 and 51 produce eulerian trails indeed. Moreover, on comparing Fleury's algorithm with the Splitting Algorithm, it is also clear that the functioning of the latter implies the functioning of the former, and that the choice of e_{i+1} in Step 1.1 of Fleury's algorithm corresponds to the choice of e_2, e_3 in the Splitting Lemma (if v_i is not an endvertex of G_i). Moreover, checking connectedness after splitting away two edges, can be done in polynomial time and it needs to be done only once per application of the splitting procedure. Therefore, the first two algorithms run in polynomial time; and so does Hierholzer's Algorithm which runs even faster. Note that the Splitting Algorithm can be adapted to produce eulerian trails in digraphs, $P(G)$-compatible eulerian trails in graphs, and non-intersecting eulerian trails in plane graphs: this adaption takes place in Step 1.1 by making specific choices of f_2 and f_3, in each of the respective cases. Clearly, these choices do not essentially alter the polynomial running time of the respectively adapted Splitting Algorithm.

4.2. MAZES

Mazes can be represented by graphs by replacing doors by vertices and edges joining doors that can be reached from one to the other without passing by or through a third door. Figure 2.8 shows an example.

In what follows we present the two most important maze search algorithms which operate on connected graphs and produce bidirectional

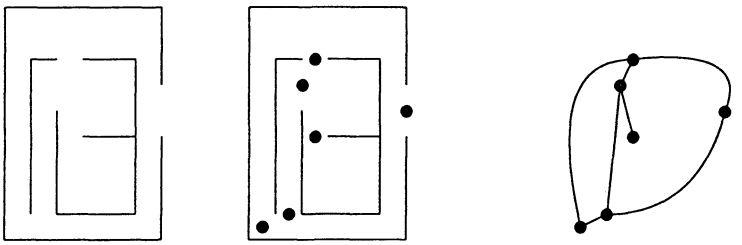


Figure 2.8 (a) a maze, (b) the maze and its doors, (c) the graph of the maze

double tracings (cf. Corollary 35 and Theorem 38). These algorithms operate with local information only.

Algorithm 52 (Tremaux's Algorithm)
Input: *A connected graph G with $E(G) \neq \emptyset$ which is unknown to the person walking in the maze.*
Hypothesis: *Whenever $v \in V(G)$ has been reached in the course of constructing a covering walk $W, \lambda_W(e)$ is known for all $e \in E_v$, with $\lambda_W(e)$ referring to the section of W performed at the moment of arriving at v.*

Step 0: *Set $i := 0$, choose an arbitrary $v_0 \in V(G)$, set $W = v_0$.*
Step 1: *Starting at $v_i \in V(G)$, walk along any edge $e_i \in E_{v_i} - E(W)$. Define $W := W, e_i, v_{i+1}$, possibly $v_i = v_{i+1}$. Set $i := i + 1$.*
Step 2: *Suppose $W = v_0, e_0, v_1, \ldots, v_{i-1}e_{i-1}, v_i$ has been constructed. If v_i is not an endvertex and $v_i \neq v_j, 0 \leq j < i$ then go to Step 1.*
Step 3: *(v_i is an endvertex or $v_i = v_j$ for some $0 \leq j < i$.) If $\lambda_W(e_{i-1}) > 1$ go to Step 4. Otherwise, ($\lambda_W(e_{i-1}) = 1$) set $e_i = e_{i-1}$, $v_{i+1} = v_{i-1}$ and $W := W, e_i, v_{i+1}$. Set $i := i + 1$ and go to Step 4.*
Step 4: *If $\lambda_W(e) > 1$ for all $e \in E_{v_i}$, then go to Step 5. Otherwise choose $e \in E_{v_i}$ so that $\lambda_W(e)$ is minimum. Set $e_i = e, v_{i+1} = y$ where $y = v_i$ if e is a loop, and $e \in E_y \cap E_{v_i}$ otherwise. Set $W := W, e_i, v_{i+1}$ and $i = i + 1$. Go to Step 2.*
Step 5: *The final W is a bidirectional double tracing of G.*

The justification of this algorithm lies basically in the second part of Step 3 and Theorem 38. Namely: if upon traversing an edge $e = xy$ for the *first* time one reaches y, say, which had been reached before or if y is an endvertex, then one backtracks on e from y to x. This is tantamount to detach the single edge e from y thus creating a new endvertex y' incident with e. If one performs such a 'detachment' at every step in question, then one ultimately ends up with a tree since G is connected. Correspondingly, by Theorem 38, the output of the algorithm is *necessarily* a bidirectional double tracing of G.

Algorithm 53 (Tarry's Algorithm)
Input: *A connected graph G with $E(G) \neq \emptyset$.*
Hypothesis: *If $v \in V(G)$ is reached in constructing W then $E^o_{v,W} \subseteq E_v$ (the set of edges used away from v by W) is known. $e_{in}(v) \in E_v$ by which v is reached for the first time is also known.*

Step 0: *Set $i = 0$. Choose $v_0 \in V(G)$, set $W := v_0$. Set $\{e_{in}(v_0)\} := \emptyset$.*
Step 1: *Beginning at $v_i \in V(G)$, walk along any edge $e_i \in E_{v_i} := E_{v_i} - (E^o_{v_i,W} \cup \{e_{in}(v_i)\})$. Define $W := W, e_i, v_{i+1}$ where $e_i \in E_{v_i+1}$, possibly $v_i = v_{i+1}$. Set $i := i + 1$.*
Step 2: *Consider $W = v_0, e_0, \ldots, e_{i-1}, v_i$. If $E_{v_i} \neq \emptyset$, then go to Step 1; otherwise, go to Step 3.*
Step 3: *If $\{e_{in}(v_i)\} \subseteq E^o_{v_i,W}$ then go to Step 4. Else set $e_i := e_{in}(v_i)$, $W := W, e_i, v_{i+1}$ ($e_i \in E_{v_{i+1}}$), $i := i + 1$ and go to Step 2.*
Step 4: *W is a bidirectional double tracing.*

Here, the justification of the algorithm is based on the following two facts:

1) $\langle \{e_{in}(v_j) : v_j \in W\} \rangle$ is a spanning tree of G;

2) for an eulerian trail T in a weakly connected eulerian digraph D, if one marks at every vertex x other than the initial vertex v_0 of T, the arc by which x is being reached for the first time in the course of traversing T, then the set of marked arcs form a spanning out-tree of D rooted at v_0.

Whence one can interpret Tarry's Algorithm as producing an eulerian trail in the eulerian digraph obtained from G by replacing every edge by two oppositely oriented arcs (cf. Corollary 34) and such eulerian trail corresponds to a bidirectional double tracing of G.

If one operates with the combined hypotheses of Tremaux's and Tarry's algorithms and proceeds according to Tarry's algorithm choosing, however, in Step 1 e_i with minimum $\lambda_W(e_i)$, then one clearly obtains a bidirectional double tracing as well. However, if the input is a connected eulerian graph, then the edge sequence obtained by listing the edges according to their second traversal, defines an eulerian trail.

For more details on algorithms producing eulerian trails and on maze search algorithms, we refer to [16, Vol. 2, Ch. X].

5. HAMILTONIAN CYCLES AND VERTEX-COVERING WALKS

Let's start with discussing some conjectures.

Conjecture 54 (Tait's Conjecture, 1880) *Every planar 3-connected, 3-regular graph has a hamiltonian cycle.*

That the condition of 3-connectedness is important in the above conjecture is shown by Figure 2.9.

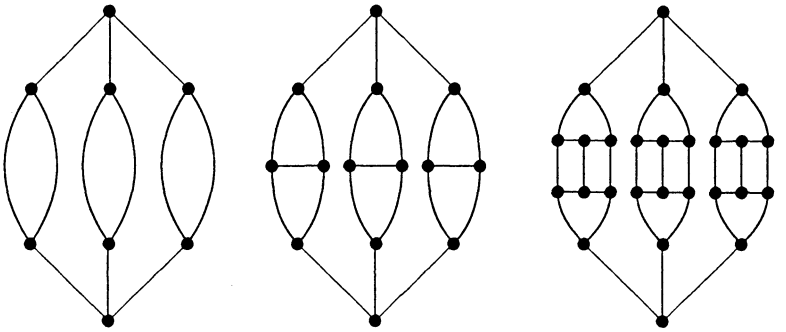


Figure 2.9 (a) 2-connected non-hamiltonian graph, (b) 2-connected simple non-hamiltonian graph, (c) 2-connected simple bipartite non-hamiltonian graph

The conjecture was disproved in 1946 by W. T. Tutte by giving a counterexample G_3 with $\lambda_c(G_3) = 3$. Later on counterexamples with $\lambda_c(G_3) = 5$ were found.

Note: For a plane 3-regular G_3, $\lambda_c(G_3) \leq \min l(C) =: g(G)$ =girth of G, where C is a cycle of G.

Conjecture 55 (Tutte) *Any 3-connected bipartite 3-regular graph has a hamiltonian cycle.*

This conjecture has been disproved by Horton who developed a counterexample having 96 vertices.

Conjecture 56 (Barnette, Bosak, Tutte) *Any planar, 3-connected, bipartite 3-regular graph has a hamiltonian cycle.*

This is still an open problem.

Observing that the classes of graphs addressed in the above conjectures have few edges in comparison to their order p (their size is $q = 3p/2$), one wonders whether a simple graph is hamiltonian (i.e., has a hamiltonian cycle) if it is of sufficiently large size. Of course, K_p is hamiltonian and has $\frac{1}{2}(p-1)!$ hamiltonian cycles (viewed as different subgraphs); it's of size $\binom{p}{2}$. Trivially, the graph obtained from K_p by deleting all but one of the edges at some fixed vertex, is non-hamiltonian and of size $\binom{p}{2} - (p-2)$. However, the following result of G. A. Dirac (proved in 1952) points already into the right direction. In the sequel, a hamiltonian path of G is a path containing all vertices of G.

Theorem 57 *Any simple graph G of order $p \geq 3$ with $d(v) \geq \frac{p}{2}$ for every $v \in V(G)$ has a hamiltonian cycle.*

This result was improved in 1960 by O. Ore who showed the following to be true.

Theorem 58 *Let G be a simple graph of order $p \geq 3$ such that $d(x) + d(y) \geq p$ for all $x, y \in V(G)$ satisfying $xy \notin E(G)$. Then G is hamiltonian.*

Proof. We proceed by contradiction. Among all counterexamples for some fixed $p \geq 3$, choose one of *maximum* size q. $G \ncong K_p$ since K_p is hamiltonian. Thus G contains $x, y \in V(G)$, $x \neq y$, such that $xy \notin E(G)$. Form $G_1 := G \cup \{xy\}$; then G_1 has a hamiltonian cycle C_1 containing xy. Number the vertices of G in accordance with a run through C_1 such that

$$x = v_1, v_2, \ldots, v_{p-1}, v_p = y; v_i v_{i+1} \in E(G) \cap E(C_1) \text{ for } 1 \leq i \leq p-1.$$

So $C_1 - \{xy\}$ is a hamiltonian path in G connecting x and y. Observe that if for some $1 \leq i \leq p-1$, $v_1 v_{i+1}, v_p v_i \in E(G)$, then G_1 has a hamiltonian cycle not containing $xy = v_1 v_p$. For, $P_1 = P(v_1, v_i)$, $P_2 = P(v_p, v_{i+1}) \subset C_1$ would yield a hamiltonian cycle $C = P_1, v_i v_p, P_2, v_{i+1} v_1, v_1$ not containing $v_1 v_p$, so that C is a hamiltonian cycle in G, a contradiction. This implies for every j, $2 \leq j \leq p$, if $x v_j \in E(G)$, then $y v_{j-1} \notin E(G)$. Set $d = d(v)$. So if $N(x) = \{v_{i_1}, \ldots, v_{i_d}\}$, then $\{v_{i_1 - 1}, \ldots, v_{i_d - 1}\} \cap N(y) = \emptyset$. Thus $d(y) \leq (p-1) - d$, whence $d(x) + d(y) \leq d + (p-1) - d = p - 1$, a contradiction to the hypothesis.

This implies that the counterexample G does not exist, therefore the theorem is true. ■

Arguing along the same lines as in the preceding proof one obtains the following for bipartite graphs.

Theorem 59 *Suppose G is simple and bipartite, $V(G) = V_1 \dot{\cup} V_2$, $|V_1| = |V_2| = k$. Suppose there is $\alpha \in \mathbb{N} \cup \{0\}$ such that $k > 2\alpha$ and $\delta(G) \geq k - \alpha$. Then G has a hamiltonian cycle.*

Corollary 60 *If G is simple and bipartite of order $n = 2k$, $k = |V_1| = |V_2|$, and $\delta(G) > \frac{n}{4}$, then G has a hamiltonian cycle.*

Reconsidering the proof of Theorem 58, we conclude the following.

Corollary 61 *Given a simple graph G. Suppose there are $x, y \in V(G)$ such that $xy \notin E(G)$ and $d(x) + d(y) \geq p$. If $G \cup \{xy\}$ has a hamiltonian cycle, then so does G.*

This leads us to the following construction due to Bondy and Chvatal.

Consider a simple graph G of order p and let $0 < k \leq p$ be given. If there are $x, y \in V(G)$ such that $xy \notin E(G)$ and $d(x) + d(y) \geq k$, form $G_1 = G \cup \{xy\}$. If there are $x_1, y_1 \in V(G_1) = V(G)$ such that $x_1y_1 \notin E(G_1)$ and in G_1 holds $d(x_1) + d(y_1) \geq k$, then form $G_2 = G_1 \cup \{x_1y_1\}$. Continue this procedure until reaching a graph G_r such that for all $u, v \in V(G_r) = V(G)$ with $uv \notin E(G_r)$ it follows that $d(u) + d(v) < k$ holds in G_r. Then G_r is called the *k-closure* of G and is denoted by $c_k(G)$.

Note: (1) In the course of constructing $c_k(G)$, it can happen that $d_{G_j}(u) + d_{G_j}(v) \geq k$ for some $u, v \in V(G_j) = V(G)$ with $uv \notin E(G_j)$ although $d_G(u) + d_G(v) < k$. Then uv can be added to G_j to form G_{j+1}.

(2) $c_k(G)$ is uniquely determined.

Thus $c_k(G)$ is well-defined.

Corollary 62 *Let G be a simple graph of order $p \geq 3$. G has a hamiltonian cycle if and only if $c_p(G)$ has a hamiltonian cycle.*

Note that in the case of Ore's Theorem (Theorem 58) $c_p(G) \cong K_p$. In their article [6], Bondy and Chvatal show for several results proved until then, that in each of these cases the p-closure of the corresponding graph is complete. Since then, many results have been proved on hamiltonian graphs, which are also based on degree conditions and where the p-closure is not complete. We also note that – based on the proof of Theorem 59 and on Corollary 60 – one can introduce the *bipartite closure* of a bipartite graph, [6].

The graphs for which we have positive answers re. hamiltonian cycles so far, have "many" edges. So, we may ask which are the conditions one could use instead of degree conditions to ensure that a graph is hamiltonian. If for instance we consider simple planar graphs, we have graphs with few edges but many other nice properties. In fact, if G is a simple planar graph of order $p \geq 3$, then it has at most $3p - 6$ edges, and every simple maximal plane graph has precisely $3p - 6$ edges. Note that such a plane graph (also called *triangulation of the plane*) is a plane graph to which one cannot add any edge without creating a parallel edge or a crossing with another edge.

Thus simple planar graphs have only a few edges. However, if the connectivity of a simple planar graph is high enough, then such graphs are hamiltonian.

Theorem 63 ([41]) *Every 4-connected planar graph has a hamiltonian cycle.*

We note in passing that this theorem by W. T. Tutte generalizes a result of H. Whitney proved in the 1930ies and stating that every 4-

connected triangulation of the plane has a hamiltonian cycle.

For the following conjecture and proposition, see, e.g., [16, Vol. 1, VI.112–VI.114].

Conjecture 64 (H. Fleischner) *Every simple eulerian triangulation of the plane has an A-trail.*

Proposition 65 *Conjectures 56 and 64 are equivalent.*

Thus we have a hamiltonian problem which can be treated as an eulerian problem. However, the following result should be viewed as a strategy in dealing with the problem of constructing hamiltonian cycles in graphs with relatively few edges (see [17] for the following and for detailed references). To state this and subsequent results, we need some new concepts.

Let M be a set (e.g., a graph) and let $\mathcal{S} = \{M_1, \ldots, M_p\}$ be a set of subsets of M such that $M = M_1 \cup M_2 \cup \cdots \cup M_p$. The *intersection graph* of $\mathcal{S}$, denoted by $I(\mathcal{S})$, is defined by

$$V(I(\mathcal{S})) = \mathcal{S} \text{ and}$$

$$M_iM_j \in E(I(\mathcal{S})) \text{ if and only if } i \neq j \text{ and } M_i \cap M_j \neq \emptyset.$$

Given a graph G, the *k-th power* G^k of G is defined by

$$V(G^k) = V(G) \text{ and}$$

$$xy \in E(G^k) \text{ if and only if } d_G(x,y) \leq k \text{ for } x,y \in V(G).$$

Finally, define the prism $\mathcal{P}(G)$ of the graph G as obtained from G and a copy G' of G by adding an edge vv' for every $v \in V(G)$, $v' \in V(G')$ where v' is the vertex corresponding to v.

Theorem 66 *An arbitrary graph G has a hamiltonian cycle if and only if there exists a set of cycles $\mathcal{S} = \{C_1, \ldots, C_k, k \geq 1\}$ such that*

1 $V(\mathcal{S}) := \bigcup_{i=1}^{k} V(C_i) = V(G)$

2 $C_i \cap C_j = \emptyset$ *or* $\cong K_2$, $1 \leq i, j \leq k$, $i \neq j$

3 $I(\mathcal{S})$ *is a tree.*

If such a set $\mathcal{S}$ exists then a hamiltonian cycle of G is defined by the edges belonging to precisely one element of $\mathcal{S}$.

This strategy has proved fruitful in proving the following two results.

Theorem 67 ([10, 11, 34]) *If G is 2-connected, then G^2 is hamiltonian.*

Theorem 68 ([15]) *If G is a planar 2-connected 3-regular graph, then $\mathcal{P}(G)$ is hamiltonian.*

The proof of these two theorems rests on the construction of special types of spanning subgraphs in the respective G. To see how the proof of Theorem 68 makes use of Theorem 66, consider a bipartite graph B having the following properties:

(a) $1 \leq d(v) \leq 3$ for every $v \in V(B)$;
(b) B is connected;
(c) any two cycles of B are disjoint;
(d) if $d(v) = 3$, then v lies on a cycle of B.

Whence the edges of B not lying on a cycle of B, are the bridges of B; and by (d), the set of these bridges induces a *linear forest*, i.e., a forest F each of whose components is a path. Hence B can be written as

$$B = E \cup F$$

where E is a set of pairwise disjoint even cycles, and E and F are edge-disjoint.

To see that $\mathcal{P}(B)$ is hamiltonian, construct a set of cycles according to Theorem 66: For a cycle $C = x_1, x_1x_2, \ldots, x_{2r}x_1, x_1$ of E $(l(C) = 2r)$ let $C' = x'_1, x'_1x'_2, x'_2, \ldots, x'_{2r}x'_1, x'_1$ be the corresponding cycle in the copy B' of B, and for a component $P = y_1, y_1y_2, \ldots, y_k$ of F, let $P' = y'_1, y'_1y'_2, \ldots, y'_k$ correspond to P.

Now set

$$H(C) := \Big\langle \{\, x_{2i-1}x_{2i}, x'_{2i}x'_{2i+1} : 1 \leq i \leq r, 2r+1 := 1, 2r = l(C) \,\} \cup \{\, x_jx'_j : 1 \leq j \leq 2r \,\} \Big\rangle$$

for arbitrary cycle $C \subset E$, and

$$H(P) := \Big\langle \{\, y_iy_{i+1}, y'_iy'_{i+1} : 1 \leq i \leq k-1 = l(P) \,\} \cup \{y_1y'_1, y_ky'_k\} \Big\rangle$$

for every component P of F. Now, each $H(C)$ and each $H(P)$ is a cycle such that

$$S := \{\, H(C), H(P) : C \text{ is a component of } E, P \text{ is a component } of F \}$$

is a set of cycles satisfying conditions *1-3* of Theorem 66. The hard part of proving Theorem 68 is to show that every planar 2-connected

3-regular graph G_3 has a *spanning* bipartite subgraph B as described above, provided one *does not* make use of the *Four Color Theorem* in its equivalent form whereby such G_3 has a 1-factorization. For if one has such 1-factorization $\{L_1, L_2, L_3\}$ at hand, then setting $E := \langle L_1 \cup L_2 \rangle$ and $F := \langle L_3' \rangle$ where $L_3' \subset L_3$ is of minimum size such that $E \cup F$ is connected, yields B as required. For more details of this construction of B, see [15].

However, the proof of Theorem 67 makes use of Theorem 66 as well, albeit in a more intricate and more implicit manner. Suffice it to say in this context that one starts by proving that every connected bridgeless graph has a connected spanning subgraph $S = E \cup F$ resembling - to some extent - B above. Namely: E is an eulerian graph, F is a linear forest, the edges of F are the bridges of S, and E and F are edge-disjoint (for details, see [10, 11]).

Unfortunately, if one drops in either of the two Theorems 67, 68 the condition that the respective graph is 2-connected and requires only connectedness, then the corresponding statement is false, in general. Figure 2.10 is the smallest example of a graph whose square is non-hamiltonian.

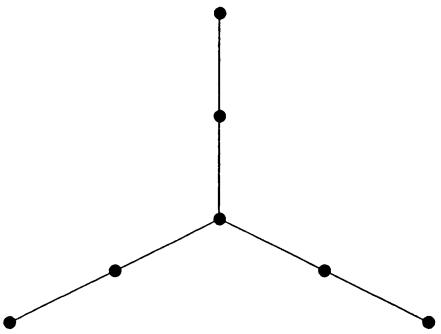


Figure 2.10

On the other hand, the following is true (call a graph *hamiltonian connected* if any two vertices can be joined by a hamiltonian path).

Theorem 69 ([39]) *If G is a connected graph, then G^3 is hamiltonian connected.*

Proof. It suffices to prove the theorem for any spanning tree T of G; T exists by Theorem 11. For $p := |V(T)| \in \{1, 2, 3\}$ this is trivially true, so assume w.l.o.g. $p \geq 4$, and choose $x, y \in V(T)$ arbitrarily.

Case 1: x, y are non-adjacent. Consider $P(x,y) \subset T$, $P(x,y) = x, xx', x', x'y', \ldots, y$; $x' \neq x, y$ (possibly $y' = y$). Form $T - x'y' = T_1 \dot{\cup} T_2$, where T_1, T_2 are trees ($x'y'$ is a bridge of T). W.l.o.g. $x' \in V(T_1)$, $y' \in V(T_2)$. It follows that $x \in V(T_1)$, $y \in V(T_2)$.

By induction T_1^3 has a hamiltonian path $P(x,x')$. Consider next $T_2^+ = T_2 \cup \{x', x'y'\} \subset T$. By induction $(T_2^+)^3$ has a hamiltonian path, $P_2(x', y)$.

Observing that T_1^3, $(T_2^+)^3 \subseteq T^3$, and because $V(T) = V(T_1) \cup V(T_2^+)$ we conclude that $P_1(x,x')$, $P_2(x',y)$ is a hamiltonian path in T^3 joining x, y.

Case 2: x, y are adjacent. $T = T_1 \cup \{xy\} \cup T_2$ where $T_1 \cap T_2 = \emptyset$, $x \in T_1$, $y \in T_2$, T_1 and T_2 are trees. Suppose $E(T_1) \neq \emptyset \neq E(T_2)$, then let x' be adjacent to x in T_1, y' adjacent to y in T_2. $d_T(x', y') = 3$, so $x'y' \in E(T^3)$. Let $P_1(x,x')$ be a hamiltonian path in T_1^3 joining x, x' and let $P_2(y', y)$ be a hamiltonian path in T_2^3 joining y', y. Then $P_1(x,x')$, $x'y'$, $P_2(y',y)$ is a hamiltonian path in T^3 joining x and y.

If w.l.o.g. $E(T_1) = \emptyset$, then let y' be adjacent to y in T_2. In T_2^3, we have a hamiltonian path $P_2(y', y)$. Note $xy' \in E(T^3)$ since $d_T(x, y') = 2$. Since $E(T_1) = \emptyset$, it follows that $x, xy', P_2(y', y)$ is a hamiltonian path in T^3 as required. ■

Theorems 67 and 69 also indicate that if a graph G has locally many edges, then one may hope to some extent that G is hamiltonian. Calling a $K_{1,3}$ a *claw* we define a graph G to be *claw-free* if no vertex-induced subgraph is a claw. A special type of a claw-free graph is the line graph $L(G)$ of a graph G; it is defined by

$$V(L(G)) = E(G), \quad \text{and}$$

$$ef \in E(L(G)) \quad \text{if and only if} \quad e, f \text{ are adjacent edges in } G.$$

Conjecture 70 (Matthews and Sumner, [32]) *Every 4-connected claw-free graph is hamiltonian.*

Conjecture 71 (Thomassen, [40]) *Every 4-connected line graph is hamiltonian.*

Noting that $\kappa(L(G)) = \lambda_e(G)$ and that $L(G)$ is hamiltonian if and only if G has a dominating connected eulerian subgraph (i.e., every edge of G is incident with a vertex of this subgraph) we conclude that the following is equivalent to Conjecture 71.

Conjecture 72 *Every essentially 4-edge-connected graph has a dominating connected eulerian subgraph.*

In fact, it has been shown that Conjectures 46, 70, 71, 72 are equivalent, [19, 38]. This brings us back to the consideration of 3-connected 3-regular graphs, for which it has been shown a long time ago that the decision problem whether such graphs have hamiltonian cycles, is $\mathcal{NP}$-complete even in the planar case (see e.g., [24]) – and this is the reason why the A-trail problem for 3-connected planar eulerian graphs is $\mathcal{NP}$-complete (see above). The same conclusion can be drawn w.r.t. dominating cycles: to see this, take any 3-regular 3-connected graph G (planar or not) and replace every vertex with a triangle, thus creating $G_\triangle$ (see Figure 2.11) which is also 3-regular and 3-connected.

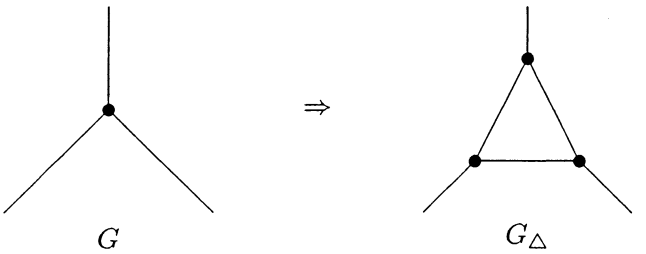


Figure 2.11

In fact, $G_\triangle$ has a dominating cycle if and only if G has a hamiltonian cycle.

However, Conjecture 46 addresses 3-regular graphs G with $\lambda_c(G) \geq 4$, whereas $\lambda_c(G_\triangle) = \lambda(G) = \kappa(G) \leq 3$ in general. This fact may be compared with the $\mathcal{NP}$-completeness of the decision problem regarding the existence of connected spanning eulerian subgraphs in arbitrary 3-edge-connected graphs (where it reduces to the hamiltonian problem in the 3-regular case) vis-à-vis the theorem that every 4-edge-connected graph has a connected spanning eulerian subgraph, [28]. – So, while deciding the existence of hamiltonian cycles is $\mathcal{NP}$-complete for planar 3-connected graphs, each of these graphs admits a closed walk passing through every vertex at least once and at most twice, [23].

We finish this section by referring the interested reader again to Bondy's survey article [7], and to Bermond's survey article [5].

6. ELEMENTS OF MATCHING THEORY

As origins of matching theory one may view Petersen's Theorem (Theorem 31) and Tait's equivalent formulation of the Four Color Problem in

terms of the existence of a 1-factorization in planar 2-connected 3-regular graphs (see the last part of section 3). Let us start with the question under which conditions certain classes of graphs have a 1-factor. For example, if a 3-regular graph G is such that it has a vertex which does not belong to any cycle of G, then G cannot have a 1-factor. So, what is a meaningful sufficient condition for a k-regular graph to have a 1-factor? When does a bipartite graph have a 1-factor? When does an arbitrary graph have a 1-factor? It is clear that in answering these questions it suffices to restrict considerations to simple graphs; and clearly, if a graph has a 1-factor, then it is of even order. The following results are answers to the above questions. Rather than give detailed references, we refer to *the* book on matching theory by Lovász and Plummer, [31].

Theorem 73 (Bäbler's Theorem) *Let $k \geq 1$, and let G be a k-regular graph of even order. If G is $(k-1)$-edge-connected, then G has a 1-factor.*

Theorem 74 (Marriage Theorem or Frobenius' Theorem) *A bipartite graph G with vertex bipartition $V(G) = A \dot{\cup} B$ has a 1-factor if and only if $|A| = |B|$ and $|A_1| \leq |N(A_1)|$ for every $A_1 \subseteq A$ ($N(A_1)$ is the set of vertices of B which are adjacent to at least one vertex in A_1).*

Theorem 75 (P. Hall's Theorem) *Let G be a bipartite graph with vertex bipartition $V(G) = A \dot{\cup} B$. G has a matching of A into B (i.e., a matching M such that every $a \in A$ is incident to some $e \in M$) if and only if $|N(A_1)| \geq |A_1|$ for every $A_1 \subseteq A$.*

Clearly, Theorem 75 implies Theorem 74. We shall see, however, that Theorem 74 also implies Theorem 75. For the following theorem, let $c_o(G)$ denote the number of components of G having odd order.

Theorem 76 (Tutte's 1-Factor Theorem) *A graph G has a perfect matching if and only if $c_o(G - S) \leq |S|$ for every $S \subseteq V(G)$.*

Before presenting the proof of this theorem as developed in [31], we need some preliminary discussion. Call a simple graph G *saturated non-factorizable* if G has no 1-factor, but $G \cup \{xy\}$ has a 1-factor for any $x, y \in V(G)$, $xy \notin E(G)$.

Lemma 77 *If G is saturated non-factorizable and of order p, and if S is the set of vertices of degree $p-1$, then the components of $G - S$ are complete graphs.*

Proof. Suppose for some $ab, bc \in E(G - S)$ (if such a, b, c exist) that $ac \notin E(G - S)$. It follows by definition of S that there is $d \in V(G)$ such that $bd \notin E(G)$. By the maximality of G, both $G \cup \{ac\}$ and $G \cup \{bd\}$ have a 1-factor, F_1 and F_2 respectively. Consider the symmetric difference $F_1 \triangle F_2 := (F_1 - F_2) \cup (F_2 - F_1)$: it consists of alternating cycles

(w.r.t. F_1 and F_2), which thus have even length. Denote them such that $ac \in E(C_1)$, $bd \in E(C_2)$.

Case 1: $C_1 \neq C_2$. Then $F_3 := F_1 \triangle E(C_1)$ is a 1-factor of G, contradicting the hypothesis.

Case 2: $C := C_1 = C_2$. Then C can be written w.l.o.g. as

$$C = b, bd, d, dx, x, \ldots, y, ya, a, ac, c, \ldots$$

(i.e., in traversing C from b along bd, a is being reached before c - note that both $ab, bc \in E(G - S)$). Let $P(b,a) \subset C$ be the path from b to a (in C). Then $C' := P(b,a)$, ab, b is an alternating cycle w.r.t. F_2 and $E(G) - F_2$. Thus $F_4 := F_2 \triangle E(C')$ is a 1-factor in G, again a contradiction.

Whence it follows in both cases that if G' is a component of $G - S$ of order $p' \geq 3$, then no two vertices of G' are of distance 2 apart, i.e., G' is a complete graph. ■

Lemma 78 *G is saturated non-factorizable if and only if precisely one of the following statements is true.*

1. *G is a complete graph of odd order.*
2. *G is of even order and contains disjoint complete subgraphs $\langle S_0 \rangle$, $G_1, \ldots, G_k$ covering all of $V(G)$, where $S_0 \subseteq V(G)$ and $k = |S_0| + 2$; G_i is of odd order, and every $x \in V(G_i)$ is adjacent to every $s \in S_0$ for every $i \in \{1, \ldots, k\}$ (see Figure 2.12 - note that $S_0 = \emptyset$ and/or $G_i \simeq K_1$ may hold for some i).*

Proof. Suppose G is saturated non-factorizable. If G is of odd order, then it must be a complete graph, trivially (no graph of odd order has a 1-factor). Whence assume G is of even order, and let S_0 be the set of vertices of degree $p - 1$; possibly $S_0 = \emptyset$. In any case, by definition of S_0, $\langle S_0 \rangle$ is a complete graph. Let $G_1, \ldots, G_k$ be the components of $G - S_0$ having odd order. By Lemma 77, each G_i is a complete graph. Moreover, by definition of S_0, every $x \in V(G_i)$ is adjacent to every $s \in S_0$. If $k \leq |S_0|$, possibly $k = 0$, then consider for each component G' of $G - S_0$ of even order p' a 1-factor F', and if $k \geq 1$, take a 1-factor F_i in $G_i - x_i$ for fixed $x_i \in V(G_i)$, $1 \leq i \leq k$ ($F_i = \emptyset$, *possibly*). Since $k \leq |S_0|$, there exists a matching M_0 of $\{ x_i : 1 \leq i \leq k \}$ into S_0. The set of yet unmatched vertices of S_0 induces a complete graph of even order (since G is of even order), which has a 1-factor F_0. Thus

$$\bigcup_{p' even} F' \cup \bigcup_{i=1}^{k} F_i \cup M_0 \cup F_0$$

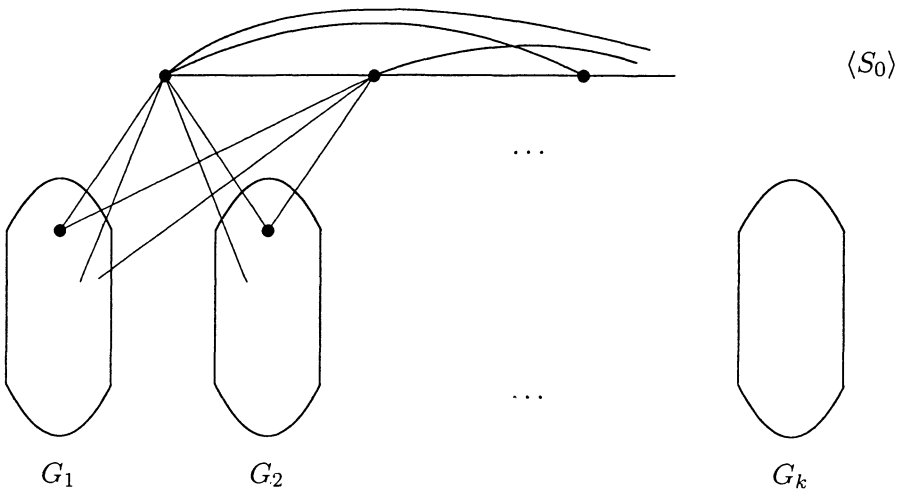


Figure 2.12

is a 1-factor of G, a contradiction. Thus $k \geq |S_0| + 1$, and since G is of even order it follows by parity that $k \geq |S_0| + 2$. In case $k \geq |S_0| + 3$ add an edge xy for $x \in V(G_{k-1})$, $y \in V(G_k)$. Then

$$c_o(G \cup \{xy\} - S_0) = k - 2 \geq |S_0| + 1 > |S_0|.$$

However, $G^+ := G \cup \{xy\}$ has a 1-factor F^+ at least $k-2$ edges of which are incident with a vertex in S_0 (at least one edge per odd component of $G^+ - S_0$). Thus, $|S_0| \geq k - 2$ which is an obvious contradiction to the last inequality. Whence $k = |S_0| + 2$.

Likewise, if $G - S_0$ had an even component G_0, form $G^+ := G \cup \{yz\}$ where $y \in V(G_k)$, $z \in V(G_0)$. G^+ having a 1-factor implies $|S_0| \geq c_o(G^+ - S_0) = k = |S_0| + 2$, again an obvious contradiction. Thus each component of $G - S_0$ is of odd order. Thus, if G is of even order, then it has the structure as described in 2.

Conversely, if G is as described in 1., then it is saturated non-factorizable. If it is as described in 2., then edges can be added only between Components of $G - S_0$ to obtain a simple graph again. Adding just one such edge to obtain G^+ and observing that $c_o(G^+ - S_0) = k$, we can construct a 1-factor F^+ of G^+ in a manner similar to the above case $k \leq |S_0|$, with F^+ containing the additional edge. Thus G is saturated non-factorizable. The Lemma now follows. ■

The proof of Theorem 76 is now easy: first of all, we may extract from the proof of Lemma 78 that if G has a 1-factor, then $c_o(G - S) \leq |S|$

for every $S \leq V(G)$. To show the converse we proceed by contradiction; i.e., we assume the validity of this inequality for some G having no 1-factor. Next, add edges between non-adjacent vertices of G until G' is reached such that G' has no 1-factor, but adding any edge between non-adjacent vertices of G' results in G'' having a 1-factor. Thus G' is saturated non-factorizable. Since $c_o(G-S) \leq |S|$ by hypothesis, we also have $c_o(G'-S) \leq |S|$, because adding edges does not increase the number $c_o(G'-S)$. This implies, in particular, that G and thus also G' must be of even order; otherwise, take $S = \emptyset$ to obtain a contradiction to $c_o(G'-S) = c_o(G') \leq |S| = 0$ since $c_o(G') \geq 1$ if G' is of odd order. Now, defining S_0 w.r.t. G' as before (possibly $S_0 = \emptyset$) we obtain by Lemma 78.2., that $c_o(G'-S_0) = |S_0|+2$. Since $c_o(G-S_0) \geq c_o(G'-S_0)$ we obtain altogether $c_o(G-S_0) \geq |S_0|+2$, contradicting the assumption $c_o(G-S) \leq |S|$ for all $S \subseteq V(G)$. The theorem now follows.

Now we use Theorem 76 to deduce the validity of Theorems 73 and 74. Suppose first, that Theorem 73 is false, and let $r \geq 1$ be an integer such that there is an r-regular $(r-1)$-edge-connected G for which the theorem fails. By Theorem 76, there is $S \subseteq V(G)$ such that $c_o(G-S) \geq |S|+2$; this inequality follows by parity since G is of even order (see the analogous argument in the proof of Lemma 78). Let $G_1, \ldots, G_k, k = c_o(G-S)$, be the odd components of $G-S$. We claim that there are at least r edges joining vertices of G_i to vertices of S, for $i = 1, \ldots, k$. There are at least $r-1$ such edges anyway since G is $(r-1)$-edge-connected. However, suppose there is an edge cut E_i of precisely $r-1$ edges separating some G_i from the rest of the graph, for some $i \in \{1, \ldots, k\}$. Let G_i^+ be obtained from G_i by introducing $z \notin V(G)$ and joining z to every $x \in V(G_i)$ by $|E_x \cap E_i|$ edges. Then $d_{G_i^+}(x) = r$ and $d_{G_i^+}(z) = r-1$. However, since G_i is of odd order, G_i^+ has an odd number of odd vertices, contradicting the Handshaking Lemma. This proves our claim.

Consequently we have $|E_i| \geq r$ for the edge cut E_i separating G_i from the rest of the graph for every $i = 1, \ldots, k$, and since these $|E_i|$ edges run between vertices of G_i and S, we have

$$|E(S, \bar{S})| \geq \sum_{i=1}^{k} |E_i| \geq rk \geq r(|S|+2) > r|S|.$$

On the other hand, G being r-regular implies $|E(S, \bar{S})| \leq r|S|$, an obvious contradiction to the preceding inequality. This proves Theorem 73.

To derive the Marriage Theorem from Theorem 76 we first observe that if G has a 1-factor F, then $|N(A_1)| \geq |A_1|$ follows of necessity for every $A_1 \subseteq A$: for the edges of F incident to A_1 cover precisely $|A_1|$ vertices in $N(A_1)$. To see that the inequality $|N(A_1)| \geq |A_1|$ for every

$A_1 \subseteq A$ implies that G has a 1-factor, we proceed again by contradiction. Consequently, a bipartite graph G with $|A| = |B|$ satisfying this inequality exists, yet it has no 1-factor. By Theorem 76, $S \subseteq V(G)$ exists such that $c_o(G - S) > |S|$. We observe that trivially $S \neq V(G)$, and $S \neq \emptyset$ since G is of even order ($|V(G)| = |A| + |B| = 2|A|$); moreover, $S \neq A, B$ since $c_o(G - A) = c_o(G - B) = |A| = |B|$. Thus we can write $S = A_1 \dot{\cup} B_1$ with $A - A_1 \neq \emptyset \neq B - B_1$. Among all possible choices for the above S, let S be chosen of maximum size.

Suppose now that $G-S$ had a nontrivial odd component G_0. Then for the corresponding bipartition $V(G_0) := A_0 \dot{\cup} B_0$ we have $A_0 = V(G_0) \cap A$, $B_0 = V(G_0) \cap B$, $|A_0| \neq |B_0|$. If $|A_0| > |B_0|$ define $S_0 := S \cup B_0$. Since $G - S_0 = (G - S) - B_0$, we then have $c_o(G - S_0) = c_o(G - S) + |A_0| - 1 \geq |S| + |A_0| > |S| + |B_0| = |S_0|$, contradicting the choice of S. Likewise, if $|B_0| > |A_0|$ we set $S_0 := S \cup A_0$ to obtain an analogous contradiction. Thus every odd component of $G - S$ is a single vertex. Thus the subgraph induced by these odd components of $G - S$ can be written in the form $A_1' \dot{\cup} B_1'$ with $A_1' \subseteq A$, $B_1' \subseteq B$, and trivially $A_1' \cap A_1 = B_1' \cap B_1 = \emptyset$. Clearly,

$$|S| = |A_1| + |B_1| < |A_1'| + |B_1'| = c_o(G - S),$$
$$\text{and } N(A_1') \subseteq B_1, N(B_1') \subseteq A_1.$$

By hypothesis, however, we also have

$$|N(A_1')| \geq |A_1'|.$$

Thus we obtain $|B_1| \geq |A_1'|$, implying

$$|A_1| + |B_1| < |A_1'| + |B_1'| \leq |B_1| + |B_1'|$$

and hence $|N(B_1')| \leq |A_1| < |B_1'|$. (*)

In any case, $N(A - N(B_1')) \subseteq B - B_1'$; and by hypothesis

$$|N(A - N(B_1'))| \geq |A - N(B_1')|.$$

Combining these two inequalities we obtain

$$|B - B_1'| = |B| - |B_1'| \geq |N(A - N(B_1'))| \geq |A - N(B_1')| = |A| - |N(B_1')|;$$

i.e., $|N(B_1')| \geq |B_1'|$, a contradiction to inequality (*). Whence we conclude that $S \subseteq V(G)$ with $|S| < c_o(G - S)$ does not exist. This finishes the proof of the Marriage Theorem.

To see finally that the Marriage Theorem implies P. Hall's Theorem, we first observe that the inequality stated in the latter theorem is a necessary condition indeed, just the same as it was in the case of the

former theorem. Now, to prove sufficiency it follows from the choice $A_1 = A \subset V(G)$ that $|B| \geq |A|$. If $|B| = |A|$ then P. Hall's Theorem reduces to the former theorem, and nothing has to be proved anymore.

Whence assume $|B| > |A|$. Construct a new bipartite graph G^+ by adding k new vertices $z_1, \ldots, z_k$ and joining each of them to every $v \in B$, where $k = |B| - |A|$. Thus we have $|B^+| = |A^+|$ for the bipartition $V(G^+) = A^+ \dot{\cup} B^+$. Note that $B^+ = B$ and $z_i \in A^+$ for $i = 1, \ldots, k$. Let $A_1^+ \subseteq A^+$ be arbitrarily chosen. If $z_j \in A_1^+$ for at least one j, then $N(A_1^+) = B$ and therefore, $|N(A_1^+)| \geq |A_1^+|$. If, however, $A_1^+ \subseteq A$, then $|N(A_1^+)| \geq |A_1^+|$ follows by the hypothesis of Hall's Theorem. It follows that G^+ has a 1-factor F_1. Setting $F := F_1 - \{ e \in F_1 \cap E_{z_j} : j \in \{1, \ldots, k\} \}$, we conclude that F is a matching in G as required.

The Marriage Theorem can also be used to deduce that *every regular bipartite graph has a 1-factor and therefore, a 1-factorization* (thus no longer requiring any connectivity condition as in Bäbler's Theorem). Namely: Considering for a connected r-regular bipartite graph G with vertex bipartition $V(G) = A \dot{\cup} B$ any $A_1 \subset A$ one deduces for the edge cut E_0 separating $\langle A_1 \cup N(A_1) \rangle$ from $\langle (A - A_1) \cup (B - N(A_1)) \rangle$ that

$$|E(\langle A_1 \cup N(A_1) \rangle)| = r|A_1| = r|N(A_1)| - |E_0| \leq r|N(A_1)|;$$

$$\text{i.e., } |A_1| \leq |N(A_1)|$$

for every $A_1 \subseteq A$. That is, since we necessarily have $|A| = |B|$, G has a 1-factor L_1 by the Marriage Theorem. Thus, $G_1 := G - L_1$ is a bipartite $(r-1)$-regular graph, and adding L_1 to a 1-factorization of G_1 (which exists by induction since a 1-regular graph has its edge set as its only 1-factor and is bipartite), one obtains a 1-factorization of G. We also note that this result is equivalent to the existence of a 2-factorization in arbitrary $2r$-regular graphs.

The following result is of central importance in matching theory. Its proof follows along the lines of [31, 1.2.2. Lemma].

Theorem 79 (König's Minimax Theorem) *Let G be a bipartite graph. The size of a maximum matching M_0 equals the minimum size of a set $V_0 \subseteq V(G)$ such that each edge of G is incident with (at least) one vertex in V_0.*

Proof. Denote $m(G) := |M_0|$ and $t(G) := |V_0|$ where M_0 and V_0 are defined as above for an arbitrary (not necessarily bipartite) graph G. For every $e \in M_0$, at least one incident vertex must lie in V_0 by definition of V_0. Hence $t(G) \geq m(G)$ for arbitrary graphs.

Now consider a maximum matching M_0 in a bipartite graph G. W.l.o.g., G is connected; otherwise, establishing $t(G_i) = m(G_i)$ for each component G_i of G yields $t(G) = m(G)$. Denote by A and B, respectively, the two classes in the vertex bipartition of $V(G)$. Let $A_0 \subseteq A, B_0 \subseteq B$ be the respective sets of vertices incident with edges of M_0. If $A_0 = A$, then every edge of G is incident with a vertex of A_0, whence $t(G) \leq |A_0| = m(G)$; so in this case $t(G) = m(G)$. We arrive at the same conclusion if $B_0 = B$. Whence assume $A - A_0 \neq \emptyset \neq B - B_0$. Set $A' := A - A_0$, $B' := B - B_0$. Also, no $a \in A'$ is adjacent to any $b \in B'$ since M_0 is a maximum matching. Thus, $N(B_0) = A$ and $N(A_0) = B$, and all edges of G are incident with some $a \in A_0$ or some $b \in B_0$.

Consider now the set of paths $P_{A'}$ and $P_{B'}$, which start at a vertex in A', B' respectively, and are alternating in the edges of $E(G) - M_0$ and M_0. It follows from the maximality of M_0 that

$$P_{A'} \cap P_{B'} = \emptyset \text{ for every such } P_{A'} \text{ and every such } P_{B'} \tag{1}$$

It may very well happen that some $e \in M_0$ belongs to no $P_{A'}$ and no $P_{B'}$. In any case, the subgraphs $G_{A'} := \langle \bigcup V(P_{A'}) \rangle$ and $G_{B'} := \langle \bigcup V(P_{B'}) \rangle$ satisfy $G_{A'} \cap G_{B'} = \emptyset$ because of (1). It also follows that

$$\begin{aligned} &\text{any } e \in E(V(G_{A'}), \overline{V(G_{A'})}) \text{ is incident to a } b_0 \in B_0 \cap V(G_{A'}), \\ &\text{any } e \in E(V(G_{B'}), \overline{V(G_{B'})}) \text{ is incident to a } a_0 \in A_0 \cap V(G_{B'}). \end{aligned} \tag{2}$$

Thus, all edges of G incident with some vertex of $G_{A'}(G_{B'})$ are covered by (i.e., incident with an element of) $B_0' := B_0 \cap V(G_{A'})$ $(A_0' := A_0 \cap V(G_{B'}))$ (see Figure 2.13).

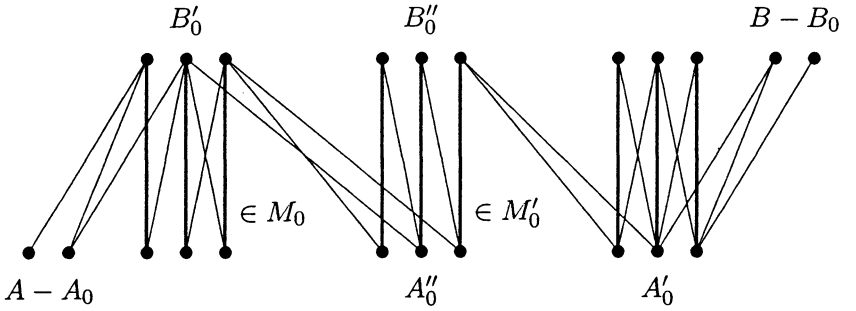


Figure 2.13

Next, set $M_0' := \{e \in M_0 - (E(G_{A'}) \cup E(G_{B'}))\}$, and let $A_0'' \subseteq A_0$, $B_0'' \subseteq B_0$ respectively, be the respective set of vertices incident with some element of M_0'. Choose $X_0 \in \{A_0'', B_0''\}$ arbitrarily. It follows from (1) and (2) that

$$m(G) = |M_0| = |A_0'| + |B_0'| + |X_0| \tag{3}$$

and that every edge of G is incident with some $x \in A'_0 \cup B'_0 \cup X_0$. Thus $t(G) \leq m(G)$ by (3) implying $t(G) = m(G)$ because of $t(G) \geq m(G)$ for arbitrary graphs. The theorem now follows. ■

The proof of Theorem 79 gives rise to new concepts which, in turn, leads to a characterization of maximum matchings in arbitrary graphs, and a good matching algorithm in bipartite graphs. To this end, consider an arbitrary graph G and an arbitrary matching M in G. Let P_M be an alternating path w.r.t. M and $E(G) - M$ (P_M need not start/end with an edge of M); call P_M an *M-alternating path.* Call $v \in V(G)$ *not covered* by M if $E_v \cap M = \emptyset$. Trivially, if an M-alternating path P connects v and w which are not covered by M, then the symmetric difference $M \triangle E(P)$ is a matching larger than M. Such P is called an *M-augmenting path.* Thus, if M is a maximum matching in G, then G contains no M-augmenting path. In fact, the converse holds as well.

Theorem 80 (Berge, [3]) *A matching M in a graph G is a maximum matching if and only if G contains no M-augmenting path.*

Proof. From what we just said it suffices to assume for a matching M in G that G has no M-augmenting path. Let M_1 be a maximum matching in G and form $M \triangle M_1$ each component of which is either an alternating cycle C w.r.t. M and M_1, or both an M-alternating and M_1-alternating path P (the case $M \triangle M_1 = \emptyset$ is trivial since it means that $M = M_1$ is a maximum matching). However, P cannot be M-augmenting by assumption. It cannot be M_1-augmenting either since M_1 is a maximum matching. Thus $\ell(P)$ is even of necessity, implying for arbitrary $K \in \{C, P\}$ $|E(K) \cap M| = |E(K) \cap M_1|$ from which $|M| = |M_1|$ follows which had to be shown. ■

Next, consider again a connected bipartite graph G with vertex bipartition $V(G) = A \dot{\cup} B$; let M be any matching in G, and let $A' \subseteq A$, $B' \subseteq B$ be the set of vertices which are not covered by M. Denote by $U \subseteq A$ the set of vertices reachable by an M-alternating path starting in A' (cf. this with the proof of Theorem 79: $U = V(G_{A'}) \cap A$). To construct U, 'grow' a maximal forest F having the following properties:

1 $d_F(b) = 2$ for every $b \in V(F) \cap B$, and some $e \in M$ is incident with b in F.

2 every component of F contains a point of A'.

It follows that $A' \subseteq V(F)$ ($a \in A'$ can be added as a singleton component), and that $U = V(F) \cap A$. The following is basically a consequence of the proof of Theorem 79.

Corollary 81 *Let G, M, A, B, A', B' and F be as above. M is a maximum matching if and only if no $b' \in B'$ is adjacent to any $a \in V(F) \cap A$.*

The following algorithm derives from the preceding considerations; the approach used in it is called the *Hungarian Method*, and the final forest constructed by it (see below) is a *Hungarian forest* (see [31, p. 15]). We use the notation explained above.

Algorithm 82 (Bipartite Matching Algorithm)
Input: *Bipartite graph G with $V(G) = A \dot{\cup} B$, any matching M.*
Step 1. *Form a maximal forest F with the above properties 1. and 2.*
Step 2. *If $xy \in E(G)$ exists such that $x \in V(F) \cap A$, $y \in B'$, then $F \cup \{y, xy\}$ contains an M-augmenting path P; $M' := M \triangle E(P)$ is a larger matching. Set $M := M'$ and go to Step 1.*
Step 3. *No $xy \in E(G)$ with the above properties exists: M is a maximum matching.*

We note in passing that one can transform this algorithm into an LP for which its implementation produces a maximum matching in polynomial time. This is even true in the case of a maximum/minimum cost perfect matching where the edges of a graph are assigned costs or weights (see below and the next section). Finally let us remark that finding a maximum (maximum/minimum cost) matching in an arbitrary graph is much more involved, and thus cannot be treated in this chapter; this will be done in other chapters of this book (see Derigs, this book). Again, the interested reader is referred to [31] for a thorough discussion of this topic which is irrevocably connected with the work of J. Edmonds.

We now turn to other algorithmic aspects of applied graph theory which are essential for the treatment of various traversal problems: determining shortest paths in a (di)graph and a minimum cost spanning tree in a graph. In both cases, the edges/arcs of the (di)graph under consideration are assigned non-negative real numbers called *the length or the cost of traversing* the respective edge/arc, depending on the context.

The Shortest Path Problem. We are given a digraph $D = V \cup A$, $A = \{a_1, \ldots, a_q\}$ and non-negative real numbers ℓ_i associated with $a_i, i = 1, \ldots, q$ (ℓ_i can be thought of as the 'length' of a_i). For $\bar{a}, \bar{b} \in V(D)$, find a path $W = \bar{a}, a_{i_1}, v_1, a_{i_2}, \ldots, a_{i_r}, \bar{b}$ such that $\ell(W) := \sum_{j=1}^{r} \ell_{i_j}$ is minimum.

The following polynomial time algorithm has been found independently by Dantzig and Dijkstra; for a discussion of this algorithm see, e.g., [22, 36, 42].

Algorithm 83 (Shortest Path Algorithm)
Input: *A digraph $D = V \cup A$ with arc lengths as described in Shortest Path Problem above.*
Step 1. *Set $t(\bar{a}) = 0$, $v_0 := \bar{a}$, $V_0 = \{v_0\}$.*
Step 2. *For every $v_i \in V_k, k \geq 0$, determine an "immediate successor" $v_j \notin V_k$ (i.e., $(v_i, v_j) \in A$) such that for $(v_i, v_j) = a_m$, $\ell_m = \min\{\ell_r : v_i \text{ is tail of } a_r\}$. $\ell_m =: \ell_m(v_i)$. Determine $v_i^* \in V_k$ such that $t(v_i^*) + \ell_m(v_i^*) = \min_{v_i \in V_k}\{t(v_i) + \ell_m(v_i)\}$. Denote by w_i^* a vertex such that (v_i^*, w_i^*) has length $\ell_m(v_i^*)$.*
Step 3. *Set $V_{k+1} = V_k \cup \{w_i^*\}$ and set $t(w_i^*) = t(v_i^*) + \ell_m(v_i^*)$. Set $k := k + 1$ and go to Step 2 if a vertex $v \notin V_{k+1}$ adjacent from some vertex in V_{k+1} exists.*
Step 4. *If $\bar{b} \in A_k$ then stop. $t(\bar{b})$ is the length of a shortest path from $\bar{a}$ to $\bar{b}$ in D. If $\bar{b} \notin A_k$ then no path from $\bar{a}$ to $\bar{b}$ exists.*

We refrain from justifying this algorithm (i.e., that it delivers what it claims to achieve - such justification can be found in the literature cited above), but note in passing that its running time is $O(n^3)$ which can be improved to $O(n^2)$ by slightly modifying the algorithm (see [36]). We also observe that the Shortest Path Problem can be rephrased as a matching problem in a subgraph of the prism $P(G)$, [31]. Matching theory can also be employed to solve the more general case in which the real numbers ℓ_i need not be non-negative.

Observe that the above algorithm can be easily adapted to the case of graphs: just replace every edge e_i, $1 \leq i \leq q$, by two oppositely oriented arcs each of which is assigned the same length ℓ_i as e_i was.

The Minimum Cost Spanning Tree Problem. In an undirected connected graph $G = V \cup E$, $E = \{e_1, \ldots, e_q\}$, every e_i is associated with a real number ℓ_i (the "length" of e_i). Find a spanning tree T in G such that $\ell(T) := \sum_{e_i \in E(T)} \ell_i$ is minimum.

The next algorithm is known as *Kruskal's Algorithm* but goes back to Boruvka, [25]. It produces a minimum cost spanning tree very fast; because of its structure it is often quoted as an example of a *Greedy Algorithm.*

Algorithm 84 (Kruskal's Algorithm)
Input : *Connected Graph G of order p, $E(G) = \{e_1, \ldots, e_q\}$, e_i associated with real number ℓ_i. W.l.o.g. $\ell_1 \leq \ell_2 \leq \cdots \leq \ell_q$.*
Step 1. *$E_1 := \{e_1\}$, $G_1 := \langle E_1 \rangle$, set $i = 1$.*
Step 2. *Find the minimum $j > max\{\, k : e_k \in E_i \,\}$ such that $\langle E_i \cup \{e_j\} \rangle$ is acyclic. Set $E_{i+1} = E_i \cup \{e_j\}$, $G_{i+1} = \langle E_{i+1} \rangle$. Set $i = i + 1$*
Step 3. *If $i = p - 1$ go to Step 4; otherwise go to Step 2.*
Step 4. *G_i is a minimum cost spanning tree of G.*

Justification of the algorithm. Since G_{p-1} is an acyclic graph on p vertices and $p-1$ edges, it is a spanning tree of G (see Theorem 12.5.); and G_{p-1} exists because G is connected: for, suppose no edge of $E(G) - E_i$ could be added to E_i for some $i < p-1$. Then there would be a component $G_i' \subseteq G_i$ such that the edge cut $E' := E(V(G_i'), \overline{V(G_i')})$ in G would be non-empty since G is connected and $|V(G_i')| \leq p-1$ (G_i' is a tree of size $\leq i$). Yet, for every $e_r \in E'$, $G_i \cup \{e_r\}$ contains a cycle C, so the two vertices x, y incident with e_r can be joined by the path $C - e_r$ in G_i, i.e., $x, y \in V(G_i')$, an obvious contradiction to $e_r \in E'$. It remains to show that $T_0 := G_{p-1}$ is of minimum cost. Let T be any minimum cost spanning tree, $E(T) = \{e_{r_1}, \ldots, e_{r_{p-1}}\}$ such that $r_1 \leq r_2 \leq \cdots \leq r_{p-1}$. W.l.o.g., $T \neq T_0$. Consequently $\ell_{r_1} \leq \ell_{r_2} \leq \cdots \leq \ell_{r_{p-1}}$ (note that by the assumption in *Input*, $j < k$ implies $\ell_j \leq \ell_k$). By the choice of $e_1, \ell_1 \leq \ell_{r_1}$. If $e_1 \notin E(T)$, then form $T \cup \{e_1\}$. It has a unique cycle C_1 which contains e_1, by Theorem 12.6. It follows that $\ell_k = \ell_1$ for every $e_k \in E(C_1)$. For otherwise, one could delete $e_k \in E(C_1)$ with $\ell_k > \ell_1$ to create a spanning tree $T_1 = (T \cup \{e_1\}) - \{e_k\}$ of G (see the discussion following Theorem 11) with $\ell(T_1) < \ell(T)$, a contradiction to the choice of T. Whence T_1 is also a minimum cost spanning tree. So set $T_1 = T$ if $e_1 \in E(T)$ and T_1 as above, otherwise. Assume, by induction, that we have constructed a minimum cost spanning tree T_s, $s \geq 1$, such that $e_{i_1}, \ldots, e_{i_s} \in E(T_s)$ and $E(T_s) - \{e_{i_1}, \ldots, e_{i_s}\} \subseteq E(T)$, where $E_{p-1} = \{e_{i_1}, \ldots, e_{i_{p-1}}\}$, $i_1 = 1$, and such that $\ell_{i_s} \leq \ell_{r_{s+1}}$. Analogously, set $T_{s+1} = T_s$ if $e_{i_{s+1}} \in E(T_s)$; otherwise, $T_s \cup \{e_{i_{s+1}}\}$ contains a unique cycle C_{s+1} and $e_{i_{s+1}} \in C_{s+1}$. By construction of $E_{s+1}, \ldots, E_{p-1}$, it follows that $E'' := E(C_{s+1}) \cap (E(T_s) - E(T_0)) \neq \emptyset$. For $e_k \in E''$ it follows of necessity that $\ell_k = \ell_{i_{s+1}}$; otherwise, $\ell_k > \ell_{i_{s+1}}$ yields a spanning tree $T'' := (T_s \cup \{e_{i_{s+1}}\}) - \{e_k\}$ with $\ell(T'') < \ell(T)$ contradicting the choice of T, whereas $\ell_k < \ell_{i_{s+1}}$ contradicts the choice of $e_{i_{s+1}}$ instead of e_k in constructing E_{s+1}. Whence T'' can be constructed anyway if $e_{i_{s+1}} \notin E(T_s)$, and so $T_{s+1} := T''$ is also a minimum cost spanning tree. It now follows by induction that $T_{p-1} = T_0$ is a minimum cost spanning tree.

7. THE CHINESE POSTMAN PROBLEM, THE TRAVELING SALESMAN PROBLEM, AND RELATED PROBLEMS

As we shall see below, the *Chinese Postman Problem (CPP)* – in essence – seeks to double certain edges of a given graph G such that the resulting graph is eulerian. The Hamiltonian Walk Problem (which is closely related to the Traveling Salesman Problem), on the other hand, seeks to double certain edges of a certain connected spanning subgraph G' of G such that the graph resulting from G' is eulerian.

The CPP was proposed in 1959 by Kwan Mei-Ko (= Guan Meigu) and states in its general form the following: If $G = V \cup E$ is a connected graph together with a *cost function* $c : E \to \mathbb{R}^+$, find a closed walk $W := v_1, e_1.v_2, e_2, \ldots, v_n, e_n, v_1$ that starts at a given $v_1 \in V$ and traverses each edge at least once so that the total cost $c(W) := \sum_{i=1}^{n} c(e_i)$ is minimum. Such a walk is called a *Postman Tour (PT).*

We immediately have the following set of inequalities (for $E_o \subseteq E$ we set $c(E_o) := \sum_{e \in E_o} c(e)$, and for $G_o \subseteq G$ we set $c(G_o) := c(E(G_o))$):

$$c(G) \leq c(W) \leq 2c(G)$$

The lower bound is attained if and only if G is eulerian, whereas the upper bound is attained if and only if G is a tree.

Theorem 85 *A closed covering walk (CCW) W in the connected graph G with given $c := E(G) \to \mathbb{R}^+$ is a PT if and only if the following two conditions hold:*

1 $\lambda_W(e) \leq 2$ *for every* $e \in E(G)$.

2 for every cycle C of G, $c(E_2(C)) \leq \frac{1}{2}c(C)$

$$\text{where} \quad E_2(C) = \{\, e \in E(C) : \lambda_W(e) = 2 \,\}.$$

Proof. ($\Rightarrow$) Define G^+ by replacing each edge e of G by $k_e := \lambda_W(e)$ parallel edges $e^{(1)}, \ldots, e^{(k_e)}$ joining the same vertices as e. Then W corresponds in G^+ to T^+, an eulerian trail of G^+. If e exists such that $\lambda_W(e) \geq 3$ then remove $e^{(k_e)}, e^{(k_e-1)}$ from G^+ to get a new graph that is still connected and eulerian and therefore has an eulerian trail which corresponds to a CCW W' in G with $c(W') < c(W)$, a contradiction.

If there is a cycle C such that $c(E_2(C)) > \frac{1}{2}c(C)$ then let $G^{++} := (G^+ - \{\, e^{(2)} : e \in E_2(C) \,\}) \cup \{\, e^{(2)} : e \in E(C) - E_2(C) \,\}$; it is also a connected eulerian supergraph of G (for $e \in E(C) - E_2(C)$, $e^{(2)}$ means adding an edge parallel to e). An eulerian trail T_1 of G^{++} corresponds to a closed covering walk W_1 in G having total cost $c(W_1) = c(W) - c(E_2(C)) + (c(E(C)) - c(E_2(C))) = c(W) + c(C) - 2c(E_2(C)) < c(W)$, a contradiction.

($\Leftarrow$) Let W' be a CCW of G satisfying conditions *1*, *2* and let W be a PT of G. Note that W also satisfies *1* and *2* because of the first part of the proof. We need to show that $c(W') = c(W)$, i.e. that W' is also a PT. Define $E_2(G)$ analogously to $E_2(C)$ above and set

$$G_1 := \langle \{E_2(G) \text{ with respect to } W'\} \rangle_G$$

$$G_2 := \langle \{E_2(G) \text{ with respect to } W\} \rangle_G$$

$$G_0 := G_1 \triangle G_2$$

It follows for every $v \in V(G)$ that $d_{G_1}(v) \equiv d_{G_2}(v) \equiv d(v) \pmod 2$; thus G_0 is eulerian.

We may assume $E(G_0) \neq \emptyset$; otherwise, $c(G_1) = c(G_2)$ and hence $c(W') = c(W)$. However, $E(G_0) \neq \emptyset$ implies that G_0 has a nontrivial cycle decomposition $\{C_1, \ldots, C_r\}, r \geq 1$, so that $\sum_{i=1}^{r} c(C_i) = c(G_0)$. We have by assumption $c_{i,1} := c(E(C_i) \cap E(G_1)) \leq \frac{1}{2}c(C_i)$ and since W is PT, we also have $c_{i,2} := c(E(C_i) \cap E(G_2)) \leq \frac{1}{2}c(C_i)$, $i = 1, \ldots, r$. However, $c_{i,1} + c_{i,2} = c(C_i)$ implying equality in the preceding inequalities. Therefore, $c(E(G_0) \cap E(G_1)) = c(E(G_0) \cap E(G_2)) = \frac{1}{2}c(G_0)$ and therefore $c(G_1) = c(G_2)$ implying again $c(W') = c(W)$. It follows that W' is also a PT. ■

However, as interesting as Theorem 85 is (Guan Meigu proved it originally only for the case $c \equiv 1$), it is not feasible per se for solving the CPP algorithmically since the number of cycles in G might be exponentially large when compared with the size of G. Nonetheless, the structure of $E_2(G)$ points into the right direction.

Let W be a PT in G with given $c : E(G) \to \mathbb{R}^+$; set $H := \langle \{ e \in E(G) : \lambda_W(e) = 2 \} \rangle$. By the above,

1. H is acyclic.

2. $d_G(v) \equiv d_H(v) \mod 2$ for every $v \in V(G)$.

If an arbitrary $H \subseteq G$ (H not necessarily connected) satisfies both conditions above then one can get a CCW W_H such that $\lambda_{W_H}(e) = 2$ for every $e \in E(H)$, and $\lambda_{W_H}(e) = 1$ for every $e \in E(G) - E(H)$, but W_H may not be a PT. However, 1 and 2 give rise to a decomposition of $E(H)$ into k paths $P_1, \ldots, P_k$ such that every odd vertex is an endvertex of precisely one such path, where $\{v_1, \ldots, v_{2k}\} = V_{odd}$ is the set of odd vertices of G. So if P_i connects $v_{j_{2i-1}}$ and $v_{j_{2i}}$ ($V_{odd} = \{v_{j_1}, v_{j_2}, \ldots, v_{j_{2k}}\}$), then these k paths partition V_{odd} into classes of size 2 : $\Pi(V_{odd}) = \{ \{v_{j_{2i-1}}, v_{j_{2i}}\} : 1 \leq i \leq k \}$. Now, Dijkstra's algorithm (adapted to graphs; see above) yields in polynomial time a path $P(v_{j_{2i-1}}, v_{j_{2i}})$ such that its length $c^*_{j_{2i-1}, j_{2i}}$ is minimum. Setting $c^*_\Pi = \sum_{i=1}^{k} c^*_{j_{2i-1}, j_{2i}}$ we can formulate the next result.

Theorem 86 *Given a connected graph G with associated cost function $c : E(G) \to \mathbb{R}^+$ finding a PT is equivalent to finding a partition $\Pi(V_{odd})$ such that c^*_Π is minimum.*

Observe that once such $\Pi(V_{odd})$ with minimum c^*_Π has been found, the corresponding k paths will be automatically pairwise edge-disjoint,

and creating an eulerian supergraph G^+ by doubling the edges of these k paths, any eulerian trail in G^+ corresponds to a CCW W in G such that W is a PT in G. Moreover, we then have $c(W) = c(G) + c^*_\Pi$, where c^*_Π is minimum.

Create an auxiliary graph K_{2k} whose vertices are the odd vertices of G and define a cost function $c^* : E(K_{2k}) \to \mathbb{R}^+$ by

$$c^*(v_iv_j) = c^*_{i,j} = \min_{P(v_i,v_j)\subseteq G} c(P(v_i, v_j))$$

Dijkstra's algorithm yields not only the number $c^*_{i,j}$ but also a corresponding path in polynomial time. It is clear that every $\Pi(V_{odd})$ corresponds to a perfect matching in (K_{2k}, c^*), and vice versa. Thus the following is true.

Theorem 87 *Solving the CPP in (G, c) is equivalent to finding a minimum cost perfect matching in (K_{2k}, c^*) where* $c^*_{i,j} := c^*(v_i, v_j) = \min_{P(v_i,v_j)\subseteq G} c(P(v_i, v_j))$.

Thus the cost of a PT W in G is $c(W) = c(G) + c^*(M)$ where M is a minimum cost perfect matching in (K_{2k}, c^*).

Remark. Suppose (G, c) has a perfect matching. Then finding a minimum cost perfect matching can be transformed to the problem of finding a maximum cost perfect matching M in G with cost function $c_1 : E(G) \to \mathbb{R}^+$, where $c_1(e) := K - c(e)$, where K is some real number greater than $\max_e c(e)$. Then $c_1(M) = \frac{p}{2}K - \sum_{e\in M} c(e) = \frac{p}{2}K - c(M)$ where $p = |V(G)|$. Thus $c_1(M)$ is maximum if and only if $c(M)$ is minimum. Thus a polynomial time algorithm for one problem will lead to a polynomial time algorithm for the other, since the transition from c to c_1 described above is linear in $|E(G)|$.

Algorithm 88
Step 0: *Given (G, c), $c : E(G) \to \mathbb{R}^+$, set $V_0(G) = V_{odd} = \{v_1, \ldots, v_{2k}\}$. If $V_0(G) = \emptyset$ then G is eulerian and any eulerian trail of G is a PT.*
Step 1. *Determine values $c^*_{i,j} := c^*(v_i, v_j) := \min_{P(v_i,v_j)\subseteq G} c(P(v_i, v_j))$ for $1 \le i, j \le 2k$, $i \ne j$.*
(Note that $c^* : E(K_{2k}) \to \mathbb{R}^+$ can be determined in polynomial time).
Step 2. *Find a minimum cost perfect matching M in (K_{2k}, c^*) where $V(K_{2k}) = V_0(G)$.*
(Note that there are polynomial time algorithms for finding such minimum cost perfect matchings).
Step 3. *For every $v_iv_j \in M$ consider the path $P(v_i, v_j)$ constructed in the course of determining $c^*_{i,j}$ (so $c(P(v_i, v_j))$ is minimum) and double in G the edges of $P(v_i, v_j)$. Denote the resulting (eulerian) graph by G^+.*

Step 4. *Find an eulerian trail T in G^+ (again, this can be done in polynomial time) and translate it into a CCW W in G.*
Step 5. *W is a PT of G.*

Since this algorithm does not repeat any step and since each step can be done in polynomial time, therefore the algorithm itself terminates in polynomial time.

The preceding considerations also lead to the following.

Theorem 89 *Given a connected graph G with cost function $c : E(G) \to \mathbb{R}^+$, the CPP can be formulated as an Integer Linear Programming Problem as follows:*

$$\begin{aligned} &\textit{Determine} && x_e \in \{0,1\}, e \in E(G) \\ &\textit{such that} && \sum_{e \in E_v} c(e)x_e \textit{ is minimum} \\ &\textit{subject to} && \sum_{e \in E_v} (1 + x_e) \equiv 0 \mod 2 \textit{ for every } v \in V(G). \end{aligned}$$

Given (G, c) *the Maximum Cost Cycle Packing Problem (MCPP)*, asks for a set S of edge-disjoint cycles such that $c(S) := \sum_{C_i \in S} c(C_i)$ is maximum.

Theorem 90 *The MCPP is equivalent to CPP in the sense that any solution of the CPP yields a solution of the MCPP. In particular, if W is a PT for a given connected graph G and $G_W := \langle\{ e \in E(G) : \lambda_W(e) = 1 \}\rangle$ then G_W is eulerian and any cycle decomposition S_W of G_W is a solution of the MCPP. Conversely, if S_0 is a solution of the MCPP then G has a PT W_0 such that $\lambda_{W_0}(e) = 1$ if $e \in E(S_0)$ and $\lambda_{W_0}(e) = 2$ otherwise.*

Proof. Let W be a PT. Set $H_W := \langle\{ e \in E(G) : \lambda_W(e) = 2 \}\rangle$. Note $V_{odd}(H_W) = V_{odd}(G)$ so $G_W := G - H_W$ is eulerian and has a cycle decomposition S_W. So $c(G) = c(G_W) + c(H_W) = c(S_W) + c(H_W)$.

On the other hand if H is an arbitrary subgraph with $V_{odd}(H) = V_{odd}(G)$ then $G_H := G - H$ is eulerian and thus has a cycle decomposition S_{G_H} and $c(G) = c(S_{G_H}) + c(H)$. Since H_W corresponds to a PT, $V_{odd}(H_W) = V_{odd}(G) = V_{odd}(H)$, hence $c(H_W) \leq c(H)$. Note that H gives rise to a CCW $W(H)$ in G such that

$$\lambda_{W(H)}(e) = \begin{cases} 1 & if\ e \notin E(H) \\ 2 & if\ e \in E(H) \end{cases}$$

Observing further that this correspondence between $H \subseteq G$ with $V_{odd}(H) = V_{odd}(G)$ and eulerian subgraphs $G_H := G - H \subseteq G$ is a

bijection, we conclude that $c(S_W) = \max c(S)$ where the maximum is taken over all sets S of edge-disjoint cycles.

Conversely, consider a solution S_0 of the MCPP,

$$H_0 := G - \bigcup_{C_i \in S_0} E(C_i).$$

Also, consider $G_H := G - H$ for H with $V_{odd}(H) = V_{odd}(G)$, so G_H is eulerian. Let S_{G_H} be a cycle decomposition of G_H. Since $c(S_{G_{H_0}}) = c(S_0)$ is maximum over all S_{G_H} as defined above, and $V_{odd}(H) = V_{odd}(G) = V_{odd}(H_0)$, this implies that there is a CCW W_0 in G such that $c(W_0) = c(G) + c(H_0)$ and $c(H_0)$ is minimum. Thus W_0 is a minimum cost CCW of G and hence a PT. ■

Corollary 91 *The MCPP can be solved in polynomial time.*

The Minimum Cost Cycle Covering Problem (MCCP) can be stated as follows. Given a bridgeless graph G, $E(G) \neq \emptyset$, $c : E(G) \to \mathbb{R}^+$, find a cycle cover S such that $c(S) = \sum_{C_i \in S} c(C_i)$ is minimum. Let S be any cycle cover of G. Define $\lambda_S(e)$ to be the number of elements of S containing e, for any edge e of G. Set $k(e) = \lambda_S(e) - 1$ $\forall e \in E(G)$ and add $k(e)$ parallel edges to G. Call the graph thus obtained $G^+(S)$; it is eulerian (S corresponds to a cycle decomposition S^+ of $G^+(S)$). $G^+(S)$ is connected if and only if G is connected. An eulerian trail T^+ in $G^+(S)$ corresponds to a CCW W^+ in G such that $c(W^+) = c(S)$. Solving the CPP for (G, c), let W be a PT in G. Thus $c(W) \leq c(W^+)$, implying that $c(W) \leq c(S)$.

Theorem 92 *If G is a connected bridgeless graph with cost function $c : E(G) \to \mathbb{R}^+$ then any PT W and any solution S of the MCCP satisfy $c(W) \leq c(S)$.*

Note that if G^+ is an eulerian supergraph of G resulting from doubling certain edges of G the cycle decomposition S^+ of G^+ may not correspond to a cycle cover of G. If $C = C(e, e') = x, e, y, e', x \in S^+$ for $e' \in E(G^+) - E(G)$, then C does not correspond to a cycle of G. However, if G^+ has a cycle decomposition S^* such that for every e with $\lambda_W(e) = 2$ (where W is a PT corresponding to G^+) the elements of S^* containing e, e' correspond to cycles in G, then S^* corresponds to a cycle cover S in G such that $c(S) = c(W)$.

However, for the Petersen Graph, with $c \equiv 1$, any PT W satisfies $c(W) = 20$, whereas any solution S of the MCCP satisfies $c(S) = 21$. This fact follows from the nonexistence of an X_5-compatible cycle decomposition in K_5 (see Figure 2.6 and the discussion of it). Thus, starting

with the Petersen graph one can construct an infinite set of 3-regular graphs for which the inequality in Theorem 92 is strict. In the case of planar graphs however, the situation is quite different. Call a transition system X of an eulerian graph G with $\delta(G) \geq 4$ *non-separating* if no $t \in X$ is an edge cut.

Theorem 93 ([12]) *Given a planar eulerian graph G with $\delta(G) \geq 4$ and a nonseparating system X of transitions, there is an X-compatible cycle decomposition of G.*

In fact, Theorem 93 can be proved quite easily by applying the Four Colour Theorem. However, the proof in [12] does not rely on the latter. Consequently, Theorem 93 is a nontrivial necessary condition for this famous result. – However, Theorem 93 is key in proving the next result (see [18]).

Theorem 94 *Let G be a planar bridgeless graph, $c : E(G) \to \mathbb{R}^+$ a cost function. If W is a PT and S is a MCC then $c(W) = c(S)$.*

Note that Theorem 94 does *not* say that finding a MCC S in G is equivalent to finding a PT in G. But finding an X-compatible cycle decomposition in a planar eulerian graph G can be done in polynomial time using the Four Colour Theorem. Thus finding an MCC for (G, c), where G is planar and bridgeless, can be done in polynomial time by first solving the CPP in (G, c). – For generalizations of Theorem 93, see [43].

Just as the CPP has been phrased for graphs, one might pose the analogous problem for digraphs:

Given a digraph D and a cost function $c : A(D) \to \mathbb{R}^+$, find a CCW in D of minimum cost.

This problem is called the *Directed Postman Problem (DPP).*

It is easy enough to see that for given (D, c), the DPP has a solution if and only if D is strongly connected. We note in passing that *directed postman tours (DPTs)* can be characterized in a way similar to Theorem 85, [20]. In the case of a DPT W in (D, c), however, an arc of D may very well be traversed more than twice by W. This is also expressed in the following ILP corresponding to DPP w.r.t. (D, c). Namely:

$$\begin{aligned} &\text{determine} && x_a \in N \cup \{0\},\ a \in A(D), \\ &\text{such that} && \sum_{a \in A(D)} c(a) x_a \text{ is minimum} \\ &\text{subject to} && \sum_{a \in A_v^+} x_a - \sum_{a \in A_v^-} x_a = d^-(v) - d^+(v) \end{aligned}$$

Nonetheless, the basic idea of creating an eulerian superdigraph D^+ of D by adding parallel arcs such that the additional costs are minimum, is analogous to the case of graphs. The idea of transforming the DPP to a matching problem is also analogous to the case of graphs: now, however, one is faced with finding a minimum cost perfect matching M in a complete bipartite graph whose order depends on the numbers $d^+(v)-d^-(v), v \in V(D)$. This problem is a typical case of an Assignment Problem. Once such M has been found, the transformation of D into the eulerian superdigraph D^+ with the help of the paths corresponding to elements of M, and reinterpreting an eulerian trail of D^+ as a DPT in D, is analogous to the case of graphs. The paths however, may no longer be arc-disjoint since arcs can be traversed only in one direction. This is also the reason why in rephrasing Theorem 92 for strongly connected digraphs, a solution W of the DPP and a solution S of the MCCP satisfy $c(W) = c(S)$.

As for the *Mixed Postman Problem (MPP)*, one is given $H = V(H) \cup E(H) \cup A(H)$, $c : E(H) \cup A(H) \rightarrow \mathbb{R}^+$, and has to determine a CCW W in H such that $c(W) = \sum_{f \in E \cup A} \lambda_W(f) c(f)$ is minimum.

To determine the existence of a solution of MPP for H one first determines if H has a CCW. This can be done by constructing a digraph D_H from H by replacing each $e \in E(H)$ by two oppositely oriented arcs joining the same vertices as e and then checking if D_H is strongly connected. In fact, H has a CCW if and only if D_H is strongly connected.

We refrain from discussing the DPP and MPP more detailed since they will be treated in other chapters. We note in passing, however, that the DPP, like the CPP, can be solved in polynomial time, whereas the MPP is, in general, an $\mathcal{NP}$-complete problem; this is also true for the Windy Postman Problem in which every edge of a graph is assigned two costs, depending on the direction in which the edge is traversed by a CCW.

We finish this section by 'touching' on the *Hamiltonian Walk Problem (HWP)* and the *Traveling Salesman Problem (TSP)*. The former can be phrased as follows.

Given a connected graph G and cost function $c : E(G) \rightarrow \mathbb{R}^+$, find a closed vertex-covering walk W such that $c(W)$ is minimum.

If W_{HWP} is a solution of the HWP and W_{CPP} a solution of the CPP for (G, c) then $c(W_{CPP}) \geq c(W_{HWP})$ because every CCW is a closed

vertex-covering walk. However, in general the HWP is $\mathcal{NP}$-complete.

In fact even if (G, c) has a hamiltonian cycle C, this by no means implies that *some* hamiltonian cycle will constitute a solution of the HWP. However, one can transform the HWP to a minimum cost hamiltonian cycle problem. To this end, let $V(G) = \{v_1, \ldots, v_p\} = V(K_p)$ and determine for every $i \neq j$ a path $P_{i,j}$ connecting v_i and v_j such that $c(P_{i,j}) =: c_{i,j}$ is minimum. Define $c^* : E(K_p) \to \mathbb{R}^+$ by setting $c^*(v_i, v_j) = c_{i,j}$.

Theorem 95 *Solving the HWP for (G, c) (G being connected) is equivalent to finding a hamiltonian cycle C in (K_p, c^*) such that $c^*(C)$ is minimum ($c := E(G) \to \mathbb{R}^+$; for extending c to c^*, see above).*

Finding such hamiltonian cycle C in K_p with minimum $c^*(C)$ is, however, also an $\mathcal{NP}$-complete problem.

This equivalent formulation of the HWP is known as the *Traveling Salesman Problem (TSP)* for the special case where the cost function satisfies the triangle inequality, i.e., $c_{i,j} + c_{j,k} \geq c_{i,k}$. Of course, the TSP can be phrased in K_p for arbitrary cost function c^*. If c^* does not satisfy the triangle inequality, then HWP and TSP might have different solutions in (K_p, c^*) indeed. This is exemplified by K_3 and the cost function c^* which assigns the value 1 to two edges of K_3 and the value 3 to the third edge. So while the TSP for this (K_p, c^*) has a unique solution C with $c^*(C) = 5$, the HWP also has a unique solution W for which however $c^*(W) = 4$ holds.

For a general formulation of TSP as an ILP, see [33, p. 308–309].

However, in some instances the following upper bound may be even better than the one above involving a PT; it can be computed faster, anyway.

Theorem 96 *If (G, c) is connected and if W is a hamiltonian walk then $c(W) \leq 2c(T_0)$ where T_0 is a minimum cost spanning tree of G.*

8. ELEMENTS OF NETWORK THEORY

A flow in a digraph D (see below) can be understood in a way similar to the construction of directed postman tours (see above) – provided the flow values are positive integers – in that one duplicates arcs in accordance with their flow values such that the resulting digraph is eulerian. However, the subsequent discussion will not resort to such an interpretation (although certain aspects of this discussion can be better understood if one keeps this interpretation in mind).

In this section, we are concerned practically exclusively with digraphs. We follow, basically, the notation of [4]. We also refer the interested reader to the more recent book [8].

Let $D = V \cup A$ be a digraph. A simple closed chain C in D is called a *circuit*; in other words, C is a circuit if C corresponds to a cycle in the underlying graph G_D. In the case of arc cuts, the terminology is also analogous: Let $V_0 \subseteq V$, and denote the coboundary $\omega_{V_0} := A(V_0, \bar{V_0})$; call ω_{V_0} *elementary* or *simple* or a *cocircuit* if both $\langle V_0 \rangle$ and $\langle V - V_0 \rangle$ are weakly connected. However, just as we dealt with different types of connectedness, we distinguish between different types of coboundaries in the case of digraphs. Denote

$$\omega_{V_0} = \omega^+_{V_0} \dot{\cup} \omega^-_{V_0}$$

where $\omega^+_{V_0}$ consists precisely of those arcs in ω_{V_0} whose tail lies in V_0, whereas $\omega^-_{V_0}$ consists precisely of those arcs of ω_{V_0} whose head lies in V_0. If, for a simple coboundary ω_{V_0}, either $\omega_{V_0^+} = \emptyset$ or $\omega_{V_0^-} = \emptyset$, then we call ω_{V_0} a cocycle. Note that $A_v = A^+_v \cup A^-_v$ is a coboundary of D for every $v \in V$.

A mapping $f : A \to \mathbb{R}$ is called a *flow* if for every $v \in V$

$$\sum_{a \in A^+_v} f(a) = \sum_{a \in A^-_v} f(a) \tag{1}$$

Also, we call a mapping $t : V \to \mathbb{R}$ a *potential* with which we associate a *potential difference* or *tension* $g : A \to \mathbb{R}^+$ by setting for every $a = (x, y) \in A$,

$$g(a) = t(y) - t(x). \tag{2}$$

Conversely, a mapping $g : A \to \mathbb{R}$ is called a tension in D if there exists a potential $t : V \to \mathbb{R}$ such that g is associated with t.

In fact, if D is weakly connected and g a tension in D, then for every potential t and every $c \in \mathbb{R}$, if g is associated with t, then it is also associated with $t + c$; and if two potentials t_1, t_2 give rise to the same tension, then $t_2 = t_1 + k$ for some constant $k \in \mathbb{R}$.

In fact, for given D, the set of flows (tensions) in D forms a vector space called *flow space (tension space)*. These two vector spaces are better described as subspaces of $\mathbb{R}^q$ by first labeling the arcs of D: $A = \{a_1, \ldots, a_q\}$. Then we associate with every flow f the vector $(f(a_1), \ldots, f(a_q))$ and with every tension g the vector $(g(a_1), \ldots, g(a_q))$, calling them flow vector and tension vector, respectively. In fact, we will not distinguish between these mappings and the corresponding vectors

since they are bijectively related to each other, once the arc labeling is fixed.

We observe that circuits and flows, coboundaries and tensions are closely related. Let C be a circuit in D, $\omega_0 := \omega_{V_0}$ a cocircuit for some $V_0 \subseteq V$. Assign a sense of traversal to C and set $A(C) = A^+(C) \cup A^-(C)$, where $A^+(C)$ contains those arcs of C whose orientation coincides with the sense of traversal of C, and $A^-(C)$ contains the remaining arcs of C. Set

$$\mathbf{v_c} = (v_c^{(1)}, \ldots, v_c^{(q)})$$

$$\text{such that } v_c^{(i)} = \begin{cases} 1 & \text{if } a_i \in A^+(C) \\ -1 & \text{if } a_i \in A^-(C) \\ 0, & \text{otherwise.} \end{cases}$$

Likewise, for $V_0 \subseteq V$ set

$$t(x) = \begin{cases} 1 & \text{if } x \in V - V_0 \\ 0 & \text{if } x \in V_0 \end{cases}$$

and – as a logical consequence – define for

$$\mathbf{v_0} = (v_{0,1}, \ldots, v_{0,q})$$

$$v_{0,i} = \begin{cases} 1 & \text{if } a_i \in \omega_0^+ \\ -1 & \text{if } a_i \in \omega_0^- \\ 0, & \text{otherwise} \end{cases}$$

We call $\mathbf{v_c}$ the *circuit vector* associated with C and $\mathbf{v_0}$ the *coboundary vector* associated with ω_0. The following is an immediate consequence from the above definitions.

Lemma 97 *Every circuit vector is a flow (vector), every coboundary vector is a tension (vector), and they are orthogonal.*

To determine bases for the flow space, tension space respectively, of a weakly connected digraph $D = V \cup A, A = \{a_1, \ldots, a_q\}$, consider a spanning tree $T \subseteq D$, w.l.o.g. $A(T) = \{a_1, \ldots, a_{p-1}\}$.

For every $a_i, i \in \{p, \ldots, q\}$, there is a unique circuit C_i in $A(T) \cup \{a_i\}$ and C_i contains a_i; define the sense of traversal of C_i in accordance with the orientation of a_i, and let $\mathbf{v_i}$ be the corresponding circuit vector.

Similarly, for every $a_j \in A(T), T - a_j$ has precisely two weakly connected components T_j', T_j''. W.l.o.g. T_j' contains the tail of a_j; set $V_j := V(T_j')$, and let $\omega_j = A(V_j, \overline{V_j})$; $a_j \in \omega_j^+$ follows. Let $\mathbf{w}_j$ be the corresponding cocircuit vector. Observing that C_i (ω_j) contains none of

the arcs in $\{a_p, \ldots, a_q\} - \{a_i\}$ ($\{a_1, \ldots, a_{p-1}\} - \{a_j\}$) we conclude that $\mathbf{v}_p, \ldots, \mathbf{v}_q$ are linearly independent, and so are $\mathbf{w}_1, \ldots, \mathbf{w}_{p-1}$. Moreover, one can show that every flow can be expressed as a linear combination of $\mathbf{v}_p, \ldots, \mathbf{v}_q$, and every tension can be expressed as a linear combination of $\mathbf{w}_1, \ldots, \mathbf{w}_{p-1}$. Therefore the following is true.

Theorem 98 *Let $D = V \cup A$ be a weakly connected digraph, $A = \{a_1, \ldots, a_q\}$ and let $\mathbf{v}_i, \mathbf{w}_j, p \leq i \leq q, 1 \leq j \leq p-1$, be defined as above. For the flow space $V_{\mathcal{F}}$ and the tension space $V_{\mathcal{T}}$ of D, the following is true.*

1 *$\mathcal{B}_{\mathcal{F}} := \{\mathbf{v}_i : p \leq i \leq q\}$ is a basis of $V_{\mathcal{F}}$; hence $\dim V_{\mathcal{F}} = q - p + 1$.*

2 *$\mathcal{B}_{\mathcal{T}} := \{\mathbf{w}_j : 1 \leq j \leq p-1\}$ is a basis of $V_{\mathcal{T}}$; hence $\dim V_{\mathcal{T}} = p - 1$.*

3 *$V_{\mathcal{F}}$ and $V_{\mathcal{T}}$ are orthogonal complements in $\mathbb{R}^q$.*

However, in practical applications one needs to construct flows or tensions satisfying certain constraints (apart from optimizing a certain objective function). Consequently, let $k_j, l_j \in \mathbb{R}, 1 \leq j \leq q, k_j \leq l_j$,be chosen, and associate the closed interval $I_j := [k_j, l_j]$ with arc a_j.

Theorem 99 *Let $D = V \cup A$ be a digraph, $A = \{a_1, \ldots, a_q\}$, with Interval $I_j = \{k_j, l_j\}$ associated with $a_j \in A, 1 \leq j \leq q$. The following is true.*

1 *A tension $(g_1, \ldots, g_q)$ satisfying $k_j \leq g_j \leq l_j$, $1 \leq j \leq q$, exists if and only if for every circuit C of D*

$$\sum_{a_i \in A^-(C)} l_i \geq \sum_{a_i \in A^+(C)} k_i \,, \quad \sum_{a_i \in A^+(C)} l_i \geq \sum_{a_i \in A^-(C)} k_i.$$

2 (A.J. Hoffman) *A flow $(f_1, \ldots, f_q)$ satisfying $k_j \leq f_j \leq l_j$, $1 \leq j \leq q$, exists if and only if for every cocircuit $\omega = \omega^+ \cup \omega^-$*

$$\sum_{a_i \in \omega^-} l_i \geq \sum_{a_i \in \omega^+} k_i \,, \quad \sum_{a_i \in \omega^+} l_i \geq \sum_{a_i \in \omega^-} k_i.$$

In many practical problems however, these constraints can be relaxed. In this context, the following problems are of central importance.

The Maximum Tension Problem. Let $D = V \cup A, A = \{a_1, \ldots, a_q\}$, be given; distinguish two vertices $\bar{v}, \bar{w} \in V$ with $(\bar{w}, \bar{v}) \in A$, w.l.o.g. $a_1 = (\bar{w}, \bar{v})$. For $j = 2, \ldots, q$, let $I_j := [k_j, l_j]$ be associated with a_j. Find a tension $(g_1, g_2, \ldots, g_q)$ such that

$$k_j \leq g_j \leq l_j, j = 2, \ldots, q,$$

g_1 is maximum.

In fact, this problem can be transformed into the *Generalized Shortest Path Problem.* Here, we have $D = V \cup A$ with distinguished vertices $\bar{v}, \bar{w}$ given such that every other vertex of D lies on some path connecting $\bar{v}$ and $\bar{w}$. With every $a \in A$ we associate a real number $l(a)$ (which need not be nonnegative). Finally we assume that D contains no cycle with $l(C) < 0$.

Algorithm 100
Input: $D = V \cup A, V = \{v_1, v_2, \ldots, v_n\}, \bar{v} = v_1, \bar{w} = v_n$, *with properties as above.*
Step 0. *Find a spanning out-tree T of D with root $\bar{v}$ (it exists since every vertex lies on a path from $\bar{v}$ to $\bar{w}$). For every $v_i, 1, \ldots, n$, define the potential t by $t_i := t(v_i) = l(P_T(\bar{v}, v_i))$ where $P_T(\bar{v}, v_i)$ is the unique path connecting $\bar{v}$ and v_i in T.*
Step 1. *If $t(v_j) - t(v_i) \leq l(v_i, v_j)$ for every $(v_i, v_j) \in A$ go to Step 3. Otherwise choose $(v_i, v_j) \in A$ such that $t(v_j) - t(v_i) > l(v_i, v_j)$.*
Step 2. *Set $t'(v_j) = t(v_i) + l(v_i, v_j)$, and for every v_k in T which lies on a $P_T(v_j, v_k)$, set $t'(v_k) = t(v_k) - (t(v_j) - t'(v_j))$. Define $A' := (A(T) - (A(T) \cap A^-_{v_j})) \cup ((v_i, v_j))$. Set $t := t', T = \langle A' \rangle$ and go to Step 1.*
Step 3. *$t(\bar{w})$ is the length of a shortest path from $\bar{v}$ to $\bar{w}$ in D. Such path is exhibited by going from $\bar{v}$ to $\bar{w}$ in the final T.*

We refrain from elaborating on the justification and efficiency of Algorithm 100; we just observe that it is the absence of cycles C with $l(C) < 0$ which guarantees that the algorithm terminates after a finite number of steps; and that it is, therefore, of polynomial running time.

Following [4] we define a *capacitated network* or just *network* as a digraph $D = V \cup A, A = \{a_1, \ldots, a_q\}$, having two distinguished vertices $\bar{v}, \bar{w}$ such that w.l.o.g. $(\bar{w}, \bar{v}) = a_1$ and $\bar{v}$ ($\bar{w}$) is the source (sink) of $D - \{a_1\}$; thus $\bar{v}$ is called the *entry* and $\bar{w}$ is called the *exit* of the network, and $a \in A^+_{\bar{v}}$ is called an *entry arc*, whereas $a \in A^-_{\bar{w}}$ is called an *exit arc* of the network. We also assume that every other vertex lies on a path from $\bar{v}$ to $\bar{w}$ in D. Finally, associate with every $a_j \neq a_1$ a *capacity* $c_j \geq 0, j = 2, \ldots, q$ (i.e., the capacities correspond to the intervals $[0, c_j]$).

Maximum Flow Problem. Find a flow $\mathbf{f} = (f_1, f_2, \ldots, f_q)$ in a network (as described above) such that $0 \leq f_j \leq c_j, j = 2, \ldots, q$, and f_1 is maximum (f_1 is called the *(flow) value* of $\mathbf{f}$).

For the following considerations we consider a simple chain $\bar{P}$ connecting $\bar{v}$ and $\bar{w}$ in a network D, such that $a_1 \notin A(\bar{P})$, with an orientation of $\bar{P}$ being defined by walking from $\bar{v}$ to $\bar{w}$ in the underlying path P_0

in G_D. Correspondingly, we define $A^+(\bar{P})$ as the set of arcs of $\bar{P}$ whose orientation coincides with that of $\bar{P}$, while $A^-(\bar{P}) := A(\bar{P}) - A^+(\bar{P})$. Suppose a *feasible flow* $\mathbf{f} = (f_1, \ldots, f_q)$ is given in D, i.e., $0 \leq f_i \leq c_i$ for $i = 2, \ldots, q$. Call $\bar{P}$ **f**-*augmenting* if $f_i < c_i$ for every $a_i \in A^+(\bar{P})$ and $f_j > 0$ for every $a_j \in A^-(\bar{P})$. Define $d_{\mathbf{f}} := \min\{c_i - f_i, f_j : a_i \in A^+(\bar{P}), a_j \in A^-(\bar{P})\}$.

Lemma 101 *Let D be a network with feasible flow* **f**. *If an* **f**-augmenting *simple chain $\bar{P}$ exists, define $\mathbf{f}^+ = (f_1^+, \ldots, f_q^+)$ by $f_1^+ = f_1 + d_{\mathbf{f}}$*

$$f_j^+ = \begin{cases} f_j + d_{\mathbf{f}} & \text{for} \quad a_j \in A^+(\bar{P}) \\ f_j - d_{\mathbf{f}} & \text{for} \quad a_j \in A^-(\bar{P}) \\ f_j & \text{for} \quad a_j \notin A(\bar{P}). \end{cases} \quad j = 2, \ldots, q$$

Then $\mathbf{f}^+$ is a feasible flow of larger value than **f** *(we say* **f** *has been* augmented *to yield $\mathbf{f}^+$).*

Proof. The lemma follows from the definition of $\mathbf{f}^+$ (the sums of equation (1) in the definition of a flow remain unaltered, or are both increased or decreased by $d_{\mathbf{f}}$), and the definition of **f** itself ($d_{\mathbf{f}} > 0$ since $\bar{P}$ is **f**-augmenting). ■

The existence of an **f**-augmenting simple chain is key to the following considerations.

Algorithm 102 *(Ford-Fulkerson Max-Flow Algorithm for integral capacitated networks)*
Input: *Network D with capacities $c_j \in \mathbb{N} \cup \{0\}, j = 2, \ldots, q$, together with an integral flow $\mathbf{f}_0$ (i.e., $f_i \in \mathbb{N} \cup \{0\}, i = 1, \ldots, q$) - e.g., $\mathbf{f}_0 = \mathbf{0}$. Set $\mathbf{f} = \mathbf{f}_0$. Set $g(v) = |V| + 1$ for every $v \in V$ (these are the unmarked vertices).*
Step 1. *Mark $\bar{v}$, set $g(\bar{v}) = 1$. Call the vertices with $g(v) < |V| + 1$* marked vertices.
Step 2. *Choose v with minimum $g(v)$ such that there is an unmarked w with either $a_i = (v, w) \in A$ and $f_i < c_i$ or $a_i = (w, v) \in A$ and $f_i > 0$, if such v exists. Set $g(w) = \max\{g(v) + 1 : g(v) < |V| + 1\}$. If no such v exists, go to Step 4.*
Step 3. *If $\bar{w}$ has not been marked go to Step 2. If $\bar{w}$ has been marked, find a simple chain $\bar{P}$ from $\bar{v}$ to $\bar{w}$ in $D_0 \subseteq D$ induced by the arcs used in the marking procedure in Step 2 ($\bar{P}$ is* **f**-*augmenting). Augment* **f** *to obtain $\mathbf{f}^+$ in accordance with Lemma 101. Unmark all marked vertices $v \neq \bar{v}$. Set $\mathbf{f} = \mathbf{f}^+$ and go to Step 2.*
Step 4. **f** *is a maximum flow and $\bar{w}$ is unmarked.*

Before justifying this algorithm and drawing further conclusions, some remarks seem in order.

(1) Apart from the requirement that the capacities be nonnegative integers, there are two crucial items in the Ford-Fulkerson algorithm: the choice of the integral flow $\mathbf{f}_0$ in *Input*, and the construction of the $\mathbf{f}$-augmenting simple chain $\bar{P}$. There are examples (see below) showing that these choices cannot be arbitrary if one expects the algorithm to reach *Step 4* in polynomial time.

(2) Of course, it is natural to start with $\mathbf{f}_0 = \mathbf{0}$. However, if one starts with a flow whose components are not integers, one might be able to produce an infinite sequence of flows $\mathbf{f}_i, i = 0, \ldots, n \ldots$, by using $\mathbf{f}_i$-augmenting chains, whose values form a strictly increasing and convergent sequence, yet the limit of this sequence need not be the optimal flow! Such an example has been exhibited by Lovász and Plummer in [31, p. 47–48]. Such problem cannot arise if the initial flow is integral, since $\mathbf{f}^+$ is then also integral by definition of $d_{\mathbf{f}}$ (see Lemma 101). Thus we obtain a strictly increasing sequence of integral flow values, bounded above by the largest (integral) capacity.

(3) If the graph is small, but $\min\{c_j : j = 2, \ldots, q\} = 1$ and $M := \max\{c_j : j = 2, \ldots, q\}$ is a large integer, then, even starting from $\mathbf{f}_0 = \mathbf{0}$, say, a repeated 'bad choice' of the $\mathbf{f}$-augmenting chain $\bar{P}$ may result in a number of iterations of Step 3 which is proportional in M, whereas the size of the input is $O(\log M)$; i.e., the running time of the algorithm may be exponential in the input (see [22, p. 127]).

(4) Extending the algorithm to networks with rational capacities does not create any extra difficulties: just multiply the capacities with their least common denominator n to obtain integral capacities and multiply the maximum flow in this modified capacitated network with $1/n$ to obtain the maximum flow in the original network.

(5) If, however, the capacities are arbitrary nonnegative real numbers, then, starting from $\mathbf{f}_0 = \mathbf{0}$ one may get an infinite sequence of feasible flows (using $\mathbf{f}$-augmenting simple chains again) whose values are strictly increasing and converge. Yet, the limit of this sequence of flow values again is not the value of a maximum flow. Such an example has been exhibited already by Ford-Fulkerson in their book, [21, p. 21–22]; a simpler example can be derived from the one quoted in (2) above.

(6) However, all these difficulties and problems quoted in *(2) – (5)* can be overcome by a simple idea due independently to Edmonds and Karp on the one hand, and Dinits (see [22, 31] for detailed references) on the other hand. Namely, choose $\bar{P}$ in Step 3 in such a way that it corresponds to a shortest path in the underlying graph G_{D_0}. And such

a shortest path can be found 'fast' by Dantzig's and Dijkstra's Algorithm 83. Once Step 3 of Algorithm 102 has been modified this way, the modified algorithm terminates with a maximum flow in polynomial time, even if the capacities are arbitrary real numbers.

For the justification of Algorithm 102 some additional notation will be useful. Consider in a capacitated network $D = V \cup A$ a set $\bar{V} \subset V$ such that $\bar{v} \in \bar{V}$ and $\bar{w} \notin \bar{V}$. Then we call $A^+_{\bar{V}} := A^+(\bar{V}, V - \bar{V})$ a *cut of the network* D, and $\bar{c} := \sum_{a_i \in A^+_{\bar{V}}} c_i$ the *capacity of the cut* $A^+_{\bar{V}}$.

Now, in justifying Algorithm 102, we basically prove the converse of Lemma 101; that is, if no $\mathbf{f}$-augmenting path exists, then there is no feasible flow whose value is larger than that of $\mathbf{f}$. To this end, observe first the following general fact: if $D = V(D) \cup A(D), A(D) = \{a_1, \ldots, a_q\}$, is an arbitrary digraph and $\omega = \omega^+ \cup \omega^-$ is an arbitrary coboundary of D, then for any flow $\mathbf{f} = (f_1, \ldots, f_q)$ in D we have the equation

$$\sum_{a_i \in \omega^+} f_i = \sum_{a_i \in \omega^-} f_i$$

(this equation readily follows from equation (1) in the definition of a flow). Applying this equation to a capacitated network $D = V \cup A$ we obtain for an arbitrary cut $A^+_{\bar{V}}$ with capacity $\bar{c}$ and a feasible flow $\bar{\mathbf{g}} = (g_1, \ldots, g_q)$ the inequality

$$g_1 \leq \sum_{a_i \in A^-_{\bar{V}}} g_i = \sum_{a_i \in A^+_{\bar{V}}} g_i \leq \bar{c} \qquad (*)$$

(note that for every cut $A^+_{\bar{V}}, a_1 \in A^-_{\bar{V}}$ since $\bar{w} \notin \bar{V}$).

Whence let $\mathbf{f}$ be the final flow constructed by Algorithm 102: $\mathbf{f}$ is obtained in a finite number of steps due to Lemma 101 ($d_{\mathbf{f}} \in \mathbb{N}$), and because the capacities are positive integers. For this $\mathbf{f}$ we mark $\bar{v}$, but the marking procedure of Step 2 leaves $\bar{w}$ unmarked (otherwise Step 3 yields an $\mathbf{f}$-augmenting chain $\bar{P}$ which would allow to augment $\mathbf{f}$). Let $\bar{V} :=$ {marked vertices}. We observe that for every $a_i \in A^+_{\bar{V}}$ we have $f_i = c_i$ (otherwise, the head of a_i would be marked in Step 2), and for every $a_i \in A^-_{\bar{V}} - \{a_1\}$ we have $f_i = 0$ (otherwise, the tail of a_i would be marked in Step 2). Therefore, equality holds in $(*)$ for this $\mathbf{f}$ and this $\bar{V}$, also implying that $A^+_{\bar{V}}$ must be a cut of minimum capacity. This and $(*)$ imply that no flow of larger value than f_1 can exist in this network.

As a consequence of the preceding argument we also obtain the following.

Theorem 103 (Max-Flow Min-Cut Theorem) *If D is a capacitated network with entry $\bar{v}$ and exit $\bar{w}$, then the maximum value of any feasible flow equals the minimum capacity of any cut of D.*

With the help of Algorithm 102 and Theorem 103 one can solve several problems in applied network theory but also derive graph theoretical results proved above. As for the former type of results, we refer to [4, 8, 21, 26, 36]. As for deriving Menger's Theorem and Proposition 24 (the 'line version' of Menger's Theorem) and corresponding versions for digraphs, we refer to [31] (as for Menger's Theorem, this has been done already in [21]). As for deriving Theorem 79 from Theorem 103 we refer to [31, 36].

References

[1] Andersen, L.D, and Fleischner, H.; The $\mathcal{NP}$-completeness of finding A-trails in Eulerian graphs and of finding spanning trees in hypergraphs, Discrete Appl. Math 59 (1995), 203-214.

[2] Appel, K., Haken, W., Koch, J.; Every planar map is four-colorable. Illinois J. Math. 21 (1977) 429–567.

[3] Berge, C.; Two theorems in graph theory, Proc. Natl. Acad. Sci. U.S., 43(1957), 842–844.

[4] Berge, C., and Ghouila-Houri, A.; Programmes, jeux et réseaux de transport. Dunod, Paris 1962 (German translation by Teubner Verlagsges. 1967)

[5] Bermond, J.-C.; Hamiltonian Graphs, in: Selected Topics in Graph Theory (L.W. Beineke and R.J. Wilson, eds.). Academic Press, N.Y. 1978.

[6] Bondy, J.A., and Chvátal, V.; A method in graph theory, Discrete Math. 15 (1976), 111-135.

[7] Bondy, J.A.; Basic Graph Theory: Paths and Circuits, in Handbook of Combinatorics Vol. I (R.L. Graham, M. Grötschel, L. Lovász, eds.). North Holland, Amsterdam 1995.

[8] Carre, B.; Graphs and Networks, Oxford Appl. Math. and Comp. Sc. Ser., Oxford University Press 1979.

[9] Dirac, G.A.; Short proof of Menger's graph theorem, Mathematika 13 (1966), 42-44.

[10] Fleischner, H.; On Spanning Subgraphs of a Connected Bridgeless Graph and Their Application to DT-Graphs, JCT 16, 1 (1974), 17-28.

[11] Fleischner, H.; The Square of Every Two-Connected Graph is Hamiltonian, JCT 16, 1 (1974), 29-34.

[12] Fleischner, H.; Eulersche Linien und Kreisüberdeckungen, die vorgegebene Durchgänge in den Kanten vermeiden, JCT **B** 29 (1980), 145-167.

[13] Fleischner, H.; Cycle Decompositions, 2-Coverings, Removable Cycles And The Four-Color-Disease, in: Progress in Graph Theory (J.A. Bondy and U.S.R. Murty, eds.), Academic Press, 1984, 233-246.

[14] Fleischner, H.; Some Blood, Sweat, but no Tears in Eulerian Graph Theory, 250th Anniversary Conference on Graph Theory (Fort Wayne, IN, 1986). Congr. Numer. **63**(1988), 8–48.

[15] Fleischner, H.; The Prism of a 2-connected, Planar, Cubic Graph is Hamiltonian (a Proof Independent of the Four Colour Theorem), in: Graph Theory in Memory of G.A. Dirac (L.D. Andersen et al., eds.), Ann. Discrete Math. 41, North-Holland, Amsterdam 1989, 141-170.

[16] Fleischner, H.; Eulerian Graphs and Related Topics, Part 1, Vols. 1 and 2. Ann. Discrete Math. 45 and 50, North Holland, Amsterdam (1990 and 1991).

[17] Fleischner, H.; (Some of) The Many Uses of Eulerian Graphs in Graph Theory (Plus Some Applications), to appear in: Proceedings of the Paul Catlin Memorial Conference (A. Hobbs ed.), Discrete Math.

[18] Fleischner, H., and Guan, M.-G.; On the Minimum Weighted Cycle Covering Problem for Planar Graphs, Ars Combin. 20 (1985), 61-67.

[19] Fleischner, H., and Jackson, B.; A Note Concerning some Conjectures on Cyclically 4-Edge Connected 3-Regular Graphs, in: Graph Theory in Memory of G. A. Dirac (L.D. Andersen et al.,eds.), Ann. Discrete Math. 41, North-Holland, Amsterdam 1989, 171-178.

[20] Fleischner, H., and Wenger, E.; Characterizing Directed Postman Tours, in: Topics in Combinatorics and Graph Theory (R. Bodendiek, R. Henn, eds.), Physica-Verlag, Heidelberg 1990, 257-262.

[21] Ford Jr., L.R., and Fulkerson, D.R.; Flows in Networks, Princeton University Press, Princeton, N.J. 1962.

[22] Frank, A.; Connectivity and network flows, in: Handbook of Combinatorics Vol. I (R.L. Graham, M. Grötschel, L. Lovász, eds.). North Holland, Amsterdam 1995.

[23] Gao, Z.C., and Richter, R.B.; 2-walks in circuit graphs, JCT **B** 62(1994),2, 259-267.

[24] Garey, M.R., and Johnson, D.S.; Computers and Intractability: A Guide to the Theory of NP-Completeness. Freeman, N. Y. 1979.

[25] Graham, R.L., and Hell, P.; On the history of the minimum spanning tree problem, Ann. Hist. Comput. 7(1985),1, 43-57.

[26] Grötschel, M., Lovász, L., Schrijver, A.; Geometric Algorithms and Combinatorial Optimization; Second Corrected Edition, Springer-Verlag, Berlin 1993.

[27] Harary, F.; Graph Theory. Addison-Wesley Publ., Reading MA 1969.

[28] Jaeger, F.; A note on sub-eulerian graphs, J. Graph Theory **3**, (1979) 91-93.

[29] Jaeger, F.; A Survey of the Cycle Double Cover Conjecture, in: Cycles in Graphs (B. Alspach et al., eds.), Ann. Discrete Math. 27, North-Holland, Amsterdam (1985) 1-12.

[30] König, D.; Theory of Finite and Infinite Graphs (translation of the original version in German). Birkhäuser, Boston 1990.

[31] Lovász, L., and Plummer, M.D.; Matching Theory, Ann. Discrete Math. 29, North-Holland, Amsterdam, 1986.

[32] Matthews, M., and Sumner, D.; Hamiltonian results in $K_{1,3}$-free graphs, J. Graph Theory 8, (1984), 139-146.

[33] Papadimitriou, C.H., and Steiglitz, K.; Combinatorial Optimization, Prentice Hall, New Jersey 1982.

[34] Riha, S.; A new proof of the theorem by Fleischner, JCT **B** 52 (1991),1, 117-123.

[35] Roberts, F.S.; Graph Theory and Its Applications to Problems of Society. CBMS-NSF Regional Conference Series in Applied Mathematics, Vol. 29, 1978.

[36] Roberts, F.S.; Applied Combinatorics, Prentice-Hall, New Jersey 1984.

[37] Robertson, N., Sanders, D., Seymour, P.D., Thomas, R.; The four-colour theorem. JCT **B** 70 (1997),1, 2–44.

[38] Ryjacek, Z.; On a closure concept in claw-free graphs, JCT **B** 70 (1997), 217-224.

[39] Sekanina, M.; On an ordering of the set of vertices of a connected graph. Publ. Fac. Sci. Univ. Brno **412**(1960), 137-141.

[40] Thomassen, C.; Reflections on graph theory, J. Graph Theory 10, (1986), 309-324.

[41] Tutte, W.T.; A Theorem on Planar Graphs, Trans. Amer. Math. Soc. 82 (1956) 99-116.

[42] Volkmann, L.; Graphen und Digraphen, Springer, Wien 1991.

[43] Zhang, C-Q.; Integer Flows and Cycle Covers of Graphs. Marcel Dekker, Inc., New York 1997.

Chapter 3

MATCHING: ARC ROUTING AND THE SOLUTION CONNECTION

Ulrich Derigs
University of Cologne, Germany

1. Introduction 89
2. Matching: Applications 91
   2.1 Team Selection 91
   2.2 Task Scheduling 92
   2.3 Processor Scheduling 93
   2.4 Route Connection 94
   2.5 Arc Routing 95
   2.6 Node Routing 98
   2.7 General Routing 101
   2.8 Set Partitioning 104
3. Matching: Combinatorial Aspects 107
4. Matching: Polyhedral Aspects 112
5. Matching Algorithms: Linking Combinatorial and Polyhedral Results 116
6. Matching Algorithms: Implementation Issues 119
   6.1 Start Procedures: Constructing the Initial Extreme Matching 120
   6.2 Organization of Dual Updates 122
   6.3 Price and Reoptimize 123

1. INTRODUCTION

Given an undirected graph $G = (V, E)$, a *matching* $M \subseteq E$ is a subset of edges no two of which are incident with a common vertex. For any $M \subseteq E$, we define $V(M)$ as the set of vertices incident to some edge in M. A matching M in G is called a *maximum cardinality matching* in G if $|M| \geq |M'|$ for all matchings M' in G. A perfect matching is a matching M with $V(M) = V$. Note that for the existence of perfect matchings $|V|$ has to be an even number.

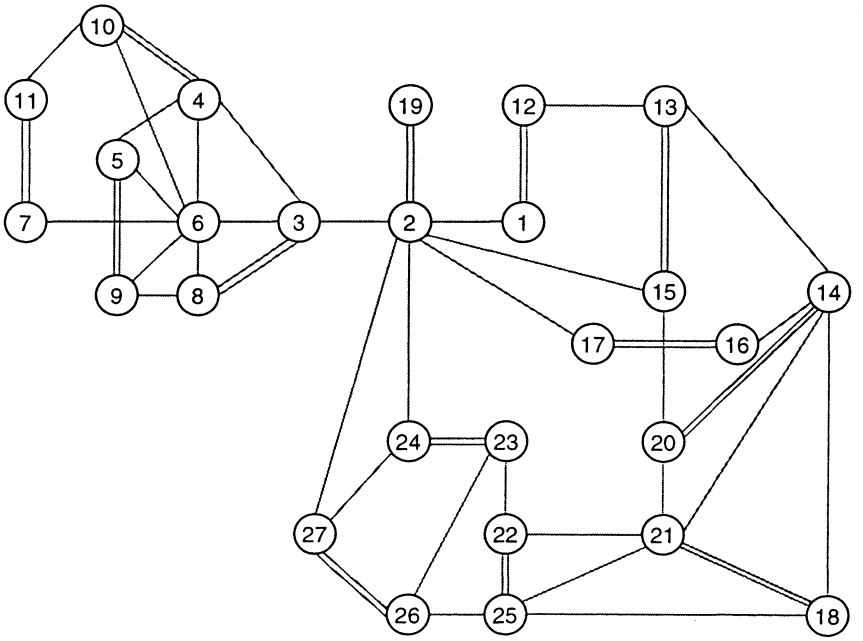


Figure 3.1 Graph with a non-perfect maximum cardinality matching (matching edges are doubly lined).

Given a cost c_{ij} for each $(i,j) \in E$, we define $c(F) := \sum_{(i,j)\in F} c_{ij}$ for every $F \subseteq E$. Then a *minimum cost perfect matching* is a perfect matching M that minimizes $c(M)$. Here, we assume w.l.o.g. $c_{ij} \geq 0$ since this can always be obtained by adding a large constant to all costs without changing the problem.

The *minimum cost perfect matching problem* (MP) can be formulated as a mathematical program:

$$\min \sum_{(i,j)\in E} c_{ij} \cdot x_{ij} \qquad s.t. \tag{3.1}$$

$$\sum_{j:(i,j)\in E} x_{ij} = 1 \qquad \text{for } i \in V \tag{3.2}$$

$$x_{ij} \in \{0,1\} \quad \text{for } (i,j) \in E, \tag{3.3}$$

where we interpret

$$x_{ij} = \begin{cases} 1, & \text{if } (i,j) \in M \\ 0, & \text{if } (i,j) \notin M. \end{cases}$$

It is well known that for bipartite graphs G the integrality conditions can be relaxed to

$$x_{ij} \geq 0 \qquad \text{for } (i,j) \in E \tag{3.4}$$

and MP becomes a linear program the optimal solution of which will be $(0,1)$-valued automatically.

Matching theory and the matching problem are one of the cornerstone concepts in combinatorial optimization/ mathematical programming. They constitute a class of genuine integer programs which are efficiently solvable. Since the matching model arises in various domains, the ability to solve large matching instances is of practical interest.

In this chapter we first motivate and show the relevance of matchings especially for arc routing by introducing several well-known applications. Then we discuss the graph theoretical background and the polyhedral results which are the foundations for matching algorithms. Finally, we review algorithmic techniques which have been developed within the last years and which enable the solution of large matching instances in reasonable time.

Sources on matching are the books by Lovasz and Plummer [1986], Lawler [1976], Derigs [1988a] and the survey article by Gerards [1995].

We close this introduction with some notation. For $W \subseteq V$, we define $\delta(W) := \{(i,j) \in E \mid i \in W, j \notin W\}$ and $\gamma(W) = \{(i,j) \in E \mid i,j \in W\}$, where we write $\delta(i)$ instead of $\delta(\{i\})$.

2. MATCHING: APPLICATIONS

In this section we outline applications of the bipartite and the non-bipartite matching model in various domains. The problem of finding an optimal matching in a bipartite graph is one of the most celebrated problems taught in the first course in Operations Research. The interpretation of the matching model as pair wise associations of entities from different classes like male and female persons, jobs and machines etc. is obvious and has led to the more common terminology as *marriage problem* or *assignment problem.*

2.1. TEAM SELECTION

A clear and illustrative example for the assignment problem was given by Machol [1961]. A swimming coach must select a medley relay team from his four best swimmers to swim the four strokes (back, breast, butterfly, and free-style). The (average) time of each swimmer in each stroke

is known from former tests.

Representing each swimmer and each stroke as a node and connecting each swimmer-node with each stroke-node by an edge the costs of which is the associated time the swimmer needs for performing that stroke leads to a complete 4×4 bipartite graph G. Now every perfect matching in G gives one possible team and the problem of finding the fastest team is to find the perfect matching with the least costs.

2.2. TASK SCHEDULING

A "more practical" application of bipartite matching arises in scheduling where the problem of minimizing the number of processors or the total processing time to perform a set of tasks can be modeled as an assignment problem.

Let $V = \{v_1, ..., v_n\}$ be a set of tasks. The ordered pair (v_i, v_j) is said to be *compatible* if the same processor can perform tasks v_i and v_j in this sequence. Now a subset $D = \{v_{i_1}, ..., v_{i_k}\}$ of V is said to be a processor *duty* if $(v_{i_j}, v_{i_{j+1}})$ is a compatible pair of tasks, $j = 1, ..., k-1$, i.e. the tasks in D can be performed by the same processor. Then a feasible *schedule* is a family $S = \{D_1, ..., D_r\}$ of processor duties, such that each task $v \in V$ belongs to exactly one duty $D_j, j = 1, ..., r$.

In the domain of "vehicle scheduling" the tasks v_j may represent a sequence of customers which have to be served by a vehicle, i.e. a vehicle-tour, but where because of capacity limitations or time constraints the vehicle has to visit a depot for a certain duration before beginning and after ending the tour. Let

f_i, the duration of the dead heading trip from the depot to the first customer in tour v_i,

l_i, the duration of the dead heading trip from the last customer in tour v_i to the depot,

d_i, the duration of tour i,

s_i, the required start time for tour i and,

d_p, the duration at the depot between two tours,

then the ordered pair (v_i, v_j) of tours is compatible if there exists a slack $t_{ij} \geq 0$ such that

$$s_i + d_i + l_i + d_p + f_j + t_{i,j} = s_j.$$

t_{ij} gives the so-called idle time of a vehicle spent at the depot between the performance of tour v_i and v_j.

Now consider the bipartite graph $G = (S, T, E)$ where S and T are the node sets with $|S| = |T| = |V|$. Every node $i \in S$ corresponds to the ending terminal of task v_i and every node $j \in T$ corresponds to the starting terminal of task v_j and $E := \{(s_i, t_j) | (v_i, v_j)$ are compatible tasks$\}$. Note that $(s_i, t_i) \in E$ for all $v_i \in V$.

Now there is a one-to-one correspondence between (vehicle) schedules and matchings in G. Any (vehicle) duty, say $D_i = \{v_{i_1}, ..., v_{i_k}\}$ can be represented in G as the set of edges

$$M_i = \left\{(s_{i_1}, t_{i_2}), (s_{i_2}, t_{i_3}), ..., (s_{i_{k-1}}, t_{i_k})\right\}.$$

Since for each $v_{i_j} \in D_i$ at most one edge in M_i is incident to node s_{i_j} and t_{i_j} and since each trip belongs to only one of the vehicle duties $D_1, ..., D_r$ the set

$$M := M_1 \cup ... \cup M_r$$

is a matching in G.

Similarly any matching M of G can be interpreted as a (vehicle) schedule with the following interpretation (cf. Carraresi and Gallo [1984]): If no matching edge is incident either to s_{i_j} or to t_{i_j}, then task v_j is performed as a single duty for its own, i.e. a vehicle runs the single tour v_j.

Introducing edge costs the construction of optimal schedules can be reduced to finding minimum cost perfect matchings in the complete graph over the node sets S and T. Setting

$$c_{i,j} = \begin{cases} 0 & \text{for } (s_i, t_j) \in E, i \neq j \\ 1 & \text{else} \end{cases}$$

each least cost perfect matching induces a schedule with a minimum number of processors (vehicles) necessary, i.e. the optimal matching determines the optimal fleet size.

Minimizing the idle time is accomplished by setting

$$c_{i,j} = \begin{cases} t_{i,j} & \text{for } (s_i, t_j) \in E, i \neq j \\ 0 & \text{else.} \end{cases}$$

2.3. PROCESSOR SCHEDULING

An early application of non-bipartite cardinality matching arises again in the area of processor scheduling. Assume two identical processors and a set of n jobs, all requiring the same processing time with a partial ordering relation " $\leq$ " prescribing precedence constraints for the jobs,

i.e. if $i \leq j$ then job i must be completed before job j can be started by either processor.

The precedence constraints can be represented as an acyclic directed graph $G = (N, A)$ with N representing the set of jobs and $(i, j) \in A$ if $i \leq j$ holds. For G we can construct the so-called compatibility graph $G^* = (N, E)$. G^* has the same nodes as G and there is an (undirected) edge (i, j) if and only if there is no directed path from i to j or from j to i in G, i.e. if nodes i and j are adjacent in G^* then job i and job j can be processed at the same time (by the two processors) given.

Now a maximum cardinality matching M in G^* indicates the maximum number of jobs that can be processed simultaneously, and thus, yields a lower bound on the total processing time and can be used to obtain an optimal schedule (cf. Fujii et al. [1969]).

2.4. ROUTE CONNECTION

Hasselström [1976] gives an early application of non-bipartite weighted matching in connection with the process of urban transportation planning by modeling the "optimization of route connections" as a matching problem.

The route network in a city usually contains several points of intersection where several routes meet. The routes at such a point can be categorized into two classes

1 routes passing through and

2 routes with one of their terminal points at this point of intersection.

If the routes that are passing through are cut into two parts at the point of intersection, the resulting network consists of route-legs having one end in common which would be served by vehicles in both directions.

It is now possible to reconnect these route-legs in several different ways, i.e. every matching of end points leads to a different route network. The quality of the resulting network is dependent on several different factors, the passenger's waiting time, number of transfers, number of vehicles etc., and the representation of these factors by edge costs is a nontrivial task. The whole procedure is illustrated in Figure 3.2.

An important application of non-bipartite matching is its use in solving certain routing problems as for instance the Chinese postman problem and the traveling salesman problem.

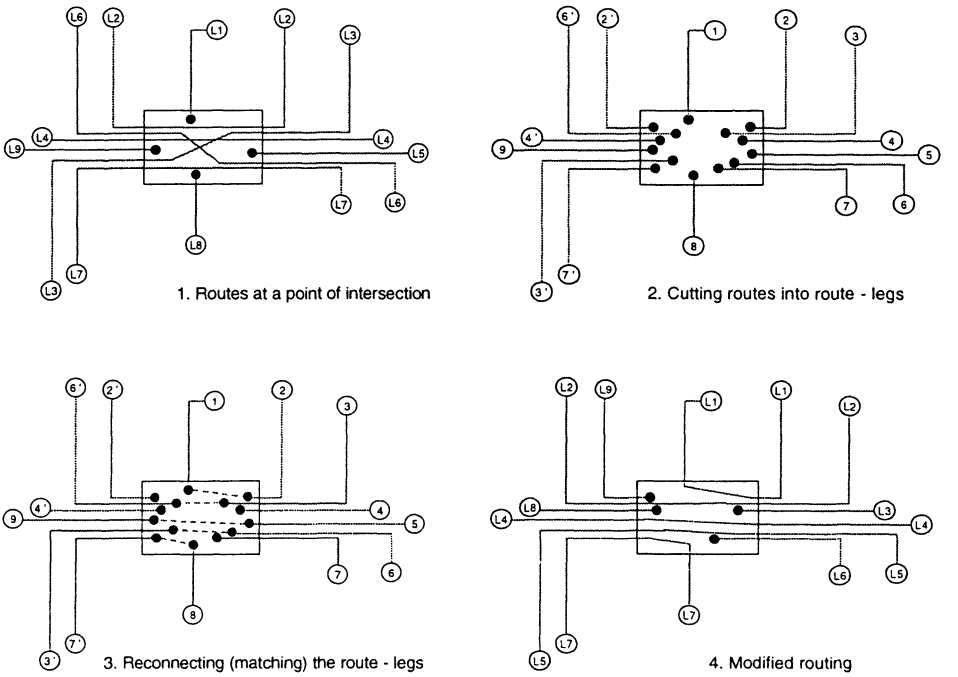


Figure 3.2 Optimization of route connections.

2.5. ARC ROUTING

Let a connected graph $G = (V, E)$ and non-negative weights c_j for $e_j \in E$ be given. Then a tour in G is a sequence $T = (v_1, e_1, v_2, e_2, ..., v_k, e_k, v_{k+1} = v_1)$ of nodes v_i and edges e_i such that e_i meets the nodes v_i and v_{i+1}. A *postman tour* in G is a tour which contains every edge at least once and an *Eulerian tour* of G is a tour which contains every edge exactly once. The length of a tour $T = (v_1, e_1, v_2, ..., e_k, v_1)$ is defined by $l(T) = \sum_{i=1}^{k} c_k$.

Now the *Chinese postman problem* (CPP) is to find the minimum length postman tour in G. This problem was first solved by Mei Ko Kwan [1962]. He considered this problem on the practical background of a postman delivering the daily mail for a certain district of streets. Thus, the problem is referred to as "Chinese" postman problem. The first polynomial algorithm was given by Edmonds and Johnson [1973] who related this problem to matching theory.

It is obvious that whenever there is an Eulerian tour in the graph, then it solves CPP. A necessary and sufficient condition for the existence of such a tour was given by Euler [1736]:

"each node must be incident to an even number of edges".

Thus, if this condition is fulfilled the CPP reduces to the problem of finding an Eulerian tour.

Given a postman tour every edge of G is traversed at least once. So let $1+t_j$ be the number of times that edge e_j is contained in the tour. Now we construct a (multi-)graph G' from G introducing t_j additional copies of edge e_j into G. Then the postman tour in G becomes an Eulerian tour in G'. Thus CPP can be reformulated in the following way:

Chinese postman problem (CPP)

$$\min \sum_{e_j \in E} t_j \cdot c_j$$

$$\sum_{e_j \in \delta(v_i)} (t_j + 1) = 0 \mod 2 \quad \text{for } v_i \in V$$

$$t_j \geq 0, \text{ integer} \quad \text{for } e_j \in E,$$

i.e. find values t_j for $e_j \in E$ s.t. after adding t_j copies of e_j every node has even degree and $\sum_{e_j \in E} t_j \cdot c_j$ is minimized.

Now if node v has odd degree in G then an odd number of incident edges has to be added such that v gets even degree in G'. If node v is an even degree node in G, then an even number of incident edges resp. no edge has to be added.

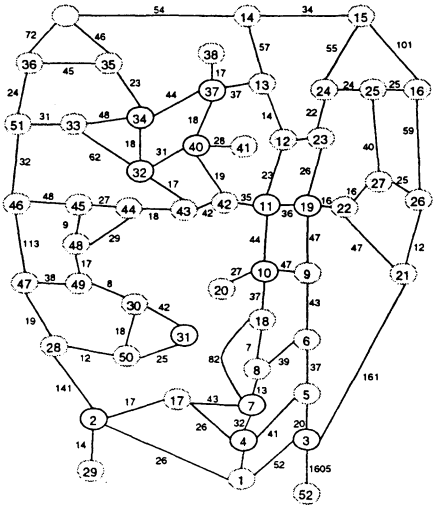
Therefore the process of duplicating edges leads to a collection of paths starting and ending at odd degree vertices the edges of which have to be duplicated. Thus, one has to decide which pairs of odd degree nodes (there is always an even number of nodes with odd degree) are to be joined together by a path of duplicated edges.

This problem can be solved in the following way:

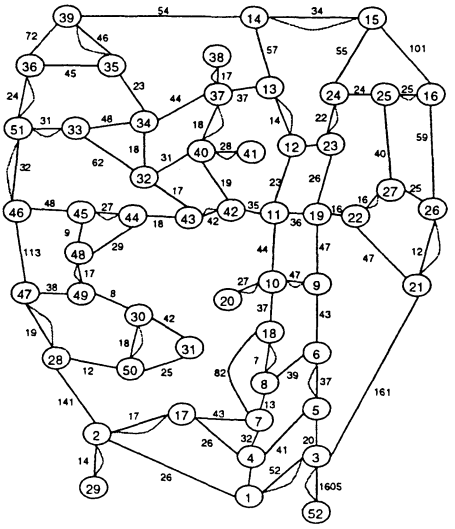
Chinese postman algorithm

1 Determine for every pair $v_i, v_j \in V$ of odd degree nodes the shortest path P_{ij} joining these two nodes and define d_{ij} to be the length of path P_{ij}.

2 Construct the complete graph $\overline{G} = (\overline{V}, \overline{E})$ where $\overline{V}$ denotes the set of all odd degree nodes in G'. Associate with every edge (i,j)



Street-network with nodes of odd-degree shaded
total cost of edges: 6452



Multi-graph with duplicated edges from optimal matching
cost of optimal matching 2248
cost of optimal postman tour 6700

Figure 3.3 Solving the Chinese postman problem.

joining $v_i, v_j \in \overline{E}$ the edge weight d_{ij} and solve the associated minimum cost perfect matching problem.

3 The edges (i, j) of the optimal matching M correspond to the paths P_{ij} the edges of which have to be duplicated to obtain the optimal postman tour.

Figure 3.3 shows the procedure and result for a refuse collection problem in a rural area modeled as CPP on a graph with 52 nodes.

This algorithm was developed by Edmonds and Johnson [1973] who describe a labeling technique which enables us to solve the shortest path and the matching problem simultaneously. Consequently, this labeling technique combines elements from shortest path computation and elements from matching algorithms. The problem to be solved is then to construct an Eulerian tour in G'. This is done by using the so called next-node-algorithm presented in Edmonds and Johnson [1973]. A FORTRAN-code for solving the Chinese postman problem can be found in Burkard and Derigs [1980].

An interesting application or extension of the above concept is that of minimizing the "up"-motion of a pen plotter as described by Reingold

and Tarjan [1981] and Iri et al. [1983].

Consider the problem of plotting a graph $G = (V, E)$ where the nodes are given by their (x, y) coordinates. If the graph contains an Eulerian tour (or path) then it can be drawn without any wasted pen movement. Otherwise the pen must be moved in the "up" position resulting in wasted pen movement. The graph can be made Eulerian by matching the edges of odd degree, i.e. introducing an edge which, when drawing the graph, i.e. drawing the postman tour (or path), has to be traversed with the pen lifted up in the "up"-position. Note that the significant difference between the classical Chinese postman problem and this variant is the fact that for the plotter-problem the graph G can be made Eulerian by using connections not present in the graph and thus the calculation of shortest paths between odd nodes is trivial and the problem reduces to a pure matching problem.

2.6. NODE ROUTING

A *traveling salesman tour* is a tour that meets every node exactly once. Now assume $G = (V, E)$ a complete graph and $c : E \rightarrow \mathbb{R}_+$ a cost function fulfilling the triangle condition, i.e. $c_{ij} + c_{jk} \geq c_{ik}$ for all $i, j, k \in V$.

We define $\mathcal{T}$:= the set of all traveling salesman tours in G, then the *traveling salesman problem* (TSP) can be formulated:

$$z_{OPT} = \min_{T \in \mathcal{T}} c(T).$$

Assume $|V|$ an even number , then every traveling salesman tour T can be uniquely partitioned into two perfect matchings M_1 and M_2 in G by introducing the edges in T alternately into M_1 and M_2 respectively and the following relation holds

$$c(T) = c(M_1) + c(M_2) \geq 2 \cdot min\{c(M_1), c(M_2)\}.$$

With $\mathcal{M}$:= the set of all perfect matchings in G the following property holds

$$\min_{M \in \mathcal{M}} c(M) \leq \frac{1}{2} \min_{T \in \mathcal{T}} c(T).$$

A *spanning tree* S in G is a subset of the edges such that

(i) $|S| = |V| - 1$

(ii) S does not contain a cycle (subtour) in G.

Obviously, given a traveling salesman tour $T \in \mathcal{T}$, deleting an arbitrary edge from T gives a spanning tree S. Thus, with $\mathcal{S} :=$ the set of all spanning trees in G the following property holds

$$\min_{S \in \mathcal{S}} c(S) \leq \min_{T \in \mathcal{T}} c(T).$$

For determining *minimal spanning trees* in a graph rather efficient algorithms have been developed (Dijkstra [1959], Kruskal [1956]).

For the minimal spanning tree $S' \in \mathcal{S}$ let $O(S')$ be the set of nodes having odd degree in S'. Note that $O(S')$ contains always an even number of nodes. Now $O(S')$ induces a complete subgraph G' of G. If we denote by $\mathcal{T}'$ and $\mathcal{M}'$ the set of traveling salesman tours and perfect matching in G' respectively, then the following relation holds:

$$\min_{M \in \mathcal{M}'} c(M) \leq \frac{1}{2} \min_{T \in \mathcal{T}'} c(T) \leq \frac{1}{2} \min_{T \in \mathcal{T}} c(T).$$

Adding the edges of a least cost perfect matching M' in G' to S' we obtain an Eulerian graph $\overline{G}$, i.e. a graph where each node has even degree. Let $\overline{P}$ be a postman tour in $\overline{G}$, then

$$c(\overline{P}) = c(S') + c(M') \leq \frac{3}{2} z_{OPT}.$$

Shortcutting all the subtours of $\overline{P}$ the postman tour can be transformed into a traveling salesman tour $\overline{T}$ in G with

$$z_{OPT} \leq c(\overline{T}) \leq c(\overline{P}) \leq \frac{3}{2} z_{OPT}.$$

Here shortcutting means that when traversing the postman tour $\overline{P}$ we would instead of traveling from a node i to a node j which has already been visited immediately travel to the successor of node j in the postman tour, node k say. Because of the triangle condition we thereby save some distance, i.e. shorten the tour. If node k has been visited already we would travel immediately to the successor of k etc.

Thus, this procedure which has been proposed by Christofides [1976] establishes a heuristic with a worst-case bound of $\frac{1}{2}$ for the relative error. With respect to this criterion the Christofides-heuristic is the best known TSP-heuristic up today.

Yet, for practical purpose a maximum error of 50% is not acceptable. It can be shown that this error is attained for rather constructed examples though (cf. Cornuejols and Nemhauser [1978]), but applying this heuristic to the well-known 120-city-problem of Grötschel [1977],

we obtain a "Christofides-tour" which is 37% longer than the optimal tour. This seems to indicate that the "pure" Christofides-heuristics is not a sufficiently valuable approach. Therefore we combined this procedure with a penalty technique and a local optimizer (cf. Derigs [1981b], [1981c]).

Instead of spanning trees we may construct *1-trees* in G. Here a 1-tree is a spanning tree plus a least cost non-tree edge incident with an arbitrary node. For 1-trees the same relations hold with respect to traveling salesman tours as it is the case for spanning trees. Obviously, the least cost 1-tree B gives a slightly better, i.e. higher lower bound than the least cost spanning tree. We define $\mathcal{B} :=$ the set of all 1-trees in G.

Now let $\pi = (\pi_1, ..., \pi_n) \in \mathbb{R}^n$ be arbitrary node-penalties and $c^\pi_{ij} := c_{ij} + \pi_i + \pi_j$ for $i, j \in V$ modified edge costs. Let B^π the minimum 1-tree with respect to c^π, then the following relations hold:

$$c^\pi(B^\pi) \leq \min_{T \in \mathcal{T}} c^\pi(T) = z_{OPT} + 2 \cdot \sum_{i=1}^{n} \pi_i \quad \text{or}$$

$$w(\pi) := c^\pi(B^\pi) - 2 \cdot \sum_{i=1}^{n} \pi_i \leq z_{OPT}$$

and thus

$$\underline{z} := \max_{\pi} w(\pi) \leq z_{OPT}.$$

It is easy to see that w is a concave, continuous and piecewise linear function and for maximizing $w(\pi)$ several ascent-methods using subgradients have been developed (cf. Held and Karp [1971]).

During the subgradient optimization the function w is evaluated for a sequence $(\pi^1, \pi^2, ..., \pi^k)$ of parameters constructing the associated optimal 1-trees $(B^{\pi_1}, B^{\pi_2}, ..., B^{\pi_k})$ with respect to the modified edge costs $(c^{\pi_1}, c^{\pi_2}, ..., c^{\pi_k})$.

Every such 1-tree B^{π_j} can now be used to construct a traveling salesman tour T_j using the Christofides-heuristic. With

$$\overline{z}^j := c(T_j) \quad \text{and}$$

$$\underline{z}_j := c^{\pi_j}(B^{\pi_j}) - 2 \cdot \sum_{i=1}^{n} \pi_i^j \,, \quad j = 1, ..., k$$

we obtain a (not necessarily monotone) sequence of lower and upper bounds.

Now let T_0 be the best traveling salesman tour constructed, i.e.

$$\overline{z}^0 := c(T_0) = \min_{j=1,\ldots,k} c(T_j) \quad \text{and}$$

$$\underline{z}_0 := \max_{j=1\ldots k} \underline{z}_j$$

the best lower bound, then T_0 is an approximate traveling salesman tour with relative error not exceeding

$$\epsilon = \frac{\overline{z}^0 - \underline{z}_0}{\underline{z}_0}.$$

Our computational experience on TSP-instances has shown that when additionally applying the well-known *3-opt local optimizer* (cf. Lin and Kernighan [1973]) to T_0, rather good approximations can be expected with the error ϵ ranging around 1% only for large problem sizes (cf. Derigs [1981c]. The results of this procedure for the 120-city-problem are depicted in Figure 3.4 and 3.5.

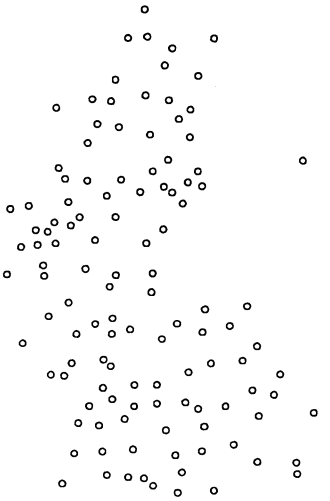
2.7. GENERAL ROUTING

The Chinese postman problem and the traveling salesman problem have been introduced as the two classical routing models. In real-life routing problems arising in waste collection, street sweeping, delivery of goods etc. variants of these pure models are applied. CPP/ TSP variants are well studied in literature, with an overview and classification given in Bodin and Golden [1981]. With regard to the application of the matching concept and matching algorithms CPP-generalization are most relevant.

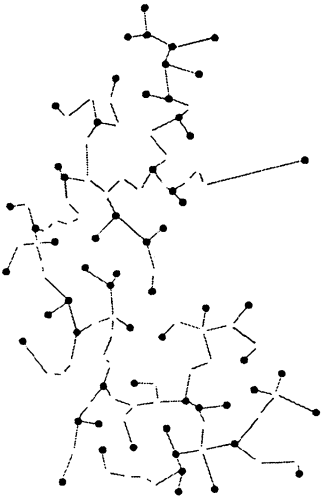
The situation that the underlying street network consists of one-way-streets only can be modeled by a directed graph. Edmonds and Johnson [1973] have given a network-flow-based algorithm for solving the associated *"directed" Chinese postman problem.* Yet, when the network contains both directed and undirected arcs, CPP becomes hard to solve (i.e. the so-called *"mixed" Chinese postman problem* is $\mathcal{NP}$-complete).

If there is a limit on the vehicle capacity or restrictions on the duration of a tour then CPP is modified to find a set of cycles (tours) which traverse every edge at least once such that traveling costs are minimized. A heuristic for this so-called *capacitated Chinese postman problem* (CCPP) which is based on the CPP approach has been described by Christofides [1973].

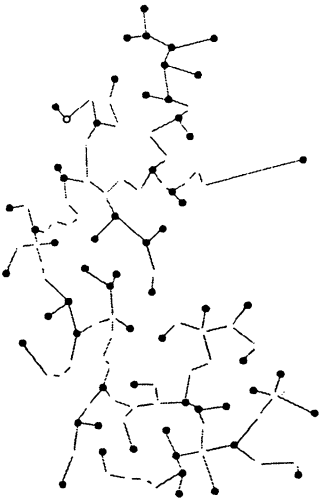
In some applications it is required that only a specified subset of street segments (edges) has to be traversed while all other edges may be traversed for dead heading. This routing problem is called the *rural postman*



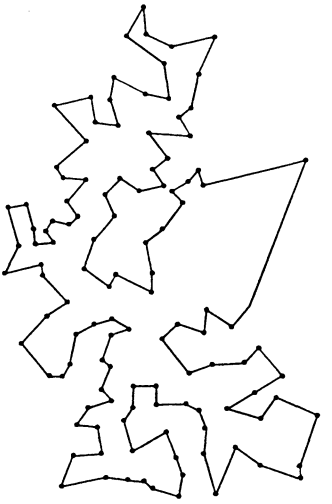
Points in the 120-City-Problem



Minimal spanning tree of length 5905 km
(nodes of odd degree are shaded)

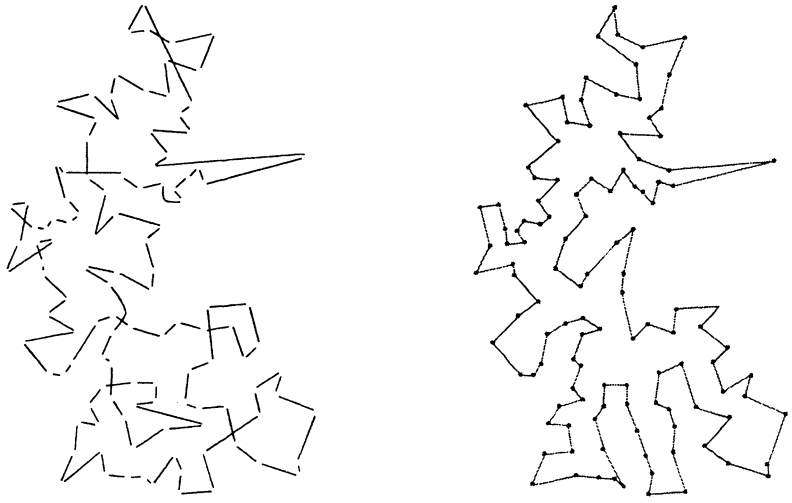


Minimal spanning tree of length 5905 km
+ Minimum perfect matching
of nodes with odd degree 2497 km
= Postman tour of length 8402 km



Optimal traveling salesman tour of length 6942 km

Figure 3.4 Application of the Christofides heuristic to the 120-city-problem (part 1).



Shortcutting the postman tour leads to a traveling salesman tour of length 7353 km

Traveling salesman tour of length 6975 km obtained through penalty technique/subgradient optimization and application of 3-OPT to best tour obtained

Figure 3.5 Application of the Christofides heuristic to the 120-city-problem (part 2).

problem (RPP) [cf. Win [1988]]. If routing is to be done on a specified subset of edges and a specified subset of nodes then the problem is called the *general routing problem* (GRP). Pandit and Muraldharan [1995] have given a heuristic procedure for solving the *capacitated general routing problem* (CGRP) on mixed graphs. In the following we give a short outline of this heuristic which is based on the concepts for solving the classical CPP, i.e. for instance matching odd degree nodes by shortest paths.

Let $G_0 = (V_0, S_0)$ be the mixed graph describing the complete street network where V_0 is the set of nodes representing the intersection of streets and locations of customers and S_0 is the set of street segments and let non-negative length c_e for $e \in S_0$ given. Let $V_1 \subseteq V_0$ be the specified set of nodes which must be visited and $p_i \geq 0$, for $i \in V_1$, the demand on node $i \in V_1$. Let $A_1 \subseteq S_0$ be the set of directed arcs representing one-way streets which have to be traversed and let $E_1 \subseteq S_0$ be the set of edges representing streets allowing bi-directional travel which have to be traversed with $q_e \geq 0$, for $e \in A_1 \cup E_1$, the demand on segment e.

Let node 1 represent a central depot and let $W \geq 0$ be the capacity of a vehicle, i.e. every tour has to start and end in node 1 and the total demand fulfilled on a single tour may not exceed W. Then the problem

is to find a set of cycles each passing through node 1 and satisfying the capacity constraint which satisfies the demands of the nodes in V_1 and the demands of the arcs and edges in $A_1 \cup E_1$. Pandit and Muraldharan [1995] propose the following heuristic:

CGRP-heuristic

1 Extract the subgraph G_1 consisting of edges, arcs and nodes that need to be traversed or visited, respectively.

2 If G_1 is not connected, connect the components of G_1, i.e. construct G_2 by introducing the edges of a minimum spanning tree between the components of G_1.

3 If G_2 is not strongly connected, make G_2 strongly connected, i.e. construct G_3 by introducing in a least cost manner additional links such that for every pair of vertices u and v there exists a path from u to v and a path from v to u.

4 If G_3 contains nodes of odd degree, connect pairs of odd degree nodes by shortest paths using the min-cost matching concept from CPP to obtain G_4.

5 Assign an orientation to the undirected arcs in G_4 such that every node has equal in- and out-degree solving a min-cost network flow problem as described in Edmonds and Johnson [1973] to obtain G_5.

6 Construct a giant Eulerian tour in G_5.

7 Break the giant Eulerian tour into smaller tours subject to the capacity constraints.

2.8. SET PARTITIONING

The final application of matching we present is a very general one: Nemhauser and Weber [1979] have shown that every set partitioning problem can be reformulated as a matching problem with additional side-constraints.

Let $A \in \{0,1\}^{m \times n}$ and $c \in I\!R^n$ then the *set partitioning problem* (SPP) reads

$$\begin{aligned} \min \quad & c'x \\ & Ax = 1 \\ & x \in \{0,1\}^n. \end{aligned}$$

This problem has a high significance from a theoretical as well as a practical point of view: quite a number of discrete optimization problems

(with linear objective function) can be reduced to SPP and quite a number of problems from routing and scheduling has been modeled as SPP.

Note that if A has the property

$$\sum_{i=1}^{m} a_{ij} = 2 \quad \text{for} \quad j = 1, ..., n$$

then A can be interpreted as the incidence matrix of a (multi-)graph G and SPP is equivalent to MP over G. This connection is the basis of the Nemhauser-Weber approach.

Let $a^1, ..., a^n$ be the columns of A, where w.l.o.g. we may assume that A does not contain identical columns, i.e. $a^i \neq a^j$ for $i \neq j$, since for identical columns only that column with minimum c-value may become part of an optimal SPP-solution.

Now assume a column a^j with p_j non-zero values. Then we associate with a^j a set of $s_j = \lfloor (p_j+1)/2 \rfloor$ column-vectors $\hat{a}^j_1, \hat{a}^j_2, ..., \hat{a}^j_{s_j}$ each having at most 2 non-zero elements and fulfilling

$$a_j = \hat{a}^j_1 + ... + \hat{a}^j_{s_j}.$$

The cost coefficients $\hat{c}_{j1}, ..., \hat{c}_{js_j}$ for the new columns are defined to fulfill the relation

$$c_j = \hat{c}_{j1} + ... + \hat{c}_{js_j}.$$

Then we can formulate a new set partitioning problem

$$(SPP') \qquad \min \quad \hat{c}' \hat{y}$$
$$\hat{A} \hat{y} = 1$$
$$\hat{y} \in \{0,1\}^s$$

$$\text{with} \quad s = \textstyle\sum_{j=1}^{n} s_j \quad \text{and} \quad \hat{A} \in \{0,1\}^{m \times s}.$$

Adding to (SPP') the so-called *column joining constraints*

$$\hat{y}^j_k = \hat{y}^j_{k+1} \text{ for } k = 1, ..., s_j - 1 \text{ and } j = 1, ..., n$$

yields a problem the feasible solutions of which are in one-to-one correspondence with the feasible solutions of (SPP) and the respective objective values coincide.

If $\hat{A}$ contains only columns having exactly two non-zero elements the transformation is complete and we have transformed SPP into an equivalent perfect matching problem with side-constraints. Otherwise let J_1 be

the set of indices for those columns containing only one non-zero entry and let

$$L_i := \{j \in J_1 \mid \hat{a}_{ij} = 1\} \text{ for } i = 1, ..., m.$$

For each $L_i \neq \emptyset$ we add a row-vector $\overline{a}_i$ with

$$\overline{a}_{ij} = \begin{cases} 1 & \text{for } j \in L_i \\ 0 & \text{else} \end{cases}$$

to our problem. Thus, we end up with the following problem

$$\begin{array}{ll} \min & \hat{c}' \hat{y} \\ & \hat{A}\, \hat{y} = 1 \\ & \overline{A}\, \hat{y} \leq 1 \\ & S\, \hat{y} = 0 \\ & \hat{y} \in \{0,1\}^s \end{array}$$

where $S\hat{y} = 0$ represents the set of column joining constraints.

The matrix $(\hat{A}|\overline{A})^\top$ fulfills the property that every column contains exactly two non-zero elements and thus induces a graph G. Let $\mathcal{M}$ be the set of matchings in G and for $M \in \mathcal{M}$ let x_M be the incidence vector associated with M. Then (SP) is equivalent to the following *matching problem with side-constraints* (MPS)

$$\begin{array}{lll} (MPS) & \min & c(M) \\ & & M \in \mathcal{M} \\ & & V(M) \supseteq \{1, ..., m\} \\ & & Sx_M = 0. \end{array}$$

In Derigs and Metz [1992a] we describe an approach for solving (MPS) based on Lagrangean relaxation and the construction of k-best matchings. In a first phase the column joining constraints are relaxed and the associated Lagrangean dual is solved using the bundle-approach, an advanced subgradient technique. Here a sequence of simple unconstrained matching problems has to be solved. In a second phase the optimal constrained matching is constructed through the generation of sequences of k-best matchings.

The method is exact in the sense that it always stops with an optimal matching fulfilling all column joining constraints and thus, induces an optimal SPP-solution. Yet, the method becomes rather inefficient and infeasible for SPP with dense matrices, i.e. matrices A having columns with many non-zero entries. In that case the procedure can be modified to obtain approximate solutions.

The problem of solving *matching problems with generalized upper bound constraints* (GUB-MP) has been investigated by Ball et al. [1990] and Derigs and Metz [1992b] . Derigs and Metz [1992c] have applied the idea of transforming (SPP) into (MPS) and the concepts for solving (GUB-MP) to a *delivery/ pickup vehicle routing problem with time windows* arising in express airline flight scheduling.

The examples introduced in this chapter should have demonstrated the great potential of the matching model, especially in the domain of routing and scheduling. Other applications in this area which we could not introduce are

- scheduling crews and vehicles in mass transit-systems (Ball, Bodin and Dial [1983])
- matching based improvement algorithms for vehicle routing (Altinkemer and Gavish [1991], Dror and Levy [1986], Desrochers and Verhoog [1991], Bachem and Malich [1993]),
- capacitated vehicle routing (Miller [1995]).

Finally, the following references should give a small insight into the use of matchings in other domains:
Sports: creating pairings in chess tournaments (Olafson [1990])
Telecommunication: image transmission (Riskin et al. [1994]), and
Statistical physics: simulation of ground state energy and magnetization of two-dimensional randomizing spin models (Bendisch, Derigs and Metz [1994]).

3. MATCHING: COMBINATORIAL ASPECTS

An *alternating path* with respect to a matching M in G is a path the edges of which are alternately in and not in M. *Alternating cycles* are defined analogously. An *augmenting path* is an alternating path between two unmatched nodes. Given an augmenting path P we can perform the following exchange operation

$$M \rightarrow M \oplus P := (M \setminus P) \cup (P \setminus M). \tag{3.5}$$

Then $M \oplus P$ is again a matching in G. For alternating cycles C we define $M \oplus C$ analogously.

The importance of augmenting paths stems from the following theorem (cf. Berge [1957]).

Theorem 3.1 *A matching M is a maximum cardinality matching if and only if M does not allow an augmenting path.*

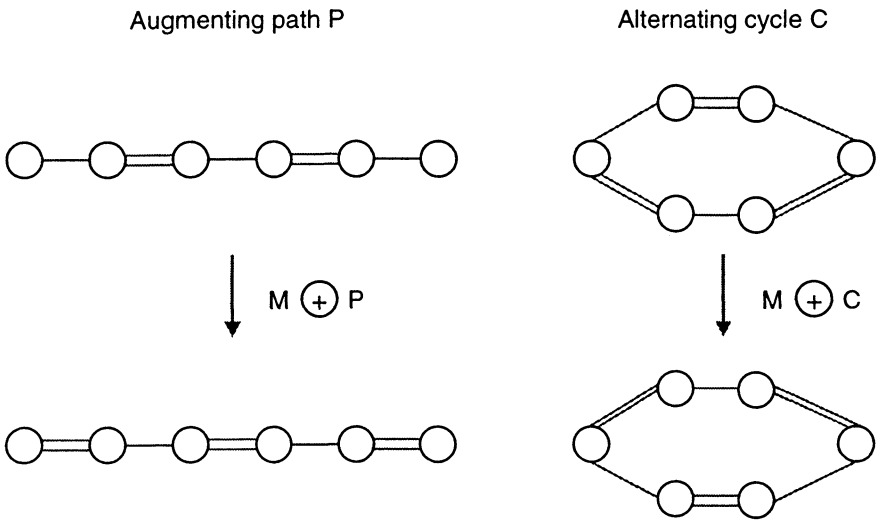


Figure 3.6 The process of augmentation.

A matching M is called *maximal* or *saturated* if any augmenting path has at least three edges, i.e. $M \cup \{e\}$ is not a matching for all $e \in E \setminus M$.

To determine augmenting paths, the concept of *alternating trees* introduced by Edmonds [1965a] has shown to be a powerful tool.

A rooted alternating tree is a tree $T = (V(T), E(T))$ with distinguished root vertex $r \in V \setminus V(M)$ and the property that the paths from r to each vertex in T are alternating paths and $E(T) \cap M$ is a perfect matching with respect to $V(T) \setminus \{r\}$. We designate vertices in a rooted alternating tree as *even* or *odd* depending on whether the number of edges in the path from r to the vertex is even or odd.

The information given by alternating trees is twofold:

- if an unmatched vertex is adjacent to an even vertex of an alternating tree, then an augmenting path can be obtained by appending the unmatched vertex to the tree;
- if all even vertices of the tree are connected to odd vertices of the tree only, then there does not exist an augmenting path starting at the root vertex r. Trees with this property are called *Hungarian trees.*

There are two basic operations for growing/ manipulating an alternating tree such that after at most $|V|$ of these operations one of the above cases will occur:

GROW-OPERATION

If a non-tree vertex $j \in V(M)$ is adjacent to an even vertex i in the alternating tree, then we can enlarge T by adding the vertex j and the vertex which is matched with j, say k, as well as the two edges (i,j) and (j,k).

SHRINK-OPERATION

If two even vertices of T are adjacent, then adding this edge to the tree will create an odd cycle $C = (V(C), E(C))$ with $|M \cap E(C)| = \frac{1}{2}(E(C) - 1)$, a so-called *blossom*. $V(C)$ can be determined by backtracking the (alternating) paths from both vertices to the root. This blossom is now *shrunk* to a *pseudonode* v_C. This is done by forming the graph $G' = (V', E')$, where

$$\begin{aligned} V' &= V \setminus V(C) \cup v_C \\ E' &= E \setminus \{(i,j) |\ i \text{ or } \ j \in V(C)\} \\ &\quad \cup \{(v_C, j) | \exists (i,j) \in E, i \in V(C), j \notin V(C)\}. \end{aligned}$$

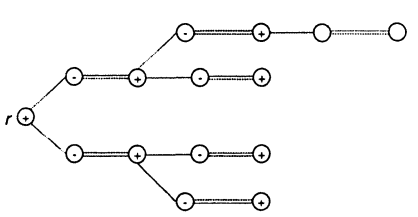
By using this definition, M uniquely defines a matching M' in G' and v_C becomes an even vertex of an alternating tree T' in G' which is induced by T.

After this SHRINK operation T' gives the same information as T and we can operate in G' further on. Given a pseudovertex v_C we may *expand* v_C by reversing the process described above. During the course of the algorithm we may perform the SHRINK operation a number of times. In particular, SHRINK may be invoked recursively in the sense that $V(C)$ may contain pseudonodes. We call the current graph $G' = (V', E')$ with (pseudo-) nodes not contained within a pseudonode the *surface graph*. Note that for bipartite graphs SHRINK can not occur and we will therefore operate on the original graph throughout the procedure.

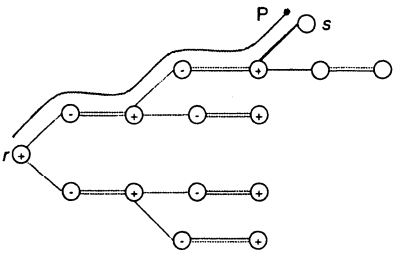
The operations on alternating trees for constructing augmenting paths are summarized in Figure 3.7.

For the bipartite and non-bipartite case efficient data structures have been developed to perform these operations and the augmenting steps. We will not discuss these problems of handling the "combinatorial structures" occurring in matching algorithms in this paper and we refer to Lawler [1976], Gabow [1976], and Gabow and Tarjan [1983].

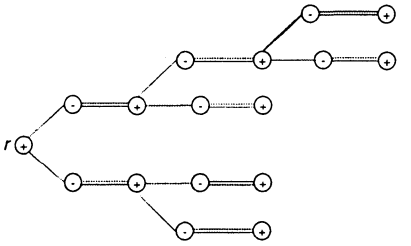
For solving weighted matching problems we are interested in augmenting paths having special properties with respect to the costs c_{ij}.



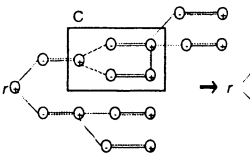
1. alternating tree T rooted at node *r*



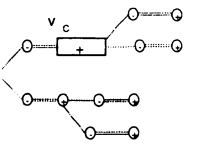
2. alternating tree T with augmenting path P after appending the unmatched node *s*



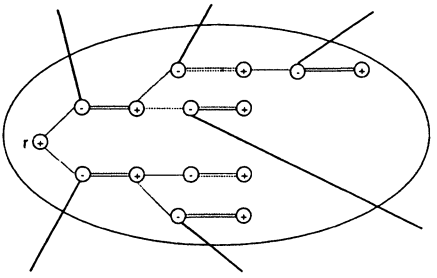
3. alternating tree after GROW - Operation



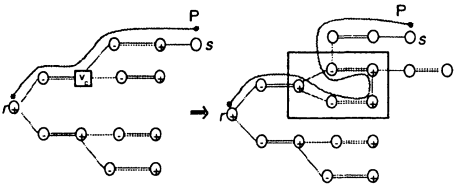
4. alternating tree with blossom C



5. alternating tree after SHRINK-Operation



5. Hungarian tree



6. Reconstructing an augmenting path after expanding a pseudonode

Figure 3.7 Operations on alternating trees (outer nodes are (+) labeled, inner nodes are (−) labeled.

For any augmenting path P and alternating cycle P with respect to M we define

$$l(P) := c(P \setminus M) - c(P \cap M).$$

This value is called the marginal cost of P or length of P since the following relation holds

$$c(M \oplus P) = c(M) + l(P).$$

Now an alternating cycle P is called a *negative alternating cycle* with respect to M if $l(P) < 0$. The following theorem, which we attribute to the matching-folklore gives a purely combinatorial optimality condition for a minimum cost perfect matching:

Theorem 3.2 *A perfect matching M is a minimum cost perfect matching if and only if M does not allow negative alternating cycles.*

A (not necessarily perfect) matching M which does not allow negative alternating cycles is called *extreme.* Theorem 3.2 also motivates a matching approach:

Negative cycle approach.
"Start from any perfect matching and improve the matching successively over negative alternating cycles until an extreme matching is obtained."

Extreme matchings in G can be constructed following the procedure which is inherent in the following theorem:

Theorem 3.3 *Let M be an extreme matching and P a shortest augmenting path connecting two unsaturated nodes. Then $M \oplus P$ is also an extreme matching.*

This theorem motivates the following approach:

Shortest augmenting path approach.
"Start from any extreme matching $M(M = \emptyset$ possibly) and augment successively using shortest augmenting paths until a perfect matching is obtained."

It can be shown that all known efficient graph theoretical matching algorithms are based on either of these two approaches (cf. Derigs [1988a]) which are related in the following way:

Given a perfect matching M with $(i, j) \in M$ then the negative cycle approach would ask whether there exists a negative alternating cycle C

containing edge (i,j). This question can be settled in the following way: Set $M' := M \setminus (i,j)$, then M allows a negative alternating cycle if and only if $l(P) < c_{ij}$ holds for the shortest augmenting path with respect to M'.

Thus, the problem of finding negative alternating cycles can be solved by computing shortest alternating paths and hence the problem of efficiently constructing shortest augmenting paths seems to be the key-problem in all efficient matching algorithms.

For $s \in V \setminus V(M)$ we define $\mathcal{P}_s(M)$ to be the set of all augmenting paths with start node s. Let P_0 be the shortest alternating path contained in $\mathcal{P}_s(M)$. Due to the fact that $P_0 \in \mathcal{P}_s(M)$ can be found by growing an alternating tree rooted at node s we call this version the *SAP-tree* method.

If we define by $\mathcal{P}(M)$ the set of all augmenting paths with respect to M, we can modify the method outlined above by treating $\mathcal{P}(M)$ instead of $\mathcal{P}_s(M)$. This modification is called *SAP-forest* version since the shortest augmenting path $P_0 \in \mathcal{P}(M)$ can be found by growing all the alternating trees rooted at the unmatched nodes $s \in V$ simultaneously.

4. MATCHING: POLYHEDRAL ASPECTS

From the results of Edmonds [1965b] it is well known that in the non-bipartite case the integrality conditions can only be relaxed to the non-negativity conditions if a set of additional constraints is added simultaneously.

Let $\mathcal{R} := \{R \subseteq V \mid |R| \geq 3 \text{ odd }\}$, then either of the following sets of constraints has to be added:

- the set of *blossom inequalities*

$$\sum_{(i,j)\in\gamma(R)} x_{ij} \leq \frac{1}{2}(|R|-1) \qquad \text{for } R \in \mathcal{R} \tag{3.6}$$

- the set of *cut inequalities*

$$\sum_{(i,j)\in\delta(R)} x_{ij} \geq 1 \qquad \text{for } R \in \mathcal{R}. \tag{3.7}$$

The importance of these inequalities is that any extreme solution to

$$\min\left\{\sum_{(i,j)\in E} c_{ij}\cdot x_{ij} \mid x \text{ fulfills } (3.2),(3.6),(3.4)\right\} \tag{3.8}$$

or

$$\min \left\{ \sum_{(i,j)\in E} c_{ij} \cdot x_{ij} \mid x \text{ fufills } (3.2), (3.7), (3.4) \right\} \tag{3.9}$$

is a solution for MP.

Further on we will call the system using the blossom inequalities "blossom characterization" and the system using the cut inequalities the "cut characterization" accordingly.

Obviously, both characterizations are equivalent. In the following we focus on the cut characterization since implementing matching algorithms based on this characterization lead to optimality conditions and processing rules with a more intuitive semantic.

When using the cut characterization, we get the dual program (DUAL-MP)

$$max \quad \sum_{i\in V} y_i + \sum_{R\in\mathcal{R}} y_R \qquad s.t. \tag{3.10}$$

$$y_i + y_j + \sum_{R:(i,j)\in\delta(R)} y_R \leq c_{ij} \quad \text{for} \quad (i,j) \in E \tag{3.11}$$

$$y_R \geq 0 \quad \text{for} \quad R \in \mathcal{R} \tag{3.12}$$

and the complementary slackness conditions are given by

$$x_{ij} > 0 \Rightarrow y_i + y_j + \sum_{R:(i,j)\in\delta(R)} y_R = c_{ij} \tag{3.13}$$

$$y_R > 0 \Rightarrow \sum_{R:(i,j)\in\delta(R)} x_{ij} = 1. \tag{3.14}$$

Edmonds [1965b] showed the correctness of the blossom/ cut characterization algorithmically, i.e. he developed an algorithm, the famous *blossom-algorithm*, which for any given instantiation of (MP) constructs a perfect matching M and a feasible dual solution fulfilling complementary slackness. Using this proof-technique Edmonds killed two birds with one stone: He did not only establish the so-called *linear characterization* of perfect matchings, but also showed the correctness of his (non-simplex) matching algorithm.

Direct non-algorithmical proofs for Edmonds' characterization of the matching polytope were later given by several authors (see e.g. Schrijver [1983]).

In fact, the class of odd sets $R \in \mathcal{R}$ defining necessary inequalities can be further restricted to the subclass $\mathcal{S} \subseteq \mathcal{R}$ of so-called *hypomatchable sets* or *shrinkable sets.* (cf. Edmonds and Pulleyblank [1974]). Here, a set $R \in \mathcal{R}$ is called hypomatchable (shrinkable) if it has the property that whenever any node in R is matched with a node not in R, the remaining set of nodes in R can be matched using edges in $\gamma(R)$ only.

Let $G' = (V', E')$ be the graph obtained from G by shrinking some hypomatchable sets to pseudonodes. Then the importance of this class stems from the fact that whenever an augmenting path or perfect matching in G' is found this path or matching can canonically be extended to an augmenting path or perfect matching in G.

Obviously, any odd cycle in G is a hypomatchable set and the algorithm for finding augmenting paths given in section 3 can be interpreted as successively shrinking all hypomatchable sets of the odd cycle subclass which are detected and thereby shrinking more complex hypomatchable sets of nested odd cycles.

Edmonds and Pulleyblank [1974] have shown that any hypomatchable set is spanned by such a system of nested odd cycles. Thus, we extend the notation of a blossom to the set of hypomatchable sets, i.e. a hypomatchable set S in G is called a *blossom* with respect to a matching M if $\mid M \cap \gamma(S) \mid \; = \; \frac{1}{2} \; (|S| - 1)$.

Given a dual solution y we define

$$\bar{c}_{ij} := c_{ij} - y_{ij} - y_j - \sum_{R:(ij)\in\delta(R)} y_R \tag{3.15}$$

the *reduced cost* of edge $(i,j) \in E$ with respect to y.

An edge with $\bar{c}_{ij} = 0$ is called *tight* with respect to y and we define $G(y) := (V, E(y))$ with $E(y) := \{(i,j) \in E \mid \bar{c}_{ij} = 0\}$ the subgraph of tight edges.

Analyzing the primal and dual solution produced by the blossom algorithm the necessary and sufficient optimality conditions can be reformulated as follows:

A perfect matching M in G is optimal if and only if there exists a dual solution y for the cut-characterization or the blossom-characterization, respectively, with the following property

$$M \subseteq E(y), \text{ i.e. } M \text{ contains tight edges only} \tag{3.16}$$

$$y_R > 0 \Rightarrow \mid M \cap \delta(R) \mid = 1. \tag{3.17}$$

Since M is a perfect matching in $G(y)$ we can deduce from (3.17) that the following equivalent property holds, too:

$$y_R > 0 \Rightarrow \ | M \cap \gamma(R)| = \frac{1}{2} \ (|R| - 1). \tag{3.18}$$

From the results of Edmonds [1965b] more precisely from the output of the blossom algorithm, we know that for characterizing optimal matchings it suffices to consider a subset of dual vectors $y \in \mathbb{R}_+^{V \cup \mathcal{S}}$ which we will refer to as *strongly dual feasible.* Here a vector $y \in \mathbb{R}_+^{V \cup \mathcal{S}}$ is called strongly dual feasible if y is dual feasible and y fulfills

$$y_S > 0 \Rightarrow \ S \text{ hypomatchable in } G(y). \tag{3.19}$$

The advantage of this subclass of dual solutions is the fact that they are algorithmically more easy to handle. Given a strongly dual feasible vector y we define $\mathcal{S}(y) := \{S \in \mathcal{S} \mid y_s > 0\}$. A set $C \in \mathcal{S}(y)$ is called maximal if there does not exist a set $R \in \mathcal{S}(y)$ with $C \subset R$. Then we obtain the so-called *surface graph* $G \times \mathcal{S}(y)$ by shrinking all maximal sets $C \in \mathcal{S}(y)$ using the transformation given in section 3. Then the dual variable y_R can be interpreted as the (dual) variable of the pseudonode v_R representing the blossom R in the surface graph.

A not necessarily perfect matching M and a strongly dual feasible vector y are called a *compatible pair* if they fulfill (3.16), (3.18) and (3.19). Now the following theorem can be shown (cf. Derigs [1986]).

Theorem 3.4 *M is extreme if and only if there exists a strongly dual feasible y such that (M,y) is a compatible pair.*

The following theorem concerning alternate dual solutions which was first proved in Derigs [1982] and Ball and Derigs [1983] is essential for the efficient matching implementations using the "Price and Reoptimize" strategy (cf. section 6).

Theorem 3.5 *Let M be an extreme matching and $i \in V$ arbitrarily chosen then there exists a dual vector y such that (M,y) is a compatible pair and $y_S = 0$ for all $S \in \mathcal{S}$ with $i \in S$ (i.e., any node $i \in V$ can be "forced" into the surface graph $G \times \mathcal{S}(y)$).*

It is important to note that given any compatible pair (M,y) a compatible pair (M,y') fulfilling this additional property can be constructed by essentially one application of the shortest augmenting path labeling method presented in Derigs [1981a].

5. MATCHING ALGORITHMS: LINKING COMBINATORIAL AND POLYHEDRAL RESULTS

Edmonds' algorithm [1965b] combines in a fascinating way polyhedral aspects, especially results from LP-duality with graph theoretical/combinatorial concepts, an approach which initiated a very successful field or research area: *polyhedral combinatorics.*

Throughout the procedure Edmonds' algorithm keeps a compatible pair (M, y). The matching M is grown using augmenting paths P made up entirely of tight edges *ensuring* that after the augmentation the augmented matching $M \oplus P$ and the dual solution y are compatible again. To ensure the fulfillment of condition (3.18) hypomatchable sets $R \in \mathcal{S}$ with $y_R > 0$ are shrunk.

Now we start growing an alternating tree T in $G(y) \times \mathcal{S}(y)$ rooted at an unmatched node r. If T becomes an Hungarian tree, i.e. the surface graph does not contain an augmenting path P rooted at r the dual solution is successively altered to y' such that M and y' are compatible and the Hungarian tree can be grown in the new surface graph.

Thus, if the alternating tree T grown in $G(y) \times \mathcal{S}(y)$ becomes a Hungarian tree we add a nonnegative value ϵ to the dual variable y_v for all outer nodes/ pseudonodes and we subtract ϵ from y_v for all inner nodes/ pseudonodes, i.e. we set

$$y'_k = \begin{cases} y_k + \epsilon & \text{for } k \text{ outer (pseudo-)node in } G(y) \times \mathcal{S}(y) \\ y_k - \epsilon & \text{for } k \text{ inner (pseudo-)node in } G(y) \times \mathcal{S}(y). \end{cases} \tag{3.20}$$

The constant ϵ is chosen as large as possible subject to the constraint that the complementary slackness conditions are fulfilled after the dual change, i.e. (M, y') is a compatible pair. We determine

$\epsilon_1 := \min\{\bar{c}_{i,j} |$ edge (i,j) joins an outer node to a node not in $T\}$,

$\epsilon_2 := \min\{\bar{c}_{ij}/2 |$ edge (i,j) joins two outer nodes$\}$,

$\epsilon_3 := \min\{y_R \,| v_R$ is an inner pseudonode$\}$,

and we set $\epsilon := \min\{\epsilon_1, \epsilon_2, \epsilon_3\}$.

If $\epsilon = \epsilon_1$, then after the dual change T can be grown by a new tight edge, if $\epsilon = \epsilon_2$, adding the tight edge determining ϵ_2 to the tree creates an odd cycle C which has to be shrunk into a pseudonode. If $\epsilon = \epsilon_3$, then after the dual change an inner pseudonode v_R has received a zero dual value and the outermost blossom R represented by v_R can be expanded. In any case we then try to grow the alternating tree in the new surface graph $G(y') \times \mathcal{S}(y')$.

From this description it becomes apparent that the blossom-algorithm can be categorized as primal-dual method and extension of the so-called Hungarian method for solving the assignment problem to the non-bipartite case (cf. Papadimitriou and Steiglitz [1976]) and there are two central routines to be combined:

- a routine for finding augmenting paths and
- a routine for updating dual solutions.

Analyzing different implementations and approaches for finding optimal matchings Derigs [1988a] showed that the determination of "shortest augmenting paths" can be identified as the driving concept for all combinatorial approaches either explicitly or implicitly. Thus, it turns out that every augmenting path in a subgraph $G(y)$ is a shortest augmenting path. Because of this fundamental property the relationship between (shortest) augmenting paths in G and (shortest) augmenting paths in $G \times \mathcal{S}(y)$ is summarized in the following proposition. For a proof see Derigs [1981a].

Proposition 3.1 *Let (M, y) be a compatible pair with M_S the matching in $G \times S(y)$ induced by M. Then the following properties hold:*

(i) Any alternating cycle C_S in $G \times \mathcal{S}(y)$ induces an alternating cycle C in G with $l(C) = \bar{l}(C)$.

(ii) Any augmenting path P_S connecting two unmatched nodes i and j in $G \times \mathcal{S}(y)$ induces uniquely an augmenting path P in G connecting i and j with $l(P) = \bar{l}(P) + y_i + y_j$.

(iii) Let P be an augmenting path in G which is induced by an augmenting path in $G \times \mathcal{S}(y)$ and for $R \in \mathcal{S}(y)$ let $P_R := E(R) \cap E(P)$, i.e. the subpath of P contained in R, then $\bar{l}(P_R) = 0$.

Thus, if the shortest M-augmenting path in G is induced by an M-augmenting path in the surface graph, then it can be found by determining the shortest M-augmenting path in $G \times \mathcal{S}(y)$ with respect to the reduced cost $\bar{c}_{ij}$. Note that in the shortest augmenting path algorithm of Derigs [1981a] all compatible pairs (M, y) fulfill the additional property

$$M \cap \delta(i) = \emptyset \Rightarrow y_i = 0 \text{ for } i \in V$$

and thus, in proposition 3.1 *(ii)* we obtain the relationship $l(P) = \bar{l}(P)$.

Now for all edges the reduced cost $\bar{c}_{ij}$ are non-negative and $\bar{c}_{ij} = 0$ for matching edges. And thus, we obtain

$$l(P) = \bar{l}(P) = \sum_{(i,j) \in P} \bar{c}_{ij}$$

and we can apply Dijkstra-like labeling methods for finding the shortest alternating path in the surface graph.

However, the shortest M-augmenting path in G does not necessarily be induced by an augmenting path in $G \times \mathcal{S}(y)$. The matching algorithm must be able to, first, detect whether this might be the case and, second, overcome this situation. The following proposition gives a method to control this situation.

Proposition 3.2 *Let P be an M-augmenting path in G which is not induced by an augmenting path in the surface graph but which is induced by an augmenting path in $G' = G \times (\mathcal{S}(y) \setminus R)$, i.e. the graph obtained by expanding an outermost blossom $R \in \mathcal{S}$ then*

$$l(P) = \bar{l}(P) + 2y_R$$

For a proof see Derigs [1981a].
We say that the augmenting path P is hidden in the pseudonode v_R.

Thus, as long as growing the alternating tree in $G \times \mathcal{S}(y)$ guarantees to find an augmenting path the length of which does not exceed the reduced cost length of an augmenting path hidden in a pseudonode v_R by more than $2y_R$, the shortest path found in $G \times \mathcal{S}(y)$ will also induce a shortest augmenting path in G_0.

In Derigs [1981a], [1988a] a Dijkstra-like labeling method for constructing minimum cost perfect matchings via constructing a sequence of compatible pairs and augmenting paths with respect to the extreme matchings is described in detail (and FORTRAN-codes are given in Burkard and Derigs [1980], and Derigs [1988b]). The labeling method assigns two distance-labels to nodes giving the length of (shortest) alternating paths of odd and even length from a root node, respectively, and maintains control variables which indicate the possibility of the existence of shorter candidate paths which are presently hidden in the surface graph.

Ball and Derigs [1983] demonstrate that this procedure can be interpreted as an implementation of the primal-dual/blossom algorithm where the possibly many dual updates within one augmentation phase are comprised to one single dual update performed after the augmentation and the distance labels are the aggregation of the ϵ-values of the dual changes that have been postponed.

Conversely, the blossom algorithm can be interpreted as a shortest augmenting path method. Yet the problem of finding such a shortest augmenting path is reduced to a sequence of "easier" problems, i.e. to decide whether any augmenting path in $G \times \mathcal{S}(y)$ exists. Such a reduction is the common philosophy in so-called primal-dual algorithms.

6. MATCHING ALGORITHMS: IMPLEMENTATION ISSUES

A first and rather straightforward implementation of the blossom algorithm leads to a complexity of $\mathcal{O}(|V|^2|E|)$. A first improvement to $\mathcal{O}(|V|^3)$ based on a more efficient data-structure for storing the blossoms and managing the augmentations and dual changes was given by Lawler [1976]. Using more involved data structures and concepts like scaling the cost coefficients the time bound could be further reduced by Gabow [1990] to $\mathcal{O}(|V|(|E| + |V|log|V|))$ and to $\mathcal{O}\left(|E|\ log\ (|V| \cdot N)\ \sqrt{|V| \cdot \alpha\ (|V|, |E|)\ log\ |V|}\right)$ by Gabow and Tarjan [1991] with N the magnitude of the largest edge cost and $\alpha(n, m)$ the inverse of the Ackermann function introduced by Tarjan [1983].

For the *Euclidean matching problem* in which the nodes of the graph are given as points in the plane and the cost of an edge (i, j) is defined to be the distance between i and j, Vaidya [1989] showed the complexity of $\mathcal{O}\left(|V|^{5/2}\ (log\ |V|)^4\right)$.

According to our knowledge these very refined implementations were never materialized into machine-executable codes and tested on (large scale) instances.

Parallel to the development of implementations with improved theoretical complexity several researchers have focused on the "practical performance" of matching algorithms and have developed a bundle of techniques, heuristics, tricks etc. to reduce the running time of matching codes enabling the solution of rather large instances and thus, proving that matching is indeed a well-solvable class of combinatorial programming.

In this section we focus on this aspect and present some of these approaches. Some enhancements were developed and evaluated for the special case of the Euclidean matching problem and thus, their applicability is limited, i.e. their generalization and their computational advantage for arbitrary graphs remains questionable.

In the following we describe several key-components which have shown to be significant time savers in several independent studies. These components or enhancements may be applied simultaneously, but then their combined behavior may be difficult to analyze and will depend on the special instance - euclidean/ non-euclidean graph, size of the graph, range of the cost-coefficients etc. So far, no single combination has shown to be dominating the other, but one can say that besides the use of an efficient basic data-structure for handling blossoms and augmentations the use of some of these enhancements is necessary to be able to solve large instances in reasonable time. The enhancements which we describe in the remainder of this section concern

- the choice of the initial (extreme) matching,
- the organization of the dual changes, and
- the adaption of concepts used for solving large scale linear programs, i.e. the strategy to determine optimal matchings for sparse subgraphs and use outpricing and reoptimization.

6.1. START PROCEDURES: CONSTRUCTING THE INITIAL EXTREME MATCHING

The blossom algorithm and hence the shortest augmenting path method can be initialized with any compatible pair (M, y). Here a trivial start would be to choose $M = \emptyset$ and $y \equiv 0$. Yet, obviously the quality of the initial pair (M, y) has significant influence on the computational effort, i.e. the running time for identifying and performing the remaining augmentations. Such a "good" initial pair is usually constructed in a so-called *preprocessing phase.* While early implementations of the shortest augmenting path method like the SMP-code given in Burkard and Derigs [1980] were focussing on producing initial compatible pairs (M, y) with a matching of large cardinality $|M|$ and were ignoring the importance of a "good" y-vector, recent applications of the shortest augmenting path labeling method in connection with results on postoptimal analysis demonstrated the importance of a good pair (M, y). Note that given a dual solution y with $y_S = 0$ for $S \in \mathcal{S}$ any matching $M \subseteq E(y)$ can be used to form an initial compatible pair (M, y).

The SMP-code presented in Burkard and Derigs [1980] uses a greedy-like routine to determine an initial compatible pair (M, y) which can be described by the following high-level language program assuming $V = \{1, ..., n\}$:

procedure SMP-Start
Let $M := \emptyset$ and $y \equiv 0$

For $i = 1$ until n
    if i unmatched do
        begin
        set $\delta := min\{c_{ij} - y_j | (i,j) \in E\}$
        if exists j_0 unmatched with $c_{ij_0} - y_{j_0} = \delta$ set $M := M \cup (i, j_0)$
        set $y_i := \delta$
        end

The above method constructs a matching M and simultaneously a dual solution y with the property that M is saturated in $E(y)$. M need not be a maximum cardinality matching in $G(y)$. Yet, computational tests have shown that constructing a maximum cardinality matching M' in $G(y)$ by augmentation of M and then starting the blossom-algorithm/ shortest augmenting path method with (M', y) does not pay.

Another more efficient start procedure based on solving the so-called fractional matching problem first has been developed by Derigs and Metz [1986a].

Any basic solution of the linear system (3.2) and (3.4) is called a (perfect) *fractional matching* of G. The following theorem (Balinski [1972]) describes the combinatorial nature of fractional matchings.

Theorem 3.6 *For a perfect fractional matching let* $Ex := (\{e \in E | x_e > 0\})$. *Then* x *is* $(0, 1, 1/2)$*-valued and the components of the subgraph* $G' = (V, E(x))$ *are either a pair of nodes joint by an edge or an odd cycle.*

Given a perfect fractional matching x, let $E_1(x) := \{e \in E \mid x_e = 1\}$ then $E_1(x)$ is a matching in G. Moreover for any odd cycle component $(V_k, E_k(x))$ of G' we have $x_e = 1/2$ for $e \in E_k(x)$ and we can easily obtain a matching $M_k \subseteq E_k(x)$ of cardinality $(|V_k| - 1)/2$ leaving arbitrarily one node $v \in V_k$ unmatched. Thus, any fractional matching x gives raise to a (non unique) matching $M(x)$ with $E_1(x) \subseteq M(x) \subseteq E(x)$ of cardinality $|V|/2 - c(x)$ where $c(x)$ is the number of odd cycle components of G'.

Now consider the linear program (fractional matching problem, FMP):

$$\min\{c'x | \ x \text{ fulfills (3.2) and (3.4)}\}$$

then the following proposition holds:

Proposition 3.3 *Let* x *be an optimal perfect fractional matching in* G *and let* $y \in \mathbb{R}^{|V|}$ *be a complementary dual solution. Then any matching* $M \subseteq E(y)$ *is an extreme matching in* G *and* (M, y) *is a compatible pair.*

Note that $E(x) \subseteq E(y)$. Thus, a compatible pair (M, y) with $|M| = |V|/2 - c(x)$ can easily be obtained from an optimal perfect fractional matching x.

FMP can be solved by any linear programming algorithm. Yet, FMP has a special combinatorial nature, which allows to apply more efficient special purpose algorithms. FMP is equivalent to a perfect matching problem (assignment problem) in a bipartite graph $\tilde{G} = (V', V'', \tilde{E})$ related to G. This graph is obtained in the following way: Split each node $v \in V$ into two nodes v', v''. Split each edge $(v, w) \in E$ into edges (v', w'') and (w', v''). Let $\tilde{x}$ be the incidence vector of a perfect matching $\tilde{M}$ in G then the associated perfect fractional matching x in G is given by $x_{v,w} = (\tilde{x}_{v',w''} + \tilde{x}_{w',v''})/2$. Setting $\tilde{c}_{v',w''} := \tilde{c}_{w',v''} := c_{v,w}/2$ we get $\tilde{c}'\tilde{x} = c'x$ and hence the optimal perfect matching in $\tilde{G}$ with respect to $\tilde{c}$ induces an optimal perfect matching in G.

Now let $\tilde{x}$ be the incidence vector of an optimal perfect matching in $\tilde{G}$ and $\tilde{y}$ an optimal (complementary) dual solution, i.e.

$$\begin{array}{rcll} \tilde{y}_{v'} & + & \tilde{y}_{w''} & \leq \tilde{c}_{v',w''} \quad \text{for } (v', w'') \in \tilde{E} \\ \tilde{y}_{v'} & + & \tilde{y}_{w''} & = \tilde{c}_{v',w''} \quad \text{if } \tilde{x}_{v',w''} > 0. \end{array}$$

Note that with $\tilde{x}$ also $\hat{x}$ with $\hat{x}_{v',w''} := \tilde{x}_{w',v''}$ induces an optimal perfect matching $\hat{M}$ in $\tilde{G}$ and $\tilde{y}$ is also complementary to $\hat{x}$. Thus, setting

$$y_v := (\tilde{y}_{v'} + \tilde{y}_{v''})/2$$

we get

$$\begin{array}{rcll} y_v & + & y_w & \leq c_{v,w} \quad \text{for } (v, w) \in E \\ y_v & + & y_w & = c_{v,w} \quad \text{if } x_{v,w} > 0. \end{array}$$

Hence y is an optimal dual solution to FMP which is complementary to x and thus, $(M(x), y)$ is a compatible pair for any matching $M(x) \subseteq E(y)$.

FMP can be solved by the same concepts as the original non-bipartite matching problem, i.e. the primal-dual Hungarian method/ shortest augmenting path method (cf. Derigs [1985]). For solving large scale assignment problems the application of the "Price and Reoptimize"-approach has shown to be highly efficient (cf. Derigs and Metz [1986b]).

6.2. ORGANIZATION OF DUAL UPDATES

A dual update is necessary if the surface graph does not allow a perfect matching, i.e. for at least one unmatched node r the search for an augmenting path fails. In the so-called tree-implementation we prespecify the unsaturated node r beforehand and update the duals of the

tree nodes whenever the single alternating tree rooted at r becomes an Hungarian tree. In the forest-version we simultaneously grow alternating trees from all unmatched nodes and update the duals of all nodes contained in the forest.

Using the forest version costs some overhead but usually leads to augmenting paths which are shorter with respect to the number of edges in the paths. While at the beginning of the augmentation process short augmenting paths (in the just described sense) exist for all unsaturated nodes and therefore the tree-version has some advantages, the strategy to grow only single alternating trees for the last augmentations makes the tree version inferior to the forest-version.

This experience and argumentation has to be viewed under the additional aspect that, when using the special start procedures described above, on the average 95% of the nodes are already matched when entering the blossom algorithm/ shortest augmenting path method and thus the forest-version can be expected to work faster.

Also, for the shortest augmenting path implementation of the blossom algorithm the overhead of the forest-version is reduced, since applying the shortest path labeling method can be viewed as aggregating and postponing dual-updates until the point of augmentation.

Cook and Rohe [1998] have developed a so-called "variable ϵ-approach" for the forest-version where each alternating tree T_i has its own dual change ϵ_i. This technique proved to be superior to the standard tree- and forest-version for the Euclidean matching problem, when applied in combination with the "Price and Reoptimize"-approach (cf. section 6.3), i.e. the dual change logic was applied to rather sparse graphs.

6.3. PRICE AND REOPTIMIZE

The strategy to compute an optimal solution to a sparse subproblem first and then to use duality to check optimality and control a process of introducing variables is a standard linear programming technique for solving large problem instances.

This approach has motivated the very successful principle for solving large structured, highly constrained problems called (delayed) *column generation.* The application of this idea to solving MP was first proposed by Grötschel and Holland [1985] as key issue in their facet-generating approach, as well as by Derigs [1986] in a shortest augmenting path based "primal approach".

The basic idea is as follows:

Price and Reoptimize-algorithm

Step 1 Select a sparse working subgraph $G' = (V, E')$ of G, i.e. $E' \subseteq E$
Step 2 Construct an optimal pair (M, y) in G'
(using a matching code for sparse problem instances)
Step 3 Calculate the reduced costs of the edges in $E \setminus E'$ with respect to y if all of the reduced costs are nonnegative then STOP
((M, y) is an optimal pair in G)
else goto 4
(M may not be optimal in G)
Step 4 Select E^- (a subset) of the edges having negative reduced cost, set $E' := E' \cup E^-$
Step 5 (Re)solve the matching problem over $G' = (V, E')$ and goto 3.

When implementing this approach the following questions arise:

1 Which subgraph should be selected for initialization in Step 1,

2 which matching algorithm/ code should be used to solve the initial matching problem in Step 2,

3 how can the outpricing in Step 3 be performed efficiently,

4 which subset E^- of edges not pricing out correctly should be introduced into the working subgraph in Step 4, and

5 how can the reoptimization in Step 5 be performed efficiently?

ad (1) The general scope is to construct an initial subgraph with only small computational effort which is very sparse and at the same time has a high potential of containing an optimal perfect matching. Approaches that have shown to be effective choose the *k-nearest graph* consisting of the k least costly edges meeting each node plus the edges in the initial extreme matching obtained from solving the fractional matching problem (Derigs and Metz [1991]) or the k edges of least reduced cost with respect to the dual solution for the fractional matching problem meeting each node (Applegate and Cook [1993]). For the special case of the Euclidean matching problem choosing the edges of an approximate Delauny triangulation of the set of points has been proposed by Cook and Rohe [1998].

ad (2) Since the initial graph is rather sparse, special implementations of the shortest augmenting path algorithm which take into account this sparsity in the data structures for storing the graph, in the procedures (priority queues) for scanning nodes during the shortest path computation etc. should be used.

ad (3) In the outpricing step we have to calculate

$$\overline{c}_{ij} := c_{ij} - y_i - y_j - \sum_{R:(i,j)\in\delta(R)} y_R$$

for edges (i,j) not contained in E'.

Here it is essential to be able to identify for a node i all blossoms which contain i and are presently shrunk. Thus, the efficiency of calculating the reduced costs depends strongly on the efficiency of the data-structure for managing the nested family $\mathcal{S}(y) = \{R \mid y_R > 0\}$. Let

$$sum(i) := y_i + \sum_{R:i\in R} y_R \quad \text{for } i \in V, \text{ then}$$
$$\tilde{c}_{ij} := c_{ij} - sum(i) - sum(j)$$

is an underestimate for the reduced costs $\overline{c}_{ij}$ of edge (i,j) which is easier to compute than $\overline{c}_{ij}$ and we only need to price out correctly those edges for which $\tilde{c}_{ij} < 0$.

Derigs and Metz [1986b] and Metz [1987] developed a special analysis of the dual solution reducing the number of superfluous outpricing operations. Applegate and Cook [1993] and Cook and Rohe [1998] describe similar techniques to reduce the effort for identifying critical edges, i.e. edges which might not price out correctly in the case of Euclidean matching problems.

ad (4) Identifying the edge $(i,j) \notin E'$ with minimal reduced cost and introducing this edge into E' if the reduced cost is negative, is in some sense equivalent to the standard Dantzig-rule for controlling the pivoting process in the Simplex method. Obviously, the determination of this edge is rather costly. The special structure of the matching problem motivates another strategy:

Let node $r \in V$ be fixed and $\epsilon_r := min \ \{\overline{c}_{rj} \mid (r,j) \in E\}$ then introducing the edge defining ϵ_r into E if $\epsilon_r < 0$ and resolving the matching problem using the updating technique described in (5) makes every edge in $\delta(r)$ feasible, i.e., without being introduced into the working graph explicitly all edges $(r,j) \in E$ are dual feasible after resolving the matching problem.

Consequently, investigating $\delta(r)$ for $r \in V$ in a round-robin fashion and enlarging E by subsets of the complete neighborhood $\delta(r)$ if $\epsilon_r < 0$ and then starting the reoptimization has shown to be quite efficient, since the working subgraph stays rather sparse throughout the whole procedure. Please observe that an edge that has priced out correctly in

one iteration may become infeasible in later iterations. Thus, in order to prove optimality, this procedure has to encounter a sequence of $|V|$ consecutive phases where for each node r that is fixed all edges in $\delta(r)$ price out correctly.

ad (5) Constructing the optimal matching in the modified subgraph G' can be performed "from scratch" and this may certainly be recommendable if the number of edges introduced after performing the out-pricing step is relatively large. Yet, if this set is small or has a special structure, for instance it contains only edges meeting a common node, then reoptimization, i.e. modifying the non-optimal perfect matching via negative alternating cycles should be preferable.

Here, in a first step the new edges from E^- have to be inserted into the existing solution, that is we have to construct a modified dual solution such that $\bar{c}_{ij} \geq 0$ for all edges in G'. Derigs [1982] and Ball and Derigs [1993] have demonstrated how this can be accomplished by the standard shortest augmenting path labeling method if successively subsets $E^-(r) := E^- \cap \delta(r)$ of edges meeting one node, r say, are introduced into G'.

Assume that $(r,t) \in M$. If r is shrunk into a pseudonode, we force r into the surface graph (cf. Theorem 3.5), i.e. in a first step the dual solution is modified such that all hypomatchable sets containing r receive zero dual value. This can be obtained by adding two artificial nodes a and b and two artificial edges (a,r) and (t,b) with sufficiently large costs, setting $y_a = y_b = 0$ and using the shortest augmenting path method to find the shortest augmenting path starting at node a. The only existing augmenting path P is hidden by the pseudonode containing r and has to use edge (t,b). Since we made this edge expensive enough, P can be "detected", i.e. P can become part of the alternating tree rooted at a only after all pseudonodes containing r have been expanded and r has been forced into the surface graph. Now we delete the artificial nodes and edges and we perform the dual update to obtain a new dual solution y.

If r is not shrunk into a pseudonode (anymore), we set $M := M \setminus \{(r,t)\}$ and we simply modify the dual variable of node r by a sufficiently large $\Delta > 0$ such that $\bar{c}_{rj} \geq 0$ with respect to y' for all edges in $\delta(r)$, i.e. we calculate

$$\Delta := \min \left\{ \bar{c}_{r,j} \mid (r,j) \in E^-(r) \right\}$$

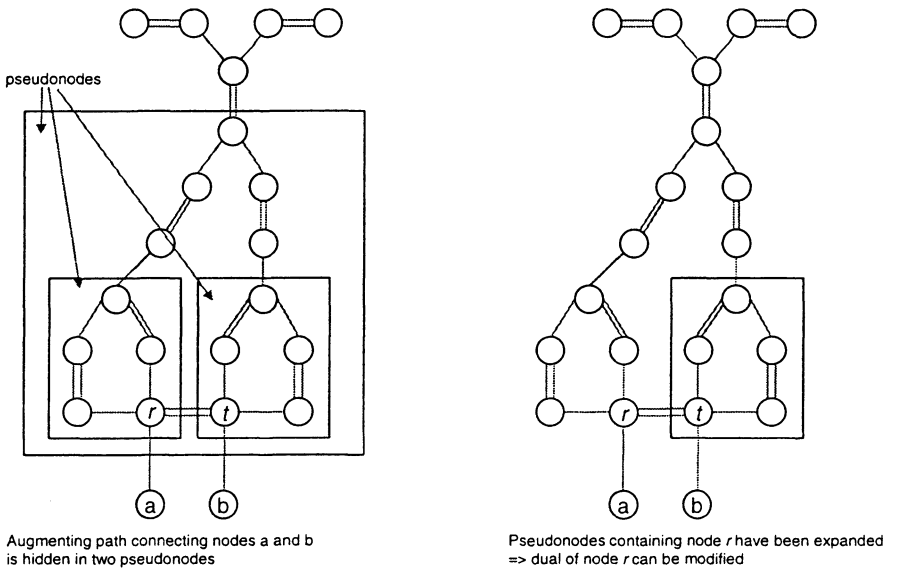


Figure 3.8 Expanding the pseudonode containing node r through SAP-steps.

and we set

$$y_i := \begin{cases} y_i' + \Delta & \text{for } i = r \\ y_i' & \text{else} \end{cases}$$

Now we could basically insert all edges meeting r, i.e. the set $\delta(r)$, into G' and (M, y) is a compatible pair in the enlarged graph, too.

After scanning the subset $E^-(r)$ we can either scan another subset, $E^-(s)$ say, or start reoptimization immediately by constructing the shortest augmenting path from node r. With both approaches we maintain the condition that throughout the whole procedure a compatible pair (M, y) is at hand and thus, the shortest augmenting path method can be used for reoptimization.

It is apparent that the choices for (4) and (5) are related. Thus, successively pricing out the neighborhood of the nodes and reoptimizing immediately has shown to be highly effective.

Cook and Rohe [1998] have introduced the concept of *careless repairs*. In contrast to the method just described each pseudonode containing node r is expanded and matching edges meeting those pseudonodes are eliminated from the matching. Thus results in a graph having a greater number of unmatched nodes compared to our approach described be-

fore. Cook and Rohe [1998] have shown that for the Euclidean matching problem applying careless repairs in combination with the variable ϵ-approach is outperforming our reoptimization procedure significantly. Yet, they also experienced that for large Euclidean instances the time necessary for completing their non-perfect matching was close to the time for computing the perfect matching from scratch.

References

[1] Altinkemer, K., and B. Gavish [1991]: *Parallel savings based heuristics for the delivery problem*; Operations Research 39, 456-469.

[2] Applegate, D., and W. Cook [1993]: *Solving large-scale matching problems*; in: Network Flows and Matching: First DIMACS Implementation Challenge (D.S. Johnson and C.C. McGeoch, editors), American Mathematical Society, 557-576.

[3] Bachem, A., and M. Malich [1993]: *The simulated trading heuristic for solving vehicle routing problems*; Operations Research 93, 16-19.

[4] Balinski, M.L. [1972]: *Establishing the matching polytope*; J. Comb. Theory, Ser. B 13, 1-13.

[5] Ball, M.O., L.D. Bodin, and R. Dial [1983]: *A matching based heuristic for scheduling mass transit crews and vehicles*; Transportation Science 17, 4-31.

[6] Ball, M.O., and U. Derigs [1983]: *An analysis of alternate strategies for implementing matching algorithms*; Networks 13, 517-549.

[7] Ball, M.O., U. Derigs, C. Hilbrand, and A. Metz [1990]: *Matching problems with generalized upper bound side constraints*; Networks 20, 703-721.

[8] Bendisch, J., U. Derigs, and A. Metz [1994]: *An efficient matching algorithm applied in statistical physis*; Discrete Applied Mathematics 52, 139-153.

[9] Berge, C. [1957]: *Two theorems in graph theory*; Proc. Natl. Acad. Sci. USA, 43, 842-844.

[10] Bodin, L. and B. Golden [1981]: *Classification in Vehicle Routing and Scheduling*; Networks 11, 97-108.

[11] Burkard, R.E., and U. Derigs [1980]: *Assignment and matching problems: Solution methods with FORTRAN-programs*; Springer Lecture Notes in Mathematical Systems No. 184.

[12] Carraresi, P., and G. Gallo [1984]: *Network models for vehicle and crew scheduling*; EJOR 16, 139-151.

[13] Christofides, N. [1976]: *Worst-case analysis of a new heuristic for the traveling salesman problem*; Management Science Research Report No. 388, Carnegie-Mellon University.

[14] Cook, W., and A. Rohe [1998]: *Computing minimum weight perfect matchings*; to appear in: ORSA J. on Computing.

[15] Cornuejols, G., and G.L. Nemhauser [1978]: *Tight bounds for Christofides traveling salesman heuristic*; Math. Programming 14, 116-121.

[16] Cunningham, W.H., and A.B. Marsh [1976]: *A primal algorithm for optimal matching*; Math. Programming Study 8, 50-72.

[17] Derigs, U. [1981a]: *A shortest augmenting path method for solving minimal perfect matching problems*; Networks 11, 379-390.

[18] Derigs, U. [1981b]: *On some experiments with a composite heuristic for solving traveling salesman problems*; Methods of Operations Research 40, 287-290.

[19] Derigs, U. [1981c]: *Another composite heuristic for solving the traveling salesman problem*; University of Maryland, Working Paper.

[20] Derigs, U. [1982]: *Shortest augmenting paths and sensitivity analysis for optimal matchings*; Report 82222-OR, Institut für Ökonometrie und Operations Research, Universität Bonn.

[21] Derigs, U. [1985]: *The shortest augmenting path method for solving assignment problems - Motivation and computational experience*; Annals of Operations Research 4, 57-102.

[22] Derigs, U. [1986]: *Solving large-scale matching problems efficiently - A new primal matching approach*; Networks 16, 1-16.

[23] Derigs, U. [1988a]: *Programming in networks and graphs - On the combinatorial background and near-equivalence of network flow and matching algorithms*; Lecture Notes in Economics and Mathematical Systems 300, Springer-Verlag.

[24] Derigs, U. [1988b]: *Solving non-bipartite matching problems via shortest path techniques*; Annals of Operations Research 13, 225-261.

[25] Derigs, U. and A. Metz [1986a]: *On the use of optimal fractional matchings for solving the (integer) matching problem*; Computing 36, 263-270.

[26] Derigs, U. and A. Metz [1986b]: *An in-core/out-of-core method for solving large scale assignment problems*; Zeitschrift für Operations Research 30, A181-A195.

[27] Derigs, U., and A. Metz [1991]: *Solving (large scale) matching problems combinatorially*; Math. Programming 50, 113-121.

[28] Derigs, U., and A. Metz [1992a]: *Über die Matching Relaxation für das Set Partitioning Problem*; in: Operations Research Proceedings 1991, 398-406.

[29] Derigs, U., and A. Metz [1992b]: *Matching problems with knapsack side constraints - A computational study -*; in: Modern Methods

of Optimization (ed. by W. Krabs and J. Zowe), Springer Lecture Notes in Economics and Mathematical Systems 378, 48-89.

[30] Derigs, U., and A. Metz [1992c]: *A matching based approach for solving a delivery/pick-up vehicle routing problem with time constraints*; OR Spektrum 14, 91-106.

[31] Desrochers, M., and T.W. Verhoog [1991]: *A new heuristic for the fleet size and mix vehicle routing problem*; Computers and Operations Research 18, 263-274.

[32] Dijkstra, E.W. [1959]: *A note on two problems in connection with graphs*; Numerische Mathematik 1, 269-271.

[33] Dror, M., and L. Levy [1986]: *A vehicle routing improvement algorithm comparison of a "greedy" and matching implementation for inventory routing*; Computers and Operations Research 13, 33-45.

[34] Edmonds, J. [1965a]: *Paths, trees and flowers*; Can J. Math. 17, 449-467.

[35] Edmonds, J. [1965b]: *Maximum matching and a polyhedron with 0,1 vertices*; J. Res. Natl. Bur. Standards 69B, 125-130.

[36] Edmonds, J., and E.L. Johnson [1973]: *Matching, Euler tours and the Chinese postman*; Math. Programming 5, 88-124.

[37] Edmonds, J. and W. Pulleyblank [1974]: *Facets of 1-matching polyhedra*; in: Hypergraph Seminar, Lecture Notes in Mathematics, 411, 214-242.

[38] Euler, L. [1736]: *Solutio problematis ad geometriam situs pertinentis*; Comment. Acad. Sci. Imp. Petropolitanae 8, 128-140.

[39] Fujii, M., T. Kasami, and N. Ninomiya [1969]: *Optimal sequencing of two equivalent processors*; SIAM J. Appl. Math 17, 784-789 [Erratum in: SIAM J. Appl. Math. 20 (1971) 141].

[40] Gabow, H.N. [1976]: *An efficient implementation of Edmond's algorithm for maximum matching on graphs*; J. of the ACM 23, 221-234.

[41] Gabow, H.N. [1990]: *Data structes for weighted matching and nearest common ancestors with linking*; Proc. of the First Annual ACM-SIAM Symp. on Discrete Algorithms, ACM, New York, 434-443.

[42] Gabow, H.N., and R.E. Tarjan [1983]: *A linear-time algorithm for a special case of disjoint set union*; Proc. 15th Annual ACM Symp. on Theory of Computing, York, N.Y., 246-251.

[43] Gabow, H.N., and R.E. Tarjan [1991]: *Faster scaling algorithms for general graph matching problems*; J. of the ACM 38, 815-853.

[44] Gerards, A.M.H. [1995]: *Matching*; in: Handbooks in OR & MS 7 (M.O. Ball, T.L. Magnanti, C.L. Monma, and G.L. Nemhauser, editors), North Holland, Amsterdam.

[45] Grötschel, M. [1977]: *Polyedrische Charakterisierung Kombinatorischer Optimierungsprobleme*; Mathematical Systems in Economics 36; Verlag Anton Hain, Meisenheim am Glan.

[46] Grötschel, M., and O. Holland [1985]: *Solving matching problems with linear programming*; Math. Programming 33, 243-259.

[47] Hasselström, D. [1976]: *Connecting bus-routes at a point of intersection*; AB VOLVO Working Paper.

[48] Held, M., and R.M. Karp [1971]: *The traveling salesman problem and minimum spanning trees: Part II*; Math. Programming 1, 6-25.

[49] Iri, M., K. Murota, and S. Matsui [1983]: *An approximate solution for the problem of optimizing the plotter pen movement*; in: R.F. Drenick and F. Kozin (eds), "System Modeling and Optimization", Proc. 10th IFIP Conf., New York, 1981, Lecture Notes in Control and Information Sciences, Vol. 38, Springer-Verlag, Berlin, 572-580.

[50] Johnson, D.S., and C.C. McGeoch [1993]: *Network Flows and Matching - First DIMACS Implementation Challenge*; American Mathematical Society.

[51] Kruskal Jr., J.B. [1956]: *On the shortest spanning subtree of a graph and the traveling salesman problem*; Proc. Amer. Math. Soc. 7, 48-50.

[52] Kuhn, H.W. [1955]: *The Hungarian method for the assignment problem*; Nav. Res. Log. Quart. 2, 83-97.

[53] Kwan, Mei Ko [1962]: *Graphic programming using odd and even points*; Chinese Math. 1, 273-277.

[54] Lawler, E.L. [1976]: *Combinatorial Optimization: Networks and Matroids*; Holt, Rinehart, and Winston, New York, N.Y.

[55] Lin, S., and B.W. Kernighan [1973]: *An effective heuristic algorithm for the traveling salesman problem*; Operations Research 21, 498-516.

[56] Lovasz, L., and M.D. Plummer [1986]: *Matching Theory*; Annals of Discrete Mathematics 29, North Holland.

[57] Machol, R.E. [1961]: *An application of the assignment problem*; Operations Research 9, 585-586.

[58] Metz, A. [1987]: *Postoptimale Analyse und neue primale Matching Algorithmen*; Diploma Thesis, University of Cologne.

[59] Miller, D.L. [1995]: *A matching based exact algorithm for capacitated vehicle routing problems*; ORSA J. on Computing 7, 1-9.

[60] Nemhauser, G., and G. Weber [1979]: *Optimal set partitioning, matchings and Lagrangean duality*; Naval Res. Log. Quart. 26, 553-563.

[61] Olafson, S. [1990]: *Weighted matching in chess tournaments*; Journal of the Operational Research Society 41, 17-24.

[62] Pandit, R. and B. Muraldharan [1995]: *A capacitated general routing problem on mixed networks*; Computers and Operations Research 26, 465-478.

[63] Papadimitriou, C.H., and K. Steiglitz [1976]: *Combinatorial Optimization: Algorithms and Complexity*; Prentice-Hall, Englewood Cliffs, NJ.

[64] Reingold, E.M., and R.E. Tarjan [1981]: *On a greedy heuristic for complete matching*; SIAM J. Comput. 10, 676-681.

[65] Riskin, E., R. Ladner, R. Wand, and L. Atlas [1994]: *Index assignment for progressive transmission of full-search vector quantization*; IEEE Transactions on Image Processing 3, 307-312.

[66] Schrijver, A. [1983]: *Short proofs on the matching polyhedron*; J. Comb. Theory, Ser. B 34, 104-108.

[67] Tarjan, R.E. [1983]: *Data structures and network algorithms*; SIAM, Philadelphia.

[68] Vaidya, P.M. [1989]: *Geometry helps in matching*; SIAM J. Comput. 18, 1201-1225.

[69] Weber, G. [1981]: *Sensitivity analysis of optimal matchings*; Networks 11, 41-56.

[70] Win, Z. [1988]: *Contributions to routing problems*; Unpublished Ph.D. dissertation Universität Ausburg.

Chapter 4

ARC ROUTING: COMPLEXITY AND APPROXIMABILITY

Moshe Dror
University of Arizona

1. Introduction: Easy and Hard Problems 133
2. CPP as a Problem in $\mathcal{P}$ 141
3. $\mathcal{NP}$-Hard Generalizations of the CPP 143
   3.1 The Mixed CPP 143
   3.2 The MCPP $\mathcal{NP}$-Completeness 144
   3.3 The Rural Postman Problem 147
   3.4 The Windy Postman Problem 148
   3.5 Non-intersecting Eulerian Circuits and A-trails in Eulerian Graphs 149
   3.6 Dominating Trails 151
   3.7 Precedence in Arc Routing 152
   3.8 Capacitated Arc Routing 154
4. Approximation Algorithms 156
5. Approximation Results for Arc Routing 159
   5.1 The Mixed CPP 159
   5.2 The Windy CPP 161
   5.3 The RPP and Other Variants 161
   5.4 The CARP 162
6. Conclusions 164

1. INTRODUCTION: EASY AND HARD PROBLEMS

The majority of arc routing problems can be viewed as variants of the classical Chinese Postman Problem (CPP). Restating the generic problem, let $G = (V, E)$ be a connected graph (undirected) with V a finite set (the nodes) and $E \subseteq V \times V$ be the set of edges. In addition, we have a real valued weight (distance) $w_{ij} \geq 0, \forall (i,j) \in E$, and a design problem: "Construct a least distance traversal sequence of all the edges in E starting at and returning to the same node." This is in essence the Chinese Postman Problem as posed by Meigu Guan (Mei-Ko Kwan)

in 1962, in the *Chinese Mathematics* journal which is the main reason why we refer to this problem as the CPP. The historical overview of arc routing and variants of CPP are eloquently described by Eiselt and Laporte (this book), however we examine the Guan (1962) work for its illustration of the computational aspects when solving the CPP and related problems. As pointed out in Edmonds and Johnson (1973), the CPP can be separated into two parts: given an arbitrary (connected) graph G, duplicate a set of edges in E of minimal total weight to transform G into $\hat{G}$ (an even degree graph) which admits an Euler tour (a closed tour which traverses exactly once every edge in the graph), and then construct an Euler tour on $\hat{G}$.

Since this chapter's focus is on computational complexity which classifies algorithms according to their performance characteristics, we need to define the appropriate mathematical notation:

Definition Given a nonnegative real function $f(x), x > 0$,
$\mathcal{O}(f(x))$ denotes the set of all real functions $g(x)$ such that $|g(x)/f(x)|$ is bounded from above as $x \to \infty$.
$\Omega(f(x))$ denotes the set of all real functions $g(x)$ such that $|g(x)/f(x)|$ is bounded from below by a (strictly) positive number as $x \to \infty$.
$\Theta(f(x))$ denotes the set of all real functions $g(x)$ such that $|g(x)/f(x)|$ is bounded from both above and below as $x \to \infty$.

It is interesting to note that Guan in his 1962 paper had most of the 'ingredients' for a good solution methodology for the CPP, however, it lacked in one important aspect. The construction of the corresponding Eulerian tour on $\hat{G}$ had an exponential worst time complexity! (See Fleischner, 1990, for details.) On the other hand, the Euler tour construction procedures described by Edmonds and Johnson (1973) are dominated in terms of time complexity by the routine required to transform G into $\hat{G}$. This routine has, for general undirected graphs, time complexity no worse than $\mathcal{O}(|V|^3)$, and for sparse graphs time complexity of $\mathcal{O}(|E||V|log|V|)$ (see Ball and Derigs, 1983). This brings us to the topic of computational complexity and its implications for arc routing problems.

Computational complexity examines the issue of tracking the 'effort' (as a measurable difficulty) required to generate a readable answer for a problem which requires computations. As Johnson (1990) put it "Given a problem, how much computing power and/or resources do we need in order to solve it?" In this chapter we address this issue focusing on arc routing problems. Since clearly this is about answers generated by computers, we need to examine the basic concepts of what a 'problem' is, and some of the related computer science terminology, in order to

do even a partial justice in addressing computational complexity. There have been a number of excellent, and much more comprehensive write-ups on this general topic (Yannakakis, 1997, Shmoys and Tardos, 1995, Johnson, 1990, Johnson and Papadimitriou, 1985, Garey and Johnson, 1979). This chapter is less comprehensive and less formal.

Following Yannakakis (1997), and Johnson (1990), we define a general computational problem as follows: Given that "$\{0,1\}^*$" represents the set of all finite strings over the alphabet of $\{0,1\}$ (any finite alphabet can serve the same purpose), a *problem* Π has a set D_Π of strings in $\{0,1\}^*$ representing the domain of the problem instances, and for each $x \in D_\Pi$ there is a set $A_\Pi(x)$ of strings in $\{0,1\}^*$ representing the corresponding possible acceptable answers for the instance x.

This technical definition of a computational problem is rich enough to represent all the combinatorial optimization problems of interest. For instance, one can easily describe any graph in the form of a finite string in $\{0,1\}$, and the same statement holds for any answer to a graph problem. Even though a number of different string representations are available for graphs, such as adjacency lists or incidence matrices, all such representations, if reasonable (not artificially padded), are related in the sense that a length of one string representation is bounded by a polynomial in the length of the other string representation. Thus, the choice of particular string representation is immaterial in practically all cases. An algorithm (i.e., a formally specified step by step sequence of 'actions' or 'moves') *solves* problem Π if it represents a mapping from D_Π to $A_\Pi \cup \emptyset$ (takes the value $\emptyset$ in case there is no answer).

Computational problems can be categorized in relation to the size of the output sets $A_\Pi(x)$. A *search problem* is a problem for which the set $A_\Pi(x)$ can have zero, or any positive number of elements. If $A_\Pi(x)$ is nonempty for all $x \in D_\Pi$, the problem is defined as *total*, and if the set $A_\Pi(x)$ has at most one member for all x, it is called *functional*. Decision problems are a special case of total functional problems and *combinatorial optimization problems* are a special case of search problems where for each instance $x \in D_\Pi$ there is a finite set of solutions and every solution has a cost. I.e., if i is a solution for $x \in D_\Pi$, we have a cost $f_\Pi(i,x)$ associated with this x.

Computing power or computing effort is measured in the context of machine computation models. Van Emde Boas (1990), describes machine computation models and the relations of time and space complexity of algorithms based on such machine models. The equivalence of the different machine models is due to the fact that (quote from the above) "each

computation in one formalism can be simulated by a computation in the other formalism." and most importantly, "a problem is unsolvable in one particular model, then it is also unsolvable for all other formalized computing devices to which this particular model is related by mutual simulation." Thus, theoretical computer science has embarked on the path of examination fundamental questions relating machine models and computational problems. One can perhaps say that this started with the classes of problems denoted as $\mathcal{P}$ and $\mathcal{NP}$, where $\mathcal{P}$ is the class of (decision) problems solvable in polynomial time by deterministic Turing machines and $\mathcal{NP}$ the class of (decision) problems solvable in polynomial time by nondeterministic Turing machines. Since these are machine-dependent definitions, computer science advanced the following two theses to set up complexity classes which are examined in machine-independent context (taken from Van Emde Boas, 1990).

Invariance Thesis: 'Reasonable' machines can simulate each other within a polynomially bounded overhead in time and a constant-factor overhead in space.

Parallel Computation Thesis: Whatever can be solved in polynomially bounded space on reasonable sequential machine model can be solved in polynomially bounded time on a reasonable parallel machine, and vice versa.

Though no precise definition of 'reasonable' machine is usually given, the basic premise is that such machines are unable to do an unrealistic amount of computation in one step, such as being able to add numbers of length 2^n in one step.

In order to make the categorization of the most important two classes of problems in this chapter $\mathcal{P}$ and $\mathcal{NP}$ even more clear, we **cite** a theorem by Cook (1971) which states the following:

Theorem 1: Let D be a decision problem. Then the following are equivalent.

i) D has the succinct certificate property.

ii) D is solvable in polynomial time by **a** nondeterministic Turing machine.

iii) D is transformable to INTEGER PROGRAMMING.

More formal definitions of the terms above, such as 'decision problem', 'succinct certificate property', and 'INTEGER PROGRAMMING', will follow below.

Is $\mathcal{P} = \mathcal{NP}$? In a chapter on computational complexity for arc routing problems one still has to state this fundamental open question about the power of deterministic machines to solve in polynomial time problems solvable in polynomial time by nondeterministic machines. We clearly do not intend to address this question here. It is just stated for completeness.

We restrict our complexity discussions mainly to the $\mathcal{NP}$ class of problems. Even for this class we restrict the discussions further by limiting the problems to decision problems only. For that we follow the definitions from Johnson (1990), where first a *function* is defined as a string relation in which each string $x \in \{0,1\}^*$ is the first component of precisely one pair, and a *decision problem* is defined as a function in which the only possible answers are "yes" and "no".

A *language* is defined as any subset of $\{0,1\}^*$ and a *decision problem* R_L *corresponding to the language* L is $\{(x,\text{yes}): x \in L\} \cup \{(x,\text{no}): x \notin L\}$. If L is a language, then its *complementary language* is $co{-}L = \{0,1\}^* - L$.

A problem instance is a string (of 0 and 1 elements) and a *size* of an instance I is represented by the number of symbols it contains. Since we have related all reasonable computing machines by the Invariance Thesis, we can now restrict the measure of a computational effort to the time requirement for solving a problem instance I in terms of the worst-case time over all instances with the number of symbols $|I|$ (the length of the instance). We have already used this measure in our introduction to the CPP by stating that the time complexity of the undirected CPP is given by $(\mathcal{O}(|V|^3)$.

The class $\mathcal{P}$ of problems contains all the so called 'easy' problems (solvable in polynomial running time). It is important to note that the *Linear Programming* problem is a member of the $\mathcal{P}$ class. The Linear Programming problem is said to be the hardest problem in class $\mathcal{P}$ (Johnson, 1990).

Following Johnson and Papadimitriou (1985), we will formalize the term *succinct certificate property* first. A *succinct certificate* for a given decision problem is a string in $\{0,1\}^*$ whose length is bounded by a polynomial in the instance length. A *polynomial-time certificate checking algorithm* is an algorithm which, given such a succinct certificate,

can verify whether the certificate is indeed valid. Also, all 'yes' instances of our decision problem have to possess at least one such certificate, and no 'no' answer can have it. What is important to note is that given a certificate for 'yes' it can be validated or discarded quickly (in polynomial-time).

Stating it a little differently, a decision problem D is said to possess the succinct certificate property if and only if there is another polynomial-time solvable decision problem C over the domain of instances of D and certificates for such instances ('short' strings in $\{0,1\}^*$) such that the problem instance warrants the answer "yes" if and only if the corresponding instance for C warrants the answer "yes".

An INTEGER PROGRAMMING problem can be formally presented in the following way:

INTEGER PROGRAMMING
Instance: An $m \times n$ integer matrix $A = (a_{ij})$, an m-vector $b = (b_1, b_2, \ldots, b_m)$ of integers.

Question: Is there an n-vector of nonnegative integers x such that $Ax = b$?

In terms of the three different though equivalent characterizations of the $\mathcal{NP}$ class presented in Theorem 1, we have described (i) and (iii). For a formal definition of nondeterministic Turing machine the reader is referred (among many others) to Van Emde Boas (1990).

The question with which this chapter will be concerned the most, is how to establish problem membership in a given class of problems. The 'tools' more frequently used to address this question are those of *reductions.* More specifically, in this chapter we restrict the reductions to *polynomial-time reductions* (or Turing reductions).

A reduction process can be defined with the help of a familiar algorithmic concept of a subroutine or more formally an oracle (or a call statement). This polynomial-time reduction construct relates polynomial-time solvability for problem X by employing an oracle (a subroutine) for problem Y. That is, problem X is polynomial-time reducible to problem Y if there is a polynomial-time solution for X which uses a subroutine for Y and counts the execution time for the subroutine as a single time unit (or a single step).

Clearly, if problem X is polynomial-time reducible to Y, and problem Y is solvable in polynomial time, then so is problem X (transitivity of polynomial-time reducibility). As Johnson and Papadimitriou (1985) state, essentially there are three kinds of reductions: (a) reductions that prove a problem easy by reducing it to a known easy problem, (b) reductions that prove a problem hard by reducing some known hard problem to it, and (c) reductions which prove nothing by reducing one problem of unknown type to another problem of unknown type.

This brings us to a point in this somewhat coarse exposition of complexity theory where we need to define the concept of *completeness.*

Definition 1: Suppose X is a decision problem and all the problems in the class $\mathcal{NP}$ are polynomial-time reducible to X. Then X is *hard* for the class $\mathcal{NP}$ (under the polynomial-time reduction). If the problem X is also a member of the class $\mathcal{NP}$, then the problem X is *complete* for $\mathcal{NP}$, or $\mathcal{NP}$*-complete* under polynomial-time reduction.

Another definition of $\mathcal{NP}$-complete problem, which does not mention polynomial-time reducibility, is based on the notion of a *transformation* from one decision problem to the other. More precisely, if X is a string relation (a decision problem) and Y is a string relation (another decision problem), a transformation is defined as a computable function f by DTM from $\{0,1\}^*$ to itself such that a problem instance I of X has a "yes" answer if and only if $f(I)$ has a "yes" answer as a an instance of Y.

Definition 2: A decision problem is $\mathcal{NP}$*-complete* if is complete for $\mathcal{NP}$ under polynomial transformations.

For the $\mathcal{NP}$ class of problems it is conjectured that both definitions of $\mathcal{NP}$-completeness, one based on polynomial-time reducibility and one based on transformation from one problem to the other, are the same. However, it is not clear if this holds also for other classes of problems (Johnson, 1990). We assume here that both definitions are equivalent for the $\mathcal{NP}$ class.

At this point one has to identify a member of the $\mathcal{NP}$-complete class of problems to establish that this class is not empty. Cook identified the first $\mathcal{NP}$-complete problem — the SATISFIABILITY problem (see Cook, 1971).

SATISFIABILITY
Instance: List of literals $U = (u_1, \bar{u}_1, u_2, \bar{u}_2, \ldots, u_n, \bar{u}_n)$, sequence of clauses $C = (c_1, c_2, \ldots, c_m)$, where each clause c_i is a subset of U.

Answer: "yes" if there is a truth assignment for the variables $u_1, u_2, \ldots, u_n$ that satisfies all the clauses in C, i.e., a subset $U' \subseteq U$ such that $|U' \cap \{u_i, \bar{u}_i\}| = 1, i = 1, 2, \ldots, n$, and such that $|U' \cap c_i| \geq 1, i = 1, 2, \ldots, m$.

By 1979 the $\mathcal{NP}$-completeness class of problems had expanded considerably. Garey and Johnson (1979) list over 300 $\mathcal{NP}$-complete problems and many more problems have been added to this list since then. (For a more recent list which includes approximability results for $\mathcal{NP}$-complete problems see Crescenzi and Kann, 1998.) However not all $\mathcal{NP}$-complete problems are equally difficult and Garey and Johnson make a distinction between two kinds of $\mathcal{NP}$-complete problems. First, there are those which can be solved by what is called a *pseudopolynomial-time* algorithm. A *pseudopolynomial-time* algorithm for a problem Π is defined as an algorithm whose running time is bounded by a polynomial if all input numbers are expressed in *unary* notation. That is, a pseudopolynomial-time algorithm will solve an instance in polynomial time provided that the numbers in the instance are bounded by a polynomial in the input size.

An example of a problem which can be solved in pseudopolynomial time is the PARTITION problem. It is solvable in $\mathcal{O}(nB)$ time, where B is the sum of the PARTITION numbers. Therefore the PARTITION problem is solvable efficiently when B is 'small'. However, the length of a binary encoding of PARTITION is of $\mathcal{O}(n \log B)$ order, and, as Garey and Johnson (1979) point out, nB is not bounded by any polynomial function in $n \log B$.

An $\mathcal{NP}$-complete problem which can be solved in pseudopolynomial-time is said to be $\mathcal{NP}$-complete *in the ordinary sense.* However, there are many $\mathcal{NP}$-complete problems (the SATISFIABILITY problem among them) which cannot be solved in even pseudopolynomial-time. These are said to be $\mathcal{NP}$-complete *in the strong sense.*

Following Johnson (1990), a search problem Π is *$\mathcal{NP}$-hard* if for some $\mathcal{NP}$-complete problem Y there is a polynomial-time Turing reduction from Y to Π. This implies that combinatorial optimization problem versions of $\mathcal{NP}$-complete decision problems are all $\mathcal{NP}$-hard. As Johnson (1990) points out, the $\mathcal{NP}$-hard problems do not constitute an equivalence class since one cannot impose an upper bound on the complexity of $\mathcal{NP}$-hard problems (because even undecidable problems can be $\mathcal{NP}$-hard). Thus, Johnson introduced the so called *$\mathcal{NP}$-easy* class of search problems by requiring that for such problem there be a polynomial-time Turing reduction to a problem in $\mathcal{NP}$. This enables the establishment

of an equivalence class of search problems that are called $\mathcal{NP}$-*equivalent* for problems that are both $\mathcal{NP}$-hard and $\mathcal{NP}$-easy.

At this point we stop with the classification of complexity classes since it has not been our intention to provide a detailed overview of complexity theory in its own right. This has been done very well by Johnson (1990) and others. Here we have only restated and defined the complexity theory terminology so we can examine arc routing problems and evaluate how hard it might be to construct optimal solutions for such problems. We are essentially grouping the different arc routing problem as 'easy' problems (in $\mathcal{P}$) and 'hard' problems ($\mathcal{NP}$-hard). We will also attempt to provide the running-time orders when they are known.

2. CPP AS A PROBLEM IN P

One of the original problems on Karp's (1972) list of $\mathcal{NP}$-complete problems is the Hamilton Circuit problem (*Input:* graph G; *Answer:* "yes" if G has a cycle which includes each node exactly once). This result establishes that the celebrated Traveling Salesman Problem (TSP) is an $\mathcal{NP}$-hard problem. In fact, Karp's reduction establishes that the TSP is $\mathcal{NP}$-hard in the strong sense. When we examine the CPP, it is also clear that one can transform (in polynomial time) the CPP into a TSP (see Mullaseril, 1996, and Laporte, 1997). However, this is not the transformation direction one would select for an 'easy' arc routing problem.

For examination of the CPP and related problems we need to define a number of new terms (see also Fleischner, this book). A *multigraph* has a finite set V of nodes but the set E of edges might have multiple edges identified by (joining) a pair of nodes in V. A *simple graph* (or just a *graph*) has no more than one edge joining any two nodes. Without trying to confuse the reader, sometimes we use G to denote a graph (simple graph) and sometimes we use G to denote a multigraph. We hope to make clear what is intended in each case. The *degree* of a node $v, d(v)$, is the number of edges incident upon v. In the case of directed edges (arcs), the *indegree* $in(v)$ (respectively *outdegree* $out(v)$), is the number of edges entering v (respectively leaving v). An *Eulerian chain* (respectively *Eulerian cycle*) is defined to be a chain (a cycle) that uses each edge exactly once. This brings us to a basic theorem (dated back to Euler, 1766) which establishes the necessary and sufficient conditions for a multigraph to possess an Eulerian chain (or an Eulerian cycle).

Theorem 2: A multigraph G has an Eulerian chain if, and only if, it is connected and the number of vertices of odd degree is 0 or 2.

Berge (1973) presents a proof based on an inductive argument which can be viewed as a constructive (algorithmic) procedure for designing an Eulerian path or cycle when there is one.

Proving that the CPP is in $\mathcal{P}$ requires demonstrating that it can be solved in polynomial-time. We have already stated this fact in the Introduction, however we pursue it further by presenting a simple algorithm which traces an Eulerian cycle on an Eulerian graph in $\mathcal{O}(|E|)$ time.

Eulerian Cycle Algorithm
INPUT: Eulerian multigraph $G = (V, E)$
Step 1: Select a node $x_0 \in V$ (any node in V will do).
Step 2: Construct a 'long' cycle from x_0 to itself. By 'long' it is implied that when moving to a next node in this 'long' cycle you do not go back to x_0 if there is still another reachable node. Denote all the edges in this cycle by E_{x_0} and all the nodes by V_{x_0}. If $E \setminus E_{x_0} = \emptyset$, STOP.
Step 3: Let $G_{x_0} = (V, E \setminus E_{x_0})$ be the graph obtained by deleting all the edges already traversed in the cycle. Select a node $x_1 \in V_{x_0}$ of degree $d(x_1) \geq 2$ in G_{x_0} (any node will do). Clearly such node exists since G is an Eulerian multigraph and there are still untraversed edges in G_{x_0}. Repeat Steps 2 and 3 with $x_i, i = 1, \ldots$

Clearly, the number of steps in this cycle construction is no more than $\mathcal{O}(|E|)$, thus the time-complexity of constructing an Eulerian cycle on a multigraph, which is not necessarily Eulerian, is dominated by the time-complexity of transforming any given graph into an Eulerian multigraph by solving what is called a matching problem (see Derigs, this book).

Note that the algorithm above can be implemented almost without change for directed multigraphs as long as such multigraph satisfies the directed Eulerian cycle conditions which are simply stated in terms of indegree being equal to the outdegree at each node. Transformation of a general multidigraph into an Eulerian multidigraph can be obtained by solving an auxiliary transportation problem over the nodes of the graph which are unbalanced in terms of indegree and out degree. Solving this auxiliary problem optimally (i.e., adding a least cost arc solution to obtain an Eulerian multidigraph) takes no more than $\mathcal{O}(|V|^3)$ time.

The above discussion of solutions for the CPP on undirected graphs and the CPP on directed graphs (digraphs) subjugates the complexity analysis for the two problems to the complexity of the corresponding matching problem for the undirected CPP and the complexity of the corresponding transportation problem for the directed CPP. Since the polynomial time solvability for the matching and transportation prob-

lems is well established (Derigs, this book) and is dominated by $\mathcal{O}(|V|^3)$, it establishes the well known fact that the CPP in both cases belongs to the class $\mathcal{P}$.

So far we restricted our discussion to either directed or undirected graphs (multigraphs). A more general case is that of a *mixed* multigraph. This is a multigraph with some edges – undirected pair of incident nodes, and arcs – a directed pair of incident nodes. Ford and Fulkerson (1974) present a necessary and sufficient condition for the existence of an Eulerian cycle in a mixed graph (true also for a mixed multigraph), which is stated as follows:

Theorem 3: (Ford and Fulkerson, 1974, Theorem 7.1) A mixed simple graph $G = (V; E, A)$ (where E is the set of edges and A is the set of arcs) contains an Euler cycle if and only if (a) G is connected; (b) every node of G is incident with an even number of arcs; (c) for every $X \subseteq V$, the difference between the number of directed arcs from X to $\bar{X}$ (the complement of X) and the number of directed arcs from $\bar{X}$ to X is less than or equal the number of undirected edges joining X and $\bar{X}$.

3. NP-HARD GENERALIZATIONS OF THE CPP

In this section we examine, starting with the CPP on a simple mixed graph, a number of a simple generalizations of the classical CPP which are all $\mathcal{NP}$-hard.

3.1. THE MIXED CPP

Given a general mixed graph $G = (V; E, A)$ which does not contain an Euler cycle, the *mixed chinese postman problem* (MCPP) is that of constructing a cycle on G which traverses every arc and every edge of G at least once while respecting the direction of each arc (Euler cycle on a corresponding multigraph). This Euler cycle requirement implies (1) transforming the graph $G = (V; E, A)$ (by duplicating arcs and/or edges of G) into a minimal cost multigraph $\hat{G} = (V; \hat{E}, \hat{A})$ which contains an Euler cycle, and (2) constructing an Euler cycle by assigning a "proper" directions to the edges in $\hat{E}$. The quandary with the MCPP is that constructing a least cost solution for this problem in the general case (i.e., selecting the 'optimal' traversal direction for the undirected arcs) is known to be $\mathcal{NP}$-hard (Papadimitriou, 1976).

As it is stated in Garey and Johnson (1979), the decision version for the chinese postman problem for mixed graphs is:

INSTANCE: Mixed graph $G = (V; E, A)$, where A is a set of directed edges and E is the set of undirected edges on V, length $l(e)$ a nonnegative integer for each $e \in A \cup E$, positive integer bound B.

QUESTION: Is there a cycle in G that includes each directed and undirected edge at least once, traversing directed edges only in the specified direction, and that has total length no more than B?

Papadimitriou (1976) provides a very clever proof that the above decision version of MCPP is $\mathcal{NP}$-complete by transformation from 3SAT (*three satisfiability problem*). We will outline below only a part of the Papadimitriou's proof to illuminate the main idea of the proof. In addition, we present a much shorter new original proof for the above $\mathcal{NP}$-completeness result. Papadimitriou (1976) also proves that the MCPP remains $\mathcal{NP}$-complete even if all edges and arcs have equal length, G is planar, and the maximum vertex degree is 3. We will restate these results as well.

3.2. THE MCPP NP-COMPLETENESS

Papadimitriou's proof for $\mathcal{NP}$-completeness of the MCPP is based on transformation from 3SAT in which he first provides a mixed graph representation for each variable in a 3SAT problem instance. Figure 4.1 (the unmarked arcs have a cost of 1 each) presents a special mixed graph for which an optimal chinese postman solution has a cost of 2 (on top of the sum of costs of individual edges and arcs added exactly once). This can be proven by enumerating all possible orientations for the undirected edges. In addition, in any optimal traversal of this graph "either both edges $(0,1),(0,3)$ enter 0 and both edges $(0,3),(0,4)$ leave 0, or vice versa." (Papadimitriou, 1976). This essentially establishes the option of assigning the value 0 or 1 to each variable in 3SAT instance and obtaining the optimal traversal path corresponding to such an assignment by either entering into node "0" or out of node "0". This optimal traversal result for the graph in Figure 4.1 carries over to the Figure 4.2 (a) graph and its symbolic representation in Figure 4.2 (b). The claim is that if a mixed graph contains m copies of the graph C, and each copy represents one occurrence of a variable in 3SAT, then the optimal traversal of such graph will have a cost of at least $2m$ (above the constant traversal cost when counting each edge and arc once). The other important details for constructing the appropriate mixed graph by connecting copies of the C graphs for each instance of 3SAT are outlined in Papadimotriou (1976). What is clear is that this is a polynomial transformation in terms of the number of steps and that the MCPP is a member of $\mathcal{NP}$ class. However, when examining the details of this transformation, it calls for an optimal MCPP solution each time a variable is 'traversed'. The number of such

calls equals to the number of variables in the 3SAT instance. Thus, this proof of $\mathcal{NP}$-completeness perhaps might be more appropriately classified as a proof by polynomial time reduction. For more details see Papadimitriou (1976).

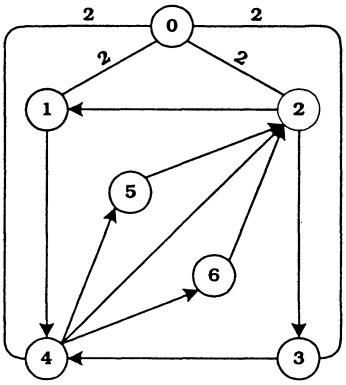


Figure 4.1 Mixed graph representing each variable in the 3SAT.

It has already been noted in the original proof of Papadimitriou (1976), the transformation of an $\mathcal{NP}$-complete (3SAT or the CO) problem to an instance of MCPP requires only two different costs for the arcs and edges of such MCPP instance. Thus, restricting the arc costs to be either 0 or 1 does not make the MCPP any easier. But an even more restrictive version of the MCPP is $\mathcal{NP}$-complete. In Papapdimitriou (1976),

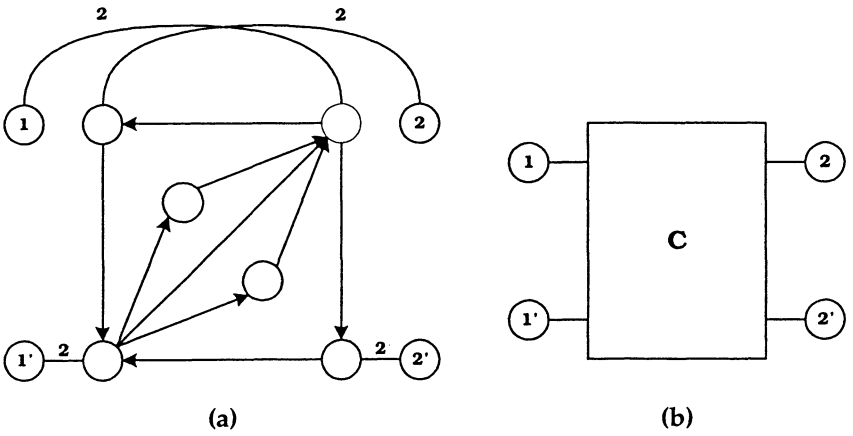


Figure 4.2 Symbolic mixed graph representation of each variable in the 3SAT.

it is proven that the MCPP remains $\mathcal{NP}$-complete even if the underlying graph is planar with total degree of nodes at most 3 and cost of all edges equal to 1. This is accomplished by modifying the MCPP instance obtained from the 3SAT transformation to admit planarity conditions, costs of 1 only, and node degrees which do not exceed 3.

Related to the MCPP is the problem called the Minimum Eulerian Graph (MEG) problem. This problem can be stated as follows: Given a mixed graph $G = (V; E, A)$, find a minimal cardinality multiset D of arcs such that the multigraph $\hat{G} = (V; E, A \cup D)$ is Eulerian. Since the emphasis in the MEG is on the cardinality of the multiset D, this problem can be expressed in terms of finding cardinality matching on an appropriately constructed bipartite graph. This task of finding cardinality set D can be accomplished in polynomial time (Derigs, this volume, and Papadimitriou, 1976). Clearly, the algorithm for MEG can be used to test if there is a covering of the graph by a single chain or a circuit (test for a *unicursal* graph).

For the sake of completeness, we also include an integer programming (IP) formulation for the MCPP problem based on the formulation given in Ralphs (1993) (see also the IP formulations in the chapters of this book by Eglese and Letchford, and by Benavent, Corberan, and Sanchis).

Let y_a be the number of additional copies of each arc $a \in A$ and denote by A_f and A_r the sets of opposite directions for an orientation of each edge $e \in E$. One must choose one orientation for each edge. Let u_e^r and y_e^r denote the first orientation of e and the additional copies of this orientation respectively. Similarly for u_e^f, y_e^f. $I(i)$ and $O(i)$ denote the arcs directed into or out of node i.

$$\min \sum_{a \in A} c_a + \sum_{a \in A \cup A_f \cup A_r} c_a y_a + \sum_{a \in A_f} c_e u_e^f + \sum_{a \in A_r} c_e u_e^r$$

subject to

$$u_e^f + u_e^r \geq 1, \forall e \in E, \tag{4.1}$$

$$x_a = 1 + y_a, \forall a \in A, \tag{4.2}$$

$$x_a = u_e^f + y_e^f, \forall e \in A_f, \tag{4.3}$$

$$x_a = u_e^r + y_e^r, \forall e \in A_r, \tag{4.4}$$

$$\sum_{a \in O(i)} x_a - \sum_{a \in I(i)} x_a = 0, \forall i \in V, \tag{4.5}$$

$$y_a, y_e^r, y_e^f \geq 0 \text{ and } integer, \forall e, a, \tag{4.6}$$

$$u_e^f, u_e^r \in \{0, 1\}, \forall e \tag{4.7}$$

So far we have examined the notion of computational complexity and in this context have established the fundamental result for arc routing that the MCPP is $\mathcal{NP}$-complete. In the remaining sections of this chapter we examine different variants of arc routing problems and classify them as either belonging to the $\mathcal{NP}$-complete class or solvable in polynomial time (i.e., in $\mathcal{P}$ class).

3.3. THE RURAL POSTMAN PROBLEM

A generalization of the Chinese Postman Problem introduced by Orloff (1974, 1976) and usually referred to as the *Rural Postman Problem* (RPP) can be stated as follows: Given a graph $G = (V = V_1 \cup V_2; E = E_1 \cup E_2)$ (where V_1 are the nodes for the multiset of edges E_1 and V_2 are the nodes of the multiset of edges E_2) together with a nonnegative cost real function on the sets E_1, E_2, construct a minimal cost Euler cycle which traverses at least once the edges in E_1. The edges in E_2 can be traversed if the solution so requires. The key point in the RPP is that the subgraph induced by E_1 need not be connected, however the graph G is a connected graph.

The $\mathcal{NP}$-hardness of the RPP, shown by Lenstra and Rinnooy Kan (1976), follows by a simple reduction from the symmetric TSP: any TSP instance can be converted into an RPP instance by replacing each vertex into two identical vertices connected by a required edge of zero cost.

However, there is a sense in which the complexity of the RPP is related to the number of connected components in the subgraph of G induced by E_2. When there is only one such component, the RPP reduces to the CPP, which as we have seen is well-solved. Moreover, Frederickson (1979) suggested a recursive algorithm for the RPP which is exponential only in the number of these components.

Note that, if the connected components in an RPP instance are 'far enough away' from each other, then in any optimal RPP solution one would traverse completely one component before moving to another component. This 'Clustered Rural Postman Problem' has been recently examined by Dror and Langevin (1997), where a solution methodology is

proposed which transforms the problem into what is called the *Generalized Traveling Salesman Problem* (GTSP). The GTSP assumes that the nodes of a given graph have been grouped into mutually exclusive and exhaustive nodes sets and the objective is to find a minimum cost cycle which includes exactly one node from each node set. Like the Frederickson (1979) approach, this algorithm is exponential only in the number of connected components induced by E_2.

One special version of the RPP with applications in manufacturing and elsewhere is the so-called *Stacker Crane Problem* (SCP). In this problem the graph G is a mixed graph $G = (V, A \cup E)$, and the set of arcs which has to be traversed is the entire set A. This problem has been proven $\mathcal{NP}$-hard using a transformation from a TSP to an instance of the SCP by Frederickson et al. (1978). The SCP has been addressed successfully in the context of printed circuit board assembly under the label of directed RPP (DRPP) (Ball and Magazine, 1988). For the integer programming formulations of the DRPP and URPP (undirected RPP) see Ball and Magazine (1988), Corberan and Sanchis (1994), and Assad and Golden (1995).

3.4. THE WINDY POSTMAN PROBLEM

In the classical CPP the cost of traversing a 'street' in one direction is assumed to be equal to the cost of traversing the same 'street' in the opposite direction. However, if in a CPP instance there is a cost distinction based on the traversal direction of an edge e (i.e., $c_{ij} \neq c_{ji}$ for some of the edges $e \in E$), then the corresponding CPP becomes what is known as the *Windy Postman Problem* (WPP). This problem was first considered by Minieka (1979), and subsequently proven $\mathcal{NP}$-hard by Guan (1984) by a simple reduction from the MCPP. In fact, the CPP, the DCPP, and the MCPP, can all be considered as special cases of the WPP. Win (1989) provides an interesting description of the WPP polyhedron based on the following integer programming formulation for the WPP.

$$\min \sum_{e=(i,j)\in E} (c_{ij} + c_{ji})$$

subject to

$$x_{ij} + x_{ji} \geq 1, \forall e = (i,j) \in E, \tag{4.8}$$

$$\sum_{j\in N(i)} (x_{ij} - x_{ji}) = 0, \forall i \in V, \tag{4.9}$$

$$x_{ij}, x_{ji} \geq 0, \forall e = (i,j) \in E \tag{4.10}$$

$$x_{ij}, x_{ji} \in \{0,1\}, \forall e = (i,j) \in E \tag{4.11}$$

where x_{ij} counts the number of times the edge $e = (i,j)$ is traversed from i to j, and $N(i)$ denotes the set of nodes j adjacent to node i by an edge $e = (i,j) \in E$.

In case the underlying undirected graph $G = (V,E)$ is Eulerian, but the costs of some edges are 'windy', then another related problem is mentioned in the literature by Guan and Pulleyblank (1985). The problem asks for a minimum cost traversal of G by an Eulerian tour which traverses each edge *exactly once*. Guan and Pulleyblank (1985) refer to this problem as the *minimum cost Eulerian orientation problem* and describe a polynomial time solution based on a transformation to a minimum cost circulation problem. Thus, the WPP on an Eulerian graph with the condition that the solution has to be an Eulerian tour is in $\mathcal{P}$. Another case of polynomially solvable WPP discovered by Guan (1984), is for graphs G in which the cost of a cycle (any cycle) does not change if the direction of cycle traversal is reversed.

3.5. NON-INTERSECTING EULERIAN CIRCUITS AND A-TRAILS IN EULERIAN GRAPHS

If, in the course of constructing an Eulerian circuit on a graph which is Eulerian, one asks the circuit to conform to some additional conditions or restrictions, the known polynomial time bound on such a circuit construction effort might be jeopardized. For instance consider a graph $G = (V,E)$ for which the edges incident to a vertex $v \in V$ have been ordered ('modulo $d(v)$') in a 'clockwise' order for every such v. Two edges are defines as *neighbors* at v if they are consecutive in the order. For a planar graph G the neighbors of an edge are the edges adjacent to it in some face of the graph. A *non-intersecting* path or circuit in a planar G is defined as one in which every two consecutive edges (v_i, v_j) and (v_j, v_k) in it are neighbors in v_j. Here we present a result due to Bent and Manber (1987), which states that the problem of deciding if a non-intersecting Eulerian path or circuit in a planar graph G exists is an $\mathcal{NP}$-complete problem. The reduction used by Bent and Manber (1987) to prove this result, reduces SAT (satisfiability) to an instance of a planar Eulerian graph in two stages. First, it uses a result proven by Lichtenstein (1982) that for every conjunctive normal form F with its associated graph $G(F)$ can be converted in polynomial time to a conjunctive normal form F' for which its graph $G(F')$ is planar. The proof then follows with a transformation of $G(F')$ to an instance of a

planar Eulerian graph for which an Eulerian circuit corresponds to the determination of satisfiability for F' and subsequently for F. This transformation is rather lengthy and we direct the interested reader to the original paper.

Anderson and Fleischner (1995) extended the above $\mathcal{NP}$-completeness result of Bent and Manber (1987) to the problem of deciding the existence of what are called A-trails in a subfamily of Eulerian graphs. Anderson and Fleischner (1995) have proven a number of other complexity results not mentioned here and outside the scope of this book since they relate to spanning trees in hypergraphs. However, we first need to define an A-trail.

Given a planar representation of an Eulerian graph G, an Eulerian circuit of G is called an *A-trail* if and only if consecutive edges of the circuit, say (v_{i-1}, v_i) and (v_i, v_{i+1}) are always neighbors in the cyclic ordering of the edges incident with v_i defined by a clockwise order in the plane representation. This definition of an A-trail taken from Anderson and Fleischner (1995) seems to coincide with the definition used in Bent and Manber (1987) of a non-intersecting Euler circuit in a graph G. To state the result of Anderson and Fleischner (1995) we need a definition of an n-connectivity for graph G which is taken from Fleischner, (this book).

Definition: Given two non-adjacent vertices $x, y \in G$, the *local connectivity* of x, y is the smallest number of vertices $v_1, \ldots, v_k$ such that $G' = G - \{v_1, \ldots, v_k\}$ is disconnected and x, y are in different components (connected subgraphs) of G'. A loopless graph G (no edges (v, v)) has connectivity n (is *n-connected*) if G contains a spanning subgraph isomorphic to K_{n+1} (complete graph with $n+1$ vertices).

Theorem 5 : (Anderson and Fleischner, 1995) Given a 3-connected planar Eulerian graph G having only 3-cycle and 4-cycle face boundaries, the problem of establishing the existence of an A-trail on G is $\mathcal{NP}$-complete.

The proof of Theorem 5 is based on transformation to an instance of a 3-connected planar Eulerian graph G having only 3-cycle and 4-cycle face boundaries, from a 3-connected, planar cubic graph G', and the question of existence of a Hamiltonian circuit on G'. Establishing the existence of a Hamiltonian circuit on G' guarantees the existence of an A-trail on G, and vice versa. For details of this transformation see Anderson and Fleischner (1995), where even stronger results are obtained.

3.6. DOMINATING TRAILS

In the previous subsection we defined a concept of an A-trail on a graph G. In this subsection we focus on yet another arc traversal related concept – a dominating trail on an undirected graph $G(V, E)$. This subsection is based on a recent work of Agnetis et al. (1999) in the context of coordinating machine set-ups for a two stage flow-shop scheduling in a manufacturing application.

A *trail* (Fleischner, this book) is defined as a not necessary simple circuit in G (i.e., might visit same nodes more than once but is not allowed to pass through any edge more than once). A *dominating trail* $\mathcal{P}_d$ is a trail in G such that each edge $e \in E$ is either a member of $\mathcal{P}_d$ or is incident to a node in $\mathcal{P}_d$. In this sense, a dominating trail "covers" all the edges of E.

The dominating trail in a graph G is related to the existence of a Hamiltonian circuit in a line graph of G denoted as $L(G)$. The line graph $L(G)$ is defined as a graph whose vertex set V' corresponds to a bijection from the edge set E, and two vertices in $L(G)$ are joined by an edge whenever the corresponding edges in G are adjacent. It is proven in Harary and Nash-Williams (1965) that a graph G has a dominating trail if and only if $L(G)$ is Hamiltonian.

The theorem proven in Agnetis et al. (1999) and reproduced below states that establishing the existence of a dominating trail on a bipartite graph is an $\mathcal{NP}$-complete problem.

DOMINATING TRAIL ON BIPARTITE GRAPH - (DTBG)
Instance: Given a bipartite graph $B = (S, T, E)$ (the vertex sets S and T and the edge set E).

Question: Is there a dominating trail?

Theorem 6: The DTBG problem is $\mathcal{NP}$-complete.
Proof: The proof is by transformation from a Hamiltonian circuit problem on cubic graphs stated as follows: Given a 3-regular graph $G = (V, E)$, is there a Hamiltonian circuit on G?

The Hamiltonian circuit problem on cubic graphs is know to be $\mathcal{NP}$-complete (Garey and Johnson, 1979), and it is clear that the DTBG problem is in $\mathcal{NP}$.

Given an instance of a 3-regular cubic graph G, we obtain a corresponding instance bipartite graph $G^b = (S, T, E')$ as follows: The set

S of vertices in G^b corresponds to the set V in G. The set of vertices T (denoted by v_e) corresponds to the set of "mid" points of each edge $e = (i, j) \in E$. Now, each vertex $v_e \in T$ is connected by an edge to the corresponding i and j vertices in S. This set of edges constitutes the set E' in G^b.

What remains is to show that if there is a dominating trail in G^b then there exists a Hamiltonian circuit in G and vice versa. The proof arguments are straight forward. For instance, consider a dominating trail P^d in G^b and a vertex $i \in S$. Since there are three edges incident to i and the three edges are "covered" by P^d, the vertex i has to belong to P^d (otherwise the edges are not "covered"). Thus, all vertices in S are on the trail P^d. On the other hand, no vertex occurs in P^d more than once because the degree of each vertex is 3. Thus, the dominating trail P^d visits the vertices in S exactly once which corresponds to a Hamiltonian circuit on G when the vertices in T are ignored. In the other direction, if there exists a Hamiltonian circuit in G, such a circuit leads to a dominating trail on G^d in a straight forward manner. The optimization version of this problem is that of constructing a dominating trail of minimal cost. Since it seems that the minimal cost dominating trail problem has not been examined in great detail (to our knowledge), we do not know of any approximation solutions for this problem.

3.7. PRECEDENCE IN ARC ROUTING

Consider an undirected connected graph $G = (V, E)$ together with a nonnegative real function $C : E \rightarrow \mathbf{R}_+$ which communicates the cost of traversing an arc in E. Assume that the edges in E correspond to a city streets in a place like "Buffalo, NY" where the winters bring a lot of snow and the roads and streets are divided into classes of 'importance' in terms of clearing them of snow. In terms of the graph G, this implies class precedence for traversals. Simply stated, the major roads should be cleared before the secondary roads, which have precedence over residential streets, etc. A similar kind of traversal precedence occurs when constructing a torch path in flame cutting of metal plates (Manber and Israni, 1984). This precedence arc traversal setting was analyzed in Dror et al. (1987), and we restate here some of the results.

Formally, let $\{E_1, E_2, \ldots, E_K\}$ be a partition P_K of the set of edges E, $(E_i \subseteq E, 1 \leq i \leq K, E_i \cap E_j = \emptyset, i \neq j, \cup_{i=1}^{K} E_i = E)$. Partial order $\prec_{P_K}$ (or simply $\prec$ if the implication is clear) of the partition P_K implies that in a CPP solution for the graph G the edges in a set E_i are traversed before any of the edges in E_j if $E_i \prec E_j$ in $\prec_{P_K}$. Clearly, there must be some traversal conditions (appropriate connectivity, etc.) satisfied for the partition P_K for this to be possible. However, the precedence

$\prec_{P_K}$ does imply that if an subset E_i has already been traversed and one constructs a traversal of E_j (of course $E_i \prec E_j$ either immediately or through the transitive closure of $\prec_{P_K}$), then in traversal of E_j one can reuse edges from E_i. Following Dror, et al. (1987), we denote by F_k as the subgraph of G induced by the union of edge subsets $E_1, E_2, \ldots, E_k$. In this case there exists a Euler cycle on graph G which respects the partial order of partition P_K if and only if the graphs $F_k, k = 1, \ldots, K$ are connected for $1 \leq k \leq K$. In the case graph G is a directed graph, one has to require that the sequence of subgraphs $F_k, k = 1, \ldots, K$ be strongly connected in order to assure the existence of an Euler cycle which satisfies the partial order P_K.

In the case that the partial order $\prec_{P_K}$ represents a chain (a set of pairwise comparable elements) over the partition P_K, and each of the subgraphs G_i induced by the corresponding subset of edges E_i is connected, then the optimal Euler cycle on G which satisfies the precedence relation $\prec_{P_K}$ can be constructed in polynomial time (Dror, et al. 1987). The time complexity of the polynomial time solution presented in Dror et al. (1987) is that of $\mathcal{O}(|V|^5)$ based on K-partite graph construction which requires calls to a matching subroutine, followed by a shortest path algorithm on that K-partite graph. In Ghiani and Improta (2000) a somewhat different polynomial time procedure has been proposed with time complexity of $\mathcal{O}(K^3|V|^3)$, which for values of K ($< |V|^{2/3}$) dominates the procedure of Dror et al. (1987).

In the case the precedence relation $\prec_{P_K}$ represents a general partial order, even if the subgraphs $G_i, i = 1, \ldots, K$ are connected, to construct a minimal cost Euler cycle on G which respects $\prec_{P_K}$ is $\mathcal{NP}$-hard. This was proven in Dror et al. (1987) by transformation from the Traveling Salesman Problem (TSP). The outline of the transformation for a TSP path between nodes s_1 and s_2 into a chinese postman path is presented in Figure 4.3 (a) and (b). Each node in the original graph (which a complete graph) is replaced by two nodes and an edge connecting the two nodes. Such an edge constitutes a single precedence class and has a 'high' traversal cost of M. Add to this graph all edges between the nodes in the original graph as illustrated in Figure 4.3 (b) except for the two extreme nodes for the TSP path nodes. The cost of those edges is the same as that in the original nodes. Now the instance of precedence relation for edge sets is setup in such a way that the set E_1 has to be traversed first before traversing the set of edges with the costs as in the original graph. All the other edges can be traversed in any order latter with the edge corresponding to the 'last' node in the TSP path to be traversed last. The difference between the optimal Chinese Postman path solution between nodes s_1^1 and s_2^1 in the transformed graph given the

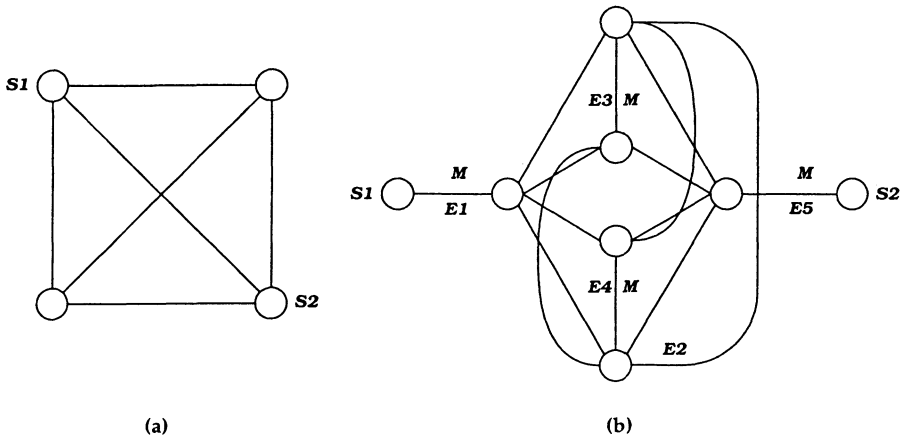


Figure 4.3 The original complete graph (a) and the transformed graph (b).

precedence relation and the optimal TSP path solution between s_1 and s_2 is $KM + A$ where A is the total cost of the arcs in the original graph.

Note however, that in the case where $\prec_{P_K}$ represents for instance a tree precedence relation, it is not known if the minimal cost Euler cycle problem is $\mathcal{NP}$-hard.

The Euler cycle construction in the case of directed graph G, given a partition P_k together with precedence $\prec_{P_K}$, and even assuming connected subgraphs $G_k, k = 1, \ldots, K$, is less clear in terms of existence of feasible solutions and construction of optimal solutions. For chain precedence relation $\prec_{P_K}$ and strongly connected subgraphs $G_k, k = 1, \ldots, K$, the polynomial procedure in Dror et al. (1987) can be extended to cover this case. However, strong connectivity of the subgraphs might not be necessary for the optimal solution and there is not much more we can contribute on this.

3.8. CAPACITATED ARC ROUTING

What happens if we introduce the notion of pickup quantity (or delivery quantity) in the context of edge traversal? This only matters if the traversing 'vehicle' has a finite collection (or delivery) capacity. For instance, in the case of a directed graph $G = (V, A)$, with pickup quantity $q_a \geq 0$ associated with each arc $a \in A$, together with a pickup limit of $Q > 0$, the *capacitated arc routing problem* (CARP) is the arc-routing analogue of the classical *vehicle routing problem* (VRP).

q_{ij} = the demand along arc $(i,j) \in R \subseteq A$,
W_v = the capacity of vehicle v,
c_{ij} = the distance *(length)* of an arc $(i,j) \in A, (c_{ij} \geq 0, \forall (i,j) \in A)$,
V = the upper bound on the number of vehicles,
x_{ijv} = the number of times vehicle v traverses the arc $(i,j) \in A$,
y_{ijv} = a binary variable which takes the value 1 if vehicle v discharges the feed along the arc $(i,j) \in R$, and takes the value 0 otherwise.

$$(CARP): \qquad \min \sum_{(i,j)\in A} \sum_{v=1}^{V} c_{ij} x_{ijv}$$

subject to

$$\sum_{k\in N} x_{kiv} - \sum_{k\in N} x_{ikv} = 0, i \in N, v = 1, 2, \ldots, V, \tag{4.12}$$

$$\sum_{v=1}^{V} y_{ijv} = 1, \forall (i,j) \in R, \tag{4.13}$$

$$\sum_{(i,j)\in R} q_{ij} y_{ijv} \leq W_v, \; v = 1, \cdots V, \tag{4.14}$$

$$x_{ijv} \geq y_{ijv}, \; \forall (i,j) \in R, \tag{4.15}$$

$$M \sum_{i\notin S, j\in S} x_{ijv} \geq \sum_{(j,k)\in A[S]\cap R} x_{jkv}, \begin{cases} \forall S \subseteq N, 1 \notin S, \\ A[S] \cap R \neq \emptyset, \\ v = 1, \ldots, V, \end{cases} \tag{4.16}$$

$$y_{ijv} \in \{0,1\}, \forall (i,j) \in R, v = 1, \ldots, V, \tag{4.17}$$

$$x_{ijv} \in Z^+, \forall (i,j) \in A, v = 1, \ldots, V, \tag{4.18}$$

where M is a large constant no smaller than the total distance of any circuit that includes all arcs in R, and $V[S]$ is the set of nodes incident to at least one arc in S.

The objective function represents the total distance traveled by all the vehicles. Note that arcs can be traversed more than once. The first set of constraints is the common 'flow conservation' constraints for network-flow formulations. The second set of constraints require that at least one traversal is made of each of the arcs in R. The third set of constraints are the capacity constraints for the vehicles. The next set of constraints require that vehicle v traverse the arc $(i,j) \in R$ if it delivers the demand to this arc. The fifth set of constraints are subtour-elimination constraints which ensure that each trip include the depot. Note that

this formulation of the CARP (taken from Dror and Leung, 1998) not only has different subtour-elimination constraints than the one given in Golden and Wong (1981) or Assad and Golden (1995), but also the x_{ijv} variables have a different interpretation. In addition, the interpretation for the y_{ijv} variable is in terms of a "fraction" of service delivered to edge (i,j) by the vehicle v. In the model above the the y_{ijv} variables are restricted to binary values. Thus, a vehicle either services an edge or does not. Later we examine the option of a partial service by a vehicle (see Dror and Langevin, this book).

Clearly, the CARP is a strongly $\mathcal{NP}$-hard problem by simple reduction from the bin-packing problem. One can setup an instance of a capacitated edge routing problem by taking one node to represent the depot (node 0) and a node for each bin packing item connected in a star structure to one additional node. The demand on each of the star edges is equal to the 'item' size from the bin packing problem (Garey and Johnson, 1979). The demand on all the other edges is set to zero. Set the distances between the depot and one of the "item" nodes to 1 and the distances of the star edges to zero. Also connect each consecutive pair of 'item' nodes by a zero cost edge (the periphery of the star). The optimal CARP solution for such graph corresponds to an optimal bin packing solution and vice versa. A different proof of $\mathcal{NP}$-hardness for the CARP is obtained after a polynomial graph transformation from an edge traversal problem to an equivalent node routing problem (see Dror and Langevin, this book). In that case the $\mathcal{NP}$-hardness reduction is from the classical vehicle routing problem. In either case, the CARP is $\mathcal{NP}$-hard for edge traversals (undirected graphs) or directed graph traversals.

4. APPROXIMATION ALGORITHMS

Approximation algorithms for hard ($\mathcal{NP}$-hard) combinatorial optimization problems have played an important part with regards to the examination and analysis of computational complexity for such problems. The informal question asked in this context is how close can we get in a 'provable way' to an optimal solution of a given $\mathcal{NP}$-hard problem while expending a modest amount (i.e., polynomial time) of computational resources? An approximation algorithm for a problem is evaluated in the worst case sense. I.e., what is the worst possible deviation from optimum taken over all problem instances? The deviation from optimum is measured in terms of δ ratio. More precisely, an algorithm $\mathcal{A}$ is said to be a δ-approximation for a minimization problem ($\delta > 1$) if for all instances of this problem the algorithm generates a solution within δ times the optimum value. In a similar manner, for maximization problems the δ value (< 1) assures that the solutions generated by such

algorithm will not be less than δ times the value of the optimal solution. The corresponding δ value is referred to as *performance guarantee* of the approximation algorithm $\mathcal{A}$. A family of algorithms which trades-off solution 'closeness' (performance guarantee) against running time is referred to as an *approximation scheme.* Following Hochbaum (1997), Arora and Lund (1997), we introduce a number of formal definitions of the concepts related to approximation algorithms.

Definition: A polynomial time algorithm $\mathcal{A}$ for a minimization problem P, is said to be a δ-approxi- mation algorithm ($\delta > 1$) if for every problem instance I of P the algorithm generates a solution which is never more than $\delta \times OPT(I)$, where $OPT(I)$ denotes the value of the optimal solution for the instance I of P. In other words, if $\mathcal{A}(I)$ denotes the solution value for instance I generated by $\mathcal{A}$, then $\delta \times OPT(I) \geq \mathcal{A}(I)$ for all instances I of P. Similarly for maximization problems.

Definition: The *absolute performance ratio* $R_{\mathcal{A}}$, of an approximation algorithm $\mathcal{A}$ is $R_{\mathcal{A}} = inf\{r \geq 1 | R_{\mathcal{A}}(I) \leq rOPT(I), \forall I\}$.

Definition: The *asymptotic performance ratio* $R_{\mathcal{A}}^{\infty}$ for $\mathcal{A}$ is, $R_{\mathcal{A}}^{\infty} = inf\{r \geq 1 | \exists n \in \mathbb{Z}^{+}, R_{\mathcal{A}}(I) \leq rOPT(I)\ \forall I, s.t.\ 1exI \geq n\}$.

In some cases, as illustrated in Hochbaum (1997), the difference between the absolute performance ratio and asymptotic performance ratio of an algorithm can be significant. The trade-off between computational time and the performance ratio for a family of approximation algorithms is captured by the two definitions below.

Definition: A family of approximation algorithms $\{\mathcal{A}_{\epsilon}\}$, is called a *polynomial approximation scheme*, if an algorithm $\mathcal{A}_{\epsilon}$ is a $(1+\epsilon)$-approximation algorithm and for a fixed ϵ its computational time is bounded by a polynomial in the length of the problem instance I.

Definition: A family of approximation algorithms $\{\mathcal{A}_{\epsilon}\}$, is called a *fully polynomial approximation scheme*, if an algorithm $\mathcal{A}_{\epsilon}$ is a $(1+\epsilon)$-approximation algorithm and for a fixed ϵ its computational time is bounded by a polynomial in the length of the problem instance I and $1/\epsilon$.

Ideally, one would like for an algorithm to assure 'closeness' to an optimal solution of a hard combinatorial problem within a computation time that is polynomial in the problem size. However, recent results in theory of approximations for $\mathcal{NP}$-hard problems suggest that computing good approximate solutions for many of these problems is just as hard as computing optimal solutions. In other words, the approximation problem of

achieving solutions very close to optimum might be $\mathcal{NP}$-hard problems in their own right. Since we prove an $\mathcal{NP}$-hardness of a problem by reduction from a known $\mathcal{NP}$-hard problem, when proving $\mathcal{NP}$-hardness of an approximation such a reduction produces a gap in the value of the optimum in the sense that the approximation scheme either gets the optimum value or it cannot get closer than a certain factor g times the optimum value. Since in this chapter we are interested in examining the known approximations for hard arc routing problems, we provide a short and informal discussion of $\mathcal{NP}$-hardness of approximations based primarily on Arora (1994), and Arora and Lund (1997). This is usually referred to as *inapproximability results.*

As Arora and Lund (1997) point out, at the present time inapproximability results divide problems into four classes based on the performance ratio that is provably hard to achieve. The performance ratios representing these classes are: $1+\epsilon$ for some fixed $\epsilon > 0$, $\Omega(\ln og)$, $2^{log^{1-\gamma}n}$ for every fixed $\gamma > 0$, and n^{δ} for some fixed $\delta > 0$, (n denotes the input size). In a manner similar to that of Garey and Johnson (1979), Arora (1994) and Arora and Lund (1997) start with a set of six so called "canonical" inapproximability problems to derive by reduction inapproximability results to many other problems of approximation. Inapproximability result implies that achieving a certain performance ratio is $\mathcal{NP}$-hard. It means that if one can prove that a certain polynomial time algorithm achieves the approximation ratio then $\mathcal{P} = \mathcal{NP}$.

A basic canonical problem in inapproximability results is that of MAX-3SAT which is an optimization version of the 3SAT problem. The objective in MAX-3SAT is to find a truth assignment, which maximizes the fraction of satisfied clauses in a given 3CNF (conjunctive normal form with three variables in each clause) formula. Note that given an instance I of 3SAT, MAX-3SAT(I) ≤ 1. Arora and Lund (1997) repeat a proof of the following fundamental theorem.

Theorem 7: There is a fixed $\epsilon > 0$ and a reduction τ from SAT to MAX-3SAT such that for every boolean formula I, if $I \in SAT$, then $MAX-3SAT(\tau(I)) = 1$, and if $I \notin SAT$, then $MAX-3SAT(\tau(I)) < 1/(1+\epsilon)$.

In other words, there is a gap in the optimum value of the objective function of MAX-3SAT depending on whether or not the boolean formula is satisfiable. This implies that achieving a performance ratio of $1+\epsilon$ for MAX-3SAT is $\mathcal{NP}$-hard.

In Arora and Lund (1997), a reduction graph is presented with the top node representing MAX-3SAT followed by MAX-3SAT(5) (each variable appears in exactly 5 clauses) and CLIQUE. The node MAX-3SAT(5) leads to CLASSI, LABEL COVER, and CLASS IV. CLIQUE node leads to COLORING and CLASS IV, and then COLORING node again leads to CLASS IV. In the other direction, LABEL COVER nodes leads to SET COVER and to CLASS III, and from SET COVER we get to CLASS II. This description includes the six canonical problems and inapproximability results for Classes I, II, III, and IV.

The above served only as a very brief outline of inapproximability introduction and results. It is not our intention to pursue this topic further here but to examine the state of approximation results for arc routing problems. Most of such results have been triggered by the early work of Frederickson (1979), which we summarize below.

5. APPROXIMATION RESULTS FOR ARC ROUTING

5.1. THE MIXED CPP

Earlier in this chapter we mentioned the result of Papadimitriou (1976) that the chinese postman problem on a mixed graph (MCPP) is $\mathcal{NP}$-complete, and gave a new and short proof of this fact. The question we ask in this subsection is how close to the optimal solution of the MCPP can we get with a polynomial time algorithm. Edmonds and Johnson (1973) have suggested a heuristic algorithm for which Frederickson (1979) proved a performance ratio of 2. In the same article Frederickson presented three other approximation algorithms for the MCPP. He starts out with an algorithm which constructs a Euler tour in an 'opposite' way to Edmonds and Johnson (1973) with performance ratio of 2 as well. He then combines the two approximation algorithms into a single heuristic with performance ratio of 5/3. In addition, for the case where the mixed graph is planar Frederickson presents a performance ratio 3/2 approximation algorithm. These results were the best known until, very recently, Raghavachari and Veerasamy (1998, 1999b) obtained a 3/2 approximation algorithm for the MCPP without the planarity assumption. We now proceed to outline the main ideas contained in these approximation schemes.

The heuristic solution procedure for MCPP proposed by Edmonds and Johnson (1973) aims at modifying the mixed graph $G = (V; E, A)$ by duplicating and directing edges from E and duplicating arcs from A, in a 'cheapest' manner to obtain the necessary and sufficient conditions for the existence of an Euler cycle on a modified G. To find this 'cheapest'

set of arcs and edges which need to be duplicated is $\mathcal{NP}$-hard. The heuristic starts by constructing a minimal-cost matching solution between the odd degree nodes of G disregarding the arc directions. Copies of the arcs and edges in this matching solution are then added to the graph. This is followed by a procedure which orients some edges and adds copies of arcs to make the indegree of each node equal to its outdegree. The procedure is formulated and solved as a min-cost flow problem followed by Frederickson's 'evenparity' adjustment which does not increase the cost over the min-cost flow solution. The end result is a modified graph which accepts a Euler cycle. The time complexity of this heuristic is $\mathcal{O}(\max\{|V|^3, |A|(\max\{|A|, |E|\})^2\})$. It is easy to see that the performance ratio of this heuristic is 2 since by duplicating all the arcs and edges of G to obtain the graph G_2, all nodes of G_2 are of even degree, and the min-cost flow algorithm provides an optimal solution to the indegree = outdegree adjustment at each node of the modified graph which is no more costly than such an adjustment on the original graph.

Frederickson (1979) noticed that the order of the so called 'matching' and 'min-cost flow' steps can be reversed to form a different heuristic solution process and that this heuristic maintains the same performance ratio of 2 for the MCPP and essentially the same time complexity. However, the two heuristics perform very differently on each others' worst-case examples. Denote by C_M the cost of the arcs in the solution of the min-cost flow problem. Subsequently, the key difference between the two heuristics lies in the fact that for the original Edmonds and Johnson's heuristic the cost of its solution does not exceed $C^* + 2C_M$ (where C^* is the cost of the optimal solution) and the 'reversed' heuristic generates a solution with cost not exceeding $2C^* - C_M$. By using a threshold of $(1/3)C^*$ for C_M one obtains the performance ration of 5/3 when both heuristic are run and the best solution is selected.

Very recently, Raghavachari and Veerasamy (1998, 1999b) improved on the above performance ratio to give a 3/2 approximation algorithm for the MCPP. Raghavachari and Veerasamy's main observation lies in the fact that the lower bound used by Frederickson can be improved by a nonnegative cost C_X which is a cost of minimum weight matching of the undirected graph obtained by shrinking all the arcs of the graph in the output of the min-cost flow algorithm in the Edmonds and Johnson's heuristic. Subsequently, they modify the first (Edmonds and Johnson's) heuristic in the Frederickson scheme and prove that its cost does not exceed $C^* + C_M$ and use a threshold value of $C^*/2$ for C_M to produce the performance ratio of 3/2. The performance ratio of 3/2 for Raghavachari and Veerasamy's heuristic is tight as demonstrated by

Frederickson (1979) examples.

At the present time we are not aware of any work on polynomial approximation schemes or inapproximability results for the MCPP.

5.2. THE WINDY CPP

Earlier in the chapter we defined the Windy CPP as a generalization of the Mixed CPP, where the underlying graph is undirected, but where the traversal cost for each edge varies according to the direction of traversal.

Win (1989) formulated the Windy CPP as a minimum-cost flow problem with side-constraints and showed that the solution to the LP relaxation of this formulation is 'half-integral' (that is, all variables have values which are multiple of one-half). Using this fact he was able to produce a 2-approximation algorithm. This was the best known result until, very recently, Raghavachari and Veerasamy (1999a) devised a 3/2-approximation algorithm.

Raghavachari and Veerasamy use the same strategy as Frederickson (1979) used for the Mixed CPP, in that they describe two different heuristics, each with a performance guarantee of 2, but which achieve a guarantee of 3/2 when used together. These heuristics are quite complicated and the proof of the guarantees are based on some deep structural properties of the LP relaxation of Win. For the sake of brevity, we do not describe them in detail.

5.3. THE RPP AND OTHER VARIANTS

Frederickson (1979) mentions very briefly that the RPP with triangle inequality can be approximated to within 3/2 by modifying the famous 3/2 heuristic for the TSP with triangle inequality, due to Christofides (1976). A more explicit construction was first provided by Benavent et al. (1985) and later independently by Jansen (1992). Jansen's heuristic extends the Christofides' heuristic to the case of RPP with triangle inequality by observing that one can 'collapse' the subgraphs spanned by each of the disconnected subsets of the required edges in E_2 into a single node each. Then, one constructs a graph whose nodes represent the collapsed subgraphs and whose edges represent shortest path links between them and computes a minimum cost spanning tree. Afterwards, one can apply the 'matching and short-cut' algorithm of Christofides to construct a 3/2-approximate RPP solution. For more detailed description of this approximation algorithm see Jansen (1992).

In Dror and Haouari (2000), the concept of generalized combinatorial optimization problems coined after the Generalized Traveling Salesman Problem is extended and a short list of classical combinatorial problems have been recast in this broader framework. One problem on this list is so called the Generalized Chinese Postman Problem. The generalization of the CPP requires traversing at least one edge from each subset of a given edge partition of the edge set E. More formally, assume that the edge set E is partitioned into K subsets $E_1, \ldots, E_K$. The *Generalized Chinese Postman Problem* (GCPP) requires finding a minimum cost tour in G which contains at least one edge from each of the subsets $E_k, k = 1, \ldots, K$.

For motivation, assume that one has only a very limited time for visiting a large museum. The museum is organized in several different departments. If the visitor does not have enough time for fully visiting all the departments he may restrict his tour to include walking at least along one wall from each department.

Obviously, when $|E_k| = 1$ for each k, the GCPP simply reduces to the CPP, and can be solved in polynomial time. However, the generalized case is hard which can be easily shown by equivalence between a special case of GCPP and the Rural Postman Problem (RPP).

We bring up the GCCP in this section on approximation results because for all cases in the list of the generalized combinatorial optimization problems described in Dror and Haouari (2000), the authors did not find 'good' (finite performance guarantee) heuristics. I.e, the GCPP problem is not only $\mathcal{NP}$-hard but also a heuristic solution with guaranteed finite error bound is not known for this problem.

Other interesting approximation results for problems such as the SCP (stacker crane problem) and the k-SCP where k tours must be constructed with the objective of balancing the tours (minimizing the length of the longest tour) can be found in Frederickson et al. (1978). For the SCP they provide a polynomial (quadratic order) algorithm with performance guarantee of 9/5, and for the k-SCP a performance guarantee of $(14/5 - 1/k)$.

5.4. THE CARP

There are essentially two approaches for generating approximation results for arc routing problems. A direct approach would be to examine capacitated arc routing heuristics. The other option is to examine capacitated node routing heuristics since capacitated arc routing problems possess an equivalent node routing representation. Historically, approx-

imation heuristics such as the Christofides result for the euclidean TSP, have been developed before any documented attempts to obtain such results for arc routing problems. This is true for the uncapacitated problem versions and even more so for the capacitated arc routing problems. Perhaps the first description of an approximation result for the CARP on an undirected graph (denoted by CAPP in Assad and Golden (1995)) can be found in Golden and Wong (1981). Moreover, Golden and Wong proved perhaps the first inapproximability result for arc routing problems, more specifically for CARP. They have shown that even 1.5-approximation for CARP is $\mathcal{NP}$-hard. The reduction proof of this result is straight forward from PARTITION and we restate it here for completeness.

INSTANCE: Finite set A and size $s(a) \in \mathbb{Z}^+$ for each $a \in A$.

QUESTION: Is there a subset $A' \subseteq A$ such that $\sum_{a \in A'} s(a) = \sum_{a \in \bar{A}'} s(a)$?

The reduction from PARTITION to CARP is as follows:

Set a single node as the zero node and connect it to another node, say node 1. The distance of this edge is one ($c_{01} = 1$), however this edge has no demand. Now create $|A|$ nodes, one for each member of the set A and connect them with node 1. Each of the edges $(1, i), i = 2, \ldots, |A| + 1$ has a demand equal to $s(a)$ when connected to a node representing the element a. This graph structure is a tree. Now set the capacity $Q = (1/2) \sum_{a \in A} s(a)$. From this construction any nonoptimal solution has a length $= 6$, whereas the optimal solution corresponding to YES answer for the PARTITION problem has a length $= 4$. This demonstrates that constructing an approximation for the CARP with performance guarantee of less than 1.5, is $\mathcal{NP}$-hard.

In Haimovich et al. (1988) a number of approximation results for the VRP are presented. Though the authors assume that points (customers) are located in euclidian space, only the triangular inequality is used to derive their approximation results. They prove that for identical customers their so called *Iterated Optimal Tour Partitioning* (IOTP) heuristic has a performance guarantee of $2 - 1/Q$ given an optimal algorithm for the TSP. However, given an α approximation algorithm for the TSP, the performance guarantee is $1 + (1 - 1/Q)\alpha$. For the case of unequal demands, their IOTP procedure provides a performance guarantee of $3 - 2/Q$ given that they use an optimal TSP procedure in the tour construction. Given an α approximation algorithm for the TSP this bound takes the form of $2 + (1 - 2/Q)\alpha$ (Altinkemer and Gavish, 1987). Since the CARP can be transformed into an equivalent VRP problem and the triangular inequality of the distance matrix is obtained by a simple

shortest path postprocessing, the same approximation performances exist for the CARP. This has been stated more explicitly by Jansen (1993).

In some applications it is interesting to examine CARP on graphs with special structure such as tree. The initial work on tree routing problems can be traced to Labbe et al. (1991).

In the case graph $G = (V, E)$ is a tree (connected graph with $|V| - 1$ edges), the CPP solution simply requires traversing twice each edge $e \in E$. In the case of capacitated arc routing on trees, this routing problem can be trivially transformed to node routing on trees by assigning the entire edge demand to the node 'away' from the route node. Thus, capacitated arc routing on trees and capacitated node routing on trees are in fact equivalent problems.

6. CONCLUSIONS

In this chapter we have presented an abbreviated overview of computational complexity theory and attempted to provide the reader with a guided tour of the different arc routing problems in terms of their complexity classification. When faced with any computational problem, in particular, with a combinatorial optimization problem, one ought to estimate a priori the 'computational effort' required for generating a solution for such a problem. In this chapter, the examination of combinatorial optimization problems is limited to arc routing. Following on the foot-steps of Edmonds and Johnson's (1973) work, many professionals working on routing problems, tend instinctively to view arc routing problems as being not as hard as node routing. The existence of polynomial time solutions for the basic CPP (directed or undirected) are at the core of such an instinct and in diametric contrast to non-known existence of polynomial time solutions to the basic TSP. However, as proven by Papadimitriou (1976), in his path breaking paper "On the complexity of edge traversing", basic arc routing problems can be just as hard as node routing. In a sense, arc routing on a mixed graph, is even harder since the restriction of node degrees to at most three, graph planarity, and 0-1 arc/edge costs, still do not make the MCPP any easier.

Some arc routing problems are easy. Building an Eulerian cycle on undirected graph with all vertices of even degree is easy (Section 2). The case of completely directed graph of all even degree nodes with in-degree equal to the out-degree for each node is also easy. Transforming a general undirected graph into a graph with all even degree nodes in a cost minimizing manner requires a little more sophistication and more computational effort (about $(|V|^3)$. The same can be said for the completely directed case. Beyond that, if the graph does not have an 'easily' recog-

nizable 'simple' structure, for instance a tree, arc routing problems are likely to be $\mathcal{NP}$-hard and therefore just as difficult to solve as any other difficult combinatorial optimization problem.

Acknowledgment: I am very grateful to Adam Letchford for his comments and suggestions which made writing this chapter a somewhat easier task. The reader does not notice the improvements but I do and I would like to thank Adam. Mistakes which remain in the text are all mine.

References

[1] Agnetis, A., P. Detti, C. Meloni, and D. Pacciarelli (1999). "Set-up coordination between two stages of supply chain", Technical Report 32-99, Dipartimento do Informatica e Sistemistica, Universita Degli di Roma "La Sapienza", Italy.

[2] Altinkemer, K. and B. Gavish (1987). "Heuristics for unequal weight delivery problems with fixed error guarantee", *Oper. Res. Lett. 6*, 149-158.

[3] Andersen, L.D., and H. Fleischner (1995). "The NP-completeness of finding A-trails in Eulerian graphs and of finding spanning trees in hypergraphs", *Discrete Applied Mathematics* 59, 203-214.

[4] Arora, S. (1994). "Probabilistic checking of proofs and hardness of approximation problems", Ph.D. thesis, U.C. Berkeley, Available via anonymous ftp as Princeton TR94-476 from http://ftp.cs.princeton.edu.

[5] Arora, S. and C. Lund, (1997). "Hardness of approximation", in Hochbaum, D.S., (ed.) *Approximation Algorithms for NP-Hard Problems*, PWS Publishing Company, Boston, Massachusetts, 399-446.

[6] Assad, A.A. and B.L. Golden (1995). "Arc routing methods and applications". In M.O. Ball, T.L. Magnanti, C.L. Monma and G.L. Nemhauser (Eds.) *Network Routing.* Handbooks of Operations Research and Management Science, 8. Amsterdam: North Holland.

[7] Ball, M.O. and U. Derigs (1983). "An analysis of alternative strategies got implementing matching algorithms", *Networks* 13, 517-549.

[8] Ball, M.O. and M.J. Magazine (1988). "Sequencing of insertions in printed circuit board assembly", *Operations Research* 36, 192-201.

[9] Belenguer J.M. and E. Benavent (1998). "The capacitated arc routing problem: valid inequalities and facets" *Computational Optimization & Applications* 10, 165-187.

[10] Benavent, E., V. Campos, A. Corberan, and E. Mota (1985). "Analisis de heuristicos para el problema del cartero rural", *Trabajos de Estadistica e Investigacion Operativa* 36, 27-38.

[11] Benavent, E., A. Corberan, and J.M. Sanchis (2000). "Linear Programming Based Methods for Solving Arc Routing Problems", in Dror, M. ed. *Arc Routing: Theory, Solutions and Applications* (this book).

[12] Bent, W. and U. Manber (1987). "On Non-intersecting Eulerian circuits", *Discrete Applied Mathematics* 18, 87-94.

[13] Berge, C. (1973). *Graphs and Hypergraphs*, North-Holland Publishing Company, Amsterdam-London.

[14] Crescenzi, P. and V. Kann (1998). "A compendium of NP optimization problems", Technical Report, Universita di Roma, Italy. Avilable at http://www.nada.kth.se/~viggo/wwwcompendium/, and as an Appendix in Complexity and Approximation, Ausiello, G., P. Crescenzi, G. Gambosi, V. Kann, A. Marchetti-Spaccamela, and M. Protasi (Eds.), Springer-Verlag, 1999.

[15] Christofides, N. (1976). "Worst case analysis of a new heuristic for the traveling salesman problem", *Management Science Research Rept.* 388, Carnegie-Mellon University, Pittsburgh, PA.

[16] Cook, S.A., (1971). "On the complexity of theorem-proving procedures", *Proc. 3rd Annual ACM Symp. Theory of Computing*, 151-158.

[17] Corberan, A. and J.M. Sanchis (1994). "A polyhedral approach to the rural postman problem", *Eur. J. Oper. Res.* 79, 95-114

[18] Derigs, U. (2000). "Matching: Arc Routing and the Solution Connection", in Dror, M. ed. *Arc Routing: Theory, Solutions and Applications* (this book).

[19] Dror, M. and M. Haouari (2000). "Generalized Steiner Problems and Other Variants", (forthcoming in J. of Combinatorial Optimization).

[20] Dror, M and A. Langevin (1997). "A Generalized Traveling Salesman Problem approach to the directed clustered Rural Postman Problem", *Transportation Science* 31, 187-192.

[21] Dror, M and A. Langevin (2000). "Exact Solutions: Transformations and Column Generation", in Dror, M. ed. *Arc Routing: Theory, Solutions and Applications* (this book).

[22] Dror, M. and J.M.Y. Leung (1998). "Combinatorial optimization in a cattle yard: Feed distribution, vehicle scheduling, lot sizing, and dynamic pen assignment", Gang Yu, ed., *Industrial Applications of Combinatorial Optimization*, Kluwer Academic Publishers, 142-171.

[23] Dror, M., H.I. Stern and P. Trudeau (1987). "Postman tour on graph with precedence relations on arcs", *Networks* 17, 283-294.

[24] Dror, M., J.M.Y. Leung and P.A. Mullaseril (2000). "Livestock Feed Distribution and Arc Traversal Problems", (this book).

[25] Edmonds, J., and E.L. Johnson (1973). "Matching, Euler Tours and the Chinese postman", *Mathematical Programming* 5, 88-124.

[26] Eiselt, H.A. and G. Laporte (2000). "A Historical Perspective on Arc Routing", in Dror, M. ed. *Arc Routing: Theory, Solutions and Applications* in Dror, M. ed. *Arc Routing: Theory, Solutions and Applications* (this book).

[27] Eiselt, H.A., M. Gendreau, and G. Laporte (1995). "Arc-routing problems, part 1: the Chinese postman problem", *Oper. Res.* 43, 231-242.

[28] Eiselt, H.A., M. Gendreau & G. Laporte (1995). "Arc-routing problems, part 2: the rural postman problem", *Oper. Res.* 43, 399-414.

[29] Eglese, R.W. and A.N. Letchford (2000), "Polyhedral Theory for Arc Routing", (this book).

[30] Fleischner, H. (1990). Eulerian Graphs and Related Topics, Part I, Volume 1. *Annals of Discrete Mathematics* 45. North-Holland, Amsterdam.

[31] Fleischner, H. (2000). "Traversing graphs: The Eulerian and Hamiltonian theme", in Dror, M. ed. *Arc Routing: Theory, Solutions and Applications* (this book).

[32] Ford, L.R. and D.R. Fulkerson, (1974). Flows in Networks, Princeton University Press, Sixth Printing.

[33] Frederickson, G.N., (1979). "Approximation algorithms for some postman problems", *J. of ACM* 26, 538-554.

[34] Frederickson, G.N., M.S. Hecht, and C.E. Kim (1978). "Approximation algorithms for some routing problems", *SIAM J. Comput.* 7, 178-193.

[35] Galil, Z. and N. Megiddo (1977). "Cyclic ordering is NP-complete", *Theoretical Computer Science* 5, 179-182.

[36] Garey, M.R. and D.S. Johnson (1979) *Computers and Intractability: a guide to the theory of NP-completeness.* San Fr.; Freeman.

[37] Ghiani, G. and G. Improta (2000). "An algorithm for the Hierarchical Chinese Postman Problem", *Operations Research Letters* 26, 27-32.

[38] Golden, B.L. and R.T. Wong (1981). "Capacitated arc routing problems" *Networks* 11, 305-315.

[39] Guan, M. (1984). "On the windy postman problem", *Discrete Applied Mathematics* 9, 41-46.

[40] Guan, M. and W. Pulleyblank (1985). "Eulerian orinentations and circulations", SIAM J. Algebraic Discr. Math. 6, 657-664.

[41] Haimovich, M., A.H.G. Rinnooy Kan and L. Stougie (1988). "Analysis of Heuristics for Vehicle Routing Problems", B.L. Golden and A.A. Assad (eds), *Vehicle Routing: Methods and Studies* , Elsevier Science Publishers B.V. (North-Holland), pp. 47-61.

[42] Harary, F., and C.St.J.A. Nash-Williams (1965). "On Eulerian and Hamiltonian graphs and line-graphs", *Canadian Mathematics Bulletin*, 701-709.

[43] Hochbaum, D.S., (ed.) (1997). *Approximation Algorithms for NP-Hard Problems* , PWS Publishing Company, Boston, Massachusets.

[44] Jansen, K., (1992). "An approximation for the general routing problem", *Information Processing Letters* 41, 333-339.

[45] Jansen, K., (1993). "Bounds for the General Capacitated Routing Problem", *Networks* 23, 165-173.

[46] Johnson, D.S. (1990). "A Catalog of Complexity Classes", J. van Leeuwen, ed., *Algorithms and Complexity, Handbook of Theoretical Computer Science*, Vol. A, Elsevier Science Publishers B.V., 68-161.

[47] Johnson, D.S. and C.H. Papadimitriou, (1985). "Computational Complexity", E.L. Lawler, J.K. Lenstra, A.H.G. Rinnooy Kan, and D.B. Shmoys, eds. *The Traveling Salesman Problem*, John Wiley and Sons Ltd., 37-85.

[48] Kappauf, C.H. and G.J. Koehler (1979). "The mixed postman problem", *Disc. Appl. Math.* 1, 89-103.

[49] Karp, R.M., (1972). "Reducibility among combinatorial problems", R.E. Miller and J.W. Thatcher, eds. *Complexity of Computer Computations*, Plenum Press, New York, 85-103.

[50] Labbe, M., G. Laporte and H. Mercure (1991). "Capacitated vehicle routing on trees", *Operations Research* 39, 616-622.

[51] Laporte G., (1997). "Modeling and solving several classes of arc routing problems as traveling salesman problems", *Com. & Ops. Res.* 24, 1057-1061.

[52] Lenstra, J.K, and A.H.G. Rinnooy Kan (1976). "On general routing problem", Networks 6, 273-280.

[53] Lichtenstein, D., (1982). "Planar formulae and their uses", *SIAM J. Comput.* 11, 329-243.

[54] Manber U. and S. Israni, (1984). "Pierce point minimization and optimal torch path determination in flame cutting", *J. Manufacturing Systems* 3, 81-89.

[55] Minieka, E. (1979). "The Chinese postman problem for mixed networks", *Management Science* 25, 643-648.

[56] Mullaseril, P.A. (1996). "Capacitated Rural Postman Problem with Time Windows and Split Delivery", Ph.D. Thesis, MIS Department, The University of Arizona, Tucson AZ 85721.

[57] Nobert, Y. and J.-C. Picard (1996). "An optimal algorithm for the mixed Chinese postman problem", *Networks* 27, 95-108.

[58] Noon, C.E. and J.C. Bean, (1993). "An efficient transformation of the generalized traveling salesman problem", *INFOR* 31, 39-44.

[59] Orloff, C.S., (1974). "A fundamental problem in vehicle routing", *Networks* 4, 35-64.

[60] Orloff, C.S., (1976). "On general routing problems: Comments", *Networks* 6, 281-284.

[61] Papadimitriou, C.H. (1976). "On the complexity of edge traversing", *J. of the A.C.M.* 23, 544-554.

[62] Raghavachari B. and J. Veerasamy, (1998). "Approximation algorithms for the mixed postman problem", R.E. Bixby, E.A. Boyd, and R.Z Rios-Mercado (eds.), *Proceedings of IPCO* VI, Vol. 1412, Springer-Verlag, 169-179.

[63] Raghavachari B. and J. Veerasamy, (1999a). "Approximation algorithms for asymmetric postman problem", *Proceedings of SODA* 1999.

[64] Raghavachari B. and J. Veerasamy, (1999b). "A 3/2-approximation algorithms for the mixed postman problem", *SIAM J. on Discrete Mathematics* 12, 425-433.

[65] Ralphs, T.K., (1993). "On the mixed Chinese postman problem", *Oper. Res. Lett.* 14, 123-127.

[66] Shmoys, D.B. and E. Tardos (1995). "Computational Complexity", in R.L. Graham, M. Grotchel, and L. Lovasz, eds. *Handbook of Combinatorics*, Vol II, North-Holland, 1599-1645.

[67] Stern, H.I. and M. Dror (1979). "Routing electric meter readers", *Com. & Ops. Res.* 6, 209-223.

[68] Van Emde Boas, P. (1990). "Machine Models and Simulations", J. van Leeuwen ed. *Handbook of Theoretical Computer Science*, Elsevier Science Publishers, B.V. 1-66.

[69] Win, Z., (1989). "On the windy postman problem on Eulerian graphs", *Mathematical Programming* 44, 97-112.

[70] Yannakakis, M., (1997). "Computational Complexity", E. Aarts and J.K. Lenstra eds. *Local Search in Combinatorial Optimization*, John Wiley & Sons Ltd., 19-55.

Chapter 5

CHINESE POSTMAN AND EULER TOUR PROBLEMS IN BI-DIRECTED GRAPHS

Ellis L. Johnson
Georgia Institute of Technology

1. Bi-directed Graphs, Euler Tours, and Postman Tours 171
2. Aircraft Routing in a Space-time Network 176
3. The Chinese Postman Problem in an Undirected Graph 181
4. Binary Group Problems and Blocking Problems 189
5. Ideal Binary Matrices 192
6. Four Problems on Planar Graphs 193

1. BI-DIRECTED GRAPHS, EULER TOURS, AND POSTMAN TOURS

Euler tours occupy an interesting position in the history of graph theory. Current interest in this area is due to problems involving tours where service is required along arcs of the tour rather than at nodes. Examples of problems of this type involve mail delivery, snow removal, street cleaning, trash pickup, etc. In considering such problems involving city streets, the nodes are intersections and the arcs are roadways between intersections. However, there are one-way streets and two-way streets. If the service must be done on both sides of a two-way street then two one-way streets (one in each direction) can replace it. However, there are situations where the street, even though two-way, need only be traversed once. For example, mail delivery in a more rural or suburban setting may require traversing the street or road only once. In a rural road, mail delivery is frequently done on one side of the road, and those who live on the other side must cross the road to get their mail. In setting up the routes, though, the road could be traversed in either direction. Thus, we consider graphs with two types of connections: directed and undirected. Other postman problems [14] have been considered.

Two basic classes of problems are addressed. The first is finding a tour on the graph as it is given. The second involves changing the graph, by duplicating arcs, so that a tour will exist. The first of these is referred

to as an Euler tour problem and the second as a postman problem. Although the second is a perfectly natural extension of the first, the Euler tour problem is much older. Serious work on the postman problem dates only to the work of Ford and Fulkerson in the 1950's (see [10]) and Meigu Guan [15] in the 1960's. In the postman problem, there are two ways to look at the problem. One is the tour itself, but the other is to focus instead on the arcs or edges that must be duplicated. This set will be defined as a postman set. In the undirected case, edges need only be duplicated once. Once a postman set has been determined and added to the graph, the postman problem reduces to the Euler tour problem. In the mixed and directed cases, edges and arcs may need to be duplicated more than once, so the order of duplication must be specified.

This section gives definitions and some results on existence of Euler tours and postman tours. The next section outlines an application in aircraft routing. Section 3 discusses the Chinese postman problem in an undirected graph and gives an algorithm using its polyhedral description. Section 4 introduces the framework of binary group problems and blocking pairs of clutters. Section 5 discusses ideal binary matrices and relates this concept to binary group problems and, in particular, the postman problem. Finally, we close with some results on four problems specialized to planar graphs. Sections 4, 5 and 6 give additional insights into the polyhedral and algebraic structures involved. The Chinese postman problem provides an interesting special case of combinatorial polyhedra and algebraic structures. These latter structures include matroids and Gomory's group problem [12].

The term mixed graph was used in [18] and the term bidirected graph was introduced in [7]. Clearly they are somewhat redundant. Bidirected seems more descriptive in that direction (or the lack of it) is the question here, so the term mixed graph will be used to mean a bidirected graph where arcs and edges are both present.

More formally, define a *bi-directed graph* G to be $(N, E\ U\ A)$ where N is the set of *nodes*, E is the set of (undirected) *edges*, and A is the set of (directed) *arcs*. An edge consists of an unordered pair of nodes, and an arc consists of an ordered pair of nodes, say (i, j). The node i is called the *tail* of the arc, and node j is called the *head* of the arc. In many of our problems, the edges will be assigned one of the two possible directions in order to specify a tour as defined below. The graph G is called *directed* (*undirected*) if it only has arcs (edges). If both E and A are non-empty, then G is called *mixed.* Since all of our graphs are bidirected graphs, we will simply refer to a graph.

The *degree* of a node i is the number of edges and arcs meeting i. The *in-degree* of node i is the number of arcs whose head is node i. The *out-degree* of node i is the number of arcs whose tail is node i. A node is *balanced* if its in-degree equals its out-degree.

A *tour* in a graph G is a sequence $T = (n_1, v_1, n_2, v_2, \ldots, v_{K-1}, n_K)$ where n_k is a node and v_k is an arc or an edge such that it meets the two nodes n_{k-1} and n_k. In addition, the first node n_1 of the tour is required to be the same as the last node n_K, and if v_k is an arc then $v_k = (n_{k-1}, n_k)$. A *simple tour* is a tour such that no arc or edge appears more than once in the tour. An *Euler tour* is a simple tour T such that every edge in E appears (exactly once) in T. A graph G is said to be *Eulerian* if it admits an Euler tour. A *postman tour* is a tour T such that every edge in E appears at least once in T. In a postman tour, an undirected edge may be used any number of times and in either or both directions. However, the directed edges, while they may appear more than once in the tour, must always be in the specified direction. Figure 5.1 shows a graph on four nodes with edges numbered 1,2,3,4,5. A postman tour traverses the edges (in order) 1,4,2,3,5,4,1, and the associated postman set is $\{1, 4\}$.

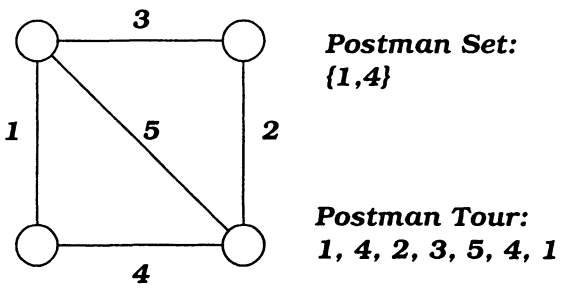


Figure 5.1 Postman set and postman tour.

The following results summarize conditions for a graph to be Eulerian and to admit a postman tour. See [9] for more details and historical perspective.

Theorem 1 (Euler) An undirected graph G is Eulerian if and only if it is connected and every node has even degree.

Proof 1: The proof is algorithmic. □

Theorem 2 [15] An undirected graph G admits a postman tour if and only if it is connected.

Proof 2: Adding a duplicate copy of every edge to the graph produces a graph where every degree is twice its original degree. Thus, it produces an even degree graph, which is Eulerian by Theorem 1. □

Theorem 3 A directed graph G is Eulerian if and only if it is connected and every node is balanced.

Proof 3: Very similar to that of Theorem 1. □

A *cut* in a connected graph G is the set of edges and arcs meeting a node in a set S and a node not in S. The cut is frequently referred to as (S,T) where $T = N/S$. When the graph G is directed or mixed, the *di-cut* associated with (S,T) is the set of arcs with a tail in S and head in T. We call the di-cut (S,T) a *strong di-cut* if the only edges or arcs in the cut (S,T) are in the di-cut. In other words, there are no edges in (S,T) and the only arcs in (S,T) have a tail in S and a head in T. For a pair of nodes s and t such that s is in S and t is in T, a cut (or di-cut) is called an *(s,t) cut* (or *(s,t) di-cut).*

Theorem 4 A directed graph G has a postman tour if and only if it is connected and has no strong di-cut.

This result can be proven using a network flow model to balance the nodes.

Theorem 5 (Ford and Fulkerson) A mixed graph is Eulerian if and only if it is connected, every node has even degree, and there does not exist a cut $C = (S,T)$ such that the number of arcs in the di-cut (S,T) is more than $1/2$ the total number of edges and arcs in the cut C.

When every node has even degree, every cut will also have an even number of edges. However, this last condition to be Eulerian is more than a condition on nodes. Requiring that no node have in-degree or out-deree more than one-half of the degree does not assure that the graph is Eulerian.

Figure 5.2 shows a non-Eulerian mixed graph. Note that every node has even degrees and satisfies the condition that there are not more arcs into (or out of) the node than other arcs or edges at the node. In other words, no node has in-degree or out-degree greater than one-half the de-

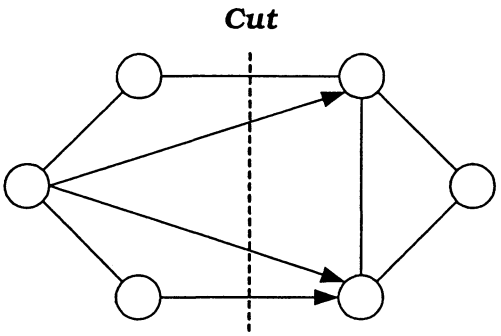


Figure 5.2 Non-Eulerian mixed graph.

gree. Yet the cut indicated has more than one-half of the arcs and edges in it directed in one direction, so by Theorem 5 the graph has no Euler tour.

Figure 5.3 shows the same mixed graph as in Figure 5.2 with the edges numbered. Although there are cuts, e.g. those having sets of arcs and edges $\{1,6,7,8\}$ or $\{2,6,7,8\}$, with more than one-half of the arcs directed in one direction, there is no strong di-cut. In this case, edges 1 and 2 must be duplicated twice, so the notion of postman set must include the number of times each edge is to be duplicated. In the directed case, the order of duplication is also needed but not in the undirected case, as has been illustrated in Figure 5.1 and mentioned there.

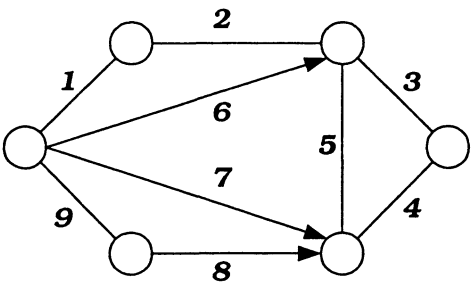


Figure 5.3 Postman tour on mixed graph.

Theorem 6 (Ford and Fulkerson) A mixed graph admits a postman tour if and only if it is connected and has no strong di-cut.

See [9] for a discussion of a network flow formulation of the above two problems. Solving the mixed postman problem is $\mathcal{NP}$-complete, but special cases were shown to be polynomially solvable in [18]. Other work has been done, see e.g. [23].

Lemma 1 A bi-directed graph contains an Euler tour if and only if it is connected and the edges can be partitioned into edge-simple tours.

Proof: The "only if" direction is trivial because a single Euler tour gives such a partition. To show the converse, an algorithm will be outlined. There is no claim that the following outlined procedure is efficient in tracing an Euler tour, but it does serve to prove the lemma. Number the edge-simple tours, say 1 to T. Start at any node r in tour number 1. If at that node there are any other tours, then move in the highest number tour to the next node in that tour and repeat. That is, any time a node is reached, look for the highest number tour and move along it. If there is no higher numbered tour, then one can proceed on the same tour to the next node of that tour unless that tour has been completely traversed. In that case there will necessarily be a lower numbered tour interrupted at that node, so choose the highest such tour and continue. To see that an Euler tour is traced, once a tour of a given number, say k, is started on, no lower number tour will be used until tour k is completed. Every edge that is traversed is used only once, so eventually tour 1 will be completed. To see that every tour is traversed, induction suffices. □

See [9] for a discussion of efficient algorithms for finding Euler tours.

Lemma 2 A bi-directed graph G contains an Euler tour if it is connected, even degree, and balanced.

Proof: Even though the undirected (directed) subgraphs of G may not be connected, each component is Eulerian. Thus the graph can be partitioned into cycles and, by lemma 1, is Eulerian. □

2. AIRCRAFT ROUTING IN A SPACE-TIME NETWORK

In scheduling an airline, usually a daily plan is first arrived at. That plan assumes that every flight leg is flown every day by the same type of plane. An aircraft routing says how to connect (or turn) the planes so that the same turns are made every day. A major requirement for many

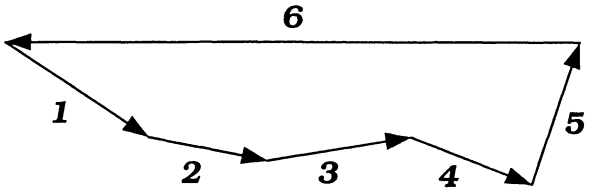
airline planners is that the daily routing for any one fleet type form a single large cycle. This required cycle is referred to as an *unlocked rotation.* We will return to this requirement after explaining the problem. This aircraft routing problem is solved after the fleet types have been assigned to all flight legs. Typically the fleet assignment model [16] is used to solve that problem. The output from the fleet assignment model includes flow values for planes by fleet type on a space-time network. The network has two types of arcs: flight leg arcs by fleet type and ground arcs by station (airport) and time. The nodes of the network are by fleet type, station, and time. There is a node for each departure of a flight leg having indices: station and time of departure and fleet type that might fly that leg. There is a corresponding node for the arrival of the leg with indices: arrival station, ready time, and fleet type. By ready time is meant the actual scheduled arrival time plus the minimum turn time. Thus the ready time represents the time at which the plane would be ready to depart. The flows on the ground arcs represent the number of planes that stay on the ground beyond the minimum turn time before departing, by fleet type.

Even if these ground arc flows are not given explicitly, they can be determined from a fleeted schedule as follows. By a fleeted schedule is meant the flight legs to be flown, the departure and arrival times, and the type of plane (fleet type) that is to fly them. For a given fleet type and a particular station, there are necessarily as many arrivals as departures. Otherwise the schedule can not be flown. Create a directed cycle with nodes for departure times of each departing leg and ready times for each arriving leg. If two of these times are the same, create just one node. Thus, for this fleet type and this station, each node has a unique time of day identifying it. Make the cycle by putting an arc from a node to the node with the next largest time except for the largest time node gets an arc back to the node with smallest time. Now, this cycle has as many departures as arrivals. Create exogenous flow requirements with a supply of one for each arrival and a demand of one for each departure. Each node has the net supply or demand over all arrivals and departures at the time associated with that node. On a cycle, there is only one independent flow needed to determine all flows. The ground arc flows can not be negative, so solve for flows so that at least one ground arc has flow of zero. This is the flow solution that uses the minimum number of planes. The number of planes needed for the fleet type can be determined by picking a global time such as midnight GMT or 2 am CST, and adding up over all ground arcs and flight leg arcs that cross that global time. The ground arc values determined in this way should be the same as from the fleet assignment problem and the plane count should also agree. The only reason not to do so is either that the schedule does

not require as many planes as available or, more likely, there are other arcs in the fleet assignment problem representing maintenance or other activities.

In any case, let us assume that correct values of flows on ground arcs are known. We now have a directed, balanced network. The only remaining question in finding the unlocked rotation mentioned at the beginning of this section is whether the network is connected. If it is connected, then the network is Eulerian, and any Euler tour of it is an unlocked rotation. Let us first address the connectivity question and return later to what the unlocked rotation means. Clearly, the network is not connected if there is a proper subset of the nodes such that all of the flight legs departing from a station in this subset arrive at another station in this subset. For US airlines, this type of disconnection hardly ever occurs because of the nature of the hub-and-spoke network flown by most US carriers. In any case, this lack of connectivity is highly undesirable because it divides the fleet into two sub-fleets that never mingle. Any planner looking at a fleeting would change the fleeting so as to avoid this problem. However, there is a more subtle cause of the network being unconnected. This cause is a disconnection is stations and time. For example, suppose there is one plane on the ground overnight in Boston and it flies early in the morning to Fort Lauderdale. Then it stays on the ground in Fort Lauderdale and flies back to Boston where it again overnights. Let us suppose further that during the time that it is on the ground in Fort Lauderdale there are no other arrivals or departures of the same fleet type and there are no other planes on the ground. This same assumption is made for Boston. Boston may have other arrivals of this fleet type during the day, but those planes all depart before the Fort Lauderdale flight returns to Boston. Then we have an isolated cycle in space-time that locks that plane into a "locked rotation". Thus, the network is connected if and only if there is no proper subset of nodes (all of which have a station and a time) such that every flight leg of the given fleet type departing from one of these nodes has an arrival node in the subset, and every ground arc for this fleet type that meets one node in the subset and one node not in the subset has zero flow.

The reason that airlines want to avoid a locked rotation is due mainly to maintenance considerations. There are longer maintenance checks that are done typically at one station every night. A plane on a locked rotation may not ever get to that station. In addition, there is the preference that all planes get maintenance at all stations and get even wear. Thus, with an unlocked rotation the planes all fly the same legs and go through all of the maintenance stations.



Flight Legs

	Departure		Arrival	
1	DFW	0800	SFO	1000
2	SFO	1130	JFK	2030
3	JFK	0830	MIA	1230
4	MIA	1400	SEA	1930
5	SEA	0900	ORD	1500
6	ORD	1600	DFW	1900

Figure 5.4 Aircraft Rotation.

For an unlocked rotation, any one plane flies the one day rotations in order until all have been flown and then starts over the same cycle. Looking at each day and all of the planes, there are as many days of flying as there are planes, and each plane flies one of these days worth of flying. The next day a plane moves to the day of flying of the plane in front of it. Thus the planes follow each other in a cyclical fashion. Of course, there may be flying at night, so a day of flying could be defined as the first departure after the global time until the leg or ground arc that next crosses the global time. Figure 5.4 provides an example of a three day cycle where each day has two flight legs and, would require three planes to fly the six legs.

Any algorithm for finding an Euler tour in a directed graph can be used to provide an unlocked rotation. However, a simple FIFO rule (that says here depart the plane that has been on the ground the longest) may not give an Euler tour but instead several cycles. These cycles would be referred to as locked rotations.

The question of how to avoid locked rotations is, thus, the same as how to avoid disconnecting the space-time network when partitioning the flight legs among the fleet types. Imposing this restriction in the fleet assignment problem greatly complicates it [2]. Inequalities much like the subtour elimination constraints in the traveling salesman problem must be adjoined to the already difficult mixed-integer program. However, in practice the hub-and-spoke nature of an airline's network generally keeps

the network for each fleet type connected.

There are two other considerations involved in finding routings: through values and maintenance. What is meant by through values is: when a plane arrives at a station, typically a hub, where there are choices as to how to connect to a departure, there may be advantages to connect so that passengers can stay on the plane. For example, if an American flight from Atlanta to DFW can connect to a flight to Albuquerque, and many Atlantans want to fly to Albuquerque, then routing the plane from Atlanta to DFW to Albuquerque allows the flight to be a 1-stop rather than a connection. Thus, there are incremental values to certain turns, mainly at hubs. However, forcing all of these desired turns may destroy the unlocked rotation, just as imposing a FIFO order may do so.

The other consideration is a constraint that must be satisfied: performing regular maintenance frequently enough. Many airlines will impose a frequency in terms of number of days between certain types of maintenance in order to make planning easier and be sure of meeting all FAA rules. This restriction, in general, turns the easy Euler tour problem into a hard problem. However, once again the hub-and-spoke network makes it fairly easy to satisfy this restriction. The main difficulty comes if the network has a significant number of point-to-point strings of flight legs without a maintenance station being encountered in the string. Frequently the difficulty can be isolated from the otherwise large network and conditions found to impose on the fleeting problem. If the maintenance condition cannot be satisfied, then either the fleeting must be changed or the routing can be examined in more detail to see if it meets the FAA requirements.

The maintenance requirement in [2] was that a short maintenance had to be done every two days and a longer maintenance including avionics had to be performed every four days. Frequently, however, the requirement involves only one type of maintenance that has to be done every three days. This requirement can be modeled, although it is not an easy problem to solve, as follows: omit all overnight return arcs and then duplicate this reduced network three times to form N^1, N^2, N^3. Now any overnight ground arc that allows maintenance (i.e. at a maintenance station and sufficiently long) returns to the first node of that station in network N^1. That is, it allows resetting the clock on maintenance. All other overnight ground arcs go to the first node of that station in the next higher super-scripted network. That is, the clock keeps running. The flight leg arcs that cross the global time selected are treated in a similar manner. They go from their departure node in one network to their arrival node in the next higher super-scripted network. The prob-

lem is to select one of the three flight leg arcs from the three networks N^1, N^2, N^3 for each flight leg so that the resulting arcs can be balanced by use of ground arcs. Plane count can be assured by simply imposing a similar constraint of the overnight ground arcs. So far, this problem is a multi-commodity flow problem, which is hard enough, but in addition the network chosen must be connected or there will be no unlocked rotation. This construction can be used within the fleet assignment problem but complicates an already difficult problem.

Another approach [1] involves strings of legs between maintenance opportunities.

3. THE CHINESE POSTMAN PROBLEM IN AN UNDIRECTED GRAPH

The problem can be stated as finding a minimum cost duplication of edges to form an Eulerian graph. Assume that the cost c_e on each edge is non-negative and that multiple edges are allowed in E. The problem is, thus, to find a minimum cost set of edges to duplicate so as to make every node have even degree. Clearly, an edge need not be duplicated more than once. Let M be a subset of edges such that M union E is Eulerian. We call such a set M a postman set. Thus, a postman set is a set of edges that allow an Euler tour when those edges are duplicated.

Define a node to be an odd node if it has odd degree and an even node if it has even degree. A set M of edges is a postman set if and only if the edges in M form a subgraph such that odd nodes are still odd degree and even nodes are still even. Then when M is unioned with E, the resulting multigraph will be even degree everywhere, hence Eulerian.

Theorem 1 (Mei-ko Kwan [15]) A postman set M is optimal if and only if every cycle C satisfies

$$\sum\{e \in C \cap M\}c_e \leq \sum\{e \in C \backslash M\}c_e.$$

This theorem gives insight into the structure of an optimum solution but does not lead directly to an algorithm. What it says in words is that a postman set is optimal if and only if for every cycle the cost of the edges in the cycle and the postman set is less than or equal to the remaining edges of the cycle.

It is easily seen that an optimal postman set M need not contain a cycle. Thus, it can be assumed to be a forest, where a forest is a subgraph (not necessarily connected) that has no cycles. The set M can be included in a spanning tree, since the graph is connected. That is,

the forest can be extended to a spanning tree. Conversely, any spanning tree determines a Chinese postman solution. Before describing the procedure, define anode to be odd if it has odd degree and even if it has even degree. The procedure is much like that for finding a basic network flow solution: for any leaf of the tree, set the edge to be in the postman set if the leaf-node is odd, otherwise it is not in the postman set. Delete the edge, changing the parity of the other incident node if the edge is in the postman set. Repeat until only one node is left, which must at that point be an even node. We summarize these remarks in Proposition 1.

Proposition 1 Every spanning tree determines a unique postman set, and every postman set can be embedded in a spanning tree (which need not be unique).

Define an *odd cut* to be a (minimal) cut $C = (S, S^c)$ where S is a set of nodes such that there are an odd number of odd nodes. The set S (or equally S^c) defines the odd cut and is identified with it. Define a *clutter* to be a family of non-nested sets. Given a clutter C, define its *blocking clutter* $B(C)$ to be the family of minimal sets each of which meets every member of C.

Proposition 2 The blocking clutter of the family of all postman sets is the clutter of odd sets.

For example, the graph in Figure 1 has as its clutter of postman sets $\{\{1,4\},\{2,3\},\{5\}\}$ and the blocking clutter is $\{\{1,2,5\},\{1,3,5\},\{2,4,5\},\{3,4,5\}\}$. Each subset of edges in the blocking clutter is an odd cut.

Because the Chinese postman problem is $\mathcal{NP}$-complete, the problem of finding a min cost odd cut is also $\mathcal{NP}$-complete. However, Padberg and Rao gave an efficient combinatorial algorithm [24].

The Chinese postman problem can be solved by combining shortest path and matching algorithms [6]. First solve for the shortest path between each pair of odd nodes. Then, form the complete graph having a node for each odd node of the original graph and a cost on each edge equal to the distance, given by the shortest path step, between those two odd nodes. Solve the minimum weight matching problem on this complete graph. Now expand every edge in the matching to the path that gave the shortest path whose distance is on that edge.

A direct algorithm [18] that does not use matching as a subproblem but uses an approach similar to the matching algorithm will now be given.

Before stating the algorithm, pseudonodes and some associated concepts will be defined. The basic idea of a pseudonode is to substitute a single node in place of a set of nodes. We call this replacement contracting the set of nodes. All edges with both ends in the set disappear from the graph, and edges with one end in the set meet the new node, called a pseudonode. Thus, the pseudonode appears as a node in the contracted graph, but also has associated with it a set of original nodes. In the algorithm, the sets associated with pseudonodes will be nested, i.e. either disjoint or one contained in the other. The *surface graph* is obtained by contracting each maximal sets to form a pseudonode. Pseudonodes are drawn in the figures as squares whereas original nodes are drawn as circles. There is a certain layering that is associated with pseudonodes. Because of the nesting of sets, each pseudonode contains some number of maximal pseudonodes.

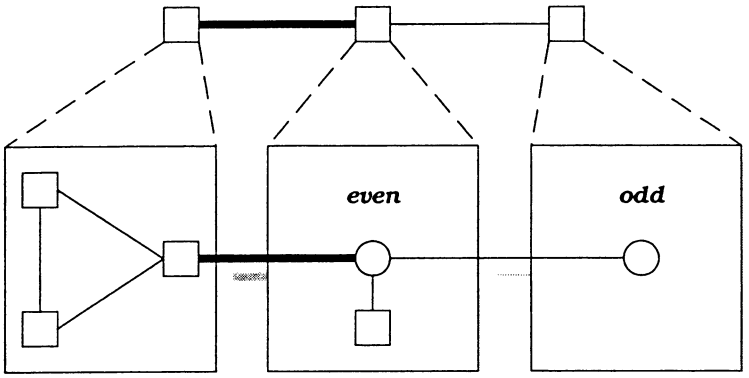


Figure 5.5 Surface graph and one level down.

Figure 5.5 shows a surface graph consisting of three pseudonodes and two edges. The *blossom* associated with each pseudonode will be explained. The blossom associated with a pseudonode will contain as nodes all of the maximal pseudonodes contained in it together with all of the original nodes contained in its subset that are not in any of these maximal pseudonodes. The three forms of blossoms are shown in Figure 5.6. The first is an odd cycle where every node is a pseudonode. The other two are both claws, i.e. a node and a set of edges meeting it. In this case, the node is an original node, and the edges meeting it need not include all of its incident edges. The claw is an odd claw if the original node is an odd node. In this case, there will always be an even number of edges in the blossom. The claw is an even claw if the original node is an even node. In this case, there will always be an odd number of edges

in the blossom. All of the other nodes in the blossom are pseudonodes, and there are an even number of them in odd claws and an odd number in even claws.

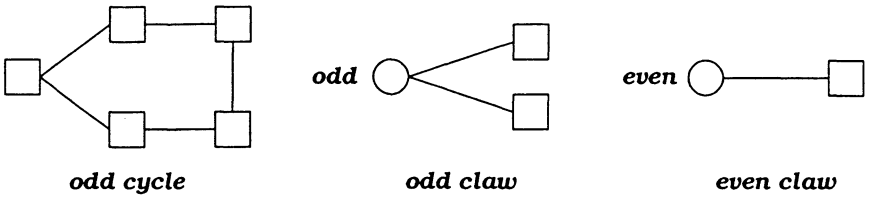


Figure 5.6 Blossoms.

Algorithm:

Start: Let every odd node be an *exposed node*. Let all primal variables $x_e, e \in E$, be 0 and all dual variables $\pi(S)$ be 0. When $x_e = 1$, the edge is called a *postman edge*, and the set of all postman edges is the postman set. Pick an exposed node and make it in M a pseudonode. Let that pseudonode be a plus-pseudeonode

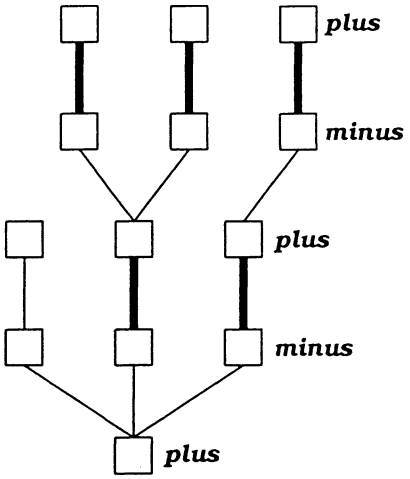


Figure 5.7 Planted Tree.

Grow a planted tree T: The form of a planted tree is alternating plus- and minus-pseudonodes as shown in Figure 5.7. Every minus-pseudonode will meet two edges, one in the postman set $x_e = 1$) and one not in the postman set ($x_e = 0$). For any plus-pseudonode k meeting an edge e

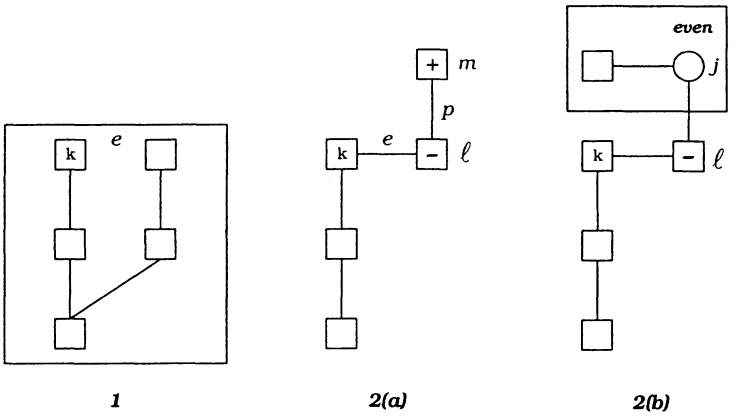


Figure 5.8 Grow a planted tree.

having $x_e = 0$ and not incident to a minus pseudonode, there are three cases for the other end ℓ: (1) ℓ is a plus pseudonode, (2) ℓ is a pseudonode not in the tree, and (3) ℓ is a node (necessarily not in the tree T). In case (1), e forms an odd cycle when adjoined to the tree T as shown in Figure 5.8(1). The odd cycle is said to form a blossom and is shrunk as described in the next step. The blossom becomes a new plus pseudonode. In case (2), there are two subcases: (a) the postman edge p meeting ℓ meets another pseudonode m, or (b) it meets a node j (see Figure 5.8, 2(a) and 2(b)). In the first case (a), ℓ becomes a minus pseudonode, and m becomes a plus pseudonode. In the second case (b), pseudonode ℓ still becomes a minus pseudonode, but now node j together with all of the postman edges meeting it, except for edge p, become a new pseudonode m that is a plus pseudonode.

In case (3) shown in Figure 5.9, there are two subcases: (a) node ℓ is exposed or its parity condition is satisfied. In case (a), node ℓ is exposed so we can augment. In the case (a), form a new plus pseudonode m subsuming pseudonode k consisting of k, e, and ℓ together with any postman edges meeting node ℓ.

Shrink a blossom: When an odd cycle occurs in the tree, the node in the cycle closest to the root is called the join and is necessarily a plus pseudonode. Contract, or shrink, the cycle to form a new pseudonode that becomes a plus pseudonode replacing the join in the tree. Its dual variable is zero but will increase at the next dual step. All of the dual variables inside the new pseudonode stay fixed until such time as this pseudonode is expanded during a dual step. The new pseudonode is in

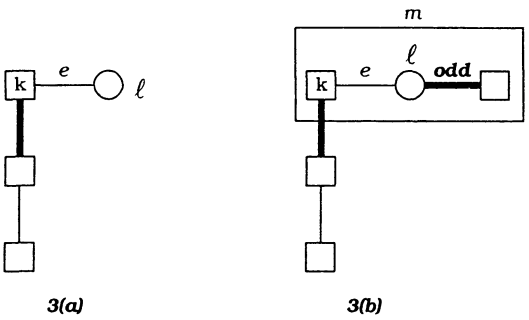


Figure 5.9 Grow a planted tree.

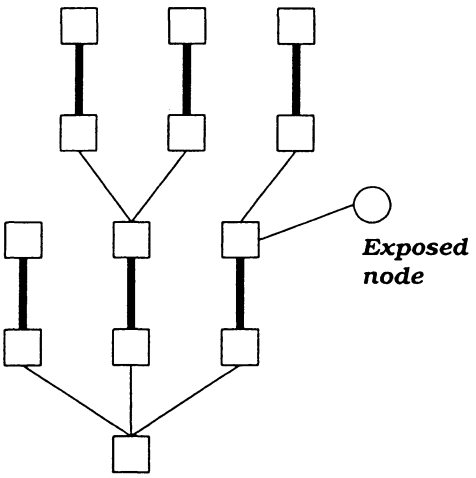


Figure 5.10 Augmenting Tree.

the surface graph but all nodes and pseudonodes in the cycle disappear from the surface graph.

Augment an alternating path: By the structure of a planted tree, there is an augmenting path in the surface graph (see Figure 5.10). What remains to be shown is that the alternating path can be extended to the original graph. The graph inside of a pseudonode can have two forms: an odd cycle of pseudonodes, or a node with any number of incident edges.

In the case of an odd cycle, if there are k pseudonodes, then there must be $(k-1)/2$ postman edges in the cycle meeting each pseudonode, except for the join, exactly once. In the case of a node and some incident edges,

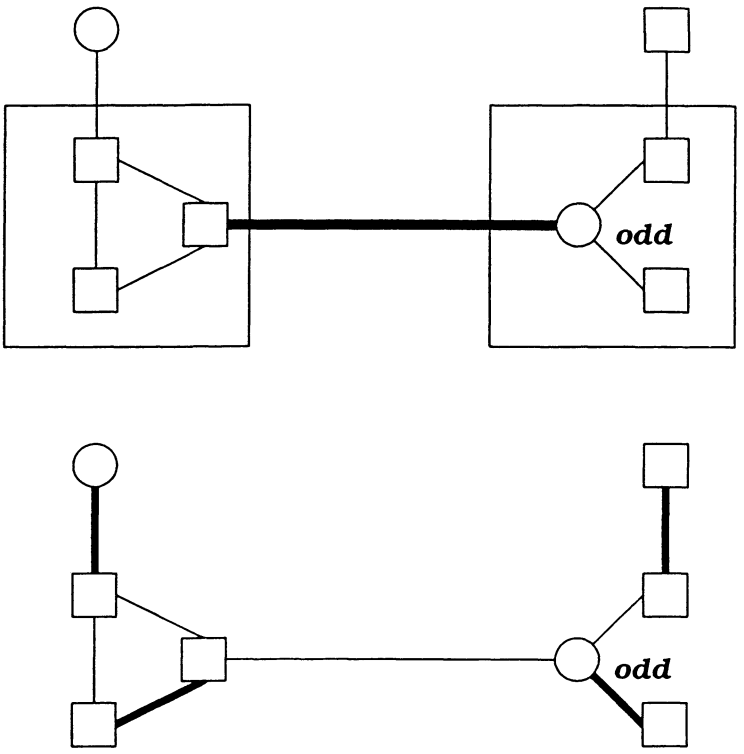


Figure 5.11 Augmenting inside pseudonodes.

if the node is an even node, there must be an odd number of incident edges, and if it is an odd node, then there must be an even number of incident edges.

Dual step: In the planted tree, increase the dual variable for each plus pseudonode and decrease the dual variable for each minus pseudonode. The amount of change is limited to be the minimum of three minimums:

1 the smallest initial dual variable for any minus pseudonode;

2 the minimum reduced cost for all edges in the surface graph with one end meeting a plus pseudonode and the other end meeting a node or pseudonode not in the planted tree; and

3 one-half times the minimum reduced cost for all edges in the surface graph having both end meeting plus pseudonodes.

If (1) limits the dual change, then the minus pseudonodes whose dual variables reach zero must be expanded before returning to the primal

step. If (2) limits the dual change, then when we return to the primal step the planted tree can be grown further. If (3) limits the dual change, then when we return to the primal step there will be a blossom that needs to be shrunk. Of course, there may be several ties for the minimum, and any subset of (1), (2), (3) may occur.

When (1) limits the dual change, it means that the dual variable corresponding to the odd cut constraint for the pseudonode has reached zero and is limited by its non-negativity requirement. In this case, the linear program would like to relax the equality in the odd cut constraint to allow inequality, i.e. to allow the slack variable to become positive. Expanding the pseudonode always allows growing the planted tree in the newly expanded subgraph to be spanning. Furthermore, if the pseudonode was not a leaf of the tree, then the tree can be reconnected using the newly expanded subgraph.

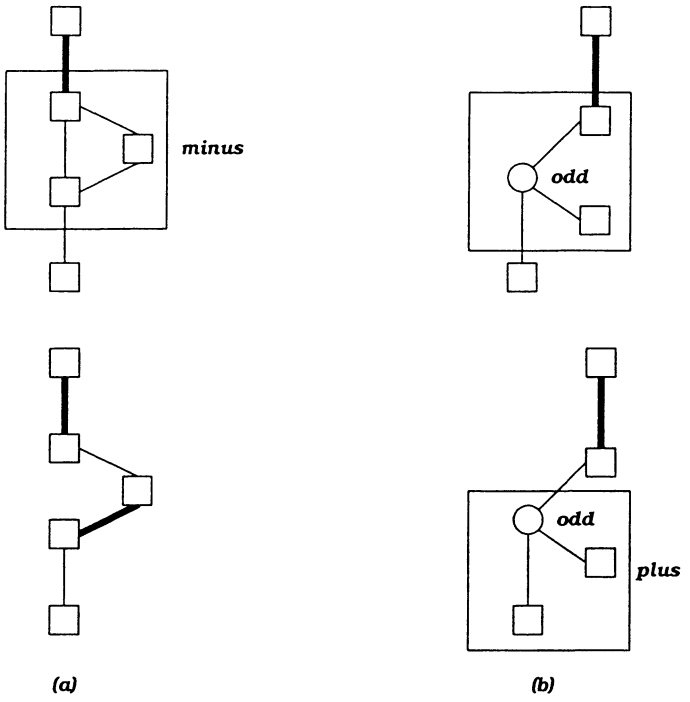


Figure 5.12 Expanding minus pseudonodes.

Figure 5.12 shows the cases (a) and (b), where (a) shows a minus pseudonode that is an odd cycle, and (b) shows a minus pseudonode consisting of an odd node and two incident edges. In case (b), a new plus pseudeonode

is formed using the rule for growing a planted tree, in this case from the root, when an original node is encountered (see Figure 5.2, case 3).

The Chinese postman problem is interesting in its own right as a graph problem with applications in routing. However, it is interesting in other ways. The first we will discuss is Gomory's group problem. Gomory developed this approach [12] as a relaxation of the integer programming problem. However, the Chinese postman problem is already a group problem, so there is no relaxation. Section 3 included a discussion of blocking clutters and blocking pairs of polyhedra. Fulkerson [11] developed these concepts independently of Gomory's work, but the Chinese postman problem provides a very nice example problem for both theories. Johnson [17] developed an algebraic construction to produce a blocking group problem whose solutions are the blocking cutter. That construction will be given here and the Chinese postman problem will be used to illustrate it.

4. BINARY GROUP PROBLEMS AND BLOCKING PROBLEMS

The general *binary group problem* is:

$$\begin{aligned} x^* &= 0, 1 \\ Mx^* &= b(\text{mod } 2) \\ &\text{Minimize } cx^*, \end{aligned}$$

where M is a 0-1 matrix, c is a non-negative real row vector, and b is any 0-1 column vector. If M is $m \times n$, then c is an n-vector and b is an m-vector. The Chinese postman problem can be stated in this way by letting M be the node-edge incidence matrix of G. There are clearly other matrices M that can be brought to the form of a node-edge incidence matrix (two 1's per column) by elementary row operations. Results of Tutte [28] show that a binary group problem can be reduced to a Chinese postman problem if and only if the matrix M does not contain certain minors [19]. Such matrices are said to be graphic, or equivalently are said to represent a graphic matroid.

The augmented matrix $[M \mid b]$ can be brought to standard form by pivoting (using modulo 2 arithmetic) on columns in a basis to bring the basis to an $m \times m$ identity I and deleting redundant rows to give the equivalent problem:

$$x^* = (x_I^*, x_N^*) = 0, 1$$

$$Ix_I^* + Nx_N^* = \bar{b}(\text{mod } 2)$$

A solution x is called basic if the only x^* equal to 1 are among x_I^*. It can be easily seen that basic solutions are minimal among all solutions so form a clutter. Gomory showed [12] that the vertices of the convex hull of integer solutions to binary (and ternary) group problems are precisely the basic solutions.

For the Chinese postman problem, the matrix I corresponds to a spanning tree, the vector $\bar{b}$ is the unique postman solution contained in that tree, and the matrix N has a column for every edge not in the tree and that column has a 1 for the edges in the path in the tree between its two end points. One could say this is an algebraic representation of the Chinese postman problem. However, this representation does not offer any easy computational advantage. It does, however, show how one postman solution can be changed to another. If an x_j^* is changed from 0 to 1, then a new solution is obtained by subtracting (modulo 2) column N^j from $\bar{b}$. Since the column N^j has 1's in the cycle formed by the out-of-tree edge, the affect of setting x_j^* to 1 and subtracting N^j from $\bar{b}$ is to change the 0's on the cycle to 1's and vice versa. Interestingly, this change in postman solution moves from one vertex to an adjacent vertex [19]. The reason that we cannot just perform the simplex method (using modulo 2 arithmetic) is that the objective function is in real arithmetic.

There is an interesting algebraic construction of the blocking problem [17]. Note that N is $m \times (n-m)$ so that an $(n-m) \times (n-m)$ identity I can be used to form the binary group problem in right-hand side form:

$$\begin{aligned} x = (x_{N^*}, x_{I^*}) &= 0, 1 \\ N^* x_{N^*} + I x_{I^*} &= 0(\text{mod } 2), \text{ where } N^* = N^t \\ \bar{b} x_{N^*} &= 1(\text{mod } 2) \end{aligned}$$

For the Chinese postman problem, the solutions will be incidence vectors of odd cuts and the congruence $\bar{b}x_{N^*} = 1 \pmod 2$ says that every odd cut meets a particular postman solution an odd number of times. The other congruence says that an odd cut meets a cycle an even number of times. Furthermore, the result here is that any subset of edges that meets any one particular postman solution contained in a spanning tree T an odd number of times and meets the cycles in the fundamental set of cycles of T an even number of times must be an odd cut. This construction works for any binary group problem [19].

The right-hand-side form used in the construction can, of course, be brought to standard form by pivoting on the row containing $\bar{b}$. Then the same construction can be used to get a right-hand-side representation

of the original binary group problem. In right-hand-side form, a basic solution is one that includes exactly one x_j equal to 1 from x_{N^*} and the values of x_{I^*} equal to that column of N^*. Again, these basic solutions are minimal among solutions and are the incidence vectors of the blocking clutter of the basic solutions of the original binary group problem.

The Chinese postman problem is characterized by the coefficient matrix being graphic. In that case, the partitioned matrix

$$[N^t I]$$

used in forming the right-hand-side form of the odd cut problem is a co-graphic matrix. However, the particular solution congruence $\bar{b}x_{N^*} = 1$ keeps the coefficient matrix of the odd cut problem from being co-graphic. Now consider a binary group problem where the coefficient matrix M is co-graphic and the right-hand side b is an arbitrary 0-1 vector. Such a problem has been called [20] a co-postman problem. Since this problem is now considered as the original problem, its standard form has congruences:

$$I\,x_I^* + Nx_N^* = \bar{b}(\mathrm{mod}\,2)$$

where for some spanning tree T, N has a column for each edge e of T and the 1's in the column of N correspond to the edges in the fundamental cut corresponding to the edge e. A row of the matrix $[\ I\ N\ ]$ is a fundamental cycle of the tree T formed by adjoining the edge e_i to the tree. We define a particular co-postman set to be those out-of-tree edges e_i such that $b_i = 1$. Define a cycle to be an odd cycle if it contains an odd number of co-postman edges. Now the general definition of co-postman sets is the sets of all edges outside of some spanning tree such that the fundamental cycle formed by adding the edge to the tree is an odd cycle. Thus, the co-postman sets are the solutions to a binary group problem when the coefficient matrix is co-graphic, for some right-hand side. The blocking clutter way to describe co-postman sets is that they are the blocking clutter of the odd cycles. Although odd cycles were defined originally based on a particular co-postman set, any co-postman set would give the same odd cycles.

For the Chinese postman problem, we usually begin with odd nodes and define postman sets in terms of meeting odd nodes an odd number of times. One way to think of that condition is that the postman set meets the odd cut of edges incident to the node an odd number of times. However, one can also begin with any fundamental set of cuts, some of which are designated as odd and some as even. Then, a postman set is a set of edges that intersects the even fundamental cuts an even number of times and the odd fundamental cuts an odd number of times. If postman is

replaced by co-postman and cut is replaced by cycle in the two preceding sentences, then we get the corresponding definition of co-postman sets. For the Chinese postman problem, minimality is assured for a postman set by being a subset of a spanning tree, i.e. not containing a cycle. Similarly, a co-postman set is minimal so long as it is a subset of the compliment of a spanning tree; i.e. does not contain a cut or in other words does not disconnect the graph.

The classical case of the Chinese postman problem designates a node as even or odd depending on the parity of its degree. Similarly, the co-postman problem can have cycles designated even or odd depending on their parity. In this case, removal of any co-postman set would leave a bipartite subgraph. Thus, a co-postman set is a complement of a maximal bipartite subgraph. For non-negative weights on the edges, finding a minimum weight co-postman set is equivalent to finding a maximum weight bipartite subgraph, a problem to be $\mathcal{NP}$-complete [13].

5. IDEAL BINARY MATRICES

Define the matrix Q^* to be the 0-1 matrix whose rows are incidence vectors of basic solutions to the binary group problem. Let Q be the corresponding matrix whose rows are incidence vectors of the basic solutions known to the blocking binary group problem. The integer program

$$\begin{aligned} x^* &= 0, 1 \\ Qx^* &\geq 1 \\ cx^* &= z(\min) \end{aligned}$$

is equivalent to the binary group problem. We reemphasize that the objective coefficients c_j are non-negative. The matrix Q is called ideal [4] when the above linear program has integer solutions for all non-negative c. By extension, the original binary group problem and the augmented matrix $[M/b]$ are also called ideal when Q is ideal. A result of Fulkerson [11] says that Q is ideal if and only if Q^* is also ideal, and then the two linear programming polyhedra are called blocking polyhedra or a blocking pair of polyhedra. By extension, we refer to the associated integer programs as being a blocking pair of integer programs. When Q and Q^* are ideal, then the solutions to the linear program with coefficient matrix Q are rows of the matrix Q^* and vice versa.

From results of Khatchian [21], it is known that when Q and Q^* are ideal, then one of their linear programs is polynomially solvable if and only if the other is also polynomially solvable. This fact is due to the equivalence of separation and optimization since separation for one linear program is optimization over the blocking clutter. Referring to these

linear programs as polynomially solvable means that the linear program is polynomially solvable in terms of the input, which is considered to be the augmented matrix $[M/b]$ rather than the linear programming coefficient matrix $[Q/1]$. The latter matrix has, in general, exponentially many rows as compared to the former.

The co-postman problem and the corresponding blocking problem are not, in general, ideal. See [5] where it is shown that they are ideal, and hence a blocking pair of polyhedra, if and only if the graph has no odd K5 minor. It is also known that in the general case i.e. when the graph may have an odd K5 minor, solving the co-postman problem is $\mathcal{NP}$-complete. Thus, solving the integer program with constraints

$$x^* \geq 0, Qx^* \geq 1,$$

is $\mathcal{NP}$-complete. However, the blocking problem is to find a minimum cost odd cycle and is polynomially solvable [13], [19]. Thus, the remark about the linear programs being both polynomially solvable when one is provided one and hence both are ideal does not hold true in the non-ideal case. The linear program for the co-postman problem is polynomially solvable because it is equivalent to separation over odd cycles. However, the integer program is equivalent to finding a maximum cost bipartite subgraph and is not polynomially solvable. For the blocking problem, the linear program is $\mathcal{NP}$-complete because it is equivalent to separation over complements of bipartite subgraphs, i.e. the co-postman problem. However, the integer program of the blocking problem is the minimum odd cycle problem and is polynomially solvable. This is an example for which it is fundamentally easier to solve for an optimum integer answer than to solve for an optimum linear programming answer, provided $\mathcal{P}$ is not equal to $\mathcal{NP}$.

6. FOUR PROBLEMS ON PLANAR GRAPHS

When a graph G is planar, it has a dual graph G^* that can be obtained by the following procedure. For any planar representation of the graph G, there is a node of G^* for each region of G, and any edge of G is in G^* and connects the two nodes of G^* corresponding to the two regions of G that the edge separates in G. For simplicity, assume that G has no cut edges and no loops. Then, G^* will have no cut edges or loops. The duality between G and G^* is that edges in a cycle in G are the edges in a cut in G^* and vice versa. Thus, there is a one-to-one mapping between cycles and cuts, where cycles and cuts are considered as sets of edges.

On planar graphs, there is no real simplification of the Chinese postman problem, but the other three problems do have significant simplifications or, at least, interpretations in terms of the dual graphs. In

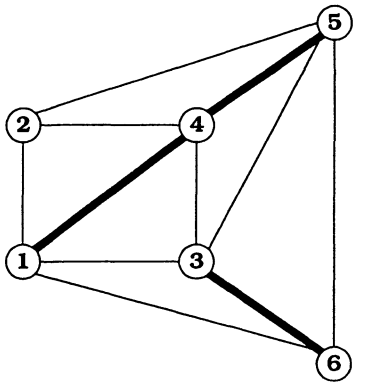


Figure 5.13 Co-postman set.

particular, the co-postman problem is the Chinese postman problem on the dual graph, where a node i^* of the dual graph is odd if the region of the original graph has an odd number of edges in its boundary. The odd cut problem is the odd cycle on the dual graph, and vice versa. Since the Chinese postman problem is polynomially solvable and the co-postman problem is not, the reduction of the co-postman problem to Chinese postman in the planar graph case is significant. Although the odd cycle problem and the odd cut problem are both polynomial, the odd cycle problem seems easier, so that reduction is interesting.

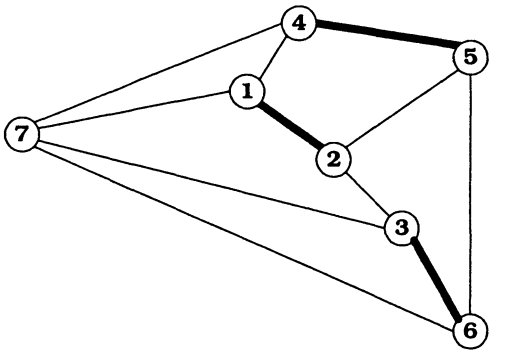


Figure 5.14 Postman set on a dual graph.

References

[1] Barnhart, C., N. Boland, L. Clarke, E.L. Johnson, G.L. Nemhauser, R. Shenoi, "Flight String Models for Aircraft Fleeting and Routing" *Transportation Science* Vol. 32, No. 3, pp. 208-220, August 1998.

[2] Clarke, L., E.L. Johnson, G.L. Nemhauser, J. Zhu, "The Aircraft Rotation Problem", *Annals of Operations Research 69*, pp. 33-46, 1996.

[3] M. Conforti, G. Cornujols, A. Kapoor and K. Vuskovic, "Perfect, Ideal and Balanced Matrices", (1996).

[4] Cornuéjols, G. "Combinatorial Optimization: Packing and Covering", Lecture Notes, Carrnegie Mellon University, May 1999.

[5] Cornuéjols, G. and B. Guenin, "On Ideal Binary Clutters and a Conjecture of Seymour", in preparation.

[6] Edmonds, J. "The Chinese Postman Problem", *Operations Research 13*, Suppl. 1, pp. 373, 1965.

[7] Edmonds, J. and E.L. Johnson, "Matching: A Well-Solved Class of Integer Linear Programs", *Combinatorial Structures and Their Applications*, proceedings from Calgary, Alberta, Canada, June 1969, pp. 89-92.

[8] Edmonds, J. and R.M. Karp, "Theoretical Improvements in Algorithmic Efficiency for Network Flow Problems", *Combinatorial Structures and Their Applications*, proceedings from Calgary, Alberta, Canada, June 1969, pp. 93-96.

[9] Eiselt, H.A. and G. Laporte, "A Historical Perspective on Arc Routing", this volume.

[10] Ford, L.R. and D.R. Fulkerson, Flows in Networks, Princeton University Press, Princeton, N.J., 1962.

[11] Fulkerson, D.R. "Blocking Polyhedra," in B. Harris, ed., *Graph Theory and its Applications* (Academic Press, NY), pp. 93-112, 1970.

[12] Gomory, R.E. "Some Polyhedra Related to Combinatorial Problems," *Linear Algebra and it Applications* 2, pp. 451-558, 1969.

[13] Grötschel, M. and W.R. Pulleyblank, "Weakly Bipartite Graphs", *Operations Research Letters 1* pp. 23-27, 1981.

[14] Grötschel, M. and Z. Win, "A Cutting Plane Algorithm for the windy Postman Problem", *Mathematical Programming* 55 (1992), pp. 339-358.

[15] Guan, M. "Graphic Programming Using Odd or Even Points", *Chinese Mathematics* 1 (1962), pp. 273-377.

[16] Hane, C., C. Barnhart, E.L. Johnson, R. Marsten, G.L. Nemhauser, G. Sigismondi, "The Fleet Assignment Problem: Solving a Large Integer Program", *Mathematical Programming* 70, pp. 211-232, 1995.

[17] Johnson, E.L. "On Binary Group Problems having the Fulkerson Property", *Combinatorial Optimization* B. Simeone (ed.), Springer-Verlag, Berlin and Heidelberg, pp. 57-112, 1989.

[18] Johnson, E.L. and J. Edmonds, "Matchings, Euler Tours and the Chinese Postman", *Mathematical Programming*, Vol. 5, pp. 88-124, 1973.

[19] Johnson, E.L. and G. Gastou, "Binary Group and Chinese Postman Polyhedra", *Mathematical Programming*, Vol. 5, pp. 88-124, 1973.

[20] Johnson, E.L. and S. Mosterts, "On Four Problems in Graph Theory," *SIAM Journal on Algebraic and Discrete Methods*, Vol. 2, pp. 163-185, 1987.

[21] Khachian, L.G. "A Polynomial Algorithm in Linear Programming", *Soviet Mathematics Doklady* 20 pp. 191-194, 1979.

[22] Lehman, A. "On the Width-Length Inequality", *Mathematical Programming* 17 (1979), pp. 403-417.

[23] Nobert, Y. and J.-C. Picard, "An Optimal Algorithm for the Mixed Chinese Postman Problem", *Networks* Vol. 27 (1996), pp. 95-108.

[24] Padberg, M. and M. Rao, "Odd Minimum Cut-Sets and B-Matchings", *Math. of Operations Research*, Vol. 7 No. 1 (1982), pp. 67-80.

[25] Roberts, F.S. and J. Spencer, "A Characterization of Clique Graphs", *Combinatorial Structures and Their Applications*, proceedings from Calgary, Alberta, Canada, June 1969, pp. 367-368.

[26] Seymour, P.D. "Matroids with the Max-Flow Min-Cut Property", *Journal of Combinatorial Theory Series B* 23 (1977), pp. 189-222.

[27] Stone, A.H. "Some Combinatorial Problems in General Topology", *Combinatorial Structures and Their Applications*, proceedings from Calgary, Alberta, Canada, June 1969, pp. 413-416.

[28] Tutte, W.T. "Lectures on Matroids", *Journal of Research of the National Bureau of Standards Section B* 69 pp. 1-47, 1965.

II

SOLUTIONS

Chapter 6

POLYHEDRAL THEORY FOR ARC ROUTING PROBLEMS

Richard W. Eglese
Lancaster University

Adam N. Letchford
Lancaster University

1. Introduction 199
2. The Basics of Polyhedral Theory 200
3. The Routing Problems Defined 203
4. Variants of the Chinese Postman Problem 205
   4.1 The CPP 205
   4.2 The DCPP 205
   4.3 The MCPP 206
   4.4 The WPP 208
5. Variants of the Rural Postman Problem 209
   5.1 The RPP 209
   5.2 The GRP 212
   5.3 The DRPP 216
   5.4 The MRPP 217
6. The Capacitated Arc Routing Problem 219
   6.1 Preliminaries 219
   6.2 Sparse Formulations of the CARP 220
   6.3 The Dense and Supersparse Formulations of the CARP 224
7. Conclusions 226

1. INTRODUCTION

As explained in Chapter 4, most realistic Arc Routing Problems are known to be $\mathcal{NP}$-hard. Therefore we can expect that there will be certain instances which are impossible to solve to optimality within a reasonable time. However, this does not mean that *all* instances will be impossible to solve. It may well be that an instance which arises in practice has some structure which makes it amenable to solution by an optimization

algorithm. Since, in addition, significant costs are often involved in real-world instances, research into devising optimization algorithms is still regarded as important.

At the time of writing, the most promising optimization algorithms for $\mathcal{NP}$-hard problems are based on the so-called *branch-and-cut* method (see Padberg & Rinaldi, 1991 and also Chapter 7 of this book). The key to producing an effective branch-and-cut algorithm for a particular class of problems is to have a good understanding of certain polyhedra which are associated with those problems.

This chapter gives a self-contained introduction to the main ideas of polyhedral theory, followed by a state-of-the-art survey of the known polyhedral results for Arc Routing. The most promising integer programming formulations of various Arc Routing Problems are reviewed and the known valid inequalities and facets of the associated polyhedra are presented. Note, however, that we are not here concerned with the *algorithmic* implications of polyhedral theory. These are discussed in detail in Chapter 7.

The outline of the chapter is as follows. In Section 2, the basic ideas and notation involved in polyhedral theory are summarized. In Section 3, the definitions of a number of routing problems are briefly reviewed. The following two sections review the polyhedral theory for single-vehicle variants of the Chinese Postman Problem and the Rural Postman Problem, respectively. Section 6 does the same for the Capacitated Arc Routing Problem. Some concluding comments are made in Section 7.

Other surveys about Arc Routing which include material on polyhedral theory can be found in Assad & Golden (1995) and Eiselt et al. (1995a, b).

2. THE BASICS OF POLYHEDRAL THEORY

This section draws on material in Weyl (1935), Grötschel & Padberg (1985) and Nemhauser & Wolsey (1988).

Like many other combinatorial optimization problems, the majority of Arc Routing Problems can be formulated as problems of the form:

$$\min\{c^T x : x \in \mathcal{S}\} \tag{6.1}$$

where $x = \{x_1, \ldots, x_n\} \in \Re^n$ is a vector of decision variables, $c = \{c_1, \ldots, c_n\} \in \Re^n$ is a vector of objective function coefficients (i.e., costs) and $\mathcal{S} \subset \Re^n$ is a set of *feasible solutions*. Often, but not always, $\mathcal{S}$ is defined by an explicit system of linear inequalities with integer coefficients,

together with an integrality condition. That is, there is an integer m, a matrix $A \in Z^{mn}$ and a vector $b \in Z^m$ such that

$$\mathcal{S} = \{x \in Z^n : Ax \leq b\}. \tag{6.2}$$

In this case, (6.1) is called an *Integer Linear Program* (ILP). Note that constraints of the 'greater than or equal to' form can be accommodated in (6.2) if they are multiplied by minus one; equations can also be accommodated since they can be written as two inequalities. Note also that m may be very large; often it is exponential in n.

The key to the polyhedral approach is the observation that the feasible solutions to (6.1) (i.e., the members of $\mathcal{S}$) are vectors in the Euclidean space $\Re^n$. It is then easy to define a polyhedron associated with a given $\mathcal{S}$. In order to show how this can be done, it is first necessary to give some formal definitions.

A set $H \subset \Re^n$ is called a *half-space* if there exists a vector $a \in \Re^n$ and a scalar $a_0 \in \Re$ such that $H = \{x \in \Re^n : a^T x \leq a_0\}$. A set $P \subset \Re^n$ is called a *polyhedron* if it is the intersection of finitely many half-spaces. A polyhedron which is bounded (i.e., not of infinite volume) is called a *polytope*. Now suppose that $x^1, \ldots, x^k \in \Re^n$ are vectors and $\lambda_1, \ldots, \lambda_k$ are scalars. A vector of the form $\lambda_1 x^1 + \cdots + \lambda_k x^k$ is called an *affine combination* of $x^1, \ldots, x^k$ if $\sum_{i=1}^k \lambda_i = 1$. An affine combination is also called a *convex* combination if $\lambda_i \geq 0$ for all i. Given some $\mathcal{S} \subset \Re^n$ with $\mathcal{S} \neq \emptyset$, the *affine* (respectively, *convex*) *hull* of $\mathcal{S}$, denoted by $\text{aff}(\mathcal{S})$ (respectively, $\text{conv}(\mathcal{S})$), is the set of all affine (convex) combinations of finitely many vectors in $\mathcal{S}$.

Note that for any $\mathcal{S}$, $\text{conv}(\mathcal{S}) \subseteq \text{aff}(\mathcal{S})$ holds. Also, $\text{aff}(\mathcal{S})$ is always a polyhedron. The situation is a little more complicated for $\text{conv}(\mathcal{S})$. It can be shown that, when $|\mathcal{S}|$ is finite, $\text{conv}(\mathcal{S})$ is always a polyhedron. When $|\mathcal{S}|$ is not finite, however, $\text{conv}(\mathcal{S})$ can fail to be a polyhedron (see, e.g., Queyranne & Wang, 1992). Fortunately, however, the sets $\mathcal{S}$ which will be of concern to us in this chapter are 'well-behaved', in that $\text{conv}(\mathcal{S})$ will always be a polyhedron.

Another important concept is that of the *dimension* of a polyhedron. This is defined using the idea of *affine independence*. A set of vectors is affinely independent if no member of the set is an affine combination of the others. The dimension of a polyhedron P, denoted by $\dim(P)$, is then defined as the maximum number of affinely independent vectors in P. Note that, if P is defined in $\Re^n$, then $\dim(P) \leq n$ holds. If equality holds, P is said to be *full-dimensional*.

Given these definitions, we can now associate a polyhedron with any instance of a combinatorial optimization problem; namely, the polyhedron conv($\mathcal{S}$), where $\mathcal{S}$ is the set of feasible solutions to (6.1). Because the vectors in $\mathcal{S}$ are assumed to be integral, conv($\mathcal{S}$) will be a polyhedron with integral vertices. We will let P_I denote such an integral polyhedron. The whole aim of the polyhedral approach is to find 'good' descriptions of P_I for various combinatorial optimization problems. In order to define what is meant by a 'good' description, we need some further definitions.

An inequality $a^T x \leq a_0$ is *valid* for P_I if $P_I \subseteq \{x \in \Re^n : a^T x \leq a_0\}$. The set $F = P_I \cap \{x \in \Re^n : a^T x = a_0\}$ is called the *face induced by* $a^T x \leq a_0$ *on* P_I. Note that F is also a polyhedron. A valid inequality is *supporting* if $F \neq \emptyset$. If $F = P_I$, then $a^T x = a_0$ is said to be an *implicit equation* of P_I. The face F of P_I induced by a valid inequality $a^T x \leq a_0$ is called a *facet* if $F \neq P_I$ and there is no other valid inequality which induces a face F' of P_I such that F is strictly contained in F'. Note that $\dim(F) = \dim(P_I)$ - 1 when F is a facet.

A 'best possible' complete linear description of P_I must therefore include all implicit equations and facet-inducing inequalities. Of course, if there are implicit equations, then any one of the equations or inequalities can be written in an infinite number of ways. All that is really needed, however, is one representative of each non-equivalent facet-inducing inequality, along with a minimal set of equations which identifies the affine hull of P_I.

When an instance of an $\mathcal{NP}$-hard combinatorial optimization problem is extremely small, it may be possible to find such a *complete linear description* of the associated P_I. However, finding a complete linear description of P_I for instances of realistic size is hopelessly difficult. Although the number of equations needed to identify the affine hull is small (only n - $\dim(P_I)$), it often happens that P_I has an immense number of facets. An example will drive this point home. Readers who are familiar with the well-known *Symmetric Traveling Salesman Problem* (STSP) will regard an STSP instance defined on a graph with only 10 vertices as trivial. Nevertheless, Christof & Reinelt (1996) have shown that over 5.1 x 10^{10} inequalities are needed to describe P_I in this case (together with 10 equations, one for each vertex in the graph).

A further negative result comes from Karp & Papadimitriou (1982), who showed that, if a combinatorial optimization problem is $\mathcal{NP}$-hard, then the problem of deciding whether or not an inequality is valid for the associated P_I is itself $\mathcal{NP}$-hard.

Despite these negative results, however, it remains true that large classes of valid, supporting and even facet-inducing inequalities, along with implicit equations, are known for many important $\mathcal{NP}$-hard problems. This will become clear in the following subsections. Such *partial* linear descriptions of polyhedra provide the basis for very effective optimization algorithms, see Chapter 7.

3. THE ROUTING PROBLEMS DEFINED

In this section, we give the definitions of twelve different fundamental routing problems for which polyhedral studies have been conducted. These will be examined in subsequent sections.

First, we consider undirected, single-vehicle problems. Given a connected, undirected graph G with vertex set V and (undirected) edge set E, a cost c_e for each edge $e \in E$, a set $V_R \subseteq V$ of *required vertices* and a set $E_R \subseteq E$ of *required edges*, the *General Routing Problem* (GRP) is the problem of finding a minimum cost vehicle route ('tour') passing through each $v \in V_R$ and each $e \in E_R$ at least once (Orloff, 1974).

The GRP contains a number of other known problems as special cases. When $E_R = \emptyset$, the GRP reduces to the *Steiner Graphical Traveling Salesman Problem* (SGTSP) (Cornuéjols et al., 1985), also called the *Road Traveling Salesman Problem* by Fleischmann (1985). On the other hand, when $V_R = \emptyset$, the GRP reduces to the *Rural Postman Problem* (RPP) (Orloff, 1974). When $V_R = V$, the SGTSP in turn reduces to the *Graphical Traveling Salesman Problem* or GTSP (Cornuéjols et al., 1985). Similarly, when $E_R = E$, the RPP reduces to the *Chinese Postman Problem* or CPP (Guan, 1962; Edmonds, 1963).

The CPP can be solved in polynomial time by reduction to a matching problem (Christofides, 1973; Edmonds & Johnson, 1973), but the RPP, GTSP, SGTSP and GRP are all $\mathcal{NP}$-hard. The GTSP and SGTSP were proved to be $\mathcal{NP}$-hard by Cornuéjols et al. (1985) and the RPP and GRP were proved to be $\mathcal{NP}$-hard by Lenstra & Rinnooy-Kan (1976).

Now we consider problems in which directed arcs are allowed. Given a connected, mixed graph G with vertex set V, (undirected) edge set E, (directed) arc set A, a cost c_e for each edge $e \in E$, a cost c_a for each arc $a \in A$, a set $E_R \in E$ of *required edges* and a set $A_R \in A$ of *required arcs*, the *Mixed Rural Postman Problem* (MRPP) is the problem of finding a minimum cost vehicle route passing through each $e \in E_R$ and each $a \in A_R$ at least once (Corberán, Romero & Sanchis, 1997; Romero, 1997).

Like the GRP, the MRPP also contains a number of other problems as special cases. When $A = \emptyset$, the MRPP reduces to the RPP mentioned above. When $E = \emptyset$, the MRPP reduces to the *Directed Rural Postman Problem* or DRPP (Christofides et al., 1986). Alternatively, it may be that $E_R = E$ and $A_R = A$, in which case the MRPP reduces to the *Mixed Chinese Postman Problem* or MCPP (Edmonds & Johnson, 1973). When $E_R = \emptyset$, the MCPP in turn reduces to the *Directed Chinese Postman Problem* or DCPP (Edmonds & Johnson, 1973).

The DCPP can be solved in polynomial time by reduction to a transportation problem (Edmonds & Johnson, 1973), but the MRPP, DRPP and MCPP are all $\mathcal{NP}$-hard. The MCPP (and therefore MRPP) was proved to be $\mathcal{NP}$-hard by Papadimitriou (1976) and the DRPP was proved to be $\mathcal{NP}$-hard by Christofides et al. (1986).

Another single-vehicle problem is known as the *Windy Postman Problem* or WPP. It is a generalization of the CPP which allows for the possibility that the cost of traversing an edge in one direction may differ from the cost of traversing the edge in the opposite direction. The WPP combines features of both undirected and directed problems. In fact, it is easy to show that it contains the MCPP as a special case. It is therefore $\mathcal{NP}$-hard, as noted by Guan (1984), although it is polynomially solvable in certain special cases (Guan, 1984; Win, 1987, 1989).

Now we consider two undirected multi-vehicle problems. The *Capacitated Arc Routing Problem* or CARP is a generalization of the RPP in which $k \geq 1$ identical vehicles are available, each of *capacity* $Q > 0$. One particular vertex is called the *depot* and each required edge has an integral *demand* $q_i \geq 0$. The task is to find a minimum cost set of k feasible routes, each one starting and ending at the depot. (A route is feasible if the sum of the demands on the route do not exceed Q.) When $E_R = E$, the CARP reduces to the *Capacitated Chinese Postman Problem* or CCPP (Win, 1987).

Since the CARP is at least as difficult as the RPP, it is $\mathcal{NP}$-hard (Golden & Wong, 1981). Perhaps more surprisingly, the CCPP is also $\mathcal{NP}$-hard. In fact, Golden & Wong (1981) showed that it is $\mathcal{NP}$-hard to find a *0.5-approximate* solution to the CCPP (i.e., a solution which has a cost less than 1.5 times the cost of the optimal solution).

4. VARIANTS OF THE CHINESE POSTMAN PROBLEM

4.1. THE CPP

Since the CPP can be solved in polynomial time, we might expect that the associated polyhedra have a simple description. This is indeed the case. We let the general integer variable x_e represent the number of times that edge e is traversed without being serviced. (That is, x_e represents the number of copies of edge e which will be added to E in order to make G Eulerian.) For each $S \subset V$, we let $\delta(S)$ denote the set of edges, commonly called the *edge cutset*, which have one end-vertex in S and one end-vertex in $V \setminus S$. When $S = \{i\}$, we write $\delta(i)$ rather than $\delta(\{i\})$ for brevity. Finally, for any $F \subset E$, we let $x(F)$ denote $\sum_{e \in F} x_e$. Then our set S of feasible solutions is defined as

$$x(\delta(i)) \equiv |\delta(i)| \pmod 2, \ (\forall i \in V) \tag{6.3}$$

$$x_e \geq 0, \ (\forall e \in E) \tag{6.4}$$

$$x \in Z^{|E|} \tag{6.5}$$

Note that the conditions (6.3) are *not* in the form of linear inequalities. They do however imply the validity of the following *odd-cut* (or *blossom*) inequalities (Edmonds & Johnson, 1973):

$$x(\delta(S)) \geq 1, \ \ (\forall S \subset V : |\delta(S)| \text{ odd}). \tag{6.6}$$

To see why these inequalities are valid, note that the vehicle must cross any given edge cutset an even number of times. Hence, if $S \subset V$ is such that $|\delta(S)|$ is odd, then the vehicle must cross the cutset at least once without servicing. Note that the number of odd-cut inequalities can increase exponentially in the size of the graph G.

Edmonds & Johnson (1973) show that P_I is completely described by the odd-cut inequalities (6.6) and the *non-negativity* inequalities (6.4). That is, the conditions (6.3) and (6.5) are unnecessary.

Corberán & Sanchis (1994) have shown that an odd-cut inequality is facet-inducing if and only if the subgraphs induced in G by S and $V \setminus S$ are each connected; they also showed that a non-negativity inequality induces a facet if and only if e is not a *cut-edge* (an edge whose removal disconnects the graph).

4.2. THE DCPP

Since the DCPP can be solved in polynomial time, we might expect that the associated polyhedra have a simple description, just as in the case of the CPP. This is indeed the case. We let the general integer

variable x_a represent the number of times that arc a is traversed without being serviced. This is analogous to the formulation for the CPP given in the previous subsection, in that x_a represents the number of copies of arc a which will be added to A in order to make G Eulerian. For any $i \in V$, we let $\delta^+(i)$ denote the set of arcs which leave i and $\delta^-(i)$ denote the set of arcs which enter i. Finally, we let $b(i)$ for each $i \in V$ denote $|\delta^-(i)| - |\delta^+(i)|$, the so-called 'unbalance' of i. Note that $b(i)$ may be positive, negative or zero. The set $\mathcal{S}$ is then given by (Edmonds & Johnson, 1973):

$$x(\delta^+(i)) - x(\delta^-(i)) = b(i), \ (\forall i \in V) \tag{6.7}$$

$$x_a \geq 0, \ (\forall a \in A) \tag{6.8}$$

$$x \in Z^{|A|} \tag{6.9}$$

Equations (6.7), which we will call *balance* equations, ensure that the vehicle leaves each vertex as many times as it enters. They describe the affine hull of P_I and in fact it can be shown that only $|V| - 1$ of them are needed (that is, any one of them can be deleted as redundant). It is also not hard to show that the non-negativity inequalities (6.8) induce facets of P_I under mild conditions.

Edmonds & Johnson (1973) showed that P_I is completely described by the balance equations and non-negativity inequalities. Thus, the integrality condition (6.9) is unnecessary.

4.3. THE MCPP

When considering how to tackle the MCPP, it appears at first sight that in addition to an integer variable for each arc, it will be necessary to have *two* integer variables for each edge (to indicate the number of times the edge is traversed in either direction). Indeed, formulations of this kind have appeared in the literature (e.g., Kappauf & Koehler, 1979; Christofides et al., 1984; Grötschel & Win, 1992; Ralphs, 1993). However, it is possible to use only one variable per edge (Nobert & Picard, 1996), as we now explain.

Each MCPP solution is defined by a family (i.e., a set with possible repeated elements) of edges and arcs which constitute what might be called a *mixed Eulerian multigraph*. In Ford & Fulkerson (1962), it is proven that a mixed multigraph is Eulerian if and only if it is *even* and *balanced*. The first condition means that the number of arcs and edges incident on any vertex is an even integer. To explain the second condition, we extend our notation a little. Given any $S \subset V$, let $\delta(S)$ be the set of all *edges* crossing from S to $V \setminus S$, $\delta^+(S)$ the set of all

arcs leaving S, and let $\delta^-(S)$ be the set of all *arcs entering* S. The condition that the multigraph be *balanced* means that, for any $S \subset V$, $|\delta^-(S)| - |\delta^+(S)| \leq |\delta(S)|$ holds. Ford & Fulkerson also provide a simple algorithm to find a tour of a mixed Eulerian multigraph.

Now let the general integer variable x_e (respectively, x_a) represent the number of times that edge e (respectively, arc a) is traversed without being serviced. That is, each variable represents the number of times a particular edge or arc will be added to $E \cup A$ to make G Eulerian. Also, for any $S \subset V$ let $b(S) = |\delta^-(S)| - |\delta^+(S)| - |\delta(S)|$, the so-called 'unbalance' of S. Note that this definition of 'unbalance' is a generalization of the definition given in the previous subsection. Finally, let $\delta^*(S)$ denote $\delta(S) \cup \delta^+(S) \cup \delta^-(S)$. Then the set $\mathcal{S}$ of feasible solutions is defined by the following conditions:

$$x(\delta^*(i)) \equiv |\delta^*(i)| \pmod 2, \ (\forall i \in V) \tag{6.10}$$

$$x(\delta^+(S)) + x(\delta(S)) - x(\delta^-(S)) \geq b(S), \ (\forall S \subset V) \tag{6.11}$$

$$x_e \geq 0, \ (\forall e \in E) \tag{6.12}$$

$$x_a \geq 0, \ (\forall a \in A) \tag{6.13}$$

$$x \in Z^{|E \cup A|} \tag{6.14}$$

The system of congruences (6.10) enforces the condition that the associated mixed multigraph be even. Similarly, the *balanced set* inequalities (6.11) enforce the condition that the multigraph be balanced. Finally, (6.12) and (6.13) are just non-negativity inequalities. Obviously, the inequalities (6.11) - (6.13) are valid for the associated P_I.

It might be thought that a 'switched' version of (6.11) would also be needed for each S, i.e., an inequality of the form $x(\delta^-(S)) - x(\delta^+(S)) + x(\delta(S)) \geq -b(S)$. However, this is easily shown to be equivalent to the balanced set inequality associated with $V \setminus S$. Using this fact, it is also possible to show that P_I is not full-dimensional in general (though this is not noted explicitly by Nobert & Picard). Suppose that $S \subset V$ is such that $|\delta(S)| = \emptyset$. Then, the balanced set inequality reduces to $x(\delta^+(S)) - x(\delta^-(S)) \geq b(S)$, whereas, the balanced set inequality for $V \setminus S$ can be written as $x(\delta^+(S)) - x(\delta^-(S)) \leq b(S)$. Thus, $x(\delta^+(S)) - x(\delta^-(S)) = b(S)$ is an implicit equation of P_I. We could call equations of this type *balance equations*. It is easily shown that they are a generalization of the balance equations (6.7) for the DCPP.

One further class of valid inequalities is presented in Nobert & Picard (1996). They note that the condition (6.10) implies that the following *blossom* inequalities are valid for P_I:

$$x(\delta^*(S)) \geq 1, \ \ (\forall S \subset V : |\delta^*(S)| \text{ odd}). \tag{6.15}$$

These are a simple generalization of the odd-cut (blossom) inequalities for the CPP.

The algorithm of Nobert & Picard (1996), based upon the formulation given here, clearly outperformed algorithms based upon formulations with two variables per edge.

In Subsection 5.4, we review a study by Corberán, Romero & Sanchis (1997) about polyhedra associated with the Mixed Rural Postman Problem (MRPP). Since the MRPP contains the MCPP as a special case, many of the results of Corberán, Romero & Sanchis also apply to the MCPP. This implies, for example, that the affine hull of P_I is described by one balance equation for each connected component in the subgraph of G induced by the (required) edges and that exactly one of these equations is redundant. It also yields necessary and sufficient conditions for the non-negativity, blossom and balanced set inequalities to induce facets of P_I. The details are not given here, for brevity.

We would like to close this subsection with a question for future research:

Research Problem: For what (mixed) graphs G (if any) does the polyhedron defined by (6.11) - (6.15) contain integer extreme points which do *not* represent feasible tours?

4.4. THE WPP

Although the WPP is notionally defined on an undirected graph, we will assume that the underlying graph is directed, with two arcs (i,j), (j,i) going in opposite directions for each edge $e = \{i,j\}$ in the original graph. Then, we can define a general integer variable x_{ij} for each arc in the directed graph, representing the number of times the vehicle travels in that particular direction. The set $\mathcal{S}$ of feasible solutions is then given by:

$$x(\delta^+(i)) - x(\delta^-(i)) = 0, \ (\forall i \in V) \tag{6.16}$$

$$x_{ij} + x_{ji} \geq 1, \ (\forall \{i,j\} \in E) \tag{6.17}$$

$$x_{ij}, x_{ji} \geq 0, \ (\forall \{i,j\} \in E) \tag{6.18}$$

$$x \in Z^{2|E|} \tag{6.19}$$

The associated polyhedron is examined in Win (1987) and Grötschel & Win (1988). They show that the *balance equations* (6.16) describe the affine hull of P_I and that only $|V| - 1$ of them are needed (any one of them can be deleted as redundant). They also show that the inequalities (6.17), which ensure that each edge is traversed at least once, together

with the non-negativity inequalities (6.18), are facet-inducing.

Grötschel and Win also show that the following *odd-cut* inequalities are valid and that they induce facets under mild conditions:

$$x(\delta^+(S)) + x(\delta^-(S)) \geq |\delta(S)| + 1, \ (\forall S \subset V : |\delta(S)| \text{ odd}). \tag{6.20}$$

They also mention that the odd-cut inequality for any S can be re-written, using the balance equations, in a variety of other forms, such as:

$$x(\delta^+(S)) \geq \frac{1}{2}(|\delta(S)| + 1)$$

or

$$x(\delta^-(S)) \geq \frac{1}{2}(|\delta(S)| + 1).$$

We close this section by mentioning a result of Win (1987, 1989), who showed that the polyhedron defined by (6.16), (6.17) and (6.18) is *half-integral*. That is, all of its extreme points have components that are an integral multiple of one-half. He also showed that this polyhedron is integral if and only if the original graph is Eulerian. Ralphs (1993) gave a similar result for an analogous formulation for the MCPP.

5. VARIANTS OF THE RURAL POSTMAN PROBLEM

5.1. THE RPP

An integer programming formulation for the RPP was given in Christofides et al. (1981), but the associated polyhedron was not examined in detail. This was done in Corberán & Sanchis (1994). In the Corberán & Sanchis (1994) formulation, x_e represents the number of times e is traversed (if $e \notin E_R$), or one less than this number (if $e \in E_R$). That is, x_e represents the number of copies of e which will be added to E_R in order to form an Eulerian multigraph. Now we let V_R denote the set of vertices incident on at least one required edge (these vertices are also 'required' since the vehicle must travel through them) and let $\delta_R(S)$ denote $\delta(S) \cap E_R$. The set $\mathcal{S}$ is then defined by

$$x(\delta(S)) \geq 2, \forall S \subset V : \begin{cases} \delta_R(S) = \emptyset, \\ S \cap V_R \neq \emptyset, \\ V_R \setminus S \neq \emptyset \end{cases} \tag{6.21}$$

$$x(\delta(i)) \equiv |\delta_R(i)| \pmod 2, \forall i \in V \tag{6.22}$$

$$x_e \geq 0, \ (\forall e \in E) \tag{6.23}$$

$$x \in Z^{|E|} \tag{6.24}$$

Corberán & Sanchis (1994) show that the associated polyhedron is full-dimensional and unbounded. Many classes of valid inequalities and facets are known. In this subsection, we outline those presented in Corberán & Sanchis (1994). The other known inequalities were discovered in the context of the *General Routing Problem* and will be described in the next subsection.

Constraints (6.21), called *connectivity inequalities*, ensure that the route is connected. They induce facets of P_I if and only if the subgraphs induced in G by S and $V \setminus S$ are connected. The non-negativity inequalities (6.23) induce facets if and only if e is not a cut-edge. Another simple class of valid inequalities is given by the following *R-odd cut* inequalities:

$$x(\delta(S)) \geq 1, \quad (\forall S \subset V : |\delta_R(S)| \text{ odd}) \tag{6.25}$$

These generalize the odd-cut (blossom) inequalities for the CPP and are valid for the same reason. Like connectivity inequalities, R-odd cut inequalities induce facets if and only if the subgraphs induced in G by S and $V \setminus S$ are each connected (Corberán & Sanchis, 1994).

In order to present the remaining inequalities, we will need some more definitions. Consider the (generally disconnected) subgraph of G obtained by deleting all non-required edges from G. We call a connected component of this subgraph an *R-component.* Also, given two disjoint subsets A and B of V, we let $E(A : B)$ denote the set of edges in E with one end-vertex in A and one in B and $E_R(A : B)$ denote $E(A : B) \cap E_R$.

A *K-component* (K-C) *configuration* is a partition $\{V_0, \ldots V_K\}$ of V, with $K \geq 3$, such that

- $V_1, \ldots V_{K-1}$ and $V_0 \cup V_K$ are clusters of node sets of one or more R-components,
- $|E_R(V_0 : V_K)|$ is positive and even,
- $E(V_i : V_{i+1}) \neq \emptyset$ for $i = 0, \ldots, K-1$.

Figure 6.1 shows a K-C configuration: the filled circles represent the V_i, the bold lines represent the required edges crossing from V_0 to V_K and the plain lines represent the non-required edges which must be present. Associated with a K-C configuration is a *K-C inequality*, which can be written as:

$$\sum_{p=0}^{K-1} \sum_{q=p+1}^{K} (q-p)x(E(V_p : V_q)) - 2x(E(V_0 : V_K)) \geq 2(K-1) \tag{6.26}$$

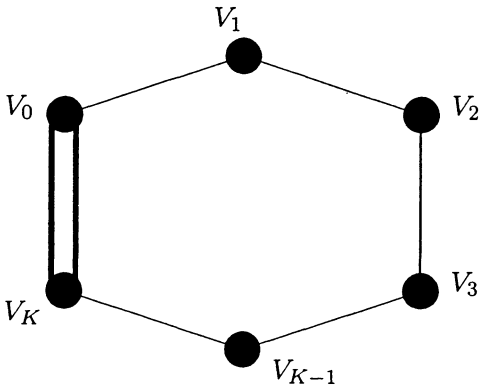


Figure 6.1 K-C configuration.

K-C inequalities induce facets when certain mild connectivity assumptions are met (Corberán & Sanchis, 1994).

The last class of inequalities presented in Corberán & Sanchis (1994) are known as *GTSP-type* inequalities. Suppose we have an RPP instance and let m be the number of R-components in G. Consider a partition of V into sets $S_1, \ldots, S_m$ such that $S_i \cap V_R \neq \emptyset$ and $\delta_R(S_i) = \emptyset$ for $i = 1, \ldots, m$. Let G_s be the (multi)graph obtained from G by shrinking each S_i into a single vertex and eliminating loops. That is, G_s has m vertices. It is now possible to define a GTSP instance on G_s and Corberán & Sanchis (1994) show that any (non-trivial) facet-inducing inequality for the resulting GTSP polyhedron is also facet-inducing for the polyhedron associated with the original RPP instance.

This is a powerful result, because GTSP polyhedra have been widely studied and many classes of facet-inducing inequalities are known for them. Space does not permit a review here, so the reader is directed to the surveys Jünger, Reinelt & Rinaldi (1995, 1997).

We close this subsection by mentioning some recent results due to Ghiani & Laporte (1997). Suppose that $V = V_R$ (a problem in which $V \neq V_R$ can be easily transformed into one in which $V = V_R$, see e.g., Christofides et al., 1981). Consider the graph obtained by 'shrinking' each R-component down to a single vertex (but *not* merging parallel edges). Find a minimum cost spanning tree T on this shrunk graph. Then Ghiani & Laporte show that there exists an optimal RPP solution such that $x_e \leq 2$ for each $e \in T$ and $x_e \leq 1$ for each $e \notin T$. This leads

to a pure 0-1 formulation for the RPP, with one binary variable for each edge, where each $e \in T$ is split into two parallel edges with the same cost.

For the Ghiani & Laporte formulation, P_I is a polytope (that is, a bounded polyhedron). This polytope is rather different from the unbounded polyhedron studied by Corberán & Sanchis. For one thing, it is typically not full-dimensional (this is easily shown by considering sets $S \subset V$ with $1 \leq |\delta(S)| \leq 3$). For another, there are new valid inequalities. For example, let $S \subset V$ and $F \subset \delta(S)$ be such that $|\delta_R(S)| + |F|$ is odd. Then the inequalities

$$x(\delta(S) \setminus F) \geq x(F) - |F| + 1 \tag{6.27}$$

are valid for the polytope but not for the unbounded polyhedron. These inequalities proved useful in the branch-and-cut algorithm of Ghiani & Laporte.

It is not immediately obvious which approach to the RPP is 'best', whether that of Corberán & Sanchis or that of Ghiani & Laporte. The authors regard this as an interesting theoretical and empirical problem. Of course, any facet-inducing inequalities for the unbounded polyhedron are also valid for the polytope. However, they cannot be guaranteed to induce facets any longer.

5.2. THE GRP

When tackling the GRP, it is helpful to assume (w.l.o.g.) that the end-vertices of each required edge are in V_R. Define for each $e \in E$ a general integer variable x_e, representing the number of times e is traversed (if $e \notin E_R$), or one less than this number (if $e \in E_R$). Then, the system (6.21) - (6.23) given in the last section defines $\mathcal{S}$ in the case of the GRP as well as in the special case of the RPP. Also, the connectivity, R-odd cut, K-C and GTSP-type inequalities are valid for the GRP as well as the RPP (Corberán & Sanchis, 1998). A number of new results are also known for the GRP and are presented in the remainder of this subsection. It should be noted that these results are also new even when specialized to the RPP.

In Corberán & Sanchis (1998), the K-C inequalities were generalized to give the *honeycomb* inequalities, which also define facets if certain mild connectivity assumptions are met. A *honeycomb configuration* is a partition of V into sets S_i such that:

- for all i, $| \delta(S_i) \setminus \delta_R(S_i) | \neq \emptyset$ and $| \delta_R(S_i) |$ is even or zero;
- there are at least two values i such that $\delta_R(S_i) \neq \emptyset$;

- there are at least two values i such that $\delta_R(S_i) = \emptyset$;
- there is a set T of non-required edges crossing between the S_i forming a tree spanning the S_i, such that each member of T crosses between sets S_i and S_j with $E_R(S_i : S_j) = \emptyset$.

Many, but not all, honeycomb configurations can be formed by 'gluing' K-C configurations together by identifying edges (Corberán & Sanchis, 1998).

The coefficients in the associated honeycomb inequality are defined as follows, apart from one exception mentioned in the next paragraph. Let α_e denote the coefficient of x_e in the honeycomb inequality. Then $\alpha_e = 0$ if and only if $e \notin \delta(S_i)$ for all i and $\alpha_e = 1$ for all $e \in T$. The coefficient α_e of any other crossing non-required edge e is equal to the number of edges traversed in T to get from one end-vertex of e to the other. For the required edges crossing between the S_i, the coefficient is 2 units less.

The exception is that, for certain complex honeycomb configurations, some of the crossing non-required edges which are not in T may have a smaller coefficient. In such cases, the α_e must be computed sequentially (by a so-called *sequential lifting* procedure).

The honeycomb inequality is then:

$$\sum_{e \in E} c_e x_e \geq 2(K-1) \tag{6.28}$$

Figure 6.2 shows two honeycomb configurations. The bold lines represent edges in $\delta_R(V_i)$ for some i and the plain lines represent edges in the spanning tree. In the corresponding inequalities, all edges shown have a coefficient equal to 1. The rhs is 6 in both cases.

In Letchford (1997a, b), the facet-inducing *path-bridge* (PB) inequalities were introduced. Like honeycomb inequalities, PB inequalities are a generalization of K-C inequalities. However, they are a generalization in a different direction. They are defined in terms of an associated *path-bridge* (PB) *configuration.* Suppose $p \geq 1$ and $b \geq 0$ are integers such that $p + b \geq 3$ and odd. Let $n_i \geq 2$ for $i = 1, \ldots, p$ also be integers. A PB configuration is (see Figure 6.3) a partition of V into vertex sets A, Z and V_j^i for $i = 1, \ldots, p$, $j = 1, \ldots, n_i$ with the following properties:

- each V_j^i is a cluster of one or more R-sets,
- $|E_R(A : Z)| = b$,

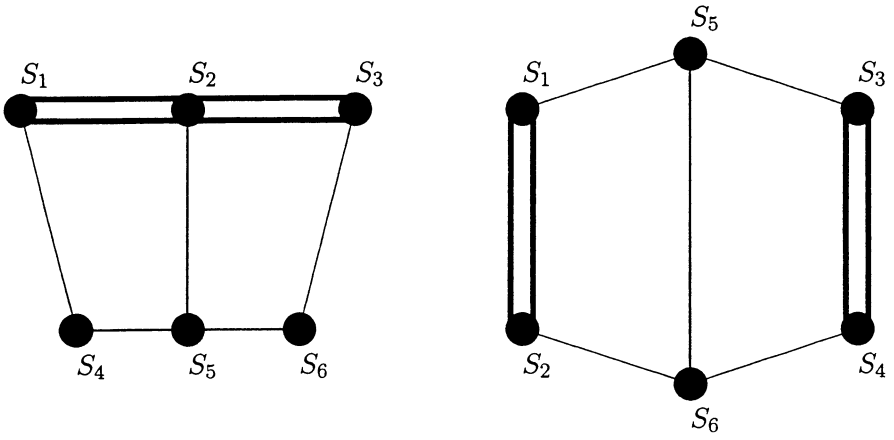


Figure 6.2 Two honeycomb configurations.

- $E(A : V_1^i) \neq \emptyset$ and $E(V_{n_i}^i : Z) \neq \emptyset$ for $i = 1, \ldots, p$,
- $E(V_j^i : V_{j+1}^i) \neq \emptyset$ for $i = 1, \ldots, p$ and $j = 1, ..., n_i - 1$.

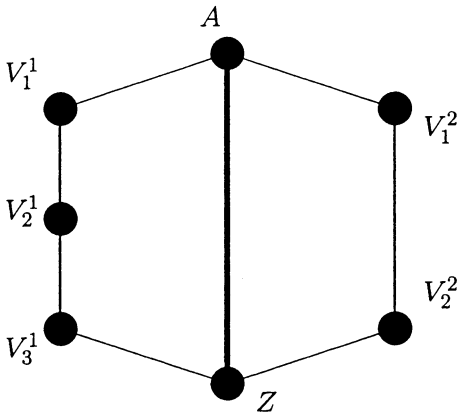


Figure 6.3 PB configuration.

The edges in $E_R(A : Z)$ constitute the *bridge*. If the bridge is empty ($b = 0$), either or both of A and Z are permitted to be empty also.

To define the coefficients of the associated PB inequality, it is helpful to identify A with V_0^i and Z with $V_{n_i+1}^i$ for $i = 1, \ldots, p$. The PB inequality

is then:

$$\sum_{e \in E} \alpha_e x_e \geq 1 + \sum_{i=1}^{p} \frac{n_i + 1}{n_i - 1} \tag{6.29}$$

where the coefficient α_e for an edge $e = \{u, v\}$ is defined as follows. If $u \in V_j^i$ and $v \in V_k^i$, $j \geq k$, then $\alpha_e = (j-k)/(n_i - 1)$; unless $u \in A$ and $v \in Z$, in which case $\alpha_e = 1$. If, however, $u \in V_j^i$ and $v \in V_k^r$, with $i \neq r$, $1 \leq j \leq n_i$, $1 \leq k \leq n_r$, then α_e equals

$$\frac{1}{n_i - 1} + \frac{1}{n_r - 1} + |\frac{j-1}{n_i - 1} - \frac{k-1}{n_r - 1}|.$$

The PB inequalities include many other known inequalities as special cases. When the bridge is empty ($b = 0$), we have the *path* inequalities of Cornuéjols, Fonlupt & Naddef (1985), valid for the STSP and GTSP. When $p = 1$, the PB inequalities reduce to K-C inequalities.

Another special case of interest is when all the n_i for $i = 1, \ldots, p$ are equal to a same value n. In such cases, the PB inequality is called *n-regular* (a term applied by Cornuéjols et al., 1985, to path inequalities). Note that K-C inequalities can be regarded as 'degenerate' n-regular PB inequalities, with $n = K + 1$. The n-regular PB inequalities have a nice description in terms of vertex sets called *handles* and *teeth*. There are $n-1$ handles, denoted by $H_1, \ldots, H_{n-1}$, and p teeth, denoted by $T_1, \ldots, T_p$ (see Figure 6.4). The first handle is defined as $H_1 = A \cup V_1^1 \cup \ldots \cup V_1^p$; the other handles are defined inductively as $H_i = H_{i-1} \cup V_i^1 \cup \ldots \cup V_i^p$. The teeth are defined as $T_j = V_1^j \cup \ldots \cup V_n^j$. The n-regular PB inequality is then:

$$\sum_{i=1}^{n-1} x(\delta(H_i)) + \sum_{j=1}^{p} x(\delta(T_j)) \;\geq\; np + n + p - 1. \tag{6.30}$$

As shown by Cornuéjols, Fonlupt & Naddef (1985), the 2-regular path inequalities are equivalent to the well-known *comb* inequalities for the STSP, which in turn include the well-known *2-matching* inequalities as a special case (see, e.g., Grötschel & Padberg, 1985). Thus, the 2-regular PB inequalities can also be regarded as a generalization of the comb and 2-matching inequalities.

Since both the GTSP-type inequalities reviewed in the previous section and the PB inequalities mentioned above are analogous to facets of GTSP polyhedra, it might be suspected that there is some general procedure for adapting polyhedral results for the GTSP into results for the GRP. This is indeed the case. In Letchford (1999), it is shown how to generalize valid or facet-inducing inequalities for the GTSP to the

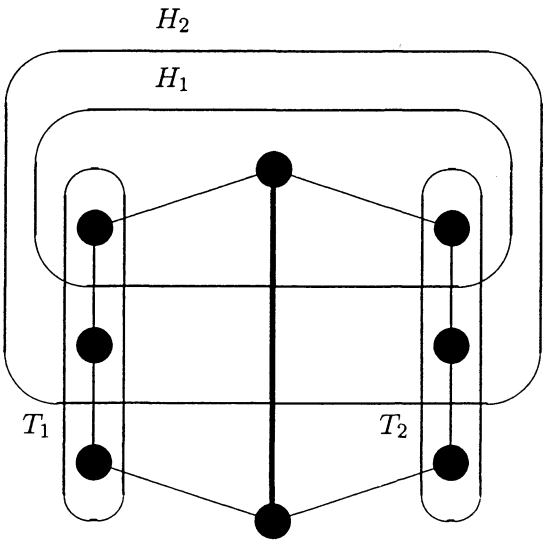


Figure 6.4 Handles and teeth in a 3-regular PB configuration.

more general SGTSP (defined in Section 3). Then, it is shown that the polyhedron P_I for any given GRP instance is a face of a polyhedron of an associated SGTSP instance. Finally, a certain *projection operation* is given which enables any class of valid inequalities for the GTSP to be adapted to the GRP. We do not describe the arguments involved in any detail for the sake of brevity; instead, we only mention a few key results:

- R-odd cut inequalities are *projected 2-matching* inequalities;
- K-C and PB inequalities are *projected path* inequalities;
- many of the honeycomb inequalities are *projected binested* inequalities.

5.3. THE DRPP

In this subsection we will need to adapt the notation somewhat. For a given $S \subset V$, we define $\delta^+(S)$, $\delta^-(S)$ and $\delta^*(S)$ as in Section 4. In addition, $A_R(S)$ will denote the required arcs with both end-vertices in S. We also set $\delta_R^+(S) = \delta^+(S) \cap A_R$, $\delta_R^-(S) = \delta^-(S) \cap A_R$ and $\delta_R^*(S) = \delta_R^*(S) \cup \delta_R^-(S)$. Finally, $b(i)$ for each $i \in V$ will denote $|\delta_R^-(i)| - |\delta_R^+(i)|$. By analogy with the DCPP (Subsection 4.2) and the MCPP (Subsection 4.3), $b(i)$ can be thought of as the 'unbalance' of vertex i.

The natural approach to the DRPP is to have a general integer variable x_a for each arc $a \in A$, representing the number of times the arc is

traversed without being serviced (see Christofides et al., 1986; Ball & Magazine, 1988). Again, this means that x_a represents the number of copies of a that will be added to A_R to form an Eulerian graph. This leads to the following definition of the set $\mathcal{S}$ of feasible solutions:

$$x(\delta^+(i)) - x(\delta^-(i)) = b(i)(\forall i \in V) \tag{6.31}$$

$$x(\delta^+(S)) \geq 1(\forall S \subset V : \begin{cases} A_R(S) \neq \emptyset, \\ A_R(V \setminus S) \neq \emptyset, \\ \delta_R^*(S) = \emptyset) \end{cases} \tag{6.32}$$

$$x_a \geq 0(\forall a \in A) \tag{6.33}$$

$$x \in Z^{|A|} \tag{6.34}$$

The associated P_I were studied independently by Savall (1990) and Gun (1993). As for the DCPP and WPP, the *balance* equations (6.31) define the affine hull of P_I and precisely $|V|-1$ of them are independent. The *postman cut* inequalities (6.32) are the directed analogue of the connectivity inequalities (6.21) for the RPP. Surprisingly, however, the conditions for them to induce facets are extremely complicated (there is not space to describe them here). Finally, the non-negativity inequalities (6.33) induce facets under mild conditions.

5.4. THE MRPP

Because the MRPP is a common generalization of the RPP, DRPP and MCPP, all of which have a complicated enough polyhedral structure, it will be clear to the reader that MRPP polyhedra are likely to be bewilderingly complicated (indeed, even the notation becomes burdensome). Nevertheless, a study has recently been made by Corberán, Romero & Sanchis (1997) and Romero (1997). To simplify the study, these authors assume that

- every $v \in V$ is incident on at least one required edge or arc
- $E = E_R$.

Instances which do not meet these assumptions are transformed by a simple procedure into instances which do.

Just as in all of the formulations examined so far (apart from the one for the WPP in Subsection 4.4), Corberán, Romero & Sanchis let x_e (or x_a) represent the total number of times that an edge (or arc) is traversed without being serviced. They define $\delta^-(S)$, $\delta^+(S)$, $\delta(S)$, $\delta^*(S)$, $\delta_R^-(S)$, $\delta_R^+(S)$, etc., as in previous subsections. Finally, they define an 'unbalance' $b(S) = |\delta_R^-(S)| - |\delta_R^+(S)| - |\delta_R(S)|$ for each $S \subset V$. The set

$\mathcal{S}$ of feasible solutions is then given by:

$$x(\delta^*(i)) \equiv |\delta^*(i)| \pmod 2, \ (\forall i \in V) \tag{6.35}$$

$$x(\delta^+(S)) \geq 1, \ (\forall S \subset V : \delta^*_R(S) = \emptyset) \tag{6.36}$$

$$x(\delta^+(S)) + x(\delta(S)) - x(\delta^-(S)) \geq b(S), \ (\forall S \subset V) \tag{6.37}$$

$$x_e \geq 0, \ (\forall e \in E) \tag{6.38}$$

$$x_a \geq 0, \ (\forall a \in A) \tag{6.39}$$

$$x \in Z^{|E \cup A|} \tag{6.40}$$

The *connectivity*, *balanced set* and *trivial* inequalities, (6.36), (6.37) and (6.38 - 6.39), respectively, are valid for the associated polyhedron P_I. Note that when $\delta(S) = \emptyset$, the balanced set inequalities for S and $V \setminus S$ imply the equation

$$x(\delta^+(S)) - x(\delta^-(S)) = b(S). \tag{6.41}$$

Corberán, Romero & Sanchis show that the affine hull of P_I is described by one such equation for each connected component in the subgraph of G induced by the (required) edges and that exactly one of these equations is redundant.

A mixed graph is *strongly connected* if and only if there is a path from any vertex to any other vertex, respecting the directions of the arcs. Note that G must be strongly connected for the MRPP to have a feasible solution. Corberán, Romero & Sanchis show that:

- A connectivity inequality induces a facet if and only if the subgraphs induced in G by S and $V \setminus S$ are both strongly connected (otherwise, they are dominated by balanced set inequalities).
- A trivial inequality induces a facet if and only if the subgraph formed by removing the corresponding edge (or arc) from G is strongly connected.
- The odd-cut (blossom) inequalities for the CPP, MCPP and RPP can be adapted to the MRPP, where they take the form:

$$x(\delta^*(S)) \geq 1 \ \ (\forall S \subset V : |\delta^*_R(S)| \text{ odd}) \tag{6.42}$$

The conditions for the balanced set and odd-cut inequalities to induce facets are rather complicated and will not be given here.

Finally, the same authors adapt the K-C and PB inequalities for the RPP (see Subsections 5.1 and 5.2) to the MRPP. Interestingly, they show that these inequalities come in two distinct (non-equivalent) 'flavours' in the mixed case. One version has the same coefficients as in the ordinary RPP, but the other has slightly different coefficients. Both versions induce distinct facets under certain (complicated) conditions.

6. THE CAPACITATED ARC ROUTING PROBLEM

6.1. PRELIMINARIES

In this section, we present the known polyhedral results for the CARP. Most of the results presented are applicable also for the CCPP, which is a special case of the CARP, except that some valid inequalities are only defined when there is more than one R-component. Since Golden & Wong (1981) proved that it is $\mathcal{NP}$-hard even to find a 0.5-approximate solution to the CCPP (see also Win, 1987), we should expect CARP polyhedra to be extremely complicated. And they are indeed.

To make matters worse, it turns out that there are a large number of competing formulations in the literature. We will attempt to explain the motivation behind each of these in the present subsection.

In real-life CARP instances, it is common for the upper bound k on the number of vehicles to be small. Moreover, real problems are frequently defined on road networks, with the result that G is very sparse (most vertices have degree smaller than 5). Under such circumstances it is 'natural' to use $\mathcal{O}(k.|E|)$ variables ($\mathcal{O}(|E|)$ for each vehicle). This is the approach taken by Belenguer (1990), Welz (1994) and Belenguer & Benavent (1998a). We will call these 'sparse' formulations.

When k is large, or when $|E_R|$ is small relative to $|E|$, an alternative approach presents itself. We can 'break' the graph G apart as follows: a complete graph G' is constructed with two vertices for each required edge, representing the two endpoints, together with an extra vertex representing the depot. An edge from one vertex to another in this expanded graph represents a shortest path between the corresponding pair of vertices in G. This leads to what we will call the 'dense' formulation, with $2|E_R|^2$ variables. This approach was explored by Letchford (1997a).

A third approach, suggested independently by Letchford (1997a) and Belenguer & Benavent (1998b), is to have only $|E|$ variables, one for each edge. Each variable represents the number of times a particular edge is traversed without being serviced. This leads to what we call a 'supersparse' formulation. Such a formulation is highly economical, elegant and easy to understand. However, it comes at a price: the individual vehicle routes are effectively 'tangled up', in that a feasible solution to such a formulation gives no indication of which vehicle traverses which edge. In fact, the problem of 'untangling' the routes appears to be $\mathcal{NP}$-hard, since it contains the $\mathcal{NP}$-hard *Bin Packing Problem* (see Garey & Johnson, 1979) as a special case. Nevertheless, the supersparse formu-

lation is extremely valuable for producing *lower bounds* on the cost of a feasible solution.

Since sparse formulations have been given most attention in the literature, we devote Subsection 6.2 to them and go into considerable detail. Subsection 6.3 reviews the results on the dense and supersparse formulations.

Still other approaches to the CARP have been proposed. Golden & Wong (1981) gave a formulation in which there were an *exponential* number of variables and constraints. It is not worth examining this, however, since it was shown in the thesis of Welz (1994) that the lower bound obtained from the LP relaxation of this formulation is always zero. Finally, one could also have a formulation in which there is a variable for each feasible vehicle route and a constraint for each required edge ensuring that the edge is serviced. This approach is not examined here as it is the subject of Chapter 8.

In the remainder of this section we will use the convention that the depot is vertex 1.

6.2. SPARSE FORMULATIONS OF THE CARP

Two different sparse CARP formulations have appeared in the literature. One can be found in Welz (1994), the other in Belenguer (1990) and Belenguer & Benavent (1998a).

The formulation of Welz bears some similarities to the Golden & Wong (1981) formulation mentioned in the previous subsection, the crucial difference being that it has a polynomial number of variables. In this formulation, the problem is effectively converted into a directed problem. That is, each edge $\{i,j\}$ is regarded as two arcs (i,j) and (j,i), with identical costs. Then, if $\{i,j\}$ is required, we require that exactly one of the pair (i,j) and (j,i) is serviced. An advantage of viewing the CARP in this way is that one obtains a pure 0-1 formulation: it is easy to show that it is never necessary for any vehicle to traverse an edge more than once in a given direction.

The variables are defined as follows:

Let x_{ij}^p take the value 1 if arc (i,j) is traversed by vehicle p, 0 otherwise.

Let l_{ij}^p take the value 1 if arc (i,j) is serviced by vehicle p, 0 otherwise.

Let A be the set of arcs in the resulting directed graph (that is, $|A| = 2|E|$) and let $A(S)$ be the set of arcs with both end-vertices in S. Welz suggests defining the set $\mathcal{S}$ of feasible solutions by the following system:

$$x^p(\delta^+(i)) = x^p(\delta^-(i)),\ (\forall i \in V, p = 1, \ldots, k) \tag{6.43}$$

$$\sum_{p=1}^{k}(l_{ij}^p + l_{ji}^p) = 1(\forall\{i,j\} \in E_R) \tag{6.44}$$

$$x_{ij}^p \geq l_{ij}^p,\ (\forall(i,j),\ p = 1, \ldots, k) \tag{6.45}$$

$$\sum_{(i,j)\in A} q_{ij} l_{ij}^p \leq Q,\ (\forall p = 1, \ldots, k) \tag{6.46}$$

$$x^p(\delta^+(S)) \geq \frac{x^p(A(S))}{|A(S)|},\ (\forall S \subseteq V \setminus \{1\}) \tag{6.47}$$

$$x \in \{0,1\}^{2k|E|}, l \in \{0,1\}^{2k|E_R|} \tag{6.48}$$

The equations (6.43) ensure that each vehicle departs from each vertex as many times as it enters. The equations (6.44) ensure that each required edge is serviced exactly once. The inequalities (6.45) ensure that each vehicle traverses each edge that it services, (6.46) impose the capacity restrictions and (6.47) are connectivity inequalities. To tighten this basic formulation Welz proposes the following *odd-cut* inequalities:

$$\sum_{p=1}^{k} x^p(\delta^+(S)) \geq \lceil |\delta_R(S)|/2 \rceil\ \ (\forall S \subseteq V \setminus \{1\} : |\delta_R(S)| \text{ odd}). \tag{6.49}$$

Also, Welz mentions that, if it is known that all k vehicles must be used, then the inequalities

$$x^p(\delta^+(1) \geq 1,\ (\forall p = 1, \ldots, k) \tag{6.50}$$

are valid also.

We would like to mention that the *connectivity* inequalities (6.47) proposed by Welz are in fact very weak. They can easily be disaggregated to give:

$$x^p(\delta^+(S)) \geq x_{ij}^p,\ (\forall S \subseteq V \setminus \{1\}, (i,j) \in A(S)).$$

We now move on to the sparse CARP formulation presented in Belenguer (1990) and Belenguer & Benavent (1998a), which uses less variables than the Welz formulation, is more 'natural' and gives better computational results. For each $e \in E_R$ and each $p = 1, \ldots, k$, let x_e^p take the value 1 if e is serviced by vehicle k, 0 otherwise. Also, for each $e \in E$ and each $p = 1, \ldots, k$, define a general integer variable y_e^p representing

the number of times e is traversed (*without* being serviced, if $e \in E_R$). The set S of feasible solutions is then given by:

$$x^p(\delta(i)) + y^p(\delta(i)) \equiv 0 \pmod 2, \ (\forall i \in V, p = 1, \ldots, k) \tag{6.51}$$

$$\sum_{p=1}^{k} x_e^p = 1, \ (\forall e \in E_R) \tag{6.52}$$

$$\sum_{e \in E_R} q_e x_e^p \leq Q, \ (\forall p = 1, \ldots, k) \tag{6.53}$$

$$x^p(\delta(S)) + y^p(\delta(S)) \geq 2x_e^p, \ (\forall S \subseteq V \setminus \{1\}, e \in E_R(S)) \tag{6.54}$$

$$x \in \{0,1\}^{k|E_R|}, y \in Z^{k|E|} \tag{6.55}$$

The reader who has persevered this far will have little difficulty interpreting the constraints in this formulation. The following results are given by Belenguer & Benavent for the associated P_I:

- The constraints (6.52) and (6.53), together with the binary conditions on the x variables in (6.55), define a so-called *Generalized Assignment polytope* (see Gottlieb & Rao, 1990a, b). Any inequality inducing a facet of this polytope (such as non-negativity inequalities $x_e^p \geq 0$ for all $e \in E_R$ and all $p = 1, \ldots, k$), induces a facet of P_I. Also, any implicit equation for this polytope (such as (6.52)) is an implicit equation for P_I and vice-versa.

- Computing the dimension of a Generalized Assignment polytope is $\mathcal{NP}$-hard, and therefore the same is true for the CARP polyhedron P_I.

- Non-negativity inequalities $y_e^p \geq 0$ for all $e \in E$ induce facets.

- If $S \subseteq V \setminus \{1\}$ is such that $|\delta_R(S)|$ is odd, then the *odd-cut* inequality

$$\sum_{p=1}^{k} y^p(\delta(S)) \geq 1 \tag{6.56}$$

  is valid and facet-inducing under mild conditions.

- If $S \subseteq V \setminus \{1\}$ is such that $\delta_R(S) \neq \emptyset$ and $F \subseteq \delta_R(S)$ is such that $|F|$ is odd, then the *parity* inequality

$$x^p(\delta_R(S) \setminus F) + y^p(\delta(S)) \geq x^p(F) - |F| + 1 \tag{6.57}$$

  is valid for $p = 1, \ldots k$ and facet-inducing under mild conditions.

- For a given $S \subset V \setminus \{1\}$, let $K(S)$ denote the minimum number of vehicles required to service $E_R(S) \cup \delta_R(S)$, due to the capacity

restrictions. Then the *capacity* inequality

$$\sum_{p=1}^{k} y^p(\delta(S)) \geq 2K(S) - |\delta_R(S)| \tag{6.58}$$

is also valid. It will frequently induce a facet when $1 < K(S) < k$. When $K(S) = 1$, it will be dominated by the connectivity inequalities (6.54). When $K(S) = k$, then the capacity and connectivity inequalities are dominated by the stronger *obligatory* inequalities

$$x^p(\delta_R(S)) + y^p(\delta(S)) \geq 2 \tag{6.59}$$

for $p = 1, \ldots k$.

- If $S \subset V \setminus \{1\}$ and $\sum_{e \in E_R(S) \cup \delta_R(S)} \alpha_e x_e^p \leq \beta$ is valid for all p due to the Generalized Assignment polytope, then the inequality

$$x^p(\delta_R(S)) + y^p(\delta(S)) \geq \frac{2}{\beta}\Big(\sum_{e \in E_R(S) \cup \delta_R(S)} \alpha_e x_e^p\Big) \tag{6.60}$$

is valid also. Note that the connectivity inequalities (6.54) are a special case of this, since $x_e^p \leq 1$ is valid for the Generalized Assignment polytope.

In Letchford (1997a), some of these inequalities are generalized.

- Let $S \subseteq V \setminus \{1\}$ be such that $\delta_R(S) \neq \emptyset$, $F \subseteq \delta_R(S)$ be such that $|F|$ is odd and $H \subseteq \{1, \ldots k\}$ be an arbitrary (non-empty) set of vehicles. Then the *general parity* inequality

$$\sum_{p \in H} (x^p(\delta_R(S) \setminus F) + y^p(\delta(S))) \geq \sum_{p \in H} x^p(F) - |F| + 1 \tag{6.61}$$

is valid. It is easy to show that the general parity inequalities include odd-cut and parity inequalities as special cases.

- Let $S \subseteq V \setminus \{1\}$ be such that $K(S)$ vehicles are required to service $E_R(S) \cup \delta_R(S)$, due to the capacity restrictions. Let $H \subseteq \{1, \ldots k\}$ be an arbitrary subset of vehicles such that $k - K(S) < |H| \leq k$. Then the *minimum crossing* inequality

$$\sum_{p \in H} (x^p(\delta_R(S)) + y^p(\delta(S))) \geq 2(|H| - k + K(S)) \tag{6.62}$$

is valid. It is easy to show that the minimum crossing inequalities include capacity and obligatory inequalities as special cases.

The issue of when these inequalities induce facets is not examined by Letchford.

Finally, Letchford (1997a) also mentions that any valid inequality for the Corberán & Sanchis RPP formulation can easily be adapted to the Belenguer & Benavent CARP formulation. That is, if any inequality of the form $\sum_{e \in E} \alpha_e x_e \geq \beta$ is valid for the former, then $\sum_{p=1}^{k} \sum_{e \in E} \alpha_e y_{ep} \geq \beta$ is valid for the latter. The odd-cut inequalities come under this category. In general, however, the resulting inequalities are unlikely to induce facets unless the demands of the required edges are small relative to the vehicle capacity Q.

6.3. THE DENSE AND SUPERSPARSE FORMULATIONS OF THE CARP

In this subsection, the dense and supersparse approaches to the CARP are reviewed. We begin with the dense formulation, which was explored by Letchford (1997a).

Assume that the required edges are numbered from 1 to $|E_R|$. Define a complete graph $G'(V', E')$, with $1+2|R|$ vertices, as follows. The depot is represented by vertex 1 in G', just as it was in G. For $i = 1, \ldots, |E_R|$, vertex $i+1$ in V' represents one arbitrary end-vertex of required edge i. Similarly, vertex $i + |E_R| + 1$ in V' represents the other end-vertex of required edge i. Note that a single vertex in V may have multiple representatives in V'.

In E', for $i = 1, \ldots, |E_R|$, the edge $\{i+1, i+|E_R|+1\}$ now represents required edge i. The other edges in E' represent shortest paths between the corresponding vertices in G. We will let E^* denote these other edges. It can be readily checked that $|E^*| = 2|E_R|^2$. A $\{0,1\}$ variable x_{ij} is now defined for every edge in E^*, taking the value 1 if a vehicle traverses between i and j, 0 otherwise.

Now let $S \subseteq V' \setminus \{1\}$ be called *unbroken* if it has the property that, for $i = 2, \ldots, |E_R| + 1$, $i \in S$ if and only if $i + |E_R| \in S$. That is, S corresponds to a set of required edges in E_R. The set $\mathcal{S}$ of feasible solutions is then given by the following linear system (when $K(S)$ is defined as in the previous subsection):

$$x(\delta(i)) = 1, \ (i = 2, \ldots, 2|E_R| + 1) \tag{6.63}$$

$$x(\delta(S)) \geq 2K(S) (\forall S \subseteq V' \setminus \{1\}, \ S \text{ unbroken}) \tag{6.64}$$

$$x \in \{0,1\}^{|E^*|} \tag{6.65}$$

Letchford (1997a) first establishes a mapping between feasible solutions for this formulation and feasible solutions to a classical formulation for the well-known *Vehicle Routing Problem* (VRP). This means that valid inequalities for the latter formulation, such as *comb*, *multistar* and *hypotour* inequalities, can be 'borrowed' to give new inequalities for the CARP (for polyhedral results on the VRP, see, e.g., Araque, 1990; Araque, Hall & Magnanti, 1990; Cornuéjols & Harche, 1993; Augerat et al., 1995).

Another class of valid inequalities are presented by Letchford for the dense formulation. A set $S \subset V' \setminus \{1\}$ is called *broken* if it is not unbroken. If S is broken, then some set $F \neq \emptyset$ of required edges lies within $\delta(S)$ in G'. If F is odd, then the *blossom* inequality $x(\delta(S)) \geq 1$ is valid.

Now define the *enlargement* of a broken set S, denoted by $en(S)$, to be the minimum unbroken set $S' \subset V'$ such that $S \subset S'$. Letchford shows that a necessary condition for a blossom inequality to induce a facet is that $|F| \geq 2K(en(S)) + 1$, since, otherwise, it is dominated by a capacity inequality (6.64).

We now move on to examine the supersparse approach, which was proposed independently by Letchford (1997a) and Belenguer & Benavent (1998b). Let the general integer variable x_e represent the number of times e is traversed without being serviced. A feasible solution then represents a collection of superimposed routes.

At this point the reader may realize that it is far from obvious how to define the set $\mathcal{S}$ of feasible solutions in terms of equations, inequalities or congruences. Perhaps surprisingly, however, that does not stop us from producing valid inequalities. For example, Letchford (1997a) proposes the following inequalities

- RPP-type inequalities. Any inequality valid for the Corberán & Sanchis RPP formulation is valid for the supersparse CARP formulation. This includes non-negativity inequalities $x_e \geq 0$ for all $e \in E$ and R-odd cut inequalities $x(\delta(S)) \geq 1$ for all $S \subset V$ with $|\delta_R(S)|$ odd.

- Capacity inequalities. As usual, let $K(S)$ for any $S \subset V \setminus \{1\}$ denote the minimum number of vehicles required to service $E_R(S) \cup \delta_R(S)$. Then the inequality

$$x(\delta(S)) \geq 2K(S) - |\delta_R(S)| \tag{6.66}$$

  is valid.

Belenguer & Benavent (1998b) propose some further inequalities for the supersparse formulation. For any $S \subset V$, let $\alpha(S)$ be a lower bound on the minimum number of times that $\delta(S)$ must be traversed without servicing. If $|\delta_R(S)|$ is even, a natural value of $\alpha(S)$ is $\max\{0, 2K(S) - |\delta_R(S)|\}$. If $|\delta_R(S)|$ is odd, a natural value is $\max\{1, 2K(S) - |\delta_R(S)|\}$. Now suppose that $S_1, \ldots S_r$ are distinct subsets of $V \setminus \{1\}$ and that $F \subset E$ is such that there is no feasible solution to the CARP in which $x(\delta(S_i)) = \alpha(S_i)$ for all i, yet $x_e = 0$ for all $e \in F$. Then the following inequality is valid:

$$\sum_{i=1}^{r} x(\delta(S_i)) + 2x(F) \geq \sum_{i=1}^{r} \alpha(S_i) + 2. \tag{6.67}$$

These inequalities are related to the *hypotour* inequalities for the STSP (e.g., Grötschel & Padberg, 1985) and the *extended hypotour* inequalities for the VRP (e.g., Augerat et al., 1995). Belenguer & Benavent (1998b) give a heuristic for identifying suitable families of sets S_i, and then show how to find an appropriate set $F \subset E$ by solving a series of minimum cost flow problems.

7. CONCLUSIONS

In this chapter we have reviewed the known polyhedral results for a number of fundamental Arc Routing Problems. It will be seen that a great deal has been learned. Nevertheless, the results in the field of Arc Routing are not as comprehensive as the results known for certain node-routing problems, especially the Symmetric and Asymmetric Traveling Salesman Problems (see Jünger, Reinelt & Rinaldi, 1995, 1997).

Of course, problems which are encountered in practice are often more complex than the problems outlined here. Only recently have researchers begun to investigate the polyhedral structure of problems with more realistic constraints. To close this chapter, we mention a paper of our own which deals with a real-life problem.

Letchford & Eglese (1998) define a variant of the RPP called the *Rural Postman Problem with Deadline Classes*, in which the edges requiring service are partitioned into a small number of classes in order of priority. The idea here is that some roads might need to be treated within two hours, some within four hours, etc. This occurs in a number of practical applications, such as postal delivery, snow ploughing or winter gritting. Letchford & Eglese give a formulation in which the route is divided into 'time phases', each with their own set of variables. The resulting polyhedron is extremely complex, yet the theoretical results which were obtained were sufficient to yield a reasonable optimization

algorithm (see Chapter 7 for more details).

The authors would like to encourage other researchers to examine more complex Arc Routing Problems from a polyhedral viewpoint.

Acknowledgments
The authors would like to thank Angel Corberán and an anonymous referee for helpful comments and corrections.

References

[1] Araque, J.R. (1990) Lots of combs of different sizes for vehicle routing. *Discussion Paper*, Centre for Operations Research and Econometrics, Catholic University of Louvain, Belgium.

[2] Araque, J.R., L.A. Hall & T.L. Magnanti (1990) Capacitated trees, capacitated routing and associated polyhedra. *Discussion Paper*, Centre for Operations Research and Econometrics, Catholic University of Louvain, Belgium.

[3] Assad, A.A. & B.L. Golden (1995) Arc routing methods and applications. In M.O. Ball, T.L. Magnanti, C.L. Monma & G.L. Nemhauser (Eds.) *Network Routing.* Handbooks of Operations Research and Management Science, 8. Amsterdam: North Holland.

[4] Augerat, P., J.M. Belenguer, E. Benavent, A. Corberán, D. Naddef & G. Rinaldi (1995). Computational results with a branch-and-cut code for the capacitated vehicle routing problem. Research report RR949-M, ARTEMIS-IMAG, France.

[5] Ball, M.O. & M.J. Magazine (1988) Sequencing of insertions in printed circuit board assembly. *Oper. Res.*, *36*, 192-201.

[6] Belenguer, J.M. (1990) The capacitated arc-routing problem polyhedron. *PhD dissertation* (in Spanish), Dept. of Stats and OR, University of Valencia, Spain.

[7] Belenguer, J.M. & E. Benavent (1998a) The capacitated arc routing problem: valid inequalities and facets. *Computational Optimization & Applications*, *10*, 165-187.

[8] Belenguer, J.M. & E. Benavent (1998b) A cutting-plane algorithm for the capacitated arc routing problem. *Working Paper*, Dept. of Stats and OR, University of Valencia, Spain.

[9] Christof, T. & G. Reinelt (1996) Combinatorial optimization and small polytopes. *Top (J. Span. Stats. & O.R. Soc.)*, *4*, 1-64.

[10] Christofides, N. (1973) The optimum traversal of a graph. *Omega*, *1*, 719-732.

[11] Christofides, N., E. Benavent, V. Campos, A. Corberán & E. Mota (1984) An optimal method for the mixed postman problem, in P. Thoft-Christensen (Ed.) *System Modelling and Optimization*, Lecture Notes in Control and Inf. Sciences, 59. Berlin: Springer.

[12] Christofides, N., V. Campos, A. Corberán & E. Mota (1981) An algorithm for the rural postman problem. *Report IC.O.R.81.5*, Imperial College, London.

[13] Christofides, N., V. Campos, A. Corberán & E. Mota (1986) An algorithm for the rural postman problem on a directed graph. *Math. Prog. Study, 26*, 155-166.

[14] Corberán, A., A. Romero & J.M. Sanchis (1997) Facets of the mixed rural postman polyhedron. *Working paper*, Dept. of Stats and OR, University of Valencia, Spain.

[15] Corberán, A., & J.M. Sanchis (1994) A polyhedral approach to the rural postman problem. *Eur. J. Opl Res., 79*, 95-114.

[16] Corberán, A., & J.M. Sanchis (1998) The general routing problem polyhedron: facets from the RPP and GTSP polyhedra. *Eur. J. Opl Res., 108*, 538-550.

[17] Cornuéjols, G., & F. Harche (1993) Polyhedral study of the capacitated vehicle routing problem. *Math. Prog., 60*, 21-52.

[18] Cornuéjols, G., J. Fonlupt & D. Naddef (1985) The traveling salesman problem on a graph and some related integer polyhedra. *Math. Prog., 33*, 1-27.

[19] Edmonds, J. (1963)The Chinese postman problem. *Oper. Res., 13*, Suppl. 1, B73-B77.

[20] Edmonds, J. & E.L. Johnson (1973) Matchings, Euler tours and the Chinese postman. *Math. Prog., 5*, 88-124.

[21] Eiselt, H.A., M. Gendreau & G. Laporte (1995) Arc-routing problems, part 1: the Chinese postman problem. *Oper. Res., 43*, 231-242.

[22] Eiselt, H.A., M. Gendreau & G. Laporte (1995) Arc-routing problems, part 2: the rural postman problem. *Oper. Res., 43*, 399-414.

[23] Fleischmann, B. (1985) A cutting-plane procedure for the traveling salesman problem on a road network. *Eur. J. Opl Res., 21*, 307-317.

[24] Ford, L.R. & D.R. Fulkerson (1962) *Flows in Networks.* Princeton University Press, Princeton, NJ.

[25] Garey, M.R. & D.S. Johnson (1979) *Computers and Intractability: a guide to the theory of NP-completeness.* San Fr.; Freeman.

[26] Ghiani, G. & G. Laporte (1997) A branch-and-cut algorithm for the undirected rural postman problem. *Working Paper*, Centre for Research on Transportation, University of Montréal.

[27] Golden, B.L. & R.T. Wong (1981) Capacitated arc routing problems. *Networks*, *11*, 305-315.

[28] Gottlieb, E.S. & M.R. Rao (1990a) The generalised assignment problem: valid inequalities and facets. *Math. Program.*, *46*, 31-52

[29] Gottlieb, E.S. & M.R. Rao (1990b) (1, k)-configuration facets for the generalised assignment problem. *Math. Program.*, *46*, 53-60.

[30] M. Grötschel & M.W. Padberg (1985) Polyhedral theory. In E.L. Lawler, J.K. Lenstra, A.H.G. Rinnooy-Kan & D.B. Schmoys (eds.) *The Traveling Salesman Problem.* New York: Wiley.

[31] Grötschel, M. & Z. Win (1988) On the windy postman polyhedron. Report No. 75, Schwerpunkt-progam der Deutschen Forschungsgeneinschaft, Universität Augsburg, Germany.

[32] Grötschel, M. & Z. Win (1992) A cutting plane algorithm for the windy postman problem. Math. Prog., *55*, 339-358.

[33] Guan, M. (1962) Graphic programming using odd or even points. *Chinese Math.*, *1*, 237-277.

[34] Guan, M. (1984) On the windy postman problem. *Discr. Appl. Math.*, *9*, 41-46.

[35] Gun, H. (1993) Polyhedral structure and efficient algorithms for certain classes of directed rural postman problem. *PhD dissertation*, Applied Math. Program, University of Maryland at College Park, Md.

[36] Jünger, M., G. Reinelt & G. Rinaldi (1995) The traveling salesman problem. In M.O. Ball, T.L. Magnanti, C.L. Monma & G.L. Nemhauser (Eds.) *Network Models.* Handbooks on Operations Research and Management Science Vol. 7. Amsterdam: Elsevier.

[37] Jünger, M., G. Reinelt & G. Rinaldi (1997) The traveling salesman problem. In M. Dell'Amico, F. Maffioli & S. Martello (Eds.) *Annotated Bibliographies in Combinatorial Optimization.* Chichester: Wiley.

[38] Kappauf, C.H. & G.J. Koehler (1979) The mixed postman problem. *Disc. Appl. Math.*, 1, 89-103.

[39] Karp, R.M. & C.H. Papadimitriou (1982) On linear characterizations of combinatorial optimization problems. *SIAM J. Comput.*, *11*, 620-632.

[40] Lenstra, J.K. & A.H.G. Rinnooy-Kan (1976) On general routing problems. *Networks*, *6*, 273-280.

[41] Letchford, A.N. (1997a) Polyhedral results for some constrained arc-routing problems. *PhD Dissertation*, Dept. of Management Science, Lancaster University.

[42] Letchford, A.N. (1997b) New inequalities for the general routing problem. *Eur. J. Opl Res.*, *96*, 317-322.

[43] Letchford, A.N. (1999) The general routing problem: a unifying framework. *Eur. J. Opl Res.*, 112, 122-133.

[44] Letchford, A.N. & R.W. Eglese (1998) The rural postman problem with deadline classes. *Eur. J. of Opl Res.*, *105*, 390-400.

[45] Nemhauser, G.L. & L.A. Wolsey (1988) *Integer and Combinatorial Optimization.* New York: Wiley.

[46] Nobert, Y. & J.-C. Picard (1996) An optimal algorithm for the mixed Chinese postman problem. *Networks*, *27*, 95-108.

[47] Orloff, C.S. (1974) A fundamental problem in vehicle routing. *Networks*, *4*, 35-64.

[48] Padberg, M.W. & G. Rinaldi (1991) A branch-and-cut algorithm for the resolution of large-scale symmetric traveling salesman problems. *SIAM Rev.*, *33*, 60-100.

[49] Papadimitriou, C.H. (1976) On the complexity of edge traversing. *J. of the A.C.M.*, *23*, 544-554.

[50] Queyranne, M. & Y. Wang (1992) On the convex hull of feasible solutions to certain combinatorial problems. *Oper. Res. Lett.*, *11*, 1-11.

[51] Ralphs, T.K. (1993) On the mixed Chinese postman problem. *Oper. Res. Lett.*, *14*, 123-127.

[52] Romero, A. (1997) The rural postman polyhedron on a mixed graph. *PhD dissertation* (in Spanish), Dept. of Stats and OR, University of Valencia, Spain.

[53] Savall, J.V. (1990) Polyhedral results and approximate algorithms for the directed rural postman problem. *PhD Dissertation* (in Spanish), Dept. of Stats and OR, University of Valencia, Spain.

[54] Welz, S.A. (1994) Optimal solutions for the capacitated arc routing problem using integer programming. *PhD Dissertation*, Dept. of QT and OM, University of Cincinnati.

[55] Weyl, H. (1935) Elementare Theorie der konvexen Polyeder. *Comentarii Mathematici Helvetica*, *7*, 290-306.

[56] Win, Z. (1987) Contributions to routing problems. *Doctoral Dissertation*, Universität Augsburg, Germany.

[57] Win, Z. (1989) On the windy postman problem on Eulerian graphs. *Math. Prog.*, *44*, 97-112.

Chapter 7

LINEAR PROGRAMMING BASED METHODS FOR SOLVING ARC ROUTING PROBLEMS

E. Benavent
Universitat de València

A. Corberán
Universitat de València

J.M. Sanchis
Universidad Politécnica de València

1. Introduction 232
2. Chinese Postman Problems 236
   2.1 The Undirected CPP 236
   2.2 The Directed CPP 238
   2.3 The Mixed CPP 239
   2.4 The Windy Postman Problem 243
3. Rural Postman Problems 244
   3.1 The Undirected RPP 244
   3.2 The General Routing Problem 250
   3.3 The Directed RPP 256
   3.4 The Mixed RPP 257
4. The Capacitated Arc Routing Problem 259
   4.1 Sparse Formulations 260
   4.2 Supersparse Formulation for the CARP 264
   4.3 Exact Methods Based on the Sparse Formulation 268
   4.4 A Cutting Plane Algorithm for the CARP Based on the Supersparse Formulation 269
5. Other Problems 269
6. Conclusions 270

1. INTRODUCTION

From the pioneering works of Dantzig, Edmonds and others, polyhedral (i.e. linear programming based) methods have been successfully applied to the resolution of many combinatorial optimization problems. See Jünger, Reinelt & Rinaldi (1995) for an excellent survey on this topic. Roughly speaking, the method consists of trying to formulate the problem as a Linear Program and using the existing powerful methods of Linear Programming to solve it.

Arc Routing Problems are no exception and it can be said that Linear Programming (LP) methods are currently among the most effective ones for solving $\mathcal{NP}$-hard Arc Routing Problems.

These methods rely for their effectiveness upon a good understanding of the polyhedron associated to the problem under study. To obtain such an understanding, it is necessary to have some grounding in the concepts and proof techniques of polyhedral theory. This is covered in depth in Chapter 6 of this book and the untrained reader is advised to read at least the first two sections of that chapter. The present chapter, on the other hand, is concerned with the application of polyhedral theory to the construction of effective optimization algorithms.

Most Arc Routing Problems can be formulated in the form:

$$\min\{c^T x : x \in \mathcal{S}\} \tag{7.1}$$

where $x = \{x_1, \ldots, x_n\}$ is a vector of decision variables, $c = \{c_1, \ldots, c_n\} \in \mathbb{R}^n_+$ is a vector of objective function coefficients (i.e., costs) and $\mathcal{S} \subset \mathbb{R}^n_+$ is a set of *feasible solutions*. Given such a problem, it is natural to define an associated polyhedron $conv(\mathcal{S})$, the convex hull of the vectors in $\mathcal{S}$. Sometimes, $conv(\mathcal{S})$ fails to be a polyhedron, but usually, if this occurs, it is easy to adjust the set of feasible solutions in such a way that the optimal solution does not change and $conv(\mathcal{S})$ is indeed a polyhedron. In what follows we will suppose this is always the case.

Usually, feasible solutions are associated with integer values of the decision variables. In such cases, $conv(\mathcal{S})$ has integral extreme points (vertices) and we use the notation $P_{\mathcal{I}} := conv(\mathcal{S})$. Given that the cost function is linear, Problem (7.1) is then equivalent to:

$$\left.\begin{array}{r} Min \;\; c^T x \\ x \in P_{\mathcal{I}} \end{array}\right\} \tag{7.2}$$

It is well known that any polyhedron can be described by a set of linear inequalities, that is, there is an integer m, a matrix $A \in Z^{mn}$

and a vector $b \in Z^m$ such that $P_{\mathcal{I}} = \{x \in I\!R^n : Ax \leq b\}$. So, at least theoretically, Problem (7.2) could be solved as a Linear Program. The set of linear inequalities given by $Ax \leq b$ is called the *linear description* of $P_{\mathcal{I}}$. Unfortunately, complete linear descriptions of $P_{\mathcal{I}}$ are only known for some easy problems and certainly not for any $\mathcal{NP}$-hard problem. It is more usual for only a small part of this description to be known. Nevertheless, as we will see, even partial linear descriptions can provide the basis for powerful optimization algorithms.

One problem which must be dealt with is that even a partial linear description frequently contains a huge number of inequalities. By 'huge', we mean a number which is exponential, or worse, in the size of the original problem. Therefore, it would not be practical to solve an LP which includes all of them explicitly.

An alternative that can be used in practice is as follows. We start with a small subset of the known inequalities and compute the optimal LP solution subject to these constraints (The polyhedron defined by a subset of the known inequalities is called a *linear relaxation* of $P_{\mathcal{I}}$). Then we check if any of the inequalities not in the current LP are violated by the optimal LP solution. If one or more violated inequalities are found, we add one or more of them to the current LP, resolve it and so on. If the LP solution obtained at the end of this process corresponds to a feasible solution of the original combinatorial optimization problem, then it is also optimal for that problem.

The linear inequalities which are added to the LP at each iteration of this process are called *cutting planes*, because, geometrically speaking, they 'cut off' the current LP solution. The entire procedure, called the *cutting plane approach*, originated in the pioneering work of Dantzig, Fulkerson and Johnson (1954) on the Symmetric Traveling Salesman Problem (STSP).

Note that the cutting-plane approach requires a method for identifying inequalities that are valid for $P_{\mathcal{I}}$ but violated by the current LP solution (An inequality $f^T x \leq f_0$ is valid for $P_{\mathcal{I}}$ if $f^T \bar{x} \leq f_0$ holds for all $\bar{x} \in S$). Normally, the known valid inequalities fall into certain well-defined classes. Therefore, for each known class of valid inequalities for $P_{\mathcal{I}}$, we are faced with the following problem:

General Identification Problem: *given a class of valid inequalities for a combinatorial optimization problem, and a point $\bar{x} \in I\!R^n$, either find an inequality in this class which is violated by $\bar{x}$, or prove that no such inequality exists.*

This problem is also sometimes known as the *separation* problem, because, geometrically speaking, we seek a hyperplane which separates $\bar{x}$ from $P_{\mathcal{I}}$. An algorithm that solves it is called an *exact separation algorithm*. Typically one finds that the identification problem can be solved efficiently for some classes of inequalities but is difficult (even $\mathcal{NP}$-hard) for others. In the latter case, we have to resort to a *heuristic separation algorithm*, which may fail to find a violated inequality in the class, even if one exists.

Let us now state more properly the scheme of a cutting plane algorithm:

Cutting plane algorithm:

Step 1 *(Initialization)* Let (LP_0) be a linear relaxation of $P_{\mathcal{I}}$. Set $k = 0$.

Step 2 *(LP solver)* Solve (LP_k). Let x^k be an optimal solution to (LP_k).

Step 3 *(Identification)* Solve the Identification Problem for x^k, and for some classes of known valid inequalities for $P_{\mathcal{I}}$.

Step 3.1 If we can conclude that $x^k \in S$, then x^k is optimal; stop.

Step 3.2 If one or more valid inequalities violated by x^k are found, define (LP_{k+1}) by adding to (LP_k) these inequalities. Set $k := k + 1$ and go to Step 2.

Step 3.3 If no violated inequality is found, stop.

The outlined cutting plane algorithm is rather rudimentary. It is quite common to use a number of refinements, such as: elimination of inequalities that are not binding, fixing of variables, using a dynamic subset of active variables, etc.

If we are lucky, this algorithm might succeed in solving a given problem instance even when the linear description of $P_{\mathcal{I}}$ is only partial. This will happen if the set of linear inequalities generated by the algorithm just happens to define a polyhedron which has the optimal solution as a vertex. Nevertheless, for the same reason, it may fail in solving another instance of the same problem. The fact that no violated inequality has been found in Step 3.2, does not mean that no such inequality exists. It may be that the violated inequality belongs to an unknown class of inequalities, or that it belongs to a known class for which we have used (without success) a heuristic separation algorithm.

If the cutting plane algorithm fails to solve a given instance, we are left with several options. One option is to use the solution cost of the final LP relaxation, which is a (typically good) lower bound on the optimal

solution value, to assess the quality of a known feasible solution found by any heuristic method. Another option is to feed the final (typically strong) linear relaxation of $P_{\mathcal{I}}$ into a classical Branch & Bound algorithm for integer programs.

A more powerful option is to use the so-called *Branch & Cut* method (see Padberg & Rinaldi, 1987 and 1991). A Branch & Cut is much like a Branch & Bound method except for the fact that valid inequalities may be added at any node of the branching tree. This leads to stronger linear relaxations at any node, which normally leads in turn to a considerable reduction in the number of nodes, in comparison to standard Branch & Bound.

Yet another option is to employ 'general purpose' (i.e., not problem specific) cutting planes, such as Gomory cuts (Gomory, 1958, 1963). These cuts are guaranteed to lead to the integer optimum in a finite number of iterations, but are not tailored to the specific $P_{\mathcal{I}}$ in question. Unfortunately, such cuts are usually weak and tend to lead to numerical problems when solving the LP. Computational experience indicates that it is generally far better to use problem specific cutting planes, especially those which define facets of $P_{\mathcal{I}}$. This is not surprising, since the facet-defining inequalities are needed in a minimal linear description of the polyhedron and are not dominated by any other valid inequality. However, some effort is required to find facet-defining inequalities and, even when a new class is discovered, one is then faced with the task of devising an exact or heuristic separation algorithm.

For an exhaustive list of problems to which the polyhedral approach has been successfully applied, see Jünger, Reinelt & Rinaldi (1995). In this chapter we are specifically concerned with Arc Routing Problems. Since the basic cutting plane algorithm is the same for each problem, and the known classes of valid and facet inducing inequalities for Arc Routing problems have been described in Chapter 6 of this book, we concentrate mainly on separation algorithms. For each of a number of important Arc routing Problems, we describe the classes of inequalities for which exact or heuristic separation algorithms have been proposed and implemented and give a description of these algorithms. Other surveys about Arc Routing, which include material on the polyhedral approach, are those of Assad & Golden (1995) and by Eiselt, Gendreau & Laporte (1995a,b).

The chapter is divided into three main sections devoted to three broad classes of Arc Routing Problems; namely, Chinese Postman, Rural Postman and Capacitated Arc Routing Problems. In each section, different variants of each problem, such as the directed, undirected and mixed

cases, are studied. However, when the same class of inequalities is valid for two or more different Arc Routing problems, we only describe the separation algorithms once, to avoid repetition.

We close this section with some notation and definitions that will be used throughout this chapter.

Let $G = (V, E)$ be an undirected graph where V is the set of nodes and E is the set of edges. Given a node subset, $S \subseteq V$, $\delta(S)$ will denote the set of edges, commonly called the edge cutset, which have one end-vertex in S and the other in $V \setminus S$, while $E(S)$ will denote the set of edges with both end-vertices in S. Given two node subsets $S, S' \subseteq V$, $(S : S')$ will denote the set of edges with one end-vertex in S and the other in S'.

Given a ground set A and a vector of decision variables $x \in I\!R^n$, for any subset $F \subseteq A$, we let $x(F)$ denote $\sum_{e \in F} x_e$. In most integer formulations of Arc Routing Problems, A is the set of edges or arcs of a graph. If this is the case and x^* is the optimal LP solution at any iteration of the cutting plane algorithm, then we define the edge-weighted graph $G(x^*)$ as the graph with vertex set equal to V, edge/arc set equal to $\{e \in A : x^*_e > 0\}$ and edge-weights equal to x^*_e. Graph $G(x^*)$ is also called the *support graph* corresponding to x^*.

Finally, we define the concept of *shrinking* a set of vertices in a weighted graph (see, e.g., Fleischmann, 1985; Padberg & Rinaldi, 1990). Given a graph $G = (V, E)$ with weights on the edges, and a set $W \subset V$, shrinking W means identifying the vertices in W, deleting any resulting loops and merging each resulting set of parallel edges, if any, into a single edge. When merging parallel edges, we give the new edge a weight equal to the sum of the original weights.

2. CHINESE POSTMAN PROBLEMS

2.1. THE UNDIRECTED CPP

Given an undirected and connected graph $G = (V, E)$ and a nonnegative cost c_e for each edge $e \in E$, the (Undirected) CPP is the problem of finding a minimum cost tour passing through each $e \in E$ at least once (Guan, 1962; Edmonds, 1963). Let x_e represent the number of copies of e that must be added to G in order to obtain an eulerian graph. The CPP can be then formulated (Edmonds & Johnson, 1973) as:

$$\text{Minimize} \quad \sum_{e \in E} c_e x_e$$

subject to

$$x(\delta(S)) \geq 1, \forall S \subset V, \quad |\delta(S)| \ \textit{is} \ \text{odd} \tag{7.3}$$

$$x_e \geq 0, \forall e \in E \tag{7.4}$$

$$x_e \ \text{ integer}, \forall e \in E \tag{7.5}$$

The CPP is closely related to the problem of finding a (minimum cost) perfect matching, since an eulerian graph can be obtained from G by adding edges to link odd degree vertices (Edmonds & Johnson, 1973; Christofides, 1973). Edmonds & Johnson solved the CPP by means of an adaptation of the blossom algorithm for perfect matching problems (Edmonds, 1965b) and proved that the associated polyhedron is completely described by the *non-negativity* inequalities (7.4) and the *odd-cut* or *blossom* inequalities (7.3). In the context of the perfect matching problem, Grötschel & Holland (1985) implemented a cutting plane algorithm which was showed to be as efficient as the existing combinatorial type matching algorithms. Then, their procedure could be also considered as an exact method to solve the CPP. The separation of *odd-cut* inequalities (7.3) can be done as follows:

2.1.1 Odd-Cut Separation.

Let $x^* \in I\!R^E$ and let $G(x^*)$ the corresponding support graph. An odd cut set in $G(x^*)$ is defined as a cut set $\delta(S), S \subset V$, such that S contains an odd number of vertices with odd degree in G. Odd cut inequalities can be separated exactly in polynomial time by finding a minimum weight odd cutset in $G(x^*)$. Using a result of Padberg & Rao (1982), this problem can be solved by computing a series of maximum flow problems on $G(x^*)$. The number of maximum flow computations needed in the worst case is equal to the number of odd degree vertices in G minus 1.

Exact algorithm (Padberg & Rao, 1982):

Step 0 Let $N = \{v_1, v_2, \ldots, v_q\}$ be the set of odd degree vertices of G. Select two vertices v_1, v_2 of N and compute the maximum flow in $G(x^*)$ between v_1 and v_2. The corresponding (minimum weight) cutset $(S : V \setminus S)$ defines two node sets, $W_1 := S$ and $W_2 := V \setminus S$.

Step 1 Let T be the tree formed by the nodes W_1 and W_2 and an edge joining them with cost equal to the maximum flow value.

Step 2 While there exists a node set W on the tree T containing more than one vertex of N, repeat:

2.1 Select one of such nodes, say W_i, and two vertices u_1, u_2 in $W_i \cap N$. Compute the maximum flow on $G(x^*)$ between u_1 and u_2.

2.2 The corresponding cutset $(S : V \setminus S)$ divides node W_i into two node sets, $W_{i_1} := W_i \cap S$ and $W_{i_2} := W_i \cap (V \setminus S)$. The other node sets of the tree T are contained either in S or in $V \setminus S$.

2.3 Substitute node W_i of T by two nodes W_{i_1} and W_{i_2} joined by an edge with cost equal to the maximum flow value. Replace each edge (W_j, W_i) of the tree incident with node W_i by an edge (W_j, W_{i_1}) or (W_j, W_{i_2}) depending on, respectively, if W_j is contained in S or in $V \setminus S$.

Step 3 If all the edges in the tree T have cost at least 1, then there is no odd-cut inequality violated by x^*. Otherwise, pick (iteratively) an edge of T having a cost less than one. If we delete it from T, we obtain two subtrees. Let S and $V \setminus S$ be the sets of vertices of G corresponding to the two subtrees. If S contains an odd number of odd-degree vertices, then $x(\delta(S)) \geq 1$ is an odd-cut inequality violated by x^*.

Heuristic algorithm:

A faster ($\mathcal{O}(|E|)$ time) heuristic algorithm was proposed by Grötschel & Holland (1985) and consists of computing the node sets $S_1, S_2, \ldots, S_q$ of the connected components of the subgraph of $G(x^*)$ induced by the edges $e \in E$ with $x_e^* > \epsilon$, where $0 \leq \epsilon < 1$ is a given parameter. If $q > 1$, then for every $i = 1, \ldots, q$, if $|\delta(S_i)|$ is odd and $x^*(\delta(S_i)) < 1$ then, $x(\delta(S_i)) \geq 1$ is a violated blossom inequality.

Usually, the heuristic algorithm is executed for different values of parameter ϵ. A given value is only tried when with the previous ones a minimum number of violated inequalities is not found. The exact algorithm is executed only when the heuristic fails.

2.2. THE DIRECTED CPP

The Directed Chinese Postman Problem (DCPP) is defined as in the undirected case except for the fact that graph G is a directed graph $G = (V, A)$ with set of arcs A. We assume that G is strongly connected because otherwise, the DCPP is infeasible. As for the undirected case, let x_{ij} denote the number of copies of arc (i, j) that must be added to G in order to obtain an eulerian graph. If G is a strongly connected graph, a least-cost Eulerian graph can be constructed by solving a transportation problem : for every vertex i, let s_i be the number of arcs entering i minus the number of arcs leaving i. Let S be the set of vertices i with $s_i > 0$ and T the set of vertices i with $s_i < 0$. If l_{ij} denotes the length of a shortest path from i to j the DCPP can be formulated as follows (Liebling, 1970; Edmonds & Johnson, 1973):

$$\text{Minimize} \qquad \sum_{i \in S} \sum_{j \in T} l_{ij} x_{ij}$$

subject to

$$\sum_{j \in T} x_{ij} = s_i, \forall i \in S \tag{7.6}$$

$$\sum_{i \in S} x_{ij} = -s_j, \forall j \in T \tag{7.7}$$

$$x_{ij} \geq 0, \forall i \in S, \quad \forall j \in T \tag{7.8}$$

This is the LP formulation of the Transportation Problem, and it is well known that it always has an integer optimal solution. Then, the DCPP can be easily solved as a Linear Program.

As we have seen up to now, the CPP and the DCPP can both be solved in polynomial time and a complete linear description of an associated polyhedron is known for them. Unfortunately, this does not hold for any other arc routing problem that will be studied in this chapter.

2.3. THE MIXED CPP

Let $G = (V, E, A)$ be a strongly connected mixed graph, with vertex set V, (undirected) edge set E and (directed) arc set A, and consider a nonnegative cost c_e for each edge and arc $e \in E \cup A$. The Mixed Chinese Postman Problem (MCPP) is the problem of finding a minimum cost tour passing through each edge and each arc of G at least once (Edmonds & Johnson, 1973).

The arcs or edges will be termed the *links* of the graph. The degree of a node is the number of links incident with it.

For this $\mathcal{NP}$-hard problem (Papadimitriou, 1976), Christofides, Benavent, Campos, Corberán & Mota (1984) proposed a formulation with a variable for each arc, two variables for each edge (representing the number of times it is traversed in either direction) and a variable for each vertex. Then, a Branch & Bound algorithm was implemented in which two different lower bounds, obtained by relaxing two types of constraints in a Lagrangian manner, were computed at each node of the search tree. A set of 34 randomly generated instances with $7 \leq |V| \leq 50$, $3 \leq |A| \leq 85$ and $4 \leq |E| \leq 39$ were solved to optimality using this algorithm.

The MCPP is an special case of the Windy Postman Problem that will be studied in section 2.4. As it will be shown there, Grötschel &

Win (1992) have proposed a procedure to solve the WPP that can be applied to the MCPP as well. Here also, two variables for each edge are used to formulate the WPP and therefore the MCPP. They solved to optimality the nine MCPP instances they tried, with $52 \leq |V| \leq 172$, $31 \leq |A| \leq 116$ and $37 \leq |E| \leq 154$.

2.3.1 A Cutting Plane Algorithm for the MCPP.

Nobert & Picard (1996) present a cutting plane algorithm that exactly solves the MCPP. They use the following characterization of an eulerian mixed graph given by Ford & Fulkerson (1962):

A strongly connected mixed graph $G = (V, E, A)$ is eulerian if and only if:

- *Evenness condition*: The degree of each node is even.
- *Balanced sets condition*: For every proper subset of vertices S, the number of arcs entering S minus the number of arcs leaving S is less than or equal to the number of edges between S and $V \setminus S$.

Let x_e represent the number of copies of link $e \in E \cup A$ that are added to G in order to obtain an eulerian graph. Note that, unlike in other formulations of the MCPP, only one variable is associated to each edge of G.

For any $S \subset V$, let $\delta^+(S)$ ($\delta^-(S)$) denote the set of arcs leaving (entering) S, and let $\delta(S)$, as usual, denote the set of edges with one end-vertex in S and the other in $V \setminus S$. The *unbalance* $b(S)$ of S is defined as $b(S) = |\delta^-(S)| - |\delta^+(S)| - |\delta(S)|$. Let us also define $odd(i) = 1$ if the degree of vertex i is odd, and $odd(i) = 0$ otherwise.

Then, Nobert & Picard (1996) formulate the MCPP as the Integer Linear Program:

$$\text{Minimize} \quad \sum_{e \in E \cup A} c_e x_e$$

subject to

$$x(\delta^+(i)) + x(\delta^-(i)) + x(\delta(i)) + odd(i) = 2z_i, \forall i \in V \tag{7.9}$$

$$x(\delta^+(S)) - x(\delta^-(S)) + x(\delta(S)) \geq b(S), \forall S \subset V,\ S \neq \emptyset \tag{7.10}$$

$$x_e, z_i \geq 0 \text{ and integer}$$

Constraints (7.9) and (7.10) are just a translation of the above conditions for a mixed graph to be eulerian. Constraints (7.10) are called *balanced set* constraints.

Nobert & Picard (1996) also introduce the *odd-cut* constraints:

$$x(\delta^+(S)) + x(\delta^-(S)) + x(\delta(S)) \geq 1, \quad \forall S \subset V, |\delta^-(S)| + |\delta^+(S)| + |\delta(S)| \text{ odd} \tag{7.11}$$

These constraints generalize the odd-cut constraints for the CPP and are used in the cutting plane algorithm instead of equations (7.9), which are introduced only to permit the generation of a Gomory cut in some situations (in practice, this possibility never occurred for the set of instances solved).

The cutting plane designed by Nobert & Picard (1996) for the MCPP starts with an initial LP containing the objective function, odd cut inequalities (7.11) associated with the odd degree vertices of G, unbalanced sets constraints (7.10) associated with 'inwards' unbalanced vertices and most unbalanced sets of G (see below how they are generated), and non-negativity constraints. At each iteration, the current LP is solved and violated inequalities of types (7.10) and (7.11) are identified and added to the LP. If no cut is found, and the LP solution is not integral, a Gomory cut is generated and added to the LP; if the solution is integer but contains an odd degree vertex, say i, equation (7.9) is added to the LP and a Gomory cut is generated on variable z_i and added to the LP. Finally, if the LP solution is integral and all vertices have even degree, it is an optimal solution for the MCPP.

Nobert & Picard(1996) were able to solve 148 instances out of 180 randomly generated instances with their pure cutting plane algorithm running on a CDC Cyber 855 with a time limit of 500 seconds. The authors report that it was never necessary to generate a z_i variable. The sizes of the instances were in the range: $10 \leq |V| \leq 169$, $2 \leq |A| \leq 2876$ and $15 \leq |E| \leq 1849$.

We now present the separation algorithms that could be used to generate violated odd-cut (7.11) and balanced set (7.10) inequalities.

2.3.2 Odd-Cut Separation.

If we consider each link in $E \cup A$ as an edge (i.e., ignoring the direction of the arcs), then G can be considered as an undirected graph, and the generalized odd-cut inequalities (7.11) are exactly the odd-cut inequalities (7.3) for the CPP. Hence, the separation algorithms presented in section 2.1.1 can be directly applied to the mixed case. Nevertheless, Nobert & Picard (1996) used only the heuristic algorithm described there with $\epsilon = 0$.

2.3.3 Balanced Set Separation.

Let x^* be the LP solution at any iteration of the cutting plane algorithm and consider the weighted graph $G'(x^*) = (V, E, A)$, with weights $w_e = x_e^* + 1$ for all $e \in E \cup A$. For any $S \subset V$, let $f(S) = w(\delta^+(S)) - w(\delta^-(S)) + w(\delta(S))$. It is easy to see that constraint (7.10) is violated by x^* for the subset S if and only if $f(S) < 0$. A set $S \subset V$ for which $f(S)$ is minimum is called a *most unbalanced set*. Nobert & Picard (1996) reduce the problem of finding a most unbalanced set to computing the maximum of a quadratic function in binary variables and they refer to previous works by Picard & Ratliff (1975) and by Picard & Queyranne (1980), where it is shown that this problem is equivalent to solving a maximum flow problem on an associated graph with $n + 2$ vertices. Here, we present a more direct proof of this important result and an explicit description of the maximum flow problem to be solved.

Exact algorithm:

Let us define $w_i^+ = w(\delta^+(i))$ and $w_i^- = w(\delta^-(i))$ for each vertex $i \in V$.

Construct the capacitated and undirected graph $H = (V_H, E_H)$ where $V_H = V \cup \{0, n+1\}$ (0 and $n+1$ are two extra vertices). The set E_H has all the edges of E with an associated capacity of w_e plus an edge from vertex 0 to every vertex $i \in V$ with capacity $w_{0i} = max\{w_i^- - w_i^+, 0\}$, and an edge from each vertex $i \in V$ to vertex $n + 1$, with capacity $w_{i,n+1} = max\{w_i^+ - w_i^-, 0\}$.

Solving on H the maximum flow problem from vertex 0 to vertex $n+1$ provides us with a minimum capacity cut in graph H. Let $S^* \cup \{0\}$ be the subset of vertices defining this cut, then S^* is a most unbalanced set in $G'(x^*)$ as it is shown in the following.

First of all, note that $f(S)$ can be written as: $f(S) = \sum_{i \in S}(w_i^+ - w_i^-) + w(\delta(S))$. Let us define $P = \sum_{i \in V} w_{0i}$ and let $\delta_H(S)$ represent an edge cutset in graph H. If we subtract the constant P from the capacity of the cutset in graph H defined by any set $S \cup \{0\}$, we obtain:

$$\begin{aligned} w(\delta_H(S \cup \{0\})) - P &= \sum_{(i,j) \in (S:V \setminus S)} w_{ij} + \sum_{i \in V \setminus S} w_{0i} + \sum_{i \in S} w_{i,n+1} - \sum_{i \in V} w_{0i} \\ &= \sum_{(i,j) \in (S:V \setminus S)} w_{ij} + \sum_{i \in S}(w_i^+ - w_i^-) = f(S) \end{aligned}$$

Then, the minimum $f(S)$ corresponds to the minimum $w(\delta_H(S \cup \{0\}))$. Therefore, any cutset with $w(\delta_H(S \cup \{0\})) < P$ provides a violated balanced set inequality.

To find a most unbalanced set in G, in order to include their corresponding inequality in the initial LP, one can simply apply the above procedure by setting $x^* = 0$. Nobert & Picard (1996) enumerate all the minimum capacity cuts in H and add them to the LP (unless its number is too large). Furthermore they split each most unbalanced set into several smaller ones in order to obtain stronger cuts. We refer to their work for the details.

2.4. THE WINDY POSTMAN PROBLEM

Let $G = (V, E)$ be an undirected graph with two costs $c_{ij}, c_{ji} \geq 0$ associated to each edge $(i, j) \in E$, i.e., the cost of traversing an edge depends on the direction of travel. The Windy Postman Problem (WPP) consists of finding a minimum cost tour traversing all edges of G at least once. This is an $\mathcal{NP}$-hard problem (Guan, 1984), although it can be solved in polynomial time if G is eulerian (Win, 1987). It also contains as special cases the CPP, DCPP and MCPP.

Win (1987) and Grötschel & Win (1992) proposed a cutting plane algorithm for the WPP based on a previous polyhedral study, that was able to solve to optimality most of the problems tested. As far as we know this was the first polyhedral approach that has been applied to the resolution of an $\mathcal{NP}$-hard Arc Routing Problem. Let x_{ij} be the number of times edge (i, j) is traversed from i to j in a WPP solution. The formulation given by Win (1987) and by Grötschel & Win (1992) is:

$$\text{Minimize} \quad \sum_{(i,j)\in E} (c_{ij}x_{ij} + c_{ji}x_{ji})$$

subject to

$$x_{ij} + x_{ji} \geq 1, \forall (i, j) \in E \tag{7.12}$$

$$\sum_{(i,j)\in\delta(i)} (x_{ij} - x_{ji}) = 0, \forall i \in V \tag{7.13}$$

$$x_{ij}, x_{ji} \geq 0 \tag{7.14}$$

$$x_{ij}, x_{ji} \text{ integer} \tag{7.15}$$

Under certain conditions, non-negativity inequalities, traversing inequalities (7.12) and the following odd-cut inequalities induce facets the corresponding polyhedron (Grötschel & Win (1988)):

$$\sum_{(i,j)\in\delta(S)} (x_{ij} + x_{ji}) \geq |\delta(S)| + 1, \forall S \subset V, \quad |\delta(S)| \text{ odd} \tag{7.16}$$

$$\sum_{i\in S, j\notin S} x_{ij} \geq \frac{1}{2}(|\delta(S)| + 1), \forall S \subset V, \quad |\delta(S)| \text{ odd} \tag{7.17}$$

$$\sum_{i\in S, j\notin S} x_{ji} \geq \frac{1}{2}(|\delta(S)| + 1), \forall S \subset V, \quad |\delta(S)| \text{ odd} \tag{7.18}$$

Odd-cut inequalities (7.16), (7.17) and (7.18) are equivalent. The form (7.16) is used in the separation algorithm. Once such a violated inequality is found, it is added to the LP in the form (7.17) or (7.18) having the minimum number of non-zero coefficients. Other valid inequalities are described in Grötschel & Win (1988), but only the ones presented above were implemented in their cutting plane algorithm.

The cutting plane algorithm starts by solving an initial LP containing the objective function and constraints (7.12), (7.13) and (7.14). Let x^* be the LP solution at any iteration. If we define the weight $w_{ij} = x^*_{ij} + x^*_{ij} - 1$ for each edge $(i,j) \in E$, then any odd-cut in $G(x^*)$ with weight less that 1 corresponds to a violated odd-cut inequality (7.16). Hence, the separation problem for inequalities (7.16) reduces to the problem of determining an odd-cut of minimum weight in $G(x^*)$. The procedures described in section 2.1.1 were used for this purpose. All the odd-cut violated inequalities are added to the LP, while odd-cut inequalities that were added in previous iterations and that are non-binding (i.e., have a positive surplus variable) are removed from the LP.

The algorithm was tested on 36 WPP instances with $52 \leq |V| \leq 264$ and $78 \leq |E| \leq 489$ and it provided an optimal WPP solution for 31 instances. When the cutting plane procedure failed to arrive at an integer solution, feasible WPP tours were derived by appropriately rounding up the fractional variables and then possibly setting some variables to zero (see Grötschel & Win (1992) for the details).

3. RURAL POSTMAN PROBLEMS

3.1. THE UNDIRECTED RPP

Let $G = (V, E)$ be a connected and undirected graph with nonnegative costs associated to its edges. Given a subset $E_R \subseteq E$ of 'required' edges, the problem of finding a minimum cost tour passing at least once through all the required edges is known as the *Rural Postman Problem (RPP)*. Note that when $E_R = E$ the RPP reduces to the CPP, but in the general case, if the graph induced by edges in E_R, $G_R = (V_R, E_R)$, is

not connected, the problem is $\mathcal{NP}$-hard (Lenstra & Rinnooy-Kan, 1976).

Christofides, Campos, Corberán & Mota (1981) implemented a Branch & Bound algorithm based on Lagrangean Relaxation for this problem and solved 24 instances with $9 \leq |V| \leq 84$, $13 \leq |E| \leq 184$ and $4 \leq |E_R| \leq 74$. Their algorithm contains the following pre-processing stage which converts any RPP instance with $V \neq V_R$ into another instance for which $V = V_R$:

Simplification routine:

Step 1. Add to $G_R = (V_R, E_R)$ a non required edge between every pair of vertices in V_R having a cost equal to the shortest path length on G.

Step 2. For each pair (if any exist) of parallel edges with the same cost, delete one member. Remove all edges $e = (i, j) \notin E_R$ such that $c_{ij} = c_{ik} + c_{kj}$ for some vertex k.

Let us denote by $G = (V, E)$ the resulting graph. This transformation can occasionally increase the number of edges in G, but it frequently decreases it. Suppose from now on that $V = V_R$. Let x_e denote the number of copies of edge $e \in E$ that might be added to G in order to obtain an eulerian graph, and let $\delta_R(S)$ denote $\delta(S) \cap E_R$. Corberán & Sanchis (1994) formulate the RPP as:

$$\text{Minimize} \qquad \sum_{e \in E} c_e x_e$$

subject to

$$x(\delta(S)) \geq 2, \forall S \subset V, \quad \delta_R(S) = \emptyset \tag{7.19}$$

$$x(\delta(i)) \equiv |\delta_R(i)| \pmod 2, \forall i \in V \tag{7.20}$$

$$x_e \geq 0 \text{ and integer } \forall e \in E \tag{7.21}$$

The convex hull in $\mathbb{R}^{|E|}$ of feasible solutions to (7.19) - (7.21), denoted by RPP(G), is an unbounded and full-dimensional polyhedron. As mentioned in the previous chapter, many classes of valid inequalities and facets are known for RPP(G), some of them described in the context of the GRP. Corberán, Letchford, & Sanchis (1998) have devised separation algorithms for:

- *connectivity* inequalities.
- *R-odd cut* inequalities.
- *K-Component (K-C)* inequalities.

- *regular path-bridge* inequalities.
- *honeycomb* inequalities.

The algorithms for connectivity, R-odd and K-C inequalities are described in the following subsections. Separation of path-bridge and honeycomb inequalities will be detailed in the next section because they were introduced in the context of the General Routing Problem (GRP), although they can be also applied to the RPP. A set $S \subset V$ will be called *R-odd* if $|\delta_R(S)|$ is odd, otherwise it will be called *R-even*. We will call the connected components of G_R *R-components*. A set of vertices defining an R-component will be called an *R-set*.

3.1.1 Connectivity Separation.

Connectivity inequalities (7.19) can be separated exactly in polynomial time. Consider the shrunk graph $G_s = (V_s, E_s)$ obtained from $G(x^*)$ by shrinking each R-set into a single node and let $\bar{x}^*$ be the resulting edge weights.

Exact algorithm:

Find a minimum weight cutset in G_s (Gomory & Hu, 1961). Each cutset with weight less than two corresponds to a violated connectivity inequality on G.

Heuristic algorithm:

Compute the connected components of the subgraph induced by the edges $e \in E_s$ with $\bar{x}^*_e > \epsilon$, where $0 \leq \epsilon < 2$ is a given parameter. Let $S_1, S_2, \ldots, S_q$ be the sets of nodes in the original graph G corresponding to the node sets of these connected components. Then $x(\delta(S_i)) \geq 2$ is a violated connectivity inequality if $q > 1$ and $\bar{x}^*(\delta(S_i)) < 2$. Note that when $q = 2$ we have $x(\delta(S_1)) = x(\delta(S_2))$, but when $q > 2$ all the inequalities are distinct.

Usually, the heuristic is executed for different values of parameter ϵ. A given value is only tried when with the previous ones a minimum number of violated inequalities have not been found. Only if the heuristic fails is the exact algorithm executed.

3.1.2 R-odd cut Separation.

Like odd-cut inequalities for the CPP, R-odd cut inequalities (Corberán & Sanchis, 1994)

$$x(\delta(S)) \geq 1, \quad \forall S \subset V, \ |\delta_R(S)| \ \textit{odd} \tag{7.22}$$

can also be separated exactly in polynomial time by using the Padberg-Rao algorithm. We only have to consider each R-odd vertex as an odd-degree vertex.

The heuristic described in section 2.1.1 has also been used by Corberán, Letchford & Sanchis (1998) for values 0, 0.25 and 0.5 of parameter ϵ.

3.1.3 K-C Inequalities Separation.

Roughly speaking, R-odd inequalities assure that every R-odd node i has to satisfy $x(\delta(i)) \geq 1$ but, if i is an R-even node and $x(\delta(i)) = 1$ holds, vertex i becomes an 'odd degree node' but no R-odd inequality is able to cut off this invalid 'solution'. This is what K-C inequalities (Corberán & Sanchis, 1994) try to do. K-Component (K-C) inequalities are defined in terms of a partition $\{V_0, \ldots V_K\}$ of V, with $K \geq 3$ where $V_1, \ldots V_{K-1}$ and $V_0 \cup V_K$ are clusters of one or more R-sets and $|(V_0 : V_K) \cap E_R|$ is positive and even. The corresponding K-C inequality can be written as:

$$F(x) = \sum_{\forall p<q} (q-p)x((V_p : V_q)) - 2x((V_0 : V_K)) \geq 2(K-1) \qquad (7.23)$$

No exact polynomial algorithm is known to separate K-C inequalities. For this problem, Corberán, Letchford & Sanchis (1998) have designed an effective heuristic algorithm, which is based on the following considerations:

- K-C inequalities try to separate solutions x^* where an R-even node u belonging to an R-component C_i satisfies $x^*(\delta(u)) \cong 1$ and $x^*(\{u\} : C_i \setminus \{u\}) \cong 0$. Thus, $\{u\}$ and $C_i \setminus \{u\}$ are suitable node sets to be considered as V_0 and V_K respectively. This idea is generalized by considering node sets V_0 with an even number of R-odd nodes forming a connected component of the subgraph induced in C_i by edges e with $x^*_e > 0$.

- For $i = 0, 1, 2, \ldots, K-1$, let $LHS(i, i+1)$ denote the sum of the x^*_e for every edge $e \in (V_p : V_q)$ with $p \leq i$ and $q \geq i+1$. It is then possible to write the left hand side of (7.23) as:

$$F(x^*) = \sum_{i=0}^{K-1} LHS(i, i+1) - 2x^*((V_0 : V_K)) \qquad (7.24)$$

  If $F(x^*) < 2(K-1)$, the K-C inequality is violated. Otherwise, it is satisfied with a slack of $F(x^*) - 2(K-1)$. Shrinking any pair of consecutive sets V_i, V_{i+1} into a single set yields a new 'smaller' (K-1)-C configuration with an associated inequality $F'(x) \geq 2(K-2)$. It is easy to see that shrinking sets V_i, V_{i+1} with $LHS(i, i+1) > 2$ leads to K-C inequalities with lower slack, as long as $K \geq 3$

remains. Hence, by shrinking iteratively pairs V_i, V_{i+1}, new K-C inequalities are obtained and checked for possible violation.

Heuristic algorithm:

- **Phase 0:**
  Let C_i be an R-set and let us call x^*-external those nodes in C_i which are adjacent to nodes not in C_i by an edge e with $x_e^* > 0$. Assume that C_i has, at least, two x^*-external nodes connected to, at least, two different R-sets.

- **Phase I: Define seeds for V_0 and V_K.**
  Consider the subgraph, $G(C_i)$, induced in G by edges e with both endpoints in C_i and $x_e^* > \epsilon$, where ϵ is a given parameter. Let u be a x^*-external node and compute its corresponding connected component in $G(C_i)$. If this component has an even number of R-odd nodes, set V_0 to it and set V_K to the complementary set in C_i.

- **Phase II: Define the node sets $V_0, \ldots, V_K$.**

  (a) Construct the graph G' obtained from the weighted graph $G(x^*)$ by shrinking the sets V_0, V_K, and the remaining R-sets into a single node each.

  (b) Compute a maximum weight spanning tree in $G' \setminus \{(V_0, V_K)\}$.

  (c) Transform the tree into a path linking V_0 and V_K, by (iteratively) shrinking each node with degree one (different from V_0, V_K) into its (unique) adjacent node. Let $V_1, V_2, \ldots, V_{K-1}$ be the nodes of this path. If $K \geq 3$, they define the node sets of the K-C configuration.

- **Phase III: Check the K-C inequality.**
  For each $i = 0, 1, \ldots, K-1$, compute $LHS(i, i+1)$ and $F(x^*)$ as in (7.24). If $F(x^*) < 2(K-1)$, the K-C inequality is violated. Otherwise, check if a violated K-C inequality could be obtained by shrinking iteratively some pairs V_i, V_{i+1}, while $K \geq 3$ holds.

The algorithm was executed with values $0, 0.25$ and 0.5 for parameter ϵ. Initially, in phase 0, C_i is set equal to each of the R-components. In further stages of the cutting plane algorithm, C_i is set equal to any pair of R-components adjacent in graph $G(x^*)$.

3.1.4 Cutting Plane and Branch & Cut Algorithms for the RPP.

In their paper on the RPP polyhedron, Corberán & Sanchis (1994) report some computational experience with a cutting-plane algorithm in which only connectivity, R-odd and K-C inequalities were considered and its identification was carried out visually. All except one of the 24 instances reported in Christofides et al. (1981) were solved to optimality and the optimal value (although not the optimal solution) was obtained for the last one. Two other real-life instances, with $|V| = 113$, $|E| = 171$ and with 10 and 11, respectively, R-components were also solved.

Corberán, Letchford & Sanchis (1998) present a cutting-plane algorithm for the General Routing Problem (that includes the RPP as a particular case) containing the heuristic and exact procedures here described for the separation of connectivity, R-odd and K-C inequalities, as well as other procedures that will be presented in the next section. This algorithm was able to solve the 26 instances of Christofides et al., (1981) and Corberán & Sanchis (1994), as well as the 92 RPP instances generated by Hertz, Laporte & Nanchen (1998). These 92 instances correspond to three classes of randomly generated graphs designed to test different heuristic algorithms for the RPP. First class graphs were obtained by randomly generating points in the plane; class 2 graphs are grid graphs generated to represent the topography of cities, while class 3 contains grid graphs with vertex degrees equal to 4. It is worth pointing out that the sophisticated heuristic procedures of Hertz, Laporte & Nanchen (1998) also produced optimal solutions for all the instances (although they could not prove optimality).

Another approach to the RPP has also been proposed by Ghiani and Laporte (1997). They used the same formulation as before, but they noted that only a small set of variables (those belonging to an SST connecting the R-components of G) may be greater than 1 in an optimal solution of the RPP and, furthermore these variables can take, at most, a value of 2 (the corresponding set of edges is denoted by E_{012}). Then, by duplicating these latter variables, the authors formulate the RPP using only 0/1 variables. Ghiani & Laporte have implemented a Branch & Cut algorithm based on connectivity inequalities (7.19), R-odd inequalities (7.22) and the following ones (that are only valid under the assumption that variables x_e are binary):

$$\sum_{e \in \delta(S) \setminus \{e_b\}} x_e \geq x_{e_b}, \quad \forall S \subset V, \quad S \text{ is } R\text{-even}, \quad e_b \in \delta(S) \tag{7.25}$$

To separate the connectivity inequalities (7.19), they use a heuristic similar to the one proposed by Fischetti, Salazar & Toth (1997) for the Gen-

eralized TSP. Starting with the support graph $G(x^*)$, each R-component is shrunk into a single vertex and a maximum spanning tree is built in the resulting graph. Each stage of the construction of this tree, produces a subset of R-components for which the corresponding inequality (7.19) is checked for violation. Once the spanning tree is completed, another check for violated connectivity constraints is made by removing in turn each edge of the tree.

To separate R-odd cut inequalities (7.22), Ghiani & Laporte (1997) use a heuristic similar to the one described in section 2.1.1. Finally, they have designed a new heuristic algorithm to separate inequalities (7.25) that can be roughly described as follows. For any R-even component of subgraph of $G(x^*)$ induced by edges with $x_e^* > \epsilon$, where ϵ is a given parameter, a maximum spanning tree is computed. If the removal of a tree edge e divides the component into two R-even subcomponents, say C' and C'', then two inequalities (7.25) are checked by considering $S = C'$, $e_b = e$ and $S = C''$, $e_b = e$. A branching step is executed whenever no violated inequality can be generated. Branching is made on fractional variables x_e nearest to 0.5 and two son subproblems are generated as usual, except in the case $e \in E_{012}$, where three son subproblems are generated.

Ghiani and Laporte (1997) report very good computational results on a set of 200 instances, corresponding to 3 classes like those in Hertz, Laporte & Nanchen (1998). Except for 6 instances, other 194 instances involving up to 300 or 350 vertices were solved to optimality in a reasonable amount of time.

3.2. THE GENERAL ROUTING PROBLEM

Given an undirected and connected graph $G = (V, E)$, a nonnegative cost c_e for each edge $e \in E$, a set $V_R \subseteq V$ of *required vertices* and a set $E_R \subseteq E$ of *required edges*, the *General Routing Problem* (GRP) is the problem of finding a minimum cost tour passing through each $v \in V_R$ and each $e \in E_R$ at least once (Orloff, 1974).

The GRP was proved to be $\mathcal{NP}$-hard by Lenstra & Rinnooy-Kan (1976). It includes as special cases the arc routing problems on an undirected graph described in previous sections (i.e., the Chinese Postman Problem, CPP, and the Rural Postman Problem, RPP) and also includes some well known routing problems where the service requirements are on the vertices of the graph: when $E_R = \emptyset$, we obtain the *Steiner Graphical Traveling Salesman Problem* (SGTSP) (Cornuéjols, Fonlupt & Naddef, 1985), also called the *Road Traveling Salesman Problem* by Fleischmann (1985); when $E_R = \emptyset$ and $V_R = V$, we obtain the *Graphical Traveling*

Salesman Problem (GTSP) (Cornuéjols, Fonlupt & Naddef, 1985). A strong relationship between the GTSP and GRP polyhedra has been described in deep by Letchford (1997b).

The integer programming formulation for the GRP is exactly the same that for the RPP. Hence, connectivity, R-odd cut, K-C and GTSP-type inequalities are valid for the GRP as well as for the RPP (Corberán & Sanchis, 1998) and can be separated with the procedures presented in sections 3.1.1, 3.1.2 and 3.1.3 for the RPP. Inversely, path-bridge (Letchford, 1997a) and honeycomb inequalities (Corberán & Sanchis, 1998) presented for the GRP are also valid for the RPP, and the corresponding separation procedures presented in next sections can be also applied to the RPP. These procedures (Corberán, Letchford & Sanchis, 1998) also rely on the assumption that $V = V_R$, because, as in the RPP, GRP instances which do not satisfy the assumption can be easily transformed into instances which do (Christofides, Campos, Corberán & Mota, 1981).

3.2.1 Honeycomb Separation.

In a K-C configuration, an R-component (or a cluster of R-components) is divided into two parts. A honeycomb configuration (Corberán & Sanchis, 1998) generalizes the K-C configuration simultaneously both in the number of parts an R-component is divided into and in the number of R-components we divide. Many honeycomb configurations can be formed by 'gluing' K-C configurations together by identifying edges.

Honeycomb inequalities for which a separation heuristic has been designed in Corberán, Letchford & Sanchis (1998) are those in which a single cluster of R-connected components is partitioned into more than 2 parts. The associated honeycomb configurations consist of:

- a partition $\{V_1, \ldots, V_L, W_1, \ldots, W_{K-1}\}$ of V, with $L \geq 3$, $K \geq 4$, such that $(V_1 \cup \ldots \cup V_L)$, $W_1, \ldots, W_{K-1}$ are clusters of one or more R-sets and $\delta(V_i)$ contains a positive and even number of required edges for all i. The required edges crossing between the V_i spans the sets V_i considered as nodes.

- a tree T spanning the sets $V_1, \ldots, V_L, W_1, \ldots, W_{K-1}$ such that the degree in T of every node V_i is 1, the degree of nodes W_j is at least 2 and the path in the tree connecting any distinct V_i, V_j is of length 3 or more.

If $L = 2$ the tree is a path and we have a K-C configuration. If $L \geq 3$, then $K \geq 4$ is needed in order to the path in the tree connecting any distinct V_i, V_j being of length 3 or more.

In the associated honeycomb inequality, the coefficient c_e of edge $e \in E$ is equal to the number of edges traversed in the spanning tree to get from one end-vertex of e to the other, except for the edges with one end-vertex in V_i and the other in V_j, $i \neq j$, when the coefficient is 2 units less. The honeycomb inequality is then:

$$\sum_{e \in E} c_e x_e \geq 2(K-1) \tag{7.26}$$

Honeycomb inequalities try to separate solutions x^* where some R-even nodes $u_1, u_2, \ldots, u_L$ (or sets with an even number of R-odd nodes) belonging to an R-component C_i satisfy $x^*(\delta(u_j)) \cong 1$ and $x^*(\{u_j\} : C_i \setminus \{u_j\}) \cong 0$, $j = 1, 2, \ldots, L$.

The idea of the algorithm is similar to that for the K-C separation. The main difference is, given an R-set or a cluster of R-sets C_i, to determine the number of node sets to divide it, and how to make the division into $V_1, V_2, \ldots, V_t$. The procedure is as follows:

Let C_i be an R-set and remember that x^*-external nodes are those in C_i which are adjacent to nodes not in C_i by an edge e with $x_e^* > 0$. Assume C_i has, at least, two x^*-external nodes and connected to, at least, two different R-sets.

Assign to each x^*-external node u_j a label k corresponding to the R-set R^k not in C_i with $x^*(\{u_j\} : R^k)$ maximum. To each of the remaining nodes in C_i, a different negative label is assigned. Starting with the edge (u, v) in $E(C_i)$ with largest x^*-weight, let l_{min} (l_{max}) be the smaller (larger) label of that of u and v. Assign the label l_{max} to all nodes in C_i having label l_{min}. This procedure is repeated until all nodes have positive label. Nodes with the same label define a partition $V_1, V_2, \ldots, V_t$ of the set of nodes of C_i. If the number of R-odd nodes in each $V_j, j = 1, \ldots, t$ is even, we are done. Otherwise, all the sets V_j with an odd number of R-odd nodes are joined forming a single set. Then, a partition $V_1, V_2, \ldots, V_L$ have been defined.

If $L \geq 3$, these sets are suitable to be considered as part of a honeycomb configuration. If $L = 2$, these sets are suitable to be considered as V_0 and V_K for a K-C configuration. Otherwise ($L = 1$), C_i is rejected.

Consider now the graph obtained by shrinking each $V_1, V_2, \ldots, V_L$ and each of the remaining R-sets into single nodes. Compute a spanning tree with large x^* weight in this shrunk graph without using edges (V_i, V_j). Then shrink (iteratively) each node with degree one on the tree (different from $V_1, V_2, \ldots, V_L$) into its adjacent node, having in mind that every node V_i must have degree one in the configuration tree.

If the above procedure has been successful, a honeycomb configuration has been defined with $V_1, V_2, \ldots, V_L$ and $W_1, W_2, \ldots, W_{K-1}$ and its corresponding inequality is checked for possible violation. If it is not violated, a similar procedure to that of section 3.1.3 for K-C inequalities shrinks pairs of sets W_i, W_j adjacent in the tree into a single one to obtain a new 'smaller' honeycomb configuration that, under certain conditions, can be violated. In further stages of the cutting plane algorithm, C_i is set equal to a pair of R-sets adjacent in graph $G(x^*)$.

3.2.2 Path-Bridge Separation.

Letchford (1997a) introduced the *path-bridge* (PB) inequalities. As K-C inequalities try to separate 'solutions' in which $x(\delta(i)) = 1$ for an R-even vertex i, PB inequalities try to separate 'solutions' in which $x(\delta(i)) = 2$ for an R-odd vertex i. The associated *path-bridge* (PB) *configuration* is presented in detail in Chapter 6. A special case are the so called *n-regular PB* inequalities, n-PB, that have an easy description in terms of *handles* and *teeth*:

There are $n-1$ handles, $H_1, \ldots, H_{n-1}$, and p teeth, $T_1, \ldots, T_p$. The first handle is defined as $H_1 = A \cup V_1^1 \cup \ldots \cup V_1^p$; the other handles are defined inductively as $H_i = H_{i-1} \cup V_i^1 \cup \ldots \cup V_i^p$. The teeth are defined as $T_j = V_1^j \cup \ldots \cup V_n^j$. The n-PB inequality is then:

$$\sum_{i=1}^{n-1} x(\delta(H_i)) + \sum_{j=1}^{p} x(\delta(T_j)) \;\geq\; np + n + p - 1 \qquad (7.27)$$

Note that 2-PB inequalities are analogous to the *comb* inequalities for the STSP (see, e.g., Grötschel & Padberg, 1979). A 2-PB inequality in which each tooth consists merely of two isolated vertices (connected by a non-required edge) is called *simple* (Letchford, 1997a). Simple 2-PB inequalities are analogous to the *2-matching* inequalities for the STSP.

No exact polynomial algorithm is known to separate general n-PB inequalities. However, simple 2-PB inequalities can be separated in polynomial time provided that x^* satisfies all connectivity inequalities (Letchford, 1997a). The algorithm is based on the *edge-splitting* idea of Padberg & Rao (1982).

Exact algorithm for simple 2-PB separation:

Given an edge $e \in E$ with both end-vertices isolated, define $\bar{x}_e^* = x_e^* + x^*(\delta(e)) - 3$. It can be shown that, if all connectivity inequalities are satisfied, then $\bar{x}_e^* \geq 0$ for all such e. If $\bar{x}_e^* < 1$, then e is called *splittable*.

Let G' be the graph obtained from G by deleting non-required edges e with $x_e^* = 0$. Label all vertices *odd* or *even* according to whether they are R-odd or R-even. Then divide each splittable edge e into two edges, called *halves*, by inserting a new *splitting vertex*, labelled *odd*, in the middle of e. One half (the *normal* half) retains the original weight, whereas the other half (the *flipped* half) gets a weight of $\bar{x}_e^*$. Then, reverse the label of any non-splitting vertex which is adjacent to an odd number of flipped halves. Each odd cut with weight less than one in the resulting *split graph* corresponds to a violated simple 2-PB inequality. The handle in the simple 2-PB inequality is composed of all original (non-splitting) vertices on one shore of the cut. The teeth are the splittable edges whose flipped halves lie in the cutset.

The above exact separation algorithm has two disadvantages. First, it is rather slow. Second, there are few violated simple 2-PB inequalities when the GRP instance has few isolated vertices (indeed, SPB inequalities are not defined at all for RPP instances). Both disadvantages can be alleviated by applying the algorithm separately to each block of G'. Each violated simple 2-PB inequality found is then expanded into a violated (not necessarily simple) 2-PB inequality.

Furthermore, Corberán, Letchford & Sanchis (1998) have devised a separation heuristic for general n-PB inequalities, which is fast and quite effective. This n-PB separation heuristic is also applied to each block separately. Nevertheless, for simplicity of notation, we assume that there is in fact only one block.

Heuristic algorithm for n-PB separation:

- **Phase I:** Select candidates for H_1
  Examine each pair S_1, S_2 of R-sets connected by at least one edge. If $x^*(\delta(S_1 \cup S_2)) + x^*((S_1 : S_2)) - 3 \leq \epsilon$, where ϵ is a given parameter, then label the edges in $E(S_1)$ and $E(S_2)$ 'strong' and the edges in $(S_1 : S_2)$ 'weak'. Store (S_1, S_2) as a 'candidate tooth'. S_1 and S_2 are the 'ends' of the candidate tooth. Examine the remaining unlabeled edges. Label such an edge e 'weak' if $x_e^* \leq \min\{\epsilon, 0.25\}$. Delete all weak edges from G' and examine each connected component C in the resulting graph. Let $b = |\delta_R(C)|$ and let p equal the number of candidate teeth with exactly one end in C. If $p + b \geq 3$ and odd, $p \geq 1$ and the p candidate teeth are vertex-disjoint, then put C into a list $\mathcal{H}$ of candidates for H_1.
  For each candidate $H \in \mathcal{H}$, repeat:

- **Phase II:** Construct the configuration nodes.
  Set $V_1^1, \ldots, V_1^p$ to be the ends of the p candidate teeth lying in H.

Set A to be the remainder of H, if any. Set $V_2^1, \ldots, V_2^p$ to be the other ends of the p candidate teeth. Set Z to be the parts of R-sets split by A which are not in A. If no such R-sets exist, then Z is initially empty. Let G_s denote the weighted shrunk graph obtained from G' by shrinking A, Z and the R-sets which are not contained in $A \cup Z$ into a single node each. Add (iteratively) edges of G_s in order of non-increasing weight so as to build the 'skeleton' of a path bridge configuration, bearing in mind that, when $b = 0$, a 'seed' for Z does not exist. In this case, try to build two structures, one in which Z is forced to be empty and one in which Z is forced to be non-empty.

Once the structure has been made, shrink iteratively vertices of degree one (different from A and Z) into their adjacent node to obtain a PB configuration, not necessarily regular. To make the configuration regular, choose n according to the length of the shortest of the p paths in the configuration, and shrink any paths which are longer than this by merging Z with adjacent V_j^i.

- **Phase III.** Check the n-PB inequality.
  Check if the n-PB inequality is violated according to (7.27). If it is not violated, the slack of the n-PB inequality can be decreased by removing any handle H_i such that $x^*(\delta(H_i)) > p + 1$ and merging adjacent V_j^i accordingly. Repeat this process iteratively, as long as $n \geq 2$ remains, until a violated regular PB inequality is obtained (if possible).

3.2.3 Cutting Plane and Branch & Cut Algorithms for the GRP.

The above heuristic and exact procedures for identifying violated Honeycomb and PB inequalities, as well as those presented in the RPP section for separating connectivity, R-odd and K-C constraints, have been implemented in a cutting plane algorithm described in Corberán, Letchford & Sanchis (1998). In the last iteration, when no violated inequality is found, an integer solution of the last LP relaxation is obtained by invoking the Branch and Bound option of CPLEX (1994). If this integer solution is a tour, then it is optimal for the GRP. Otherwise, its value is a lower bound for the cost of the optimal GRP tour.

Besides the RPP instances mentioned in section 3.1.4 and some pure GTSP instances, the cutting plane algorithm was tested on 10 GRP instances generated from the Albaida graph by selecting visually some edges as required in order to obtain 'difficult' instances and on 30 other (randomly generated) instances. This last set of instances, with up to 111 R-components, was obtained from the Albaida graph and from an-

other real-world based graph — with 196 vertices and 316 edges —, by defining an edge as required with probability p, $p = 0.7, 0.5$ and 0.3, and considering all the vertices of the graph as required. Thirty three out of these 40 were solved to optimality.

The algorithm also proved capable of solving 7 out of 8 GTSP instances (recall that the GTSP, like the RPP, is a special case of the GRP), which were formed by taking planar Euclidean TSP instances from TSPLIB and making the associated graphs sparse.

Finally, note that any GRP instance can be transformed into an RPP instance by duplicating every isolated required vertex and adding a required edge between them with zero cost. Therefore, the Branch and Cut algorithm by Ghiani & Laporte (1997) could also be applied (in principle) to solve the GRP.

3.3. THE DIRECTED RPP

Consider now a strongly connected and directed graph $G = (V, A)$ with nonnegative arc costs and a subset of required arcs $A_R \subseteq A$. The Directed Rural Postman Problem (DRPP) consists of finding a minimum cost tour traversing, at least once, all the arcs in A_R. As in the RPP case, if the subgraph induced by A_R is not connected, the DRPP is $\mathcal{NP}$-hard.

To formulate the problem, we need some new notation. Given $S \subset V$, remember that $\delta^+(S)$ ($\delta^-(S)$) denotes the set of arcs leaving (entering) S. In addition, $\delta_R^+(S) = \delta^+(S) \cap A_R$ and $\delta_R^-(S) = \delta^-(S) \cap A_R$. Finally, as for the undirected case, in order to simplify both the problem formulation and its resolution, DRPP instances are often transformed into instances which satisfy $V_R = V$ (Christofides, Campos, Corberán & Mota, 1986).

If x_a denotes the number of times an arc $a \in A$ is traversed without being serviced, the formulation of the DRPP given by Christofides et al. (1986) and Ball & Magazine (1988) is as follows:

$$\text{Minimize} \qquad \sum_{a \in A} c_a x_a$$

subject to

$$x(\delta^+(i)) + |\delta_R^+(i)| = x(\delta^-(i)) + |\delta_R^-(i)|, \forall i \in V \tag{7.28}$$

$$x(\delta^+(S)) \geq 1, \begin{cases} \forall S = \cup_{k \in Q} V_k, \\ Q \subset \{1, \ldots, p\} \end{cases} \tag{7.29}$$

$$x_a \geq 0 \text{ and integer}, \forall a \in A \tag{7.30}$$

where p is the number of connected components of the graph induced by A_R and V_k, $k = 1, 2, \ldots, p$, denote the corresponding node sets.

Note two important aspects that make the DRPP different to its undirected version. First, all DRPP tours satisfy equations (7.28), any $|V|-1$ of them linearly independent. This means that its corresponding polyhedron is not full-dimensional and, therefore, more difficult to study. Second, no small and easy-to-obtain upper bound is known for the variables in the DRPP formulation, unlike in the undirected case where all the variables are trivially bounded by 2.

Christofides et al. (1986), solved the DRPP with a Lagrangean Relaxation procedure embedded within a Branch and Bound algorithm. With this method, they were able to solve instances with $|V|$ ranging from 13 to 80, $|A|$ from 24 to 180, $|A_R|$ from 7 to 74 and a number p of R-connected components that ranges from 2 to 8. The DRPP polyhedron have been studied by Savall (1990) and by Gun (1993). Furthermore, some of the results in Romero (1997) and in Corberán, Romero & Sanchis (1999) for the Mixed RPP (see section 3.4) apply directly to the DRPP. Then, a cutting plane algorithm for the DRPP could benefit from some of the separation procedures described in that section.

3.4. THE MIXED RPP

In this section, we consider again a strongly connected mixed graph $G = (V, E, A)$ with a cost $c_e \geq 0$ associated to each link (edge or arc) $e \in E \cup A$. Furthermore, consider a subset $E_R \subseteq E$ of required edges and a subset $A_R \subseteq A$ of required arcs. Then, the problem of finding a minimum cost tour traversing, at least once, all the required links is called the Mixed Rural Postman Problem (MRPP). Note that when $A_R = \emptyset$ ($E_R = \emptyset$), the MRPP reduces to the RPP (DRPP) and that if $E_R = E$ and $A_R = A$, we obtain the MCPP (that itself generalizes the CPP and DCPP). Therefore, the MRPP contain, as special cases, almost all the Arc Routing Problems involving only one vehicle and is, obviously, an $\mathcal{NP}$-hard problem in the general case.

Recent works on the MRPP are those by Romero (1997) and by Corberán, Romero & Sanchis (1999). In both studies, a MRPP instance is transformed into a new one in which every vertex $v \in V$ is incident on, at least, one required link and where $E_R = E$ (as each non required edge $e = (i, j)$ can be replaced by two non required arcs (i, j) and (j, i)). Under these assumptions, if x_e, $e \in E \cup A$, represents the number of copies of link e that are added to G in order to obtain an eulerian graph, the MRPP formulation is then as follows:

$$\text{Minimize} \quad \sum_{e \in E \cup A} c_e x_e$$

subject to

$$x(\delta(i)) + |\delta_R(i)| + x(\delta^+(i)) + |\delta_R^+(i)| + x(\delta^-(i)) + |\delta_R^-(i)| \equiv 0 \pmod 2, \forall i \in V \tag{7.31}$$

$$x(\delta^+(S)) - x(\delta^-(S)) + x(\delta(S)) \geq b(S), \forall S \subset V \tag{7.32}$$

$$x(\delta^+(S)) \geq 1, \begin{cases} \forall S = \cup_{k \in Q} V_k, \\ Q \subset \{1, \ldots, p\} \end{cases} \tag{7.33}$$

$$x_e \geq 0 \text{ and integer}, \forall e \in E \cup A \tag{7.34}$$

where, in a similar way to that in Section 2.3, $b(S)$ is defined as $b(S) = |\delta_R^-(S)| - |\delta_R^+(S)| - |\delta_R(S)|$ and p is the number of connected components induced in G by the required links (called R-components) and subsets V_k, $k = 1, 2, \ldots, p$, are their corresponding node sets (called R-sets).

In their paper about the MRPP polyhedron, Corberán, Romero & Sanchis (1999) present some computational results with a preliminary cutting-plane algorithm including separation routines for connectivity (7.33), R-odd cut and balanced set inequalities (7.32). This algorithm was tested on a set of 100 randomly generated instances with $20 \leq |V| \leq 100$, $15 \leq |E| \leq 200$, $55 \leq |A| \leq 350$, and up to 15 R-connected components. It produced the optimal solution in 28 out of them and, on average, the bound obtained was less than 0.5% far from the cost of a feasible solution obtained by using heuristic methods. The separation algorithms (similar to that used for the undirected RPP) are described in what follows.

3.4.1 Connectivity Separation.

Consider the shrunk graph G_s obtained from $G(x^*)$ by shrinking each R-set into a single node. Note that G_s is a directed graph.

Exact algorithm:

Connectivity inequalities can be separated exactly in polynomial time by computing a maximum directed flow between every pair of vertices of G_s. Each cutset with weight less than 1 corresponds to a violated connectivity inequality.

Heuristic algorithm:

Compute the strongly connected components of the subgraph induced by the links e of G_s with $\bar{x}_e^* > \epsilon$, where ϵ is a given parameter. Let

$S_1, S_2, \ldots, S_q$ be the sets of nodes in the original graph G corresponding to the node sets of these strongly connected components. Then, $x(\delta^+(S_i)) \geq 1$ is a violated connectivity inequality if $x^*(\delta^+(S_i)) < 1$. Constraint $x(\delta^-(S_i)) \geq 1$ is not considered because $\delta(S_i) = \emptyset$ and, therefore, balanced set constraints imply $x(\delta^+(S_i)) = x(\delta^-(S_i))$.

3.4.2 *R*-odd Cut and Balanced Set Separation.

Let $\delta^*(S) = \delta^+(S) \cup \delta^-(S) \cup \delta(S)$ and $\delta_R^*(S) = \delta^*(S) \cap (E_R \cup A_R)$. R-odd cut inequalities

$$x(\delta^*(S)) \geq 1, \quad \forall \ S \subseteq V \ : \ |\delta_R^*(S)| \text{ is odd} \tag{7.35}$$

can also be separated exactly in polynomial time, in a similar way to the odd-cut inequalities separation described in section 2.1.1 for the CPP. The heuristics presented in that section also apply directly. We only have to consider each arc in A as an edge and each R-odd vertex as an odd-degree vertex.

The balanced set separation algorithm described in section 2.3.3 for the MCPP applies directly to the MRPP.

3.4.3 *K*-C and Path-Bridge Separation.

No K-C or Path-Bridge separation procedures for the Mixed RPP have been proposed. Nevertheless, some subclasses of the K-C and PB inequalities described in Corberán, Romero & Sanchis (1999) for the MRPP have the same coefficients as those for the undirected RPP. Hence, the separation algorithms described in sections 3.2.2 and 3.1.3 could be slightly modified to be applied to the mixed case by only considering every arc of the graph as an edge (ignoring its direction).

4. THE CAPACITATED ARC ROUTING PROBLEM

The Capacitated Arc Routing Problem (CARP) was introduced by Golden & Wong (1981) and is defined as follows. Let $G = (V, E)$ be a connected and undirected graph with a demand $d_e \geq 0$ and a traversing cost $c_e \geq 0$ associated to each edge $e \in E$. The subset of edges with positive demand (called *required edges*) is denoted by E_R. Given a vehicle capacity Q, the CARP consists of finding a set of vehicle routes of minimum cost that service every required edge and such that every route contains the *depot* (that will be assumed to be vertex 1) and the total demand serviced by a route does not exceed the capacity Q. A route is a closed chain containing a set of traversed edges, some of which are also serviced by the route.

A special case of this problem is the Capacitated Chinese Postman Problem (CCPP) where all the edges are required ones. Note that the number of vehicles is free in the above definition of CARP. Some authors consider different variants of the CARP in which the number of vehicles is fixed, or an upper bound for it is given.

The minimum number of vehicles that will be used by any feasible solution to the CARP may be computed by solving a Bin Packing Problem (BP) where the item weights are the edge demands and the bin capacity is Q. Let K^* denote this minimum number of vehicles. The Bin Packing Problem is $\mathcal{NP}$-hard, but it can be easily solved for relatively large instances (see Martello & Toth, 1990).

Several LP formulations of the CARP exist in the literature and a survey of them can be found in Chapter 6 of this book. Here, we will only report about those formulations that have been used to develop solution strategies for the CARP. As far as we know, the only work done on LP-based methods for the CARP is reported in Belenguer & Benavent (1992, 1994, 1998a, 1998b) and Welz (1994). Different formulations of the CARP have been used in these works. Following the terminology used in Chapter 6 of this book, these formulations have been classified into sparse and supersparse. Welz (1994) uses a sparse formulation with directed variables, while Belenguer & Benavent use both classes of formulations using non directed variables. These formulations have been computationally tested and the results are quite encouraging. To summarize their results, we first present the valid inequalities that have been obtained for the sparse formulation and the corresponding routines implemented to separate them. Afterwards, we review the corresponding work made on the supersparse formulation. Finally we will present the results obtained with a Branch & Cut code for the CARP based on the sparse formulation and a cutting plane algorithm based on the supersparse formulation.

4.1. SPARSE FORMULATIONS

Belenguer & Benavent (1992, 1994, 1998a) assume that there is an upper bound K on the number of available vehicles. This is normally taken to be equal to K^*, but need not be. They use the following decision variables to formulate the CARP:

$$x_e^p = \begin{cases} 1 & \text{if vehicle } p \text{ serves edge } e \in E_R \\ 0 & \text{otherwise} \end{cases}$$

y_e^p = number of times vehicle p traverses edge $e \in E_R$ without servicing it.

Let $I = \{1, \ldots, K\}$. Given any $S \subseteq V \setminus \{1\}$, as in previous sections, we will denote by $E_R(S) = E(S) \cap E_R$ and $\delta_R(S) = \delta(S) \cap E_R$. On the other hand, given $p \in I$, $R' \subseteq E_R$ and $E' \subseteq E$, we denote $x^p(R') = \sum_{e \in R'} x_e^p$ and $y^p(E') = \sum_{e \in E'} y_e^p$. Then, the CARP can be formulated as follows:

$$\text{Minimize} \qquad \sum_{p \in I} \sum_{e \in E_R} c_e x_e^p + \sum_{p \in I} \sum_{e \in E} c_e y_e^p$$

subject to

$$\sum_{p \in I} x_e^p = 1, \ \forall \ e \in E_R \tag{7.36}$$

$$\sum_{e \in E_R} d_e x_e^p \leq Q, \forall \ p \in I \tag{7.37}$$

$$x^p(\delta_R(S)) + y^p(\delta(S)) \geq 2x_f^p, \ \forall \ S \subseteq V \setminus \{1\}, \ f \in E_R(S) \text{ and } p \in I \tag{7.38}$$

$$x^p(\delta_R(S)) + y^p(\delta(S)) \equiv 0 \pmod 2, \ \forall \ S \subseteq V \setminus \{1\} \text{ and } p \in I \tag{7.39}$$

$$x_e^p \in \{0,1\}, y_e^p \geq 0 \text{ and integer} \tag{7.40}$$

Constraints (7.36) and (7.37) ensure, respectively, that each required edge will be serviced and that the capacity of the vehicles is not exceeded. Constraints (7.38) (called connectivity constraints) state that, if route p services edge e then it must traverse any edge cutset separating e from the depot.

In what follows, we present the separation procedures designed for each class of valid inequalities known for this formulation. Constraints (7.36) and (7.37) are usually included in the initial LP. Let (x, y) be an optimal solution of the LP containing the objective function and some set of valid constraints for the CARP. For any fixed vehicle $p \in I$, let $w_e^p = x_e^p + y_e^p$, for all $e \in E$ (for simplicity, we assume that $x_e^p = 0$ whenever $e \notin E_R$), and let $G_p(w)$ be the graph induced by the edges $e \in E$ with $w_e^p > 0$ plus the depot (node 1).

4.1.1 Connectivity Separation.

Constraints (7.38) can be separated in polynomial time by applying the following algorithm for each vehicle p. Consider the weight w_e^p as the capacity of arc $e \in E$ in graph $G_p(w)$.

Exact algorithm

1 For each node $i \in V \setminus \{1\}$:

Compute the min-cut separating nodes 1 and i. Let $\delta(S_i), S_i \subseteq V \setminus \{1\}$, be the min-cut and let F_i be the capacity of this cut.

2 For each edge $e = (i, j) \in E_R$:

If $max\{F_i, F_j\} < 2x_e^p$, then a violated connectivity constraint has been found.

Let F_e be the min-cut separating node 1 and edge $e \in E_R$. Note that $F_e \geq max\{F_i, F_j\}$. Then, $max\{F_i, F_j\} \geq 2x_e^p$ implies that no connectivity constraint is violated for this edge. On the other hand, it can be proved that, if $max\{F_i, F_j\} < 2x_e^p$, then either $j \in S_i$ or $i \in S_j$ holds. Then, $max\{F_i, F_j\} < 2x_e^p$ implies that either $F_e = F_i$ or $F_e = F_j$ holds, so $F_e < 2x_e^p$ and a violated connectivity constraint can be generated.

4.1.2 Parity Separation.

Parity constraints (7.39) are not linear. In Belenguer & Benavent (1992,1998) they are substituted by the weaker, but linear, constraints:

$$x^p(\delta_R(S) \setminus H) + y^p(\delta(S)) \geq x^p(H) - |H| + 1 \qquad (7.41)$$

were $S \subseteq V \setminus \{1\}$ and $H \subseteq \delta_R(S)$ with $|H|$ odd. To see that they are valid, note that if all edges in H are serviced by vehicle p (that is $x^p(H) = |H|$), given that $|H|$ is odd, this vehicle will traverse at least once more the edge cut set $\delta(S)$, so $x^p(\delta_R(S) \setminus H) + y^p(\delta(S)) \geq 1$. Constraints (7.41), which will also be called parity constraints in what follows, are similar to those appearing in the complete description of the 2-matching polytope (see for instance Grötschel & Holland, 1987).

Exact algorithm:

The separation of constraints (7.41) can be done in polynomial time using a procedure similar to one described in Padberg & Rao (1982).

From the support graph $G_p(w)$, a new graph G' is constructed as follows: initially, all the nodes in $G_p(w)$ are labelled as *even*; then, sequentially, each required edge $e = (i, j)$ in $G_p(w)$ is replaced by two edges (i, i_e), which gets a weight $1 - x_e$, and (i_e, j) with weight x_e, where i_e is a new node labelled as *odd*. The label of node i is changed from *even* to *odd* (or vice-versa). Finally, for each edge $e = (i, j) \in E$, if $y_e^p > 0$, an edge joining nodes i and j is added to G' with a weight of y_e^p. Compute now a minimum weight odd cut set in graph G' and let S' be the set of nodes on the shore of the depot defining this cut. Let S be the set of original nodes in S'. The set H is defined as the subset of edges $e = (i, j) \in \delta_R(S)$ such that $i_e \notin S'$. It can be easily shown that $|H|$ is odd and the weight of the cutset in G' can be written as:

$$y^p(\delta(S)) + x^p(\delta_R(S) \setminus H) + |H| - x^p(H)$$

Therefore, a violated parity inequality is found if and only if the above expression is less than 1.

4.1.3 Obligatory Cutset Separation.

Let D_T be the total demand of the required edges and let $Q_{min} = max\{D_T - (K-1)Q, 0\}$. Note that Q_{min} is the minimum demand that any vehicle must service in any feasible solution to the CARP. For any $S \subseteq V \setminus \{1\}$, let $d(S) = \sum_{e \in E_R(S) \cup \delta_R(S)} d_e$. Then, the following inequality is valid:

$$x^p(\delta_R(S)) + y^p(\delta(S)) \geq 2 \qquad (7.42)$$
$$\text{for all } p \in I \text{ and } S \subseteq V \setminus \{1\}, D_T - d(S) < Q_{min}$$

Note that $D_T - d(S) < Q_{min}$ implies that each vehicle p have to traverse the edge cutset $\delta(S)$. For this reason, (7.42) are called obligatory cutset constraints. Obviously, they only make sense if $Q_{min} > 0$. No specific algorithm was devised to separate these constraints. Instead, a heuristic was used to generate several edge cutsets for which the connectivity, parity and obligatory cutset constraints were checked for possible violation. The heuristic computes the connected components of $G_p(w)$ and that of $G_p(w) \setminus \{e\}$, where e is any cut edge of $G_p(w)$.

4.1.4 Separation of Constraints from the Knapsack Problem.

Obviously, each vehicle capacity constraint (7.37) is a Knapsack like constraint. Then, any valid inequality for the Knapsack polytope: $conv\{z : \sum_{e \in E_R} d_e z_e \leq Q\}$, generates K valid inequalities for the CARP, one for each vehicle p.

Two classes of valid knapsack constraints were used: minimal cover and $(1-k)$ configuration constraints.

A set $W \subseteq E_R$ is called a minimal cover if $d(W) > Q$ and $d(W \setminus \{f\}) \leq Q$ for all $f \in W$. If W is a minimal cover, then the constraint $x^p(W) \leq \mid W \mid -1$, for any vehicle p, is valid for the CARP.

A set $W \subseteq E_R$, an edge $f \in E_R \setminus W$ and an integer $2 \leq t \leq \mid W \mid$ are said to define a $(1-k)$ configuration if:

- $d(W) \leq Q$, and
- $W' \cup \{f\}$ is a minimal cover for every $W' \subseteq W$ such that $\mid W' \mid = t$.

Then, for any integer r, $t \leq r \leq \mid W \mid$, and any subset $T_r \subseteq W$ such that $\mid T_r \mid = r$, the inequality $(r-k+1)x_f^p + x^p(T_r) \leq r$, for any vehicle p, is valid for the CARP.

It is well known that minimal cover and $(1-k)$ configuration constraints can be lifted to produce facets of the knapsack polytope. Lifted

constraints are not guaranteed to be facets for the CARP but they can still be expected to be strong. The heuristic methods described in Crowder, Johnson & Padberg (1983) were used to separate and lift these constraints.

In the work reported in Welz (1994), the CARP is formulated using variables that take into account the direction in which each edge is traversed. This formulation is fully described in Chapter 6, so it will not be reproduced here. Welz uses separation heuristics for his odd cut and connectivity inequalities that are similar to the ones here described.

4.2. SUPERSPARSE FORMULATION FOR THE CARP

Consider any feasible solution of the CARP and let us define for each edge $e \in E$, the aggregated variable:

z_e = *total number of times that edge e has been traversed without being serviced by all the vehicles.*

For any $S \subseteq V \setminus \{1\}$, let $k(S) = \lceil d(S)/Q \rceil$. Obviously, at least $k(S)$ vehicles are needed to service the edges in $E_R(S) \cup \delta_R(S)$.

Consider the following supersparse formulation for the CARP:

$$\text{Minimize} \qquad \sum_{e \in E} c_e z_e$$

subject to

$$z(\delta(S)) \geq 2k(S) - \mid \delta_R(S) \mid \quad \forall\ S \subseteq V \setminus \{1\} \tag{7.43}$$

$$z(\delta(S)) \geq 1 \quad \forall\ S \subseteq V \setminus \{1\}, \mid \delta_R(S) \mid \text{ odd} \tag{7.44}$$

$$z_e \geq 0 \quad \forall\ e \in E \tag{7.45}$$

$$z_e \text{ integer} \quad \forall\ e \in E \tag{7.46}$$

Constraints (7.43) are called capacity constraints and they express the fact that at least $k(S)$ vehicles must traverse the edge cutset $\delta(S)$ to service the edges in $E_R(S) \cup \delta_R(S)$. Constraints (7.44) are the usual odd-cut constraints which are also valid for the CARP. Note that the objective function includes only the cost of the deadheading edges of the CARP solution; the real cost includes also that of traversing exactly once all the required edges, but this a fixed cost that cannot be minimized. Note also that the number of vehicles is free in this formulation.

On the other hand, note that the number of variables is $\mid E \mid$, which is much less than that of the sparse formulation (at least $K(\mid E \mid + \mid E_R \mid)$).

The main drawback of the supersparse formulation is that it is not complete, as it contains integer solutions that do not correspond to feasible solutions for the CARP.

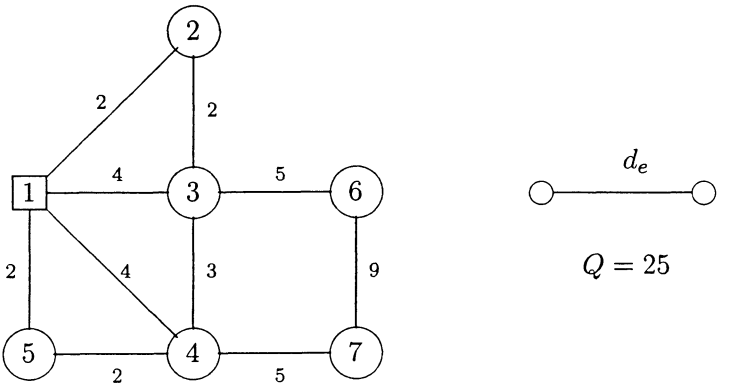


Figure 7.1 CARP instance.

Consider the CARP instance of Figure 7.1. Note that the solution $\{z_e = 0 : e \in E\}$ satisfies all the capacity and odd-cut constraints. Nevertheless, this solution cannot correspond to a feasible solution for the CARP, as we now show. Consider the edge cutset defined by $S = \{6, 7\}$, then $\delta_R(S) \cup E_R(S) = \{(3,6), (7,4), (6,7)\}, d(S) = 19$ and $k(S) = 1$. Given that no deadheading edge exists in the edge cutset $\delta(S)$, a single vehicle will have to service the edges in $\delta_R(S) \cup E_R(S)$. This vehicle will travel from the depot to node 3 or 4, service the edges in $\delta_R(S) \cup E_R(S)$ and come back to the depot. Therefore, it will use two edge-disjoint paths in graph G from the depot to nodes 3 and 4; but these paths have a total demand of at least 8 and $8 + 19 = 27$. As $Q = 25$, this vehicle would exceed its capacity.

Although the supersparse formulation is not complete, it has been used in Belenguer & Benavent (1998b) to compute a lower bound for the CARP with a cutting plane algorithm that includes constraints (7.43) and (7.44) as well as other new valid inequalities which are called Disjoint Path inequalities. There are three classes of Disjoint Path inequalities, denoted, respectively, DP1, DP2 and DP3, based on similar ideas.

For instance, the following is a DP1 valid inequality for the CARP instance depicted in Figure 7.1:

$$2z(E') + z(\delta(S)) \geq 2,$$

where $S = \{6, 7\}$ and $E' = \{(1,2), (1,3), (1,4), (1,5), (2,3), (4,5)\}$. The validity of this inequality becomes apparent by noting that, either:

(a) at least one deadheading edge in the set E' is used, so $z(E') \geq 1$, or

(b) the edges in $\delta_R(S) \cup E_R(S)$ are serviced by more than one vehicle, so $z(\delta(S)) \geq 2$.

In general, given a node subset $S \subseteq V \setminus \{1\}$, such that $2k(S) \geq | \delta_R(S) |$, and edge subset $E' \subseteq E(V \setminus S)$, the DP1 inequality associated to S and E' is:

$$2z(E') + z(\delta(S)) \geq 2k(S) - | \delta_R(S) | + 2 \qquad (7.47)$$

This inequality is valid if $d(MCFP) + d(S) > k(S)Q$, where $d(MCFP)$ is a lower bound on the total demand that $k(S)$ vehicles would have to service on their way from the depot to edge cutset $\delta(S)$ and on return, assuming that $z(E') = 0$. This lower bound can be computed by solving a Minimum Cost Flow Problem where edge costs are set equal to the edge demands.

Inequalities DP2 and DP3 are based on similar ideas, although they are defined on a more involved configuration. Thus, while inequalities DP1 are defined by a pair (S, E'), $S \subseteq V \setminus \{1\}$ and $E' \subseteq E$, inequalities DP2 and DP3 are defined by a sequence $S_0 \subseteq S_1 \subseteq \ldots \subseteq S_t \subseteq V \setminus \{1\}$ and a subset of arcs $E' \subseteq E$ which is empty in inequalities DP2. We refer to the paper by Belenguer & Benavent (1998) for the details.

Let us consider now the separation routines that have been used for the constraints in the supersparse formulation.

Let $\{z_e : e \in E\}$ be the optimal solution of an LP containing the objective function, non-negativity constraints (7.45) and a subset of other valid constraints for the supersparse formulation. Let $G(z)$ be the graph induced by the edges $e \in E$ with $z_e > 0$, plus the depot (vertex 1).

The separation of the odd-cut constraints (7.44) can be done in polynomial time with the algorithm of Padberg & Rao as explained in section 3.1.1.

4.2.1 Capacity Constraints Separation.

The problem of identifying violated inequalities of type (7.43) seems to be more difficult. In Belenguer & Benavent (1998a, 1998b) the following heuristics were used to generate sets of nodes for which (7.43) is checked for possible violation.

1 Compute the node sets of the connected components of $G(z)$.

2 Find a node set $S \subseteq V \setminus \{1\}$ for which $f(S) = \sum_{e \in \delta(S)} z_e - 2d(S)/Q+ \mid \delta_R(S) \mid$ is minimum. This can be done in polynomial time by solving a maximum flow problem on a transformed graph. If we consider the inequality that results when substituting in (7.43) $\lceil d(S)/Q \rceil$ by $d(S)/Q$, then $f(S)$ represents the slack of this inequality, so $f(S) < 0$ implies that it is violated by the LP solution and, therefore, (7.43) is also violated. Obviously, (7.43) must be checked even if $f(S) \geq 0$ as it is a stronger inequality. This procedure is similar to the one due to Harche and Rinaldi (1993) for the CVRP.

3 Substitute the demand d_e of every edge e by $(1 + p)d_e$, where $0 < p < 1$, and apply the previous procedure. Ten different values for p have been used.

4.2.2 Disjoint Path Inequalities Separation.

The identification of Disjoint Path (DP) inequalities is far more complex than that of the previous ones. The method proposed by Belenguer & Benavent(1998b) is as follows. The separation procedures used for the capacity and odd-cut constraints generate a number of edge cutsets whose corresponding node sets are stored in a pool to be used in the separation of DP inequalities. From this pool, three lists are built: INI, a list of node sets, and SEQ2 and SEQ3, two lists of node sequences. Each element of INI, SEQ2 and SEQ3, together with an edge set E', defines a candidate configuration for an inequality DP1, DP2 or DP3, respectively. The edge set E' is usually defined as $E' = \{e \in E(V \setminus S) : z_e = 0\}$ for the DP1 inequality and similarly for a DP3 inequality. Given a candidate configuration for a DP inequality, first the tentative inequality is checked for violation, and, if it is violated, it is then checked for validity, which involves the solution of a maximum flow problem.

Once a violated (and valid) DP inequality is found, an attempt is made to strengthen it in two ways: eliminating some edges of E' and using the parity requirement to decrease from 2 to 1 the coefficients of z_e for the remaining edges $e \in E'$. We refer to Belenguer & Benavent(1998b) for the details.

4.3. EXACT METHODS BASED ON THE SPARSE FORMULATION

Belenguer & Benavent(1994) have developed a Branch & Cut code for the CARP based on the sparse formulation but including also capacity constraints (7.43) and odd-cut constraints (7.44). Note that any valid inequality for the supersparse formulation can be converted into a valid inequality in the sparse formulation by substituting $z_e = \sum_{p \in I} y_e^p$.

The algorithm first looks for violated constraints of types (7.43) and (7.44). The separation routines for the other inequalities (from the sparse formulation) are called only in those iterations where no violated inequality of the former types is found.

At any iteration of the cutting plane algorithm, if the LP solution is integer and represents a feasible solution of the CARP, it is the optimal solution. Otherwise, the separation routines are called to identify violated inequalities and, if no one is found, a branching step is executed. The following branching rule was used: select the edge e with greatest demand among those for which x_e^p is not integer for some vehicle p; then, create K new nodes by fixing the variable $x_e^j = 1$ for each vehicle $j \in I$. The possibility of facing an LP solution in which all the x_e^p variables are integral but which does not represent a CARP solution (for instance, because some y_e^p were fractional) was not contemplated by the code, but this possibility never occurred in the instances tested.

The algorithm was tested on the set of 24 instances used in Benavent, Campos, Corberán & Mota (1992) having a number of vehicles less than 5 . From this set, 16 instances were solved to optimality with less than 53 nodes in the Branch & Cut tree. Nevertheless, it was observed that the instances optimally solved were precisely those for which the lower bound at the root node was equal to the optimal cost. This lower bound value was reached at the first stage of the cutting plane algorithm, where only constraints (7.43) and (7.44) are used. The only use of constraints from the sparse formulation, therefore, is to encourage integrality in the LP solutions.

Welz (1994) has implemented a Branch & Bound algorithm for the CARP based on his formulation. At the root node, he uses a cutting plane algorithm where violated odd cut and connectivity constraints are identified and added to the LP. When no violated inequality is found and the solution is not feasible, an integer solution of the last LP is obtained by invoking a Branch and Bound procedure. If this does not yield a feasible solution, he adds more inequalities and tries again. The largest

instances solved with this approach were : K=2, $|V|$=27, $|E|$=82 ; K=3, $|V|$=16, $|E|$=48 and K=4, $|V|$=12, $|E|$=50.

4.4. A CUTTING PLANE ALGORITHM FOR THE CARP BASED ON THE SUPERSPARSE FORMULATION

Belenguer & Benavent(1998b) developed a cutting plane algorithm for the CARP that uses all the previous separation routines for constraints of the supersparse formulation. The lower bound obtained with this algorithm outperforms any other existing lower bounding procedure for the CARP. Nevertheless, even in the case that the LP solution is integer, the optimal solution cannot be obtained with this approach without embedding the cutting plane algorithm into a Branch & Cut scheme. The optimality of the computed lower bound can be stated only if a heuristic CARP solution with the same cost is known. Comparing the lower bound with the best upper bound known, it was found that it was optimal for 21 out of the 24 above mentioned instances from Benavent et al. (1992). In this paper 10 more difficult instances were also introduced each with a number of vehicles greater than 7. For these instances, the lower bound obtained at the root node with the above described Branch & Cut algorithm that uses the sparse formulation was on average 4.18% over the best known upper bound, while the lower bound obtained with the cutting plane algorithm that uses the supersparse formulation had an average deviation of 2.31%. On the other hand, the cutting plane algorithm was also tested on the 23 instances used by Golden, DeArmon & Baker (1983) and Pearn (1989) (removing 2 of them that presented inconsistencies) and the lower bound was optimal in 19 of them, with an average gap over the best upper bound known of 0.33%.

5. OTHER PROBLEMS

Letchford and Eglese (1997) studied an interesting Arc Routing Problem with Time Windows: The Rural Postman Problem with Deadline Classes (RPPDC). Given that routing problems with time windows are, in general, very hard to solve to optimality, some relaxations or special cases of the general problem have been studied. In this case, customers are divided into a small number of priority classes, each class having its own time deadline. The paper describes a realistic situation with applications to the real world. The authors present a formulation of the problem, a wide number of valid inequalities and a cutting-plane algorithm useful for solving instances of moderate size.

In more detail, Letchford and Eglese (1997) deal with the problem of finding a minimum cost route traversing a subset, E_R, of the edges of a

graph where E_R is divided into a number of *deadline classes* $R^1, R^2, \ldots,$ R^L each class having its own time deadline: customers in R^1 must be serviced by time T^1, customers in R^2 must be serviced by time T^2, and so on. A feasible RPPDC route is regarded as being composed of L *phases*. In phase i the vehicle services the edges in R^i plus (optionally) some edges in $R^{i+1} \cup \ldots \cup R^L$. In the formulation of the problem, the introduction of variables z_{vp} defined as 1 if phase p ends at vertex v and 0 otherwise, allows the global route to be split into in some partial routes corresponding to the different phases, in a similar way to the case of individual vehicle routes in the multivehicle problems: for each B, $1 \leq B \leq L$, the vector $\sum_{i=1}^{B}(x^i + y^i) + z^i$, where $x^i \in I\!R^E$ represents the number of times each edge is traversed *without* servicing and $y^i \in I\!R^E$ is the incidence vector of the edges serviced in phase i, represents a solution of the "quasi-RPP" problem defined on graph G, where the edges in $R^1 \cup \ldots \cup R^B$ are required edges and, furthermore, the route must cross exactly one edge in $\{(v,0), v \in V\}$.

Hence, some valid inequalities can be obtained in a natural way from the facet-inducing inequalities for L separate RPP instances. These are called *Strong Cumulative* constraints (connectivity, R-odd, K-C, etc.) by the authors. Other constraints, representing the interaction of different phases, are also introduced.

Letchford and Eglese (1997) implement a dual cutting-plane algorithm in which violated inequalities are identified and added to an initial LP relaxation as cutting-planes. When no more violated inequalities can be found, branch-and-bound is invoked to obtain integrality. If the resulting integer solution is not feasible and violates more known inequalities, those inequalities are added to the LP and the cutting plane procedure continues again, followed by branch-and-bound and so on. The test problems were adapted from the 5 most difficult instances in Corberán and Sanchis (1994). For each one of this instances, two deadline versions were solved, with $L = 1$ and with $L = 2$.

6. CONCLUSIONS

In this chapter, we have outlined the LP based methods, cutting plane and Branch & Cut procedures, that have been implemented for several Arc Routing Problems (ARP). Special attention have been given to the inequalities for which a separation procedure have been proposed, as well as to the procedures themselves. These methods have proved to be very useful to solve ARPs, just as they are for many other Combinatorial Optimization Problems (COPs).

It can be said that they have produced best results, when compared with other methods, on almost all ARPs to which they have been applied. Nevertheless, for most of them, only cutting plane algorithms have been implemented. Although many instances have been solved using such algorithms, cutting plane methods do not guarantee to solve exactly all the instances, but only to produce a lower bound. This is therefore only a first step: to obtain good lower bounds assessing the quality of the solutions produced by the many new heuristic approaches for COPs and, particularly, for ARPs (see Chapter 9 of this book), as well as to give information about how good our understanding of their associated polyhedra is. The following step, the implementation of a complete Branch & Cut scheme producing optimal or provably good solutions (see Jünger, Reinelt & Thienel, 1994), remains to be done for most of the ARPs.

The work done so far provides a library of tools that makes it easier to attack the resolution of more complex ARPs: many classes of valid inequalities and their corresponding separation procedures are known. As it has been presented here, many ARPs share some classes of valid inequalities, maybe with minor differences, so separation algorithms can be adapted easily. The hope of the authors is that the ideas presented in this chapter can be used for the resolution of other Arc Routing Problems.

Acknowledgments

We are grateful to Adam Letchford who read an earlier version of the manuscript and made many valuable comments.

References

[1] Assad, A.A. & B.L. Golden (1995) Arc routing methods and applications. In M.O. Ball, T.L. Magnanti, C.L. Monma & G.L. Nemhauser (Eds.) *Network Routing.* Handbooks of Operations Research and Management Science, 8. Amsterdam: North Holland.

[2] Ball, M.O. & M.J. Magazine (1988) Sequencing of insertions in printed circuit board assembly. *Operations Research, 36,* 192-201.

[3] Belenguer, J.M. & E. Benavent (1992) Polyhedral Results on the Capacitated Arc Routing Problem. *Working Paper,* Dept. of Stats and OR, University of Valencia, Spain.

[4] Belenguer, J.M. & E. Benavent (1994) A Branch and Cut algorithm for the Capacitated Arc Routing Problem. Workshop on Algorithmic Approaches to Large and Complex Combinatorial Optimization Problems. Giens, France.

[5] Belenguer, J.M. & E. Benavent (1998a) The capacitated arc routing problem: valid inequalities and facets. *Computational Optimization & Applications, 10,* 165-187.

[6] Belenguer, J.M. & E. Benavent (1998b) A cutting-plane algorithm for the capacitated arc routing problem. *Working Paper,* Dept. of Stats and OR, University of Valencia, Spain.

[7] Benavent, E., V. Campos, A. Corberán & E. Mota (1992) The Capacitated Arc Routing Problem : lower bounds. *Networks, 22,* 669-690.

[8] CPLEX Optimization Inc., CPLEX, ver. 3.0, 1994.

[9] Christofides, N. (1973) The optimum traversal of a graph. *Omega, 1,* 719-732.

[10] Christofides, N., E. Benavent, V. Campos, A. Corberán & E. Mota (1984) An optimal method for the mixed postman problem, in P. Thoft-Christensen (Ed.) *System Modelling and Optimization,* Lecture Notes in Control and Inf. Sciences, 59. Berlin: Springer.

[11] Christofides, N., V. Campos, A. Corberán & E. Mota (1981) An algorithm for the rural postman problem. *Report IC.O.R.81.5,* Imperial College, London.

[12] Christofides, N., V. Campos, A. Corberán & E. Mota (1986) An algorithm for the rural postman problem on a directed graph. *Mathematical Programming Study, 26,* 155-166.

[13] Corberán, A., A. Letchford & J.M. Sanchis (1998) A Cutting Plane Algorithm for the General Routing Problem. *Working Paper.* Dept. of Stats and OR, University of Valencia, Spain.

[14] Corberán, A., A. Romero & J.M. Sanchis (1999) The General Routing Problem on a Mixed Graph. *Working paper,* Dept. of Stats and OR, University of Valencia, Spain.

[15] Corberán, A. & J.M. Sanchis (1994) A polyhedral approach to the rural postman problem. *European Journal of Operational Research, 79,* 95-114.

[16] Corberán, A. & J.M. Sanchis (1998) The general routing problem polyhedron: facets from the RPP and GTSP polyhedra. *European Journal of Operational Research, 108,* 538-550.

[17] Cornuéjols, G., J. Fonlupt & D. Naddef (1985) The traveling salesman problem on a graph and some related integer polyhedra. *Mathematical Programming, 33,* 1-27.

[18] Crowder, H.P., E.L. Johnson & M.W. Padberg (1983) Solving Large-Scale Zero-One Linear Programming Problems *Operations Research 31,* 803-834.

[19] Dantzig, G.B., D.R. Fulkerson & S. M. Johnson (1954) Solution of a large scale traveling salesman problem *Operations Research, 2*, 393-410.

[20] Edmonds, J. (1963) The Chinese postman problem. *Operations Research, 13*, Suppl. 1, B73-B77.

[21] Edmonds, J. (1965) Maximum Matching and a Polyhedron with 0,1 Vertices. *Journal of Research National Bureau of Standards., 69B*, 125-130.

[22] Edmonds, J. & E.L. Johnson (1973) Matchings, Euler tours and the Chinese postman. *Mathematical Programming, 5*, 88-124.

[23] Eiselt, H.A., M. Gendreau & G. Laporte (1995) Arc-routing problems, part 1: the Chinese postman problem. *Operations Research, 43*, 231-242.

[24] Eiselt, H.A., M. Gendreau & G. Laporte (1995) Arc-routing problems, part 2: the rural postman problem. *Operations Research, 43*, 399-414.

[25] Fischetti, M., J.J. Salazar & P. Toth (1997) A Branch and Cut Algorithm for the Symmetric Generalized Traveling Salesman Problem. *Operations Research* 45, 378-394.

[26] Fleischmann, B. (1985) A cutting-plane procedure for the traveling salesman problem on a road network. *European Journal of Operational Research, 21*, 307-317.

[27] Ford, L.R. & D.R. Fulkerson (1962) *Flows in Networks.* Princeton University Press, Princeton, NJ.

[28] Ghiani, G. & G. Laporte (1997) A Branch-and-Cut Algorithm for the Undirected Rural Postman Problem. *Centre de Recherche sur les Transports, University of Montreal.* Technical Report CRT-97-54 (December 1997).

[29] Golden, B.L. & R.T. Wong (1981) Capacitated arc routing problems. *Networks, 11*, 305-315.

[30] Golden, B.L. J.S. DeArmon & E. Baker (1983) Computational experiments with Algorithms for a class of Routing Problems. *Computers and Operations Research* 10, 47-59.

[31] Gomory, R.E. (1958) Outline of an algorithm for integer solutions to linear programs. *Bull. Amer. Math. Soc. 64*, 275-278.

[32] Gomory, R.E. (1963) An algorithm for integer solutions to linear programs. In R.L. Graves and P. Wolfe, eds., *Recent Advances in Mathematical Programming*, McGraw Hill, New York, 269-302.

[33] Gomory, R.E. & T.C. Hu (1961) Multi-terminal network flows. *SIAM Journal on Applied Mathematics., 9*, 551-570.

[34] Grötschel, M. & O. Holland (1985) Solving matching problems with linear programming. *Mathematical Programming, 33*, 243-259.

[35] Grötschel, M. & O. Holland (1987) A Cutting Plane Algorithm for Minimum Perfect 2-Matchings. *Computing, 39*, 327-344.

[36] Grötschel, M. & M.W. Padberg (1979) On the symmetric traveling salesman problem I: inequalities. *Mathematical Programming, 16*, 265-280.

[37] Grötschel, M. & Z. Win (1988) On the windy postman polyhedron. Report No. 75, Schwerpunkt-progam der Deutschen Forschungsgeneinschaft, Universität Augsburg, Germany.

[38] Grötschel, M. & Z. Win (1992) A cutting plane algorithm for the windy postman problem. *Mathematical Programming, 55*, 339-358.

[39] Guan, M. (1962) Graphic programming using odd or even points. *Chinese Mathematics, 1*, 237-277.

[40] Guan, M. (1984) On the windy postman problem. *Discrete Applied Mathematics, 9*, 41-46.

[41] Gun, H. (1993) Polyhedral structure and efficient algorithms for certain classes of directed rural postman problem. *PhD dissertation*, Applied Math. Program, University of Maryland at College Park, Md.

[42] Harche, F. & G. Rinaldi (1991) Vehicle Routing. Private communication.

[43] Hertz, A., G. Laporte & P. Nanchen (1998) Improvement Procedures for the Undirected Rural Postman Problem. *Centre de Recherche sur les Transports, University of Montreal.* Technical Report CRT-96-30 (revised March 1998).

[44] Jünger, M., G. Reinelt & G. Rinaldi (1995) The traveling salesman problem. In M.O. Ball, T.L. Magnanti, C.L. Monma & G.L. Nemhauser (Eds.) *Network Models.* Handbooks on Operations Research and Management Science Vol. 7. Amsterdam: Elsevier.

[45] Jünger, M., G. Reinelt & S. Thienel (1994) Provably good solutions for the traveling salesman problem. *ZOR - Math. Meth. Oper. Res., 40*,183-217 .

[46] Lenstra, J.K. & A.H.G. Rinnooy-Kan (1976) On general routing problems. *Networks, 6*, 273-280.

[47] Letchford, A.N. (1997a) New inequalities for the General Routing Problem. *European Journal of Operational Research, 96*, 317-322.

[48] Letchford, A.N. (1997b) The General Routing Polyhedron : a unifying framework. Forthcoming in *European Journal of Operational Research.*

[49] Letchford, A.N. & R.W. Eglese (1998) The rural postman problem with deadline classes. *European Journal of Operational Research, 105*, 390-400.

[50] Liebling, T.M. (1970) Graphentheorie in Planungs - und Tourenproblemen. *Lecture Notes in Operations Research and Mathematical Systems 21*, Springer, Berlin.

[51] Martello, S. & P. Toth (1990) Knapsack Problems: Algorithms and Computer Implementations. John Wiley.

[52] Nobert, Y. & J.C. Picard (1996) An optimal algorithm for the mixed chinese postman problem. *Networks*, *27*, 95-108.

[53] Orloff, C.S. (1974) A fundamental problem in vehicle routing. *Networks*, *4*, 35-64.

[54] Padberg, M.W. & M.R. Rao (1982) Odd minimum cut-sets and *b*-matchings. *Mathematics for Operations Research*, *7*, 67-80.

[55] Padberg, M.W. & G. Rinaldi (1987) Optimization of a 532 City Symmetric Traveling Salesman Problem by Branch and Cut. *Operations Research Letters*, *6*, 1-7.

[56] Padberg, M.W. & G. Rinaldi (1990) Facet identification for the symmetric traveling salesman polytope. *Mathematical Programming*, *47*, 219-257.

[57] Padberg, M.W. & G. Rinaldi (1991) A branch-and-cut algorithm for the resolution of large-scale symmetric traveling salesman problems. *SIAM Review*, *33*, 60-100.

[58] Papadimitriou, C.H. (1976) On the Complexity of Edge Traversing. *J. ACM* 23, 544-554.

[59] Pearn, W.L. (1989) Approximate solutions for the Capacitated Arc Routing Problem. *Computers and Operations Research* 16, 589-600.

[60] Picard, J.C. & M. Queyranne (1980) On the structure of all minimum cuts in a network and applications. *Mathematical Programming, 13*, 6-16.

[61] Picard, J.C. & H.D. Ratliff (1975) Minimum cuts and related problems. *Networks 5*, 357-370.

[62] Romero, A. (1997) On Mixed Rural Postman Problem. *PhD Dissertation* (in spanish), Dept. of Statistics and OR, University of Valencia, Spain.

[63] Savall, J.V. (1990) Polyhedral results and approximate algorithms for the directed rural postman problem. *PhD Dissertation* (in Spanish), Dept. of Statistics and OR, University of Valencia, Spain.

[64] Welz, S.A. (1994) Optimal solutions for the capacitated arc routing problem using integer programming. *PhD Dissertation*, Dept. of QT and OM, University of Cincinnati.

[65] Win, Z. (1987) Contributions to routing problems. *Doctoral Dissertation*, Universität Augsburg, Germany.

Chapter 8

TRANSFORMATIONS AND EXACT NODE ROUTING SOLUTIONS BY COLUMN GENERATION

Moshe Dror
University of Arizona

André Langevin
GERAD and École Polytechnique

1. Introduction 278
2. Transformations to Node Routing: Why? 279
   2.1 The Capacitated Rural Postman Problem 279
   2.2 Mathematical Formulation of the CARP 280
   2.3 When to Transfer to Node Routing 282
3. Arc Routing Transformations: How 283
   3.1 Transformations of Uncapacitated Arc Routing Problems 284
   3.2 Transformations of Capacitated Arc Routing Problems 285
   3.3 Transformation of CARP with Time Windows to VRPTW 287
   3.4 Split Delivery Arc Routing with Time Windows 289
4. Column Generation for Routing Problems with Non-split Delivery 290
   4.1 Revised Simplex: Chvátal's Introduction to Column Generation 290
   4.2 Set Covering, Vehicle Routing, and Column Generation 292
   4.3 The Shortest Path Subproblem 294
   4.4 An Algorithm for the Shortest Path with Resource Constraints 295
   4.5 Solving SPRCP with only Elementary Paths 297
5. Column Generation for Routing Problems with Split Delivery 305

5.1 Properties or Split Deliveries with Triangle Inequality 306
5.2 Formulation 308
5.3 Set Covering Approach for Split Deliveries 309
5.4 The Subproblem for Generating Feasible Columns for Split Delivery 311
5.5 The Computational Phase: A Mixed Integer Solver and a Dynamic Programming Algorithm 314
5.6 Another Set Covering Formulation for SDVRPTW 317
6. Conclusion 322

1. INTRODUCTION

This chapter examines arc routing problems from the "dual" perspective of node routing. It first attempts to explain and respond to a question from a "typical" engineering and applied mathematics graduate who was exposed to classical node routing problems such as the transportation and the traveling salesman problems, and ventured a little into the vast literature on problems and solutions for the different variants of vehicle routing (node routing) problems. Why shouldn't (or why should) any arc routing problem be viewed as some version of a node routing problem, pending an appropriate transformation of the corresponding graph? The first part of this chapter will examine this question addressing the issue of when such a transformation is necessary and the complementary question of when, or for what arc routing problems, from computational point of view, a transformation to node routing is inappropriate. In addition, the first part will attempt to provide a partial account of the different transformation schemes proposed over the years for arc routing problems into node routing setting. For an excellent write-up of exact solution methodologies for "hard" arc routing problems addressed without transformation to a node routing setting the reader is directed to chapters in this book by Eglese and Letchford, Benavent, Corberán, and Sanchis, and Johnson.

However, the second and the main part of the chapter focuses on providing a state-of-the-art survey of column generation methodology and its computational promise for solving node routing problems. The emphasis is on exact solutions to vehicle routing problems with time windows with and without split deliveries. The motivation stems from the fact that arc routing problems with time windows are very hard to model directly without an extensive graph modification (Mullaseril, 1996, Mullaseril and Dror, 1997, Dror, Leung, and Mullaseril, this book) and require transformation to node routing before an attempt of finding exact solutions can be made. This motivational aspect will be examined in more detail in the first part of this chapter.

2. TRANSFORMATIONS TO NODE ROUTING: WHY?

We begin this section with the description of what is called the *Rural Postman Problem* (RPP) since it encapsulates a number of well known classical node and arc routing problems. We provide a mathematical formulation of the capacitated RPP which is amenable to straightforward modification for a split delivery option. Following the capacitated RPP model we examine the problem of introducing time windows for arc delivery into such a model (for motivation see Dror and Leung, 1998, and Dror, Leung, and Mullaseril, this book). The time window modeling problem impels the necessity of the transformation from arc routing graph description of the problem into the node routing problem setting which leads into the section describing problem transformations.

2.1. THE CAPACITATED RURAL POSTMAN PROBLEM

Given a connected graph $G = (N, E \cup A)$, with N as the set of nodes (vertices), E set of edges ($E \subseteq N \times N$) and A a set of arcs ($A \subseteq N \times N$), the Rural Postman Problem (RPP) is the problem of finding a minimum cost traversal of a given subset of edges and arcs in $R \subseteq E \cup A$. The set R is usually referred to as the *required edges and arcs*. If the set of edges $E = \emptyset$, the corresponding problem is sometimes referred to as the directed RPP (or undirected RPP in case $A = \emptyset$).

We note from the outset that the (RPP) is strongly $\mathcal{NP}$-hard even for completely directed and completely undirected graphs (see Lenstra and Rinnooy Kan, 1976, Garey and Johnson, 1979, and Papadimitriou, 1976). For the additional graph theory terminology and notation the reader is directed to Fleischner (this book).

The Capacitated Rural Postman Problem, usually denoted as the Capacitated Arc Routing problem or in short as CARP, has in addition to traversal cost for each edge and arc, a positive demand value associated with each edge and arc in the subset R. Subsequently, the CARP is the problem of finding a minimum cost cover of a given subset R by a set of traversal circuits with one common node (the depot) such that the total demand of edges from R serviced (delivered) in each circuit does not exceed some value $Q > 0$ (vehicle capacity). Note that in the case that vehicle capacity Q is less than the total demand on the edges of R, the CARP solution will require some integer number (> 1) of traversal circuits to service the edges in R, versus a single circuit (vehicle) solution outcome for the RPP. Disregarding at this point the issue of the timing

of traversals for each edge in R, the formulation of the CARP for the graph G taken from Dror and Leung (1998) is presented below.

2.2. MATHEMATICAL FORMULATION OF THE CARP

Let

q_{ij} = the demand along edge (*arc*) $(i,j) \in R$,

Q_v = the capacity of traversal circuit (*trip*) v,

c_{ij} = the cost of an edge (*arc*) $(i,j) \in E\,(A)$. (Note that initially $c_{ij} \geq 0, \forall\,(i,j) \in E \cup A$),

x_{ij}^v = the number of times edge (*arc*) $(i,j) \in E \cup A$ is traversed in trip v,

V = the upper bound on the number of traversal circuits,

$y_{ij}^v = \begin{cases} 1 & \text{if the edge or arc } (i,j) \in R \text{ is covered in trip } v, \\ 0 & \text{otherwise.} \end{cases}$

Node 0 is designated as the depot.

In capacitated edge (arc) routing problems we have basically two costs (or more in case of a heterogeneous vehicle fleet) for traversing each edge in R. The unavoidable cost is that of "deadheading" (nondelivery) traversal. It has to be accounted for in any solution to the CARP and it is some constant for traversing all the edges and arcs in R, and does not have any role in solving the CARP if split edge or arc deliveries are not allowed. The c_{ij}, as denoted above, account only for the incremental cost (the additional delivery service cost) of traversing the edge (arc) (i,j).

$$(CARP): \qquad \min \sum_{(i,j)\in E} \sum_{v=1}^{V} c_{ij} x_{ij}^v$$

subject to

$$\sum_{k\in N} x_{ki}^v - \sum_{k\in N} x_{ik}^v = 0, \forall i \in N, v = 1,2,\ldots,V,$$

$$\sum_{v=1}^{V} y_{ij}^v = 1, \forall (i,j) \in R,$$

$$\sum_{(i,j)\in R} q_{ij} y_{ij}^v \leq Q_v, v = 1,\cdots V,$$

$$x_{ij}^v \geq y_{ij}^v, \forall (i,j) \in R, v = 1,2,\ldots,V,$$

$$M \sum_{i\notin N[S], j\in N[S]} x_{ij}^v \geq \sum_{(j,k)\in S} x_{jk}^v, \begin{cases} \forall S \subseteq R, \\ 0 \notin N[S], \\ v = 1,\ldots,V, \end{cases}$$

$$y_{ij}^v \in \{0,1\}, \forall (i,j) \in R, v = 1, \ldots, V,$$
$$x_{ij}^v \in Z^+, \forall (i,j) \in E, v = 1, \ldots, V,$$

where M is a large constant greater than or equal to sum of traversals of arcs and edges in any given $S \subseteq R$, and $N[S]$ is the set of nodes incident to the arc set S. In this formulation, the index v denotes a trip, and V is the maximum number of trips allowed. We come back to this formulation later when we discuss the column generation approach for routing problems.

The objective function represents the total distance covered by all traversal circuits. Note that edges are allowed to be traversed an integer number of times (that is, more than once). The first set of constraints are the common 'flow conservation' constraints for network-flow formulations. The second set of constraints require that at least one traversal is made for each of the edges in R. The third set of constraints are the capacity constraints. The next set of constraints require that the traversal circuit v covers the edge $(i,j) \in R$ if it delivers its demand. The fifth set of constraints are subtour-elimination constraints ensuring each trip is connected to the depot. Note that this formulation of the CARP is different and more direct than the one given in Golden and Wong (1981) with respect to subtour elimination constraints and the integrality requirement on x_{ij}^v variables.

In the next subsection, we introduce the issue of time windows and their relation to the CARP solution.

2.2.1 Time Window Constraints for Arc Routing.

In this section, we examine time-window constraints for the CARP and point out the difficulty of formulating the mathematical linear integer programming model for CARP with time windows, which turns out to be a much more difficult task than modeling its node-routing (VRP) counterpart.

Associate with each edge (arc) $e \in R$ a time window $[a_e, b_e]$ within which delivery must be completed, and a positive duration t_e for the traversal of edge or arc e at the delivery speed. In addition, with each arc or edge in R, we associate a delivery starting time and completion time, indicating the times when the vehicle starts the delivery traversal of the edge (arc) and completes the delivery (service) traversal of the edge (arc) respectively.

In the vehicle routing literature for node-routing problems with time windows, the original "physical" network can be augmented by adding an arc (i, j), with its edge-length equal to the 'shortest-path' distance, whenever there is a path from i to j in the original network. With this augmented network, we can then make the assumption that a node is visited no more than once, and the time-window restrictions can be modeled by the constraints:

$$x_{ij}^v(t_i^v + t_{ij} - t_j^v) \leq 0, \forall (i,j) \in A \cup E, \ \forall v,$$
$$a_i \leq t_i^v \leq b_i, \forall i \in N, \ \forall v,$$

where t_i^v is a decision variable representing the time when vehicle v arrives to deliver to node i, $[a_i, b_i]$ is the allowed time window for node i and t_{ij} is the traversal time for edge (arc) (i, j) (see Desrosiers, et al., 1995). Note that in this formulation, waiting is allowed at customer nodes.

It is important to note that in an arc routing problems such as CARP, each arc or edge $e \in R$ is serviced exactly once but can be traversed an additional number of times in a deadheading mode if so required by the minimum-distance objective. Hence, we cannot associate a unique starting and completion time for an edge (arc) $e \in E \cup A$. Moreover, it is not possible to augment the network in a manner analogous to node-routing problems to get an equivalent formulation where edges and arcs (or nodes) are visited only once. Thus, the addition of time window requirements for the edges (arcs) in R precludes a direct edge/arc routing integer linear programming model formulation for the CRPP or even for the RPP problem. This compels the modeler to seek alternative modeling approaches and to express arc routing settings with time windows by using graph transformations which necessarily cast the arc routing problems in terms of their node routing counterparts.

2.3. WHEN TO TRANSFORM TO NODE ROUTING

The theory of complexity of combinatorial problems such as CARP classifies problems as 'hard' (not known to be solvable in polynomial time complexity) or 'easy' for which known polynomial time procedures exist with guarantee of reaching an optimal solution. The majority of problems examined in this book belongs to so called $\mathcal{NP}$-class (see chapter in this book by Dror, and the book by Garey and Johnson, 1979). One of the main results laid in the foundations for the theory of $\mathcal{NP}$-completeness paper by Cook (1971), was in proving that every problem in $\mathcal{NP}$-class of decision problems can be polynomially reduced to the "satisfiability" problem. Furthermore, this property of reducibility for

the satisfiability problem is shared by entire equivalence class of problems called $\mathcal{NP}$*-complete problems.* One of the $\mathcal{NP}$-complete problems is stated as a decision version of the traveling salesman problem (TSP). Thus, in principle, one can transform in polynomial time any arc routing problem (or for that matter any problem in $\mathcal{NP}$) to a TSP problem version (instance). However, this makes computational sense only for a selective small subset of problems. The next section in this chapter describes a number of direct transformations for arc routing problems into instances of node routing problems. It is perhaps quite transparent that some arc routing problems ought not to be transformed to node routing setting since, as Johnson and Papadimitriou (1985) point out, one might be transforming "easy" problems to a very "hard" problem – the TSP.

For instance, given the classical Chinese Postman Problem (CPP) on an undirected graph, it is well known that an optimal solution to the problem can be constructed in time complexity dominated by the complexity of the corresponding minimum weight matching problem for the odd degree nodes of this graph. Since an efficient ($\mathcal{O}(|N|^3)$) algorithm exists for the corresponding matching problem (see Derigs, this book), there is clearly no computational advantage in transforming the undirected CPP to a node routing problem setting. Identical argument can be used for the directed CPP, since this problem can also be solved in its arc routing version by polynomial time algorithms dominated by the complexity of the corresponding transportation problem. It is perhaps not surprising that arc routing problems are considered "easier" than most node routing problems. This intuitive view might have its roots in the fact that "early" arc routing problems such as the Chinese Postman Problem (CPP) are solvable in polynomial time asking to prescribe an arc (or edge) 'covering' solution, whereas node routing is more of a 'arc selection and partition' process.

However, for arc routing problems which are $\mathcal{NP}$-hard, it might be of computational interest to examine solution schemes in their transformed node routing image. As indicated above, for the arc routing problems with time windows, this is absolutely essential. Moreover for some arc routing problems, like the Mixed RPP and the Stacker Crane Problem, the **only** known exact method, uses a transformation into node routing problems.

3. ARC ROUTING TRANSFORMATIONS: HOW

This section examines a number of well known and more recent graph transformations which accept a routing problem statement expressed in terms of arc/edge traversals and converts it to a node routing problem

on a related graph. The node routing solution on the modified "dual" graph is equivalent to the arc/edge traversal solution on the original graph. Clearly, one can also transform a node routing problem statement on a given graph into a arc/edge traversal problem statement on an appropriately modified graph where the arc/edge traversal solution would correspond to a node routing solution on the original graph. However, this transformation of a routing problem is almost never executed in practice for obvious computational reasons since it results in many disconnected arc routing subgraphs connected by arcs and edges which need not be traversed (i.e., the result is usually a many component RPP).

3.1. TRANSFORMATIONS OF UNCAPACITATED ARC ROUTING PROBLEMS

Laporte (1997) describes several classes of arc routing problems for which he provides a transformation into node routing problems and proceeds to solve these problems as TSPs. The transformation described by Laporte is a two phase procedure applied uniformly to the Mixed Chinese Postman Problem (MCPP), the Windy Chinese Postman Problem (WCPP), the Stacker Crane Problem (SCP), and the variations of Undirected, Directed, Mixed, and Windy Rural Postman Problems. In phase 1 of the transformation procedure, the problems are transformed to what is called the Generalized TSP (GTSP), which is defined below. In the second phase, the corresponding GTSPs are transformed into TSP problems, completing the transformation to a "classical" TSP instance. Below we outline in some detail Laporte's graph transformations.

Given a graph $G = (N, A \cup E)$, where N is a finite set of nodes, A ($A \subset N \times N$) is a set of arcs without self cycles, and E is a set of edges without self loops and a "length" function defined on $A \cup E$. The transformation can be described as follows: replace each edge $e_{ij} \in E$ by two arcs a_{ij} and a_{ji}. The length of the corresponding a's is the same as that of e. The new graph $G' = (N, A')$ is a directed graph (directed "version" of G). In the next step, a complete (directed) graph $H = (V, B)$ is constructed with two directed arcs between every pair of nodes. The set of nodes V consists of one node for each arc in the directed graph G' obtained earlier. Thus, an edge $i \in E$ is represented by two node set N_i in V and an arc $j \in A$ by only one node set N_j (a singleton) in V. Denote by K the total number of such node sets in V. For each arc $a \in B$, determine its length $c(a)$ using the shortest path distance obtained on the graph G'. For completeness, note that in the case when length of the shortest path between two nodes i and j in G' is ∞, then there is no arc (i, j) in B.

Given graph $H = (V = \cup_{i=1}^{K} N_i, B)$, the so called *Generalized Traveling Salesman Problem* (GTSP) consists of constructing a minimal length circuit on H which contains exactly one node from each subset $N_i, i = 1, \ldots, K$. Laporte (1997) continues this transformation process by implementing the rules described in Noon and Bean (1993) which transform a GTSP problem into a classical (directed) TSP instance.

The significance of Laporte's work lies in the computational part of his paper. Some problems, like the Mixed RPP, and the SCP, in their arc routing version have not been solved to proven optimality. Thus, the transformations to node routing and their subsequent solutions are encouraging since they lead to optimal solutions for the corresponding original arc routing problems. For the Mixed CPP, the more recent results on the arc routing version by Nobert and Picard (1996) outperform the results obtained by Laporte (1997) on the transformed node routing problems. For the Directed RPPs the results are inconclusive since the reported exact solutions of Christofides et al. (1986) are more than fifteen years old.

At this point it is perhaps appropriate to remark that the undirected arc routing problems require an additional phase in their transformation to node routing. Specifically, each edge creates a pair of nodes in the transformed graph only one of which has to be visited. Thus, in that case there is the need of transforming a GTSP instance before solving it into a TSP instance. In the directed arc routing case the transformed problem is already a TSP.

We also note that recently there has been a study of the Generalized Vehicle Routing Problem (GVRP), which is the problem of designing optimal capacitated delivery routes starting from a depot to a number of mutually exclusive and exhaustive clusters of customers, visiting exactly one customer in each cluster. This node routing problem was transformed into a capacitated arc routing problem (Ghiani and Improta, 2000) which then was solved using a heuristic methodology designed to solve arc routing problems (see Hertz and Mittaz, this book).

3.2. TRANSFORMATIONS OF CAPACITATED ARC ROUTING PROBLEMS

A node routing problem can be transformed into an arc routing problem simply by replacing each node $i \in N$ that requires delivery (service) of q_i with an edge. The two nodes of such edge are adjacent to all the other nodes the original node i has been adjacent to. On the other hand,

the opposite direction transformation is a little more involved. For the capacitated arc routing problem (CARP), Pearn et al. (1987) (see also Assad and Golden, 1995) describe the following transformation into an equivalent vehicle routing problem (node routing). As in subsection 2.2, consider a directed graph $G = (N, A)$, with node set N and an arc set A together with a "distance" or "cost" value for each arc $(i, j) \in A$ denoted by c_{ij} and a positive value Q (Q = vehicle capacity). Also with each such arc $(i, j) \in A$ a "demand" value $q_{ij} > 0$ (delivery or collection as the case may be) is specified. Designate node $0, 0 \in N$ as the depot. The CARP problem requires finding a set of circuits on G with node 0 in common, covering all the arcs in A at minimum cost and such that the total demand over the arcs in each circuit does not exceed the value Q.

Pearn et al. (1987) transformation proceeds as follows: First, replace each arc $(i, j) \in A$ with three nodes: $s_{(ij)}, m_{(ij)}$, and $s_{(ji)}$. The nodes are referred to as near-side, middle, and far-side nodes respectively. Let $N' = \{0\} \cup \{s_{(ij)}, m_{(ij)}, s_{(ji)} | (i, j) \in A\}$ be a set of nodes of the new graph G' for the corresponding instance of the vehicle routing problem. Denote by $\bar{d}(i, j)$ the least cost path between nodes i and j in the original graph G. In order to determine the cost of the arcs (directly going between two nodes) in N' define the following distance function $d : N' \times N'$:

$$
\begin{aligned}
d(s_{(ij)}, s_{(kl)}) &= \begin{cases} (1/4)(c_{ij} + c_{kl}) + \bar{d}(i, k) & \forall (i, j) \neq (k, l) \\ d(s_{(ij)}, s_{(kl)}) = 0 & \text{if } (i, j) = (k, l) \end{cases} \\
d(p, s_{(ij)}) &= (1/4)c_{ij} + \bar{d}(p, i), \ \forall p \in N' \\
d(s_{(ij)}, m_{(ij)}) &= d(m_{(ij)}, s_{(ji)}) = (1/4)c_{ij}
\end{aligned}
$$

and the distance from and to any $m_{(ij)}$ node to nodes other than its far-side and near-side nodes is set to ∞.

The demands at the N' nodes are set as follows: $q(s_{(ij)}) = q(m_{(ij)}) = q(s_{(ji)}) = (1/3)q_{ij}$. The depot (node 0) has no demand associated with it.

Since in the classical vehicle routing problem each customer (a node in G') can only be visited by a single vehicle (no split deliveries), this transformation guarantees that the nodes $s_{(ij)}, m_{(ij)}, s_{(ji)}$ corresponding to the same arc in the original graph G appear consecutively on the same vehicle route for the VRP solution. Since converting a single arc into three nodes adds only two arc segments, one needs to add (1/4)th of the original arc distance when entering near-side node $(s_{(ij)})$, (1/4)th of the

original arc distance when leaving the far-side node ($s_{(ji)}$), and half of the arc distance when going from $s_{(ij)}$ to $s_{(ji)}$ through $m_{(ij)}$.

This transformation by Pearn et al.(1987) converts the CARP with $|A|$ arcs into a VRP with $3|A|+1$ nodes, thus more than tripling the problem size relative to the arc routing problem version. Since the VRP exact solution methodologies based on implicit enumeration, such as branch and bound techniques, are very sensitive to problem size this transformation's tripling size effect might have implications on the solvability of a problem. As stated by Eiselt et al. (1995b), "the interest of these transformations is mostly formal and their algorithmic value has yet to be demonstrated".

3.3. TRANSFORMATION OF CARP WITH TIME WINDOWS TO VRPTW

Given any instance of a capacitated arc routing problem on a graph $G = (N, A \cup E)$, with time windows only for a subset of required arcs R in A (i.e., a special case of CARP with time windows), one can formulate an equivalent directed vehicle routing problem with time windows (VRPTW) by performing a graph transformation originated by Mullaseril (1996), reproduced in Mullaseril and Dror (1997), and similar to the transformation described independently in Laporte (1997). This transformation is described below.

Construct a complete graph $G' = (N', A')$ whose node-set N' consists of nodes which correspond to the subset of required arcs $a \in R \subseteq A$ ($|N'| = |R|$). Each arc $a \in A$ has an incident node in N at the tail of the arc and an incident node in N at head of the arc. Let the length of an arc $(i, j) \in A'$ be the shortest path (without delivery) distance between the head node of arc $e^i \in A$ and the tail node of arc $e^j \in A$ denoted by c_{ij}, where both arc e^i and e^j are required arcs (i.e., are members of $R \subseteq A$) each with its corresponding node in N'. The demand and time window associated with any node $i \in N'$ will be the same as its "originating" arc $e \in R$. Since each arc $e \in R$ has a length $c(e)$, each node $i \in N'$ has a "length" (a cost) corresponding to the length of the arc. This cost will not be used in the subsequent mathematical formulation for the transformed problem, however this cost is used to calculate t_{ij}, the time required by any trip to travel from node i to any node j in N' in the following manner:

$$t_{ij} = c_i/S_L + c_{ij}$$

Let S_L be the "service rate" (the time required to deliver service to one unit of arc length), and where c_i is the cost (length) associated with node $i \in N'$, and c_{ij} is the length of the shortest path from i to j as

described earlier. Thus the new graph G' has $|R|$ nodes and $|R(|R|-1)|$ arcs. Note that we assume a homogeneous delivery system of identical vehicle speeds.

A valid formulation for the VRPTW on the graph $G' = (N', A')$, based on the formulations from Dror and Trudeau (1990), and Desrosiers et al. (1995), is presented below. The additional parameters and variables for this formulation are:

t_{ij} = the non-negative duration of a trip from node i to node j.
q_i = the daily demand at node i.
$x_{ij}^v = 1$ if the vehicle v travels along arc (i,j), and
$x_{ij}^v = 0$ otherwise.
$y_i^v = 1$ indicates that vehicle v delivers the demand at node i, and
$y_i^v = 0$ indicates that it does not.
t_i^v = the starting time of delivery to node i by trip v.
S = the set of simple cycles on the set N which include the depot node.
V = denotes the number of vehicles.
Q_v = denotes the capacity of vehicle v.

If the objective is to minimize the total distance traveled by all trips, the VRPTW can be represented by the following mathematical formulation:

$$\text{Minimize} \quad \sum_{(i,j)\in A'} \sum_{v=1}^{V} c_{ij} x_{ij}^v$$

$$\text{subject to:} \quad \sum_{(k,i)\in A'} x_{ki}^v - \sum_{(i,k)\in A'} x_{ik}^v = 0, \forall\, i \in N',\ v = 1, \ldots, V,$$

$$\sum_{v=1}^{V} y_i^v = 1, \forall\, i \in N'$$

$$\sum_{i\in N'} q_i y_i^v \le Q_v,\ v = 1, \ldots, V,$$

$$y_i^v - \sum_{(i,j)\in A'} x_{ij}^v \le 0,\ i \in N',\ v = 1, \ldots, V,$$

$$(t_i^v + t_{ij} - t_j^v) x_{ij}^v \le 0, \forall\, (i,j) \in A',\ v = 1, \ldots, V,$$

$$a_i \le t_i^v \le b_i, \forall\, i \in N',\ v = 1, \ldots, V.$$

$$x_{ij}^v \in \{0,1\}, \forall\, (i,j) \in A',\ v = 1, \ldots, V,$$

$$y_i^v \in \{0,1\}, \forall\, (i,j) \in A',\ v = 1, \ldots, V,$$

This formulation is exactly the node routing transformed representation of the formulation for the CARP discussed in section 2.2, with includes the necessary addition of the subsection on time windows 2.2.1, and an optimal solution to this problem formulation generates an optimal solution which needs to be recast in the arc routing (CARPTW) terms.

Observe that in the transformed graph G', we have $|R|$ nodes as compared to $3|V| + 1$ nodes in the transformation suggested by Pearn et al. (1987). Also note that this transformation is bi-directional unlike the transformation in Pearn et al.(1987). The state-of-the-art VRPTW algorithms can solve about 100 node problems (Desrochers, et al. 1992). Hence by just executing this transformation, one may now solve optimally CARPTW problems of 100 required arcs using the techniques developed by Desrosiers et al. (1995).

3.4. SPLIT DELIVERY ARC ROUTING WITH TIME WINDOWS

A feed distribution problem at a large cattle yard (over 100,000 head of cattle) in Yuma, Arizona, (see Dror and Leung, 1998, and Dror, Leung, and Mullaseril, this book) has motivated an increased interest in arc routing, both in its real-life realizations such as in the case of large cattle yards, and for the related graph theoretical questions. A main characteristic of such arc routing setting in a cattle yard is that the corresponding combinatorial optimization problem is best represented by a capacitated rural postman problem with time windows and split deliveries model. The previous section describes a transformation of a CARPTW problem instance to a VRPTW problem instance without split delivery, and the model "rewrite" required to incorporate the split delivery feature into the corresponding VRPTW model is minor. The formulation has been "borrowed" from the split delivery VRP formulation described in Dror and Trudeau (1990), and the required adjustment is simply the relaxation of the binary requirement on the y_i^v variables (denoting the fraction of demand at node i delivered by vehicle v which is set to 0 or 1 possible values) and its replacement with

$$0 \leq y_i^v \leq 1, \forall (i,j) \in A', v = 1, \ldots, V,$$

Even though the mathematical model representation of the split delivery VRPTW is very similar to the non-split 0-1 integer programming model, and the fact that both problems are hard ($\mathcal{NP}$-hard in the strong sense), the sense of applied combinatorial optimization professionals community working on exact solutions for such routing problems is that the split delivery VRPTW (or even just the split delivery VRP) is "harder" than its non-split counterpart. This is specially apparent

in the case of continuous split (versus discrete split) and was computationally demonstrated by Dror et al. (1994). As there is an important real-life motivation for solving split delivery capacitated arc routing with time windows problems, in the second part of this chapter we present a detailed examination of a column generation approach for this problem since presently this seems to be the most promising solution methodology for very hard and important routing problems. We begin with a column generation survey for the non-split node delivery problems first.

As mentioned earlier, the focus of this chapter is on exact solution procedures for node routing problems based on column generation solution methodology because it has been demonstrated that transformations from arc routing to node routing settings are efficient (maintain the problem size) and in some cases absolutely necessary (e.g., problems with time windows). Thus, all the remaining sections of this chapter are about solving node routing problems.

4. COLUMN GENERATION FOR ROUTING PROBLEMS WITH NON-SPLIT DELIVERY

We start this section with a short introduction to the familiar revised simplex approach for solving linear programming problems. We follow with an overview of set covering in the context of vehicle routing, and column generation. Then the shortest path problem with resource constraints, which is central in the scheme of solution approach based on column generation, is examined in some detail based on the work of Desrochers (1988) and Desrochers and Soumis (1988). This shortest path problem with resource constraints subsection is also very important for the next section where we examine the column generation solution approach for solving the Split Delivery Vehicle Routing Problem with Time Windows (SDVRPTW) based on the work of Mullaseril and Dror (1997). This is followed with a summary of a more recent work on the shortest path problem by Guéguen, et al. (1998), where a modified algorithm is proposed and promising computational results for the VRPTW are reported.

4.1. REVISED SIMPLEX: CHVATAL'S INTRODUCTION TO COLUMN GENERATION

In order to examine column generation solution methodology for vehicle routing problems it is perhaps illuminating to start with a review of the revised simplex algorithm for solving linear programs based on a

very elegant exposition of this topic by V. Chvátal in his prize winning 1983 book "Linear programming".
Given a linear program:

$$\begin{aligned} \text{Maximize} \quad \mathbf{z} &= \mathbf{cx} \\ \mathbf{Ax} &= \mathbf{b} \\ \mathbf{x} &\geq \mathbf{0} \end{aligned}$$

where $\mathbf{A}$ is an $m \times n$ matrix of reals, $\mathbf{c}$ is the cost vector $1 \times n$, $\mathbf{b}$ is the r.h.s. vector $m \times 1$, and $\mathbf{x}$ is the (unknown or decision variables) solution vector $n \times 1$.

Denoting the nonsingular square basic matrix $m \times m$ by $\mathbf{A_B}$, its inverse by $\mathbf{A_B^{-1}}$, and the $m \times (n - m)$ nonbasic matrix in the linear program above by $\mathbf{A_N}$, a solution vector $\mathbf{x_B}$ is obtained

$$\mathbf{x_B} = \mathbf{A_B^{-1}b} - \mathbf{A_B^{-1}A_N x_N}.$$

where $\mathbf{x_N}$ is the vector of non-basic variables.

Substituting for $\mathbf{x_B}$ in the objective function of the linear program

$$\mathbf{z} = \mathbf{c_B}(\mathbf{A_B^{-1}b} - \mathbf{A_B^{-1}A_N x_N}) + \mathbf{c_N x_N} = \mathbf{c_B A_B^{-1}b} + (\mathbf{c_N} - \mathbf{c_B A_B^{-1}A_N})\mathbf{x_N}.$$

and $\mathbf{A_B^{-1}b}$ is the vector $\mathbf{x_B^*}$ specifying the current values of the basic variables.

As Chvátal explains, in the revised simplex method, the vector $\mathbf{c_N} - \mathbf{c_B A_B^{-1}A_N}$ is computed in two steps: first the $m \times 1$ vector $\mathbf{y} = \mathbf{c_B A_B^{-1}}$ is computed by solving the linear system $\mathbf{yA_B} = \mathbf{c_B}$ followed by the calculation of the vector $(\mathbf{c_N} - \mathbf{yA_N})$. The important observation for column generation is that the components of the vector $(\mathbf{c_N} - \mathbf{yA_N})$ may be calculated one by one by identifying specific columns of the matrix $\mathbf{A_N}$. For instance, if a nonbasic variable x_j corresponds to a nonbasic cost coefficient c_j and a column $\mathbf{a}_j$ of the nonbasic matrix $\mathbf{A_N}$, then its component in $\mathbf{c_N} - \mathbf{yA_N}$ corresponds to $c_j - \mathbf{ya}_j$. Thus, any nonbasic variable x_j for which $c_j - \mathbf{ya}_j > 0$ in maximization problem will improve the current objective function value z (corresponding to the current basic solution $\mathbf{x_B^*}$) by entering the basis. Clearly, in a minimization problem we would be searching for a nonbasic variable x_j such that $c_j - \mathbf{ya}_j < 0$.

The determination of the basic variable leaving the basis is obtained by solving, for the vector $\mathbf{d}$ standing for the column of $\mathbf{A_B^{-1}a}_j$ of the entering variable, another system $\mathbf{d} = \mathbf{A_B^{-1}a}_j$ and finding the largest t such that $\mathbf{x_B^*} - t\mathbf{d} \geq 0$. If no such t exists, then the problem is unbounded. If

the problem is bounded, then at least one component of $\mathbf{x}_\mathbf{B}^* - t\mathbf{d}$ is zero and its corresponding variable is leaving the basis.

Optimality is determined when there is no longer a column $\mathbf{a}_j$ of $\mathbf{A_N}$ such that $c_j - \mathbf{y}\mathbf{a}_j > 0$. In column generation terminology, the subproblem of generating an "interesting" column (entering column) is defined by the problem of maximizing the value $(c_j - \mathbf{y}\mathbf{a}_j)$, subject to column $\mathbf{a}_j$ being a member (nonbasic) of the matrix $\mathbf{A_N}$, and the cost of the column equal to c_j. If, for the minimization problem a solution (a column in $\mathbf{A_N}$) is found such that $c_j - \mathbf{y}\mathbf{a}_j < 0$, then we update our basis and repeat.

The next subsection starts with a short description of set covering problem which is a central concept for thinking about solving combinatorial optimization problems such as capacitated routing problems by generating and substituting partial solutions (corresponding to non-basic columns in $\mathbf{A_N}$) which cover (deliver to) all the nodes (customers) of a given problem instance.

4.2. SET COVERING, VEHICLE ROUTING, AND COLUMN GENERATION

In general, Set Covering Problem (SCP) can be expressed as follows: Given a (nonempty) finite set of elements S and a collection of subsets of S, say $C = \{C_i | C_i \subseteq S \ \forall i \in I\}$ and a real "cost" $c_i, i \in I$, find a subset $I' \subseteq I$ such the $\sum_{i \in I'} c_i$ is minimized and $S \subseteq \cup_{i \in I'} C_i$. On the other hand, the Set Partition Problem (SPP) for S seeks to minimize $\sum_{i \in I''} c_i$ for a subset $I'' \subseteq I$ such that $S = \cup_{i \in I''} C_i$, and such that $C_i \cap C_j = \emptyset, i \neq j, i, j \in I''$.

A set covering approach (sometimes referred to in the literature as set partitioning) for the vehicle routing problem with time windows has been proposed as far back as 1964 by Balinski and Quandt.

The approach by Balinski and Quandt is based on the premise of selecting "good" routes from a given (implicit or explicit) large set of feasible routes. Following the notation and terminology from Desrochers, et al. (1992) for the VRPTW, the set covering vehicle routing problem representation on a graph $G = (N, A)$ can be stated as follows:

$$\min \sum_{r \in \mathcal{R}} c_r x_r \tag{8.1}$$

$$\sum_{r \in R} \delta_{ir} x_r \geq 1, i \in N \setminus \{0\} \tag{8.2}$$

$$x_r \in \{0,1\}, r \in \mathcal{R} \tag{8.3}$$

The set $\mathcal{R}$ denotes all the feasible routes for the VRPTW. The constants (indicators) δ_{ir} characterize by a value 1 a route $r \in \mathcal{R}$ which visits customer $i \in N \setminus \{0\}$, and δ_{ir} takes the value 0 if route r does not visit customer i. The cost of the route r is denoted by c_r and is calculated as the sum of the costs of the arcs of this route. N is the set of customer nodes including the depot node (node 0).

The columns in the formulation above (vectors $\delta_{\mathbf{r}}$, $r \in \mathcal{R}$ of dimension $(|N| - 1) \times 1$) correspond to feasible routes and determine the decision variables for the problem. Clearly, for any reasonable size problem, the total number of columns representing all possible feasible routes can be extremely large (and exponential in the problem's size expressed by the number of customers). Since the above set covering problem is stated as a 0/1 linear programming problem with a very large number of columns, this problem presentation is addressed (solved) only in some trivially small cases. In any practical context this problem is first examined in its LP relaxed version by the revised simplex solution methodology based on column generation. The feasible columns are added usually one at a time (and more than one at a time in some implementations of the column generation scheme) by solving an optimization subproblem which both checks the optimality of the LP relaxation for the set covering problem and generates a new "better" feasible route if an improvement is possible. As indicated by Desrochers, et al. (1992), (see also Haouari et al. 1991) the solution to the LP relaxation of the above VRPTW problem formulation serves as a lower bound in a branch and bound scheme for the integer program. In general, this LP relaxation solution generates an excellent lower bound as demonstrated by the computational results of Desrochers et al. (1992).

Note that in the case where the cost/distance matrix defined on the graph G satisfies the triangle inequality, equation (12) implies the same whether it is written as equality (partition), or $\geq$ (covering). Since it is always possible to convert a general distance matrix to a matrix with triangle inequality by solving an all pairs shortest path problem, the assumption of triangle inequality allows for replacing equality in (12) with "$\geq$" and it does not have an effect on the partition solution as long as the demands at each node are strictly positive.

Unquestionably, a key to a successful column generation scheme for problems such as VRPTW lies in the ability to solve quickly the subproblem – the determination of the existence of an entering column (a column in $\mathcal{R}$ which improves on the current basic solution to the SCP).

This subproblem for the VRPTW together with its solution methodology is presented next.

4.3. THE SHORTEST PATH SUBPROBLEM

First, one has to establish what is the "right" subproblem which has to be solved in order to test the existence of an entering column which improves on the current basic solution. In principle, we are solving the minimization problem described earlier (minimize $(c_j - \mathbf{y}\mathbf{a}_j)$ subject to the constraint that the vector $\mathbf{a}_j$ represents a nonbasic column of the corresponding set covering problem). I.e., $\mathbf{a}_j$ corresponds to a circuit in graph G whose nodes' demands do not exceed the capacity of the vehicle. This leads us to the shortest path problem representation for the "right" subproblem (the "depot" node is represented by "origin" and "destination" pair of nodes converting a circuit problem into a path problem). When the visits to the nodes in G are constrained by time windows, the corresponding subproblem becomes an appropriate shortest path with time windows problem.

The shortest path problem with time windows (SPPTW) as a subproblem in a VRPTW column generation solution scheme has been described in a number of papers (see Desrosiers et al. 1995, for an extensive review). Perhaps the most successful examination of this problem has been first presented by Martin Desrochers in his 1988 working paper at GERAD, Montreal. This early work of M. Desrochers has been considered a fundamental contribution and the basis for computational implementation in real-life commercial software for optimization problems such as crew scheduling, flight scheduling, staffing problems, school busing, handicap pickup and delivery, and more. The review of the shortest path with resource constraints problem in this chapter is based on the original work of Desrochers (1988), and on the more recent modifications described by Guéguen et al. (1998).

The SPPTW is viewed by Desrochers (1988) in a general setting which includes time window restrictions on the node visits as a special case. The more general approach to this problem views time as a resource and a time window as availability restriction of this resource at a given node. The capacity of the vehicle is clearly another resource. The generalization enables the consideration of yet other resources in the same analytical framework of searching for a path which satisfies resource availability. This also explains why the problem is often referred to as the Shortest Path with Resources Constraints Problem (SPRCP). In a computational implementation such as described in Desrochers (1992), and in many papers originated with the Montreal research center GERAD, the SPRCP problem is solved by a pseudo-polynomial solution proce-

dure. (The research on this topic developed over the years in GERAD is referred to here as the DDDSS results after Desrochers, Desrosiers, Dumas, Solomon, and Soumis, to avoid excessive listing of references - see also the excellent survey by Desrosiers et al. 1995.)

4.4. AN ALGORITHM FOR THE SHORTEST PATH WITH RESOURCE CONSTRAINTS

Let $G = (N, A)$ be a directed graph with N as the set of nodes including a single source node p and a single sink node q (the nodes p and q are also referred to as origin and destination respectively). The arc set is denoted by A ($A \subseteq N \times N$), and each arc $(i, j) \in A$ has a cost c_{ij} which is a real number (positive or negative). A *path* in G always starts at node p and ends at node q and is defined as a sequence of nodes $p, i_1, \ldots, i_K, q$ such that for each consecutive nodes in the sequence (i_k, i_{k+1}) there is an arc in A. The cost of a path is the sum of the costs of the arcs in the path. The paths need not be simple (or elementary) which implies that some nodes in the path can occur more than once.

Assume that there are L resource types (finite positive integer) and each node requires some nonnegative amount of each resource. In addition, each of the nodes $i \in N$ has for each of the resources $l \in L$ a "feasibility" window $[a_i^l, b_i^l]$ (such as time window, quantity window, etc.). I.e., a resource requirement t_i^l at node i satisfies $a_i^l \leq t_i^l \leq b_i^l$ for feasibility. In addition, with each arc $(i, j) \in A$ along the path there is a nonnegative resource consumption $t_{ij}^l, l \in L$ (such as time, fuel, etc.). The resources in L are of two types: (1) "time-like" resources - L', and (2) "delivery quantity" resources - $L \setminus L'$. The feasibility window $[a_i^l, b_i^l]$ at node i for resource $l \in L'$ implies that a path with resource consumption greater than b_i^l when reaching the node i is infeasible. This feasibility restriction has to be satisfied for each of the resources in L'. A path reaching node i which consumed $\leq a_i^l$ of resource $l \in L'$, can "waste" (or wait) the difference by adjusting its consumption and is considered feasible. The resources (in L'), such as time, are set to zero at node p (the origin). I.e., $t_p^l = 0, l \in L' \subseteq L$. The resource levels for resources in L' at node i along a feasible path are computed as $t_{i_k}^l = t_{i_{k-1}}^l + t_{i_{k-1},i_k}^l, l \in L'$.

However, in this problem setting, for resources $l \in L \setminus L'$, the opposite is true. The resource levels for $l \in L \setminus L'$ at node p are set to some "capacity" value Q^l, and the levels at node i along a feasible path are computed as $t_{i_k}^l = t_{i_{k-1}}^l - t_{i_{k-1},i_k}^l, l \in L \setminus L'$. In this case, a path reaching node i with resource level greater than b_i^l is considered feasible, but if

the resource level is below a_i^l, this path is infeasible.

Path feasibility requires that the resource levels for each resource be within their node window at each node in the path or below (above) for the appropriate resource types. These resource feasibility requirements are multi-dimensional generalizations of time window and delivery constraints in vehicle routing.

An arc $(i,j) \in A$ is *infeasible* if it is never possible to visit node j after node i while respecting the feasibility requirements of both nodes i and j. The graph G can be preprocessed with this respect and all infeasible arcs excluded from A. This results in that for all arcs $(i,j) \in A$, the following conditions (constraints) are satisfied: $a_i^l + t_{ij}^l \leq b_j^l, l \in L'$, and $a_i^l - t_{ij}^l \geq a_j^l, l \in L \setminus L'$.

In summary, the Shortest Path with Resource Constraints Problem (SPRCP) is defined as follows: find a minimal cost feasible path in G from node p to node q. Since a number of $\mathcal{NP}$-hard problems, such as multi-dimensional knapsack problem and resource constrained project scheduling (Desrochers, 1988, Dror, 1994), can be formulated as a special case of the SPRCP, the SPRCP is clearly $\mathcal{NP}$-hard in the strong sense.

In order to present a dynamic programming approach for solving the SPRCP we introduce a definition of states. With each path X_{pj} from the origin node p to node j associate a state vector $(t_j^1, t_j^2, \ldots, t_j^L)$ corresponding to the resource levels at node j and a cost $C(t_j^1, t_j^2, \ldots, t_j^L)$ as a function of the resource levels vector. We restrict the discussion to feasible paths only, thus the state vector $(t_j^1, t_j^2, \ldots, t_j^L)$ describes feasible resource levels. For simplicity, let the label at node j be described by $(t_j^1, t_j^2, \ldots, t_j^L, C_j)$ and denoted by $\mathcal{T}_j$.

A transition between two feasible states $\mathcal{T}_i$ and $\mathcal{T}_j$ exists if

(1) There is an arc $(i,j) \in A$, and

(2) $t_j^l \leq b_j^l, l \in L'$, and $t_j^l \geq a_j^l, l \in L \setminus L'$ (since $t_j^l \geq t_i^l + t_{ij}^l, l \in L', t_j^l \leq t_i^l - t_{ij}^l, l \in L \setminus L'$).

Definition : Let $X_{pj}^{\#}$ and X_{pj}^{*} be two distinct paths from p to j with labels $\mathcal{T}_j^{\#}$ and $\mathcal{T}_j^{*}$ respectively. We say that $X_{pj}^{\#}$ *dominates* X_{pj}^{*} if and only if $C_j^{\#} \leq C_j^{*}, t_j^{l\#} \leq t_j^{l*}, l \in L', t_j^{l\#} \geq t_j^{l*}, l \in L \setminus L'$, and $\mathcal{T}_j^{\#} \neq \mathcal{T}_j^{*}$.

For resource $l \in L \setminus L'$, we can rewrite the resource level as $t_i'^l = -t_i^l$ and then drop the "$'$" notation so we can use only the inequality $\leq$ for determining non-dominance between labels for the same node i. This will enable a more "uniform" comparison as stated below.

The dominance relation, defined as vector comparison, determines that a path X_{pj} and its label $\mathcal{T}_j$ are non-dominated if and only if X_{pj} is the least cost path reaching node j with a state value less than or equal to $(t_j^1, t_j^2, \ldots, t_j^L)$. Clearly, it is sufficient to restrict our search to non-dominated paths. In such a search, we need only to consider extensions of non-dominated paths. The dynamic programming recurrence equations for the SPRCP are as follows:

$$C(0, \ldots, 0) = 0 \text{ for the source } p$$

$$C(t_j^1, t_j^2, \ldots, t_j^L) = \min_{i \in N}\{C(t_i^1, t_i^2, \ldots, t_i^L) + c_{ij} : \text{ such that } (t_i^1, t_i^2, \ldots, t_i^L)$$

feasible and $(i, j) \in A\}$, and $(t_j^1, t_j^2, \ldots, t_j^L)$ feasible for $j \in N \setminus \{p\}\}$

The fundamental step in a dynamic programming algorithm for shortest path problems is the "reaching" step. Starting with a label for a path X_{pi}, the algorithm determines the labels corresponding to feasible paths X_{pj} which extend the path X_{pi} for all arcs $(i, j) \in A$. In short, labels can be created for node j every time one of the nodes i with and arc $(i, j) \in A$ is reached. This node "labeling" operation might be time consuming when several resources are involved since many (non-dominated) labels (non-dominated paths) might be possible for each node. In Desrochers and Soumis (1988) (see also Denardo and Fox, 1979), an algorithm for what is called *generalized permanent labeling* is described and a concept of generalized bucket is introduced in order to handle the non-dominated node labels in an efficient manner. These ideas are further advanced in Desrochers (1988) to the SPRCP with multiple resource types (with up to 5 resources in the subsequent computational experiments).

4.5. SOLVING SPRCP WITH ONLY ELEMENTARY PATHS

Set covering approach to VRP (and VRPTW) requires that customers (nodes in G) be visited only once (elementary paths). However, the algorithm described in most of the DDDSS computational schemes for the SPRCP subproblem does not necessarily generate elementary path solutions as has been pointed out by Beasley and Christofides (1989). In

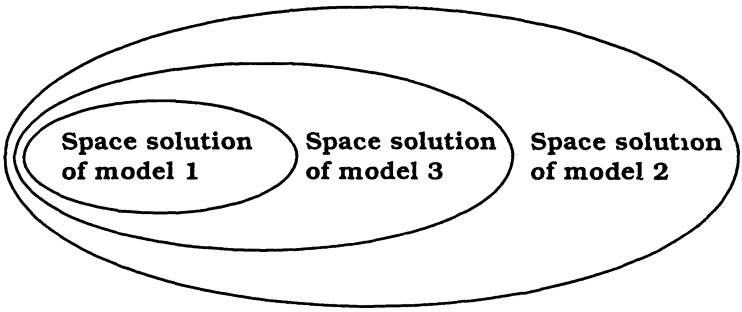


Figure 8.1 Inclusion of solution spaces.

a number of interesting cases such as the Selective VRP and the Prize Collecting Problem (see Guéguen, et al. 1999), the pseudo-polynomial solution approach to SPRCP is not feasible for the fact that the resulting solution does not necessarily have the elementary path structure. This section describes a modified algorithm which finds an optimal elementary path solution for SPRCPs of moderate size.

Desrochers et al. (1992) advanced three dynamic programming models for the solution of three different versions of the resources constrained shortest path problem based on the principles presented in Desrochers (1988). The solution space of the first model contains only feasible routes and it is stated that "the model is very time consuming to solve." The solution spaces for the two other models presented as alternatives to the first model, contain more than just the feasible routes. Generating such SPRCP routes for the models has a pseudo-polynomial time complexity. The solution space of the second model contains many different cycles, but the solution space of the third model contains cycles but no 2-cycles. See Figure 8.1 for the solution space relations between the three different shortest path problem versions. Desrochers et al. (1992) propose dynamic programming based procedures which solve efficiently relaxations of the resource constrained shortest path problem.

The master problem in the Desrochers et al. (1992) solution approach is the set covering problem presented in (8.1)-(8.3). The subproblem associated with this formulation is a Shortest Path Problem with Resources Constraints, and it is not necessary that the paths be elementary. Desrochers at al. (1992) also propose to eliminate all the 2-cycles. Subsequently, the candidate columns for the master problem consists of the non elementary paths without 2-cycles.

The second relaxation model in the Desrochers et al. (1992) approach has the largest solution space. Solving the elementary shortest path problem using their model 3 gives an optimal solution in the set $(S_3 \setminus S_1)$, then one is confronted with the following problem: how can the actual optimal solution (in the set S_1) without cycles be extracted? In the next subsection, it is shown that there are cases for which the algorithm proposed by Desrochers (1988) concludes that no elementary path (with negative costs) exists where in fact there is one.

4.5.1 Overlooking an Optimal Elementary Path.

The purpose of the example below is to show that for a case where there might be a unique one-time "profit" associated with visiting a node, Desrochers et al. (1992) algorithm is no longer a valid solution approach. It is not to say that Desrochers' algorithm is incorrect because Desrochers clearly stated that his algorithm solves the non-elementary path case.

In the case where the columns in the column generation procedure for a vehicle routing setting are restricted to elementary shortest paths, it is impossible to use an algorithm for the non-elementary shortest path problem. This fact is demonstrated via an example in Figure 8.2. In short, the transitivity of the dominance relationship defined earlier for the labels of paths which are not necessarily elementary, does not extend itself to elementary paths.

The example of Figure 8.2 proves that when considering non-elementary paths and the dominance relation proposed by Desrochers (1988), one is going to miss the optimal elementary path.

As before, the resources are restricted to two: the time and the capacity of the vehicle (Q is equal to 20). This graph has four possible paths going from the depot (node 0) to node 5, and there are no other paths because of the capacity constraint:

1 path 1 : $0 \rightarrow 1 \rightarrow 4 \rightarrow 5 : q = 20, t = 20, c = -5$

2 path 2 : $0 \rightarrow 2 \rightarrow 4 \rightarrow 5 : q = 12, t = 11, c = 45$

3 path 3 : $0 \rightarrow 2 \rightarrow 3 \rightarrow 6 \rightarrow 4 \rightarrow 5 : q = 17, t = 13, c = 15$

4 path 4 : $0 \rightarrow 2 \rightarrow 3 \rightarrow 6 \rightarrow 2 \rightarrow 4 \rightarrow 5 : q = 19, t = 14, c = -45$

If a non-elementary path is allowed, using the dominance relation of Desrochers (1988), label 4 dominates label 1, and in node 5, there remain only three labels corresponding to paths 2, 3 and 4. If one seeks to

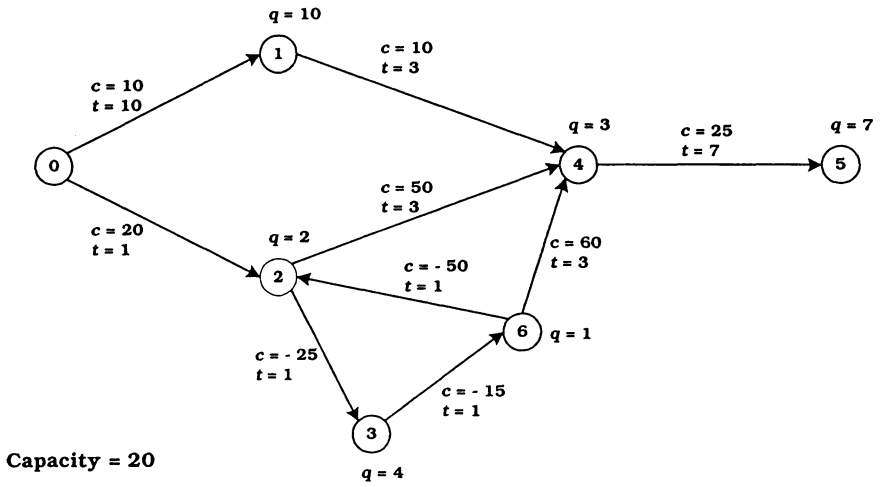


Figure 8.2 Demand and arc duration.

generate a path with negative cost between nodes 0 and 5, then paths 2 and 3 are eliminated. Label 4 is the only label remaining at node 5, but its corresponding path is non-elementary. Thus, the conclusion is that there does not exist a path with negative cost (i.e., the optimum of the column generation process has been reached). This is contrary to the fact that path 1 is elementary and has a negative cost of -5, which is going to improve the current solution.

The situation demonstrated in Figure 8.2 may occur if the graph contains negative cycles and the resources constraints are not very "tight".

4.5.2 An SPRCP – Improved Elementary Path Algorithm.

The elementary shortest path problem has been investigated by Beasley and Christofides (1989) where they proposed a procedure for finding elementary paths. It involves the addition of an extra resource for each node ($|N|$ extra resources in total), restricting the consumption value for that resource to be either fully consumed (= 1) or not consumed at all (= 0). When a path visits node k, it consumes its extra resource – $V_k = 1 (V_j = 0, j = 1, \ldots, n = |N|$, if the path did not visit node j). Beasley and Christofides did not test their procedure computationally and suggest that this formulation would be only suitable for solving rela-

tively small problems for which the number of resources is not very large.

In Guéguen et al. (1998), this idea of Beasley and Christofides has been implemented and tested to see if it can generate optimal solutions for problems of moderate size. We present the basic idea underlying the implementation of this notion. In what follows, the procedure from Guéguen et al. (1998) is presented in some detail.

First, the dominance relation was rewritten so that two node labels can be compared according to the new definition of labels. This dominance relation still allows the use of the original Desrochers (1988) algorithm since the principle of this algorithm remains the same: all non-dominated paths are the extension of a non-dominated path.

In order to eliminate efficiently the dominated labels, a new variable (new resource) f_j is added to each label at each node j. This label counts the number of nodes previously visited by the path. A label $\mathcal{T}_1$ cannot dominate a label $\mathcal{T}_2$ if it visits more nodes (because it consumes more "visitation" resources). So, instead of comparing all the resources corresponding to the n nodes, in some cases, a single comparison between two labels is sufficient to establish dominance.

With each path X_{pj} from the origin to node j we associate a state

$$\mathcal{R}_j = (t_j^1, \ldots, t_j^L, f_j, V_j^1, \ldots, V_j^n)$$

which computes the quantity of the resources used by the path, the number of visited nodes, and the visitation vector (let $f_j = \sum_{i=1}^{n} V_j^i$.).

For this new label for each node we also obtain a new dominance relation. This new dominance relation has the transitive property for all paths, elementary and non-elementary just as the "old" dominance relation.

With this new node labeling procedure, it is likely that we need to keep more labels for each node since some labels which were dominated in the previous definition of labels are no longer going to be dominated with the new one. However, we are sure to keep only labels corresponding to elementary paths. In the unlikely worst case, the labels for all the existing paths must be kept. However, this is extremely unlikely and the test results prove that this labeling procedure is quite effective.

The description of the algorithmic procedure is given below, but first, some notation has to be introduced.

Q_i - the list of feasible labels at node i,
$Succ[i]$ - the set of successors of node i,
E - the list of nodes that have to be treated,
$Ext(l_i, j)$ - the extended label at node j obtained from label l_i at node i,
$F_{ij}(Q_i)$ - the set of labels extended from node i to node j,
$EFF(L)$ - the procedure that conserves only non dominated labels in the list L of labels.

4.5.3 Description of the Algorithm.

Step 1 : *Initialization*
$Q_p = (0, \ldots, 0)$
$Q_i = \emptyset$ for $i = 1, \ldots, n, i \neq p$
$E = \{p\}$

Step 2 : *Exploration of the successors of a node*
Choose a node $i \in E$ and for $j \in Succ[i]$ such that j has not been visited from i, set $Q_j = EFF[F_{ij}(Q_i) \cup [Ext(l_i, j)]]$
If Q_j has been modified then $E = E \cup \{j\}$

Step 3 : *Reduction of E*
$E = E \setminus \{i\}$
If $E = \emptyset$ then end, else go to step 2

The important point here is that this algorithm will only generate elementary paths.

The time complexity of this algorithm is strongly related to the structure of the graph, the numbering of the nodes, and the resources constraints. For highly constrained problems, it is possible to implement this algorithm in an efficient way and reduce the required number of operations. This efficient implementation has been demonstrated in the computational experiments described in Guéguen et al. (1998).

Speedup Improvement In this section, a modification is presented which speeds up the computation time by reducing the number of labels that need to be considered. In some cases, it is not useful to consider the resources corresponding to some nodes. Labels which could not be compared considering all the resources will become comparable and some of them are going to be dominated, resulting in a "stronger" dominance relation.

Example: If you consider the label L_1= | 117 | 38 | 17 | 2 | 1 | 0 | 1 | and the label L_2= | 95 | 34 | 16 | 2 | 0 | 1 | 1 | where the different columns correspond to the time, the quantity, the cost, the number of visited nodes and the consumption of the resource associated with three nodes 1, 2 and 3. With the present relation, these two labels are not comparable.

A proposed modification leaves a resource associated with a node if it is impossible to visit this node again. For the nodes that have not been visited before and are unreachable with the given path (according to the time or the delivered quantity), indicate that the corresponding resource has been consumed. So, with the same example as above, two new labels are obtained: L'_1= | 117 | 38 | 17 | 3 | 1 | 1 | 1 | and the label L'_2= | 95 | 34 | 16 | 3 | 1 | 1 | 1 |

The label L'_2 dominates the label L'_1. If a third label L_3 dominates L_2, then it dominates also L'_2. Finally, if L_2 dominates L_3, then the transformed label L'_2 dominates the label L'_3 because all the transformations that were made on L_2 were also made on L_3.

Now we modify the previous definition for the labels to including the above label "correction". With each path X_{pj} from the origin to node j, associate a state $\mathcal{R}_j = (t_j^1, \ldots, t_j^L, f_j, V_j^1, \ldots, V_j^n)$ corresponding to the quantity of the resources used by the path, the number of nodes that are unreachable and the visitation vector with the following property: $V_j^i = 1$ if the path has visited node i or if it is impossible to reach this node with the current time value and the delivered quantity. Note that node i is now unreachable if $V_j^i = 1$.

Proposition 1: During the execution of the modified algorithm, one only needs to consider non-dominated paths.

Proof : Consider two labels in node i that are extended to node j such that $\mathcal{T}'_i \prec \mathcal{T}_i$. If it is possible to extend path X_{pi}, it is possible to extend path X'_{pi} because $V_i'^j \leq V_i^j = 0$.

More directly, $\mathcal{T}'_i \prec \mathcal{T}_i \Rightarrow C'_i + c_{ij} \leq C_i + c_{ij}$ and $t_i'^l + t_{ij}^l \leq t_i^l + t_{ij}^l$ for $l = 1, \ldots, L$, and
$t_j'^l = max\{a_j^l, t_i'^l + t_{ij}^l\}$ and $t_j^l = max\{a_j^l, t_i^l + t_{ij}^l\} \Rightarrow t_j'^l \leq t_j$ for $l = 1, \ldots, L$.

One still has to check the resource associated with each node:

- For the resource corresponding to node j: $1 = V_j'^j \leq V_j^j = 1$.
- For a resource corresponding to a node $k \neq j$:
  - if $t'_j + s_j + t_{jk} \leq b_k$ and $t_j + s_j + t_{jk} \leq b_k$: then it is possible to visit node k, thus, $V_j'^k = V_i'^k$ and $V_j^k = V_i^k$, and $V_j'^k \leq V_j^k$.
  - if $t'_j + s_j + t_{jk} \leq b_k < t_j + s_j + t_{jk}$: then it is impossible to return to k with label R. In this case set $V_j^k = 1$ and possibly $s_j = s_i + 1$, if node k was not visited by the path R before, and don't change the other label values. Thus, $V_j'^k \leq V_j^k$.
  - if $b_k < t'_j + s_j + t_{jk} \leq t_j + s_j + t_{jk}$: then it is impossible to visit node k: $V_j'^k = 1$ and $V_j^k = 1$, and $V_j'^k \leq V_j^k$.

For every case and every k, the inequality $V_j'^k \leq V_j^k$ holds. Then

$$\sum_{k=1}^{n} V_j'^k \leq \sum_{k=1}^{n} V_j^k \quad \text{i.e.} \quad f'_j \leq f_j$$

$$\Rightarrow \mathcal{T}'_j \leq \mathcal{T}_j$$

The computational experiments with the above modified SPRCP algorithm which generates only elementary paths are fully reported in Guéguen et al. (1998) and we do not reproduce the tables here. These results demonstrate that the above algorithm solves problems of moderate size and that the gain obtained when comparing to the original Desrochers (1988) approach is quite considerable. The common test bed for VRPTW and shortest path procedures in the context of column generation for VRPTW are the so-called Solomon's problems (see Solomon, 1983). The computational results clearly indicate that this new algorithm solves also VRPTW problems.

The algorithm with modification succeeded in solving 75 problems on the 87 tested problems in less than 300 seconds while the version without the modification succeeded in only 47 problems. The tests in Guéguen et al.(1999) prove that if a problem has "enough" constraints, then it is possible to optimally solve problems with 100 nodes in reasonable time and the more difficult the problems are, the more significant are the gains.

5. COLUMN GENERATION FOR ROUTING PROBLEMS WITH SPLIT DELIVERY

This section is based on the work of Mullaseril (1996), and Mullaseril and Dror (1997). Once again let $G = (N, A)$ be a directed graph with node set N and arc set A. Node 0 represents a depot while the remaining nodes corresponds to customers that have to be served. At the depot there is a "free" number of identical vehicles of capacity Q. Every node except the depot has a nonnegative demand q_i and every arc has an associated nonnegative cost c_{ij} and traversal time t_{ij}. With some danger of confusion, the service time at node i is traditionally denoted as s_i and represents the time needed to unload at i its demanded quantity q_i. Assume that the cost matrix (c_{ij}) satisfies the triangle inequality.

The classical Vehicle Routing Problem with Time Windows (VRPTW) consists of determining a set of least cost vehicle routing solution such that each route (a circuit on the graph G) starts and ends at the depot, every customer is served exactly once by one vehicle inside his time windows $[a_i, b_i]$ and the total demand of each route does not exceed the capacity of the vehicle. In this section, a version of this problem is examined in which the demand q_i can be delivered by several vehicles. This problem is called the Split Delivery Vehicle Routing Problem with Time Windows (SDVRPTW). In such a problem, the constraint that the demand of any customer is less or equal to Q is relaxed allowing for large customers whose demand might exceed the capacity of the vehicle.

The split delivery version of the classical vehicle routing problem without time windows was first introduced in Dror and Trudeau (1989, 1990), and in Dror et al. (1994) several new classes of valid constraints for the SDVRP are described in the context of a branch and bound algorithm (some problems with 10, 15 and 20 nodes have been solved). In Frizzell and Giffin (1995) a heuristic is proposed for vehicle routing with split deliveries and time windows for grid distance graph. Also in Mullaseril et al. (1997) a heuristic for split delivery arc routing with time windows is examined on some real-life problems. Finally, two other papers due to Sierksma and Tijssen (1998) and Mullaseril and Dror (1997) present optimal methods based on column generation respectively for the SDVRP and the SDVRPTW. The method proposed by Sierksma and Tijssen (1998) is used to schedule crew exchanges on off-shore locations in the North Sea. The limited number of places in a helicopter makes that it is indispensable to split the demand (operations and maintenance crews flown in) of some platforms. It is important to remark that the quantity of split (i.e. how many people of one platform are in an helicopter) is necessarily integer and this quantity is determined in the column generation subproblem. In the paper of Mullaseril and Dror (1997), only

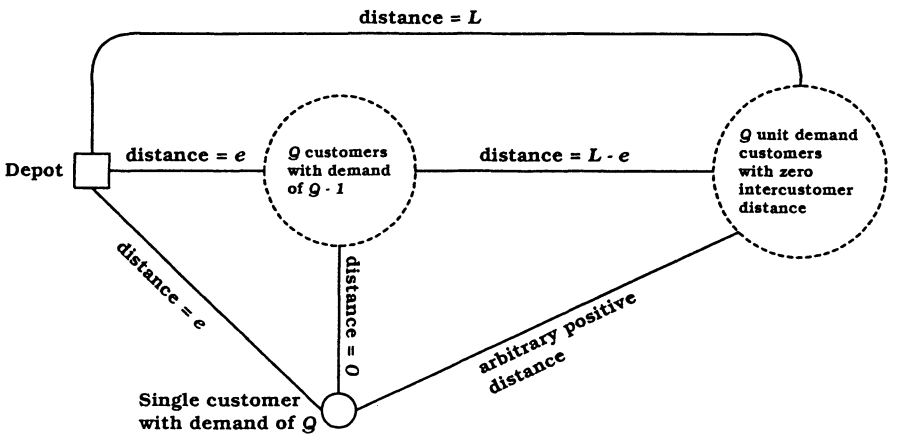


Figure 8.3 Example of $2Q + 1$ customers for which nonsplit delivery solution with no more than $Q + 1$ vehicles of capacity Q has a distance of $2QL + 2e$, where as the split solution with $Q + 1$ vehicles is of distance $2L + 2Qe$.

some preset proportions of split delivery are allowed and these proportions are also preset in the subproblem in the column generation scheme. Consequently, the solution can serve only as an "approximation" for the optimal solution in a sense similar to approximating a convex function by piecewise linear functions. I.e., based on how refined are the preset split deliveries.

In this section, a set covering formulation for the SDVRPTW is examined and an algorithm is described that solves to optimality problems of moderate size. However, first some properties of to the SDVRPTW solution are restated.

5.1. PROPERTIES OF SPLIT DELIVERIES WITH TRIANGLE INEQUALITY

It is very easy to demonstrate the potential savings accrued by allowing split deliveries to the demand points. Assume there are $(2Q + 1)$ demands points as depicted in Figure 8.3. Q points with demand 1 have a distance of L from the depot. Q points with demand Q and one point with demand $(Q - 1)$ are located at a distance $e \ll L$ from the depot. Also suppose that there are $(Q + 2)$ vehicles of capacity Q.

The optimal solution without split deliveries uses $(Q+1)$ vehicles and has a cost of $2LQ$. If split is allowed, it is possible to find a solution

of cost $2L$ in which one vehicle serves the Q points of demand 1 and the remaining demand is split between the other vehicles. The ratio of the solution value without split over the one with split deliveries for this example tends to Q as $e \rightarrow 0$. Thus, the ratio of the optimal non-split solution and the optimal split solution can be arbitrarily large. It is straightforward to extend this result to the case of deliveries with time windows.

The split delivery problem (the SDVRP or the SDVRTW) might be perhaps viewed as a "relaxation" of the discrete VRP. However, adding the option of split delivery introduces a whole new set of delivery alternatives in which each one of the routes still represents a TSP solution over its set of deliveries. Clearly, the SDVRP is $\mathcal{NP}$-hard just by reduction from the TSP (by setting the capacity Q larger than the sum of deliveries). As the formulation below indicates, the SDVRP is modeled as a mixed 0-1 programming problem with y_i^v variables (indicating the fraction of demand at i delivered by route v) taking their values between 0 and 1. It has been "conjectured" by Dror et al. (1994) that the SDVRP is harder than the classic (non-split) VRP in the sense that certain cut inducing inequalities for the VRP (the 0-1 formulation) are no longer valid for the SDVRP and the corresponding polyhedral solution approach is more difficult.

The following observations are taken from Dror and Trudeau (1990), and Dror, et al. (1994):

Observation 1: If the $\{c_{ij}\}$ matrix satisfies the triangle inequality, there exists an optimal solution of the SDVRP in which no two routes have more than one split demand point in common.

Definition (see also Dror et al., 1994) : Consider k demands points $n_1, n_2, ..., n_k$, and k routes such that route 1 includes points n_1 and n_2, route 2 includes points n_2 and n_3, ..., route $k-1$ includes points n_{k-1} and n_k and route k includes points n_k and n_1. (This implies that the points $n_1, n_2, ..., n_k$ receive split deliveries by the k respective routes and possibly other routes). This subset of demand points $\{n_i\}_{i=1}^k$ is called a *k-split cycle.*

Observation 2: If the $\{c_{ij}\}$ matrix satisfies the triangle inequality, there exists an optimal solution of the SDVRP which contains no k-split cycle (for any k).

Proofs of the observations above are given in Dror and Trudeau (1990). For the rest of the chapter it is assumed that the triangle inequality for

the time matrix, and cost matrix is satisfied. As demonstrated later, it is not possible to extend these observations for the SDVRPTW.

5.2. FORMULATION

An integer formulation for the SDVRPTW presented by Mullaseril and Dror (1997) is given below. Similar formulation is presented in Frizzell and Giffin (1995). This formulation, even though not used in the computational analysis, serves as a "stepping stone" to the set covering formulation and the column generation methodology presented later in the chapter. Let x_{ij}^v be a binary variable defined for $i \neq j$ and equal to 1 if and only if vehicle v serves customer j just after customer i. Let y_i^v be the proportion (the fraction) of the demand of customer i delivered by vehicle v and t_i^v be the time at which vehicle v starts serving customer i. Let V be an upper bound on the number of vehicles required. Let T be a large constant (for instance, $T = \max(\max_{(i,j)} (b_i + s_i + t_{ij} - a_j), 0)$). The problem can then be formulated as:

$$\text{Minimize} \sum_{i=0}^{n} \sum_{j=0}^{n} \sum_{v=1}^{V} c_{ij}^v x_{ij}^v \tag{8.4}$$

subject to

$$\sum_{i=0}^{n} x_{ij}^v = \sum_{i=0}^{n} x_{ji}^v, \quad j = 0, ..., n; v = 1, ..., V \tag{8.5}$$

$$\sum_{v=1}^{V} y_i^v = 1, \quad i = 1, ..., n \tag{8.6}$$

$$\sum_{i=1}^{n} q_i y_i^v \leq Q, \quad v = 1, ..., V \tag{8.7}$$

$$\sum_{j=0}^{n} x_{ij}^v \geq y_i^v, \quad i = 1, ..., n; v = 1, ..., V \tag{8.8}$$

$$t_i^v + s_i + t_{ij} - (1 - x_{ij}^v)T \leq t_j^v, \quad i = 0, ..., n; j = 1, ..., n; v = 1, ..., V \tag{8.9}$$

$$a_i \leq t_i^v \leq b_i, \quad i = 0, ..., n; v = 1, ..., V \tag{8.10}$$

$$x_{ij}^v \in \{0, 1\}, \quad i, j = 0, ..., n; v = 1, ..., V \tag{8.11}$$

$$0 \leq y_i^v \leq 1, \quad i = 1, ..., n; v = 1, ..., V \tag{8.12}$$

Constraints (8.5) are the flow conservation conditions. Constraints (8.6) ensure that the demand of each customer is entirely satisfied while constraints (8.7) specify that vehicle capacities are never exceeded. Constraints (8.8) are consistency constraints: if vehicle v delivers customer

i, it has to leave this customer. Constraints (8.9) and (8.10) ensure the time continuity, force service instants to be inside the time windows of the customers and forbid the formation of illegal subtours.

5.3. SET COVERING APPROACH FOR SPLIT DELIVERIES

In the set covering formulation of a split delivery vehicle routing problems, a column represents not only a vehicle trip (a circuit on the corresponding graph) but also the split deliveries made on that trip. Since the fleet is homogeneous (i.e., identical vehicles), the index v denoting a vehicle index changes its designation and now denotes a trip index since each trip starts and terminates at the depot node. Let $\{\mathbf{a}_k|\ \ k = 1, \ldots, h\}$ be a collection of column vectors representing all feasible trips in G. In vehicle routing problems where split deliveries are not allowed, the entire demand for a node is delivered by the unique trip which visits that node. Thus, a feasible trip (column), is determined only by the nodes visited in that trip and its feasibility established by the total demand of the nodes visited. However if split deliveries are allowed, a trip k must also specify the fraction of the demand delivered at each node i when visited by trip k. In this case the set covering formulation has to impose an additional constraint for each node i that the total demand for a node i is delivered by adding up the demand's fractions on some subset of trips.

Definition: A vector T_k consisting of 1's and 0's of size $|N|$, whose i^{th} elements indicate whether the trip k visits node i (represented by 1) is called a *visitation vector* for trip k.

Definition: A vector P_k with fractional elements, of size $|N|$, whose i^{th} elements indicates the fraction of the demand delivered at node i is called the *delivery vector* for trip k.

Note that in formulation (8.4)-(8.12), the x_{ij}^v's determine the visitations and the y_i^v's determine the deliveries.

A column denoting a split delivery trip k can be represented by a vector $\mathbf{a}_k$ consisting of the two vectors T_k and P_k:

$$\mathbf{a}_k = \begin{bmatrix} T_k \\ P_k \end{bmatrix}$$

Hence, a feasible trip, i.e., the column $\mathbf{a}_k$, consists of two vectors, the visitation vector for that trip and its corresponding delivery vector with

a total $|2N|$ elements in the column. Unlike the pure set covering problem, the column vector $\mathbf{a}_k$ of the set covering formulation for the split delivery problem may have fractional elements. The set covering formulation of the split delivery node routing problem (SDVRP) is given below:

(SC1)

$$\min \sum_{k\in\Omega} c_k z_k$$

$$\sum_{k\in\Omega} T_{ik} z_k \geq 1, i = 1, ..., n \tag{8.13}$$

$$\sum_{k\in\Omega} P_{ik} z_k \geq 1, i = 1, ..., n \tag{8.14}$$

$$z_k \in \{0, 1\}, \forall k \in \Omega.$$

where the variable z_k determines the inclusion ($z_k = 1$) or the exclusion ($z_k = 0$) of the trip $\mathbf{a}_k$, while c_k is its associated cost. T_{ik} and P_{ik} are the ith elements of vectors T_k and P_k. The set Ω is the set of all possible trips for the vehicles. An important point to note is that the visitation vector T_k and the delivery vector P_k must satisfy the following vector inequality:

$$P_k \leq T_k, \forall k$$

This implies that if the i^{th} element of P_k is non-zero ($0 < P_{ik} \leq 1$), the corresponding element of T_k must be 1. Constraints (8.13) insure that at least one vehicle visits a node requiring service, whereas constraints (8.14) insure that all of the demand for a node is delivered by some subset of the trips.

In principle, since $T_k \in Z^N$ and $P_k \in R^N$ (many splits are possible), the problem SC1 will have an (uncountably) infinite number of columns and, consequently of decision variables z_k, which determine the selected columns by taking the value 1 (i.e., the set Ω is of uncountable cardinality). This is very different from the non-split delivery case where the number of possible trips is finite even if such number is of exponential order as a function of the number of nodes in the graph. Thus determining the optimal solution to the problem above by searching through all feasible solutions to the problem is not tractable even for very small size problems. Instead, one must follow the column generation solution methodology and first solve a linear relaxation of the above set covering problem formulation using a standard (revised simplex) column generation approach, where no explicit listing of the entire set of feasible solutions is necessary. This LP relaxation solution provides a lower

bound for the SDVRP formulated in the previous subsection.

To solve the set covering problem one first solves the LP relaxation of SC1 using column generation methodology as described in section 4. Each column in the master problem represents a feasible tour of a subset of nodes in N and is determined by solving a subproblem, where vehicle capacity constraints and time window constraints (if necessary) are imposed. In the split delivery problem version, one also relaxes the requirement that, on every vehicle visit, the entire node (customer) demand is delivered. The generated column and its associated column variable are added to the problem SC1 and another pivot operation is performed. Similarly as in the column generation scheme for the VRP, each subproblem generates a vehicle trip which starts at the depot, visits one or more nodes and terminates at the depot node.

5.4. THE SUBPROBLEM FOR GENERATING FEASIBLE COLUMNS FOR SPLIT DELIVERY

Let the decision variable $x_{ij}^v = 1$, $(i,j) \in A$, if a trip v uses arc $(i,j) \in A$ and $x_{ij}^v = 0$ otherwise. Knowing the arcs traversed by a trip v will allow to construct the sequence in which the nodes are visited. Following Mullaseril and Dror (1997), to determine whether or not a trip visits a node $i \in N$, we compute the quantity $\sum_{j \in N} x_{ij}^v, (i,j) \in A$. Let y_i^v be the fraction of the delivery made to node $i \in N$ by trip v. The vector T_v, consists of elements $T_{iv}, i \in N$, where $T_{iv} = \sum_{j \in N} x_{ij}^v$, and the vector P_v consists of the elements $P_{iv} = y_i^v, i \in N$. Thus the column $\mathbf{a}_v$ for the master problem is constructed from the solution to the subproblem as follows:

$$\mathbf{a}_v = \begin{bmatrix} \sum_{j \in N} x_{1j}^v \\ \vdots \\ \sum_{j \in N} x_{nj}^v \\ y_1^v \\ \vdots \\ y_n^v \end{bmatrix}$$

Let c_{ij} be the length of arc $(i,j) \in A$. Let $\pi = [\alpha, \beta]$ be the vector of dual variables associated with problem SC1 (restricted to the columns generated so far; α_i and β_i, $i = 1, \ldots, n$, are the values of the dual variables associated with node i, i.e., with the corresponding two constraints for node i). Thus π consists of two vectors α and β, each of size n consisting of these dual variables:

$$\pi = \begin{bmatrix} \alpha_1 \\ \vdots \\ \alpha_n \\ \beta_1 \\ \vdots \\ \beta_n \end{bmatrix}$$

Suppose that we can generate the column $\bar{\mathbf{a}}_v$ corresponding to the minimum reduced cost $\bar{\mathbf{c}}_v$. Here we state how it can be done. The cost $\mathbf{c}_v$ of the column $\mathbf{a}_v$ is equal to the total length of a trip, or $\sum_{(i,j)\in A} c_{ij}x_{ij}^v$. To generate a new column $\bar{\mathbf{a}}_v$ the subproblem has the following objective:

$$\bar{\mathbf{c}}_v = \min\ (\mathbf{c}_v - \pi\mathbf{a}_v)$$

over all feasible columns $\mathbf{a}_v$. Which can be rewritten as

$$\bar{\mathbf{c}}_v = \min\ (\sum_{(i,j)\in A} (c_{ij} - \alpha_i)x_{ij}^v - \sum_{i\in N} \beta_i y_i^v)$$

Now we can formally present the subproblem whose solution generates a new candidate column for the "master" problem, i.e., the restricted version of SC1, in the column generation scheme.

5.4.1 Formulating the Subproblem.

The formulation is based on the work described in Dror and Trudeau (1990) and Desrosiers *et al.* (1995), and it assumes that no k-cycles can occur in the optimal solution for this problem of split-delivery with time-windows.

Denote by s_i the service time at node i and assume that this service time is additive (dependent on the fraction delivered). Also t_i denotes the time the vehicle arrived at node i. The subproblem (SP) to generate a feasible column $\mathbf{a}_v$ is (the superscript v has been omitted in this model):

$$\text{Minimize} \sum_{(i,j)\in A} (c_{ij} - \alpha_i)x_{ij} - \sum_{i\in N} \beta_i y_i \qquad (SP)$$

subject to

$$\sum_{(i,p)\in A} x_{ip} - \sum_{(p,j)\in A} x_{pj} = 0, \forall p \in N \setminus \{0\}$$

$$\sum_{(0,i)\in A} x_{0i} + \sum_{(i,0)\in A} x_{i0} = 2$$

$$y_i - \sum_{(i,j)\in A} x_{ij} \leq 0, \forall i \in N$$

$$\sum_{i\in N} q_i y_i \leq Q$$

$$(t_i + t_{ij} + s_i y_i - t_j) x_{ij} \leq 0, \forall (i,j) \in A$$

$$a_i \leq t_i \leq b_i, \forall i \in N$$

$$x_{ij} \in \{0,1\}, \forall (i,j) \in A$$

$$0 \leq y_i \leq 1, \forall i \in N$$

This subproblem can be characterized as a *simple shortest cycle* (or a shortest path from an origin node (= depot) to the destination node (= depot)) problem over a graph G with time windows and partial delivery. Observe that the costs associated with each arc may be negative. Such cycles are of finite length owing to the existence of time window constraints and hence the subproblems are bounded from below. The above problem is $\mathcal{NP}$-hard in the strong sense (see Dror, 1994).

5.4.2 Alternating between the Master Problem and the Subproblem.

The solution process for these type of column generation solution approach entails alternating between solving the master problem and the subproblems. The procedure is usually started with an initial set of columns for the master problem which forms the initial identity basis, for example, $|N|$ single node trips from the depot to each of the nodes in N, if the demand of each node does not exceed the capacity Q. For each pivot operation in the master problem, the dual variable vector π is updated and used for solving the corresponding subproblem. After solving the subproblem, the resulting column is added to the master problem and the next pivot operation performed. The column generation procedure is terminated when no more negative marginal cost columns exist. This process generates an optimal solution to a linear relaxation of the problem SC1, which is a lower bound to the set covering problem formulation for the SDVRPTW.

To our knowledge, the special structure of our master LP problem, expressed by the vector property $P_k \leq T_k$ for each of the master problem columns, has not been examined in terms of speedup in the LP or a corresponding matrix size reduction for LP operations and updates. Since it naturally occurs in the split delivery setting, it would be of interest to investigate linear systems with this type of matrix partition and column property.

5.5. THE COMPUTATIONAL PHASE: A MIXED INTEGER SOLVER AND A DYNAMIC PROGRAMMING ALGORITHM

Testing the effectiveness of the column generation method described above by running extensive computational experiments on data from the arc routing problems in cattle feed industry as well as randomly generated problems derived from well known problems in the literature is described next. The actual solution approach implemented in the tests is based on the so called 'pulling' dynamic programming algorithm discussed in Desrochers (1988) and reviewed by Desrosiers et al. (1995). This solution scheme, based on the dynamic programming algorithm, is reproduced by Mullaseril (1996) and Mullaseril and Dror (1997) and applied to problems where both split delivery and time window constraints are present.

Unlike Desrochers' (1988) problem of non-split deliveries with time windows, when split-deliveries are allowed, some basic characteristics of the subproblem change. For instance, in non-split subproblems it has been proven that so called *2-cycles* (i.e., cycles of the form (i, j, i)) do not exist in the optimal solution. With split deliveries, 2-cycles can occur in the optimal solution of the subproblem. This is easily demonstrated by the following example even when the service time is independent of the demand delivered:

Consider two nodes (node 1 and node 2) with time window of $[0, 5]$ and $[2, 3]$ respectively. Assume also service time of 1 for node 1 and service time of 1 for node 2 and travel time of 1 unit between the nodes (symmetric). Entering node 1 at time 0, the vehicle would deliver only 50% of the demand (service of 1 unit) continuing to node 2 to deliver its total demand in the specified time window of $[2, 3]$ before returning to node 1 in time and completing the delivery (in time).

Still, note that for split-delivery with time windows there are no 'double' 2-cycle solutions. I.e., routing sequence like $i \rightarrow j \rightarrow i \rightarrow j$ cannot

occur.

The above example of 2-cycle can be generalized in a straightforward fashion to existence of k-cycle ($k \geq 2$) in the optimal split delivery subproblem. The potential of such k-cycles in the optimal solution to the subproblem complicates the subsequent analysis considerably. Thus, for the sake of simplicity and full knowledge of the consequences, in the subsequent analysis the k-cycles in the subproblem's solution are not allowed. In this sense, the result should be viewed as a 'heuristic' since optimality of the solution cannot be guaranteed. Note that in the case where the time windows are substantially greater than the service time and triangular inequality for the distance matrix holds, even in the split-delivery case, no k-cycles can occur in the optimal subproblem solution.

5.5.1 Discretizing the Split Deliveries in the SPPRC Subproblem.

Since the split-delivery routing problem in general is very "hard", and the split-delivery problem with time windows "reasonably" wide does not seem to constrain this problem sufficiently and make the task of solving this problem any easier, a reasonable compromise is that of discretizing possible split-deliveries to each node. This would considerably reduce in "size" the number of alternative routings and imply a transformation (which depending on the discretization scheme could be pseudopolynomial) of the original graph into a graph with no split deliveries. Since the original SDVRPTW is $\mathcal{NP}$-hard in the strong sense, a pseudopolynomial transformation does not effect the complexity of the resulting problem. The solution approach to the discretized SDVRPTW is described in detail in this subsection.

Let $\mathcal{R} = \{Q, t\}$ represent a set of two resources: capacity (Q) and time (t). Consider 'discretizing' the continuous variable y_i in the subproblem SP. In the algorithms implemented by Mullaseril and Dror (1997) for the feedyard problem, the fractional variable y_i is restricted to values $\{0.1, 0.2, \ldots, 0.9, 1.0\}$. As a result, the optimal subproblem solution will not guarantee an overall optimal solution for the SDVRPTW but only an upper bound (heuristic) solution to the problem (the set covering formulation).

Note that discretizing the fractional variable y_i is equivalent to a 'graph' transformation where a single node becomes a "chain" of nodes (one node for each fractional value) with no split-deliveries allowed to any node. Each of the nodes in such a chain is connected to all other nodes in the graph in the same manner as the original "parent" node.

The arcs from one node to another (next) node in the chain have a zero cost but have a time span (t_{ij}) corresponding to time required to service the node i. Note that since the nodes in a "chain" are identical fractions of the original node they can be numbered consecutively such as $i_1, i_2, \ldots, i_k$, and each requires a delivery of $(1/k)q_i$. The "chained" nodes are connected by directed arcs in order. Thus, we can assume (w.l.g.) that the "fractional" nodes are delivered in the chain order.

The dynamic programming method of Desrochers (1988) involves determining the shortest resource constrained feasible path from the origin p (the depot) to the destination node q (also the depot). With each path X_{pi} from the origin p to a node i we associate a label $\mathcal{T}_i$ as before, which corresponds to the consumption of each resource (time and capacity) on a path X_{pi} and a cost C_i of this path. The rest of this solution is essentially identical to the one outlined in section 4.4 for the non-split case.

5.5.2 Discussion of Computational Experiments for the SDVRPTW.

Computational experiments for the SDVRTW are reported in Mullaseril (1996) and in Mulleseril and Dror (1997). The column generation algorithm based on Desrochers' (1988) dynamic programming approach was used to solve arc routing problems with time windows and split delivery from a cattle yard that operates in Arizona. The CARP problems encountered varied from $227 \leq |V| \leq 581$, $294 \leq |A| \leq 770$ and $16 \leq |R| \leq 348$. The algorithm was also tested on randomly generated problems based on test problems from the literature. These problems provided two types of input data, those with Euclidean distances and those with non-Euclidean distance matrix. The data for Euclidean problems were derived by taking a subset of nodes of the 75-node problem originally presented in Eilon, Watson-Gandy and Christofides (1971) and used by many others in their computational experiments. In the problems generated, we allow split delivery. However, this data set does not contain time windows associated with each node. Subsequently, the time window values for each node have been randomly generated. For more details on how the problems were constructed see Mullaseril (1996) and Mullaseril and Dror (1997). The solutions obtained for the cattle yard problems (5 problems) using the above approach were about 6% better than the heuristic solutions reported in Mulleseril et al (1997).

In the next section we examine a modified approach for solving the SDVRPTW.

5.6. ANOTHER SET COVERING FORMULATION FOR SDVRPTW

Let Ω be the set of feasible routes for the SDVRPTW, respecting the time windows of the customers. Let a_{ik} be a constant value equal to 1 if and only if route $k \in \Omega$ visits customer i. c_k represents the cost of route k and is equal to the sum of the costs of the arcs of the route.

Let x_k be a binary variable equal to 1 if and only if route k is used in the optimal solution and y_{ik} be the quantity delivered by route k to customer i. The set covering formulation consists in choosing the routes satisfying all the constraints such that the total cost is minimum. Let this model be (MP_1):

$$\text{Minimize} \sum_{k \in \Omega} c_k x_k$$

subject to

$$\sum_{k \in \Omega} a_{ik} y_{ik} \geq q_i, \quad i = 1, ..., n \tag{8.15}$$

$$x_k Q \geq \sum_{i=1}^{n} a_{ik} y_{ik}, \quad k \in \Omega \tag{8.16}$$

$$x_k \in \{0, 1\} \quad k \in \Omega$$
$$0 \leq y_{ik} \leq q_i, \quad i = 1, ..., n; k \in \Omega$$

Note that we can modify constraints $0 \leq y_{ik} \leq q_i$ for each i to simply $0 \leq y_{ik}$ for each i since if $y_{ik} \geq q_i$, and equivalent cost solution with $y_{ik} = q_i$ exists.

This model determines the set of routes in the optimal solution and the quantity that is delivered to the customers visited by each route. Constraints (8.15) ensure that the demand of each customer is satisfied. The capacity constraint of the different vehicles is represented by constraints (8.16) which is also a consistency constraint: if customer i is served by route k, the cost of this route has to be incurred.

This model has two interesting properties: (i) it finds the exact split quantity on each route and does not search the solution in a limited subset of split options such as in Mullaseril and Dror (1997), and (ii) because no 2-cycles exist in an optimal solution, it does not generate two columns which represent visits to exactly the same subset of customers

with different delivery quantities.

However, the problem with this formulation is that its LP relaxation is very far from the integer solution. Constraints (8.16) are not strong enough to force variables x_k to have a value near 0 or 1.

It is thus possible to add two kinds of constraints to model (MP_1):

$$x_k q_i \geq y_{ik}, \forall i, k |\ a_{ik} > 0 \tag{8.17}$$

$$\sum_{k \in \Omega} a_{ik} x_k \geq \left\lceil \frac{q_i}{Q} \right\rceil, \forall i = 1, ..., n \tag{8.18}$$

Constraints (8.17) are consistency constraints which force a route to be used if it serves a customer and force the variable x_k to be as big as possible and specially $x_k \geq \frac{y_{ik}}{q_i}$. If one customer is not split in route k, x_k is then equal to 1. Constraints (8.18) ensure the visit of the minimum number of vehicles according to the demand for each customer.

With constraints (8.17) and (8.18), respectively we obtain two models (MP_2) and (MP_3) that can be used to solve the problem. It is also possible to create another model (MP_4) including both sets of constraints.

It is clear that constraints (8.17) are stronger than constraints (8.18) but while there are only n constraints (8.18), one for each customer, for constraints (8.17), each time we generate a route, we have to generate all the constraints corresponding to the customers visited by this route. Subsequently, the linear relaxation of problem (MP_2) is quite difficult (time consuming) to solve due to the large number of constraints.

5.6.1 The (MP_3) Model and Column Generation Approach.

The model (MP_3) (called the master problem) is stated below.

$$\text{Minimize} \sum_{k \in \Omega} c_k x_k$$

subject to

$$\sum_{k\in\Omega} a_{ik}x_k \geq \left\lceil \frac{q_i}{Q} \right\rceil, \quad i = 1, ..., n \tag{8.19}$$

$$\sum_{k\in\Omega} a_{ik}y_{ik} \geq q_i, \quad i = 1, ..., n \tag{8.20}$$

$$x_k Q \geq \sum_{i=1}^{n} a_{ik}y_{ik}, \quad k \in \Omega \tag{8.21}$$

$$x_k \in \{0, 1\}, \quad k \in \Omega \tag{8.22}$$

$$0 \leq y_{ik}, \quad i = 1, ..., n; k \in \Omega \tag{8.23}$$

The number of variables in the above model corresponds to the number of feasible routes and, as noted before, can be "very large".

We must have an efficient algorithm that is able to "price out" (generate) all feasible routes, in particular those that have a negative reduced cost. Model (MP_3) is slightly different from the conventional set covering formulations and requires to adapt the classical column generation scheme described by Desrochers et al. (1992). Each time a column is generated (and its associated variable), we also have to generate as many variables as the number of customers that are visited by the new route. It is important to remark that these variables have 0 cost and do not appear in the objective function. We also have to add one consistency constraint.

The simplex solution provides respectively the dual variables associated to constraints (8.19), (8.20) and (8.21) for the optimal solution of the restricted master problem: α_i, β_i and ω_k. Consider a new route k_0 defined by a column with elements $(a_{ik_0})_{(i=1,...,n)}$. Its reduced cost is equal to:

$$\bar{c}_{k_0} = c_{k_0} - \sum_{i=1}^{n} a_{ik_0}\alpha_i - Q\omega_{k_0} \tag{8.24}$$

Suppose that $\bar{c}_{k_0} < 0$, we need to show that with the new constraints and the new variables y_{ik_0}, the corresponding column will enter in the basis. To do that, we have to just prove that the dual problem (D_3') of the new problem (MP_3') (with the new variables and constraints) is not feasible. The new column for x_{k_0} in (MP_3') corresponds to a constraint in (D_3') that is:

$$\sum_{i=1}^{n} a_{ik_0}\alpha_i + Q\omega_{k_0} \leq c_{k_0} \tag{8.25}$$

But since $\omega_{k_0} \geq 0$ then the constraints (8.25) can not be satisfied facing equation (8.24) and the fact that the reduced cost $\bar{c}_{k_0}$ is strictly negative. So, the dual problem (D_3') is not feasible. In (D_3'), the new variables do not operate in the objective function and in the initial constraints. So we are sure that the new solution of the dual problem will be lower than the first one. Thus, it is clear that the variable x_{k_0} is going to enter in the basis and will decrease the value of the objective function.

Because it is accounted directly in the master problem for the capacity of the vehicles, we do not have to consider these capacities in the subproblem. The subproblem is a special case of the Elementary Shortest Path Problem with Resource Constraints (see Guéguen et al., 1998, and the next section). Time and capacity are the only resources that need to be considered.

A complete description of the problem is given below. (Again, x_{ij} is equal to 1 if arc (i,j) belongs to the shortest path, 0 otherwise. t_i is the time at which node i is visited. T is still a large constant.) The subproblem can then be formulated as:

$$\text{Minimize} \sum_{(i,j)\in A} (c_{ij} - \alpha_j) x_{ij}$$

subject to

$$\sum_{(i,p)\in A} x_{ip} - \sum_{(p,j)\in A} x_{pj} = 0, \forall p \in N \setminus \{0\}$$

$$\sum_{(0,i)\in A} x_{0i} + \sum_{(i,0)\in A} x_{i0} = 2$$

$$t_i + s_i + t_{ij} - (1 - x_{ij})T \leq t_j, \forall (i,j) \in A$$

$$a_i \leq t_i \leq b_i, \forall i \in N$$

$$x_{ij} \in \{0,1\}, \forall (i,j) \in A$$

To solve the above problem, we can use the algorithm by Guéguen et al. (1998) (summarized in the next section) which is also an adaptation of the frequently used algorithm proposed by Desrochers (1988). This algorithm solves the non Elementary Shortest Path Problem with Resources Constraints. It is important to note that this approach is valid when there is a strong assumption that the service time is independent of the quantity that is delivered to a customer.

When the optimal solution is reached, one suggestion is to detect and add in the master problem the violated constraints (8.17) and try again to add new columns by solving the subproblem. This should improve

the quality of the linear relaxation of the set covering problem and it is important to remark that adding the violated constraints does not change the formulation of the subproblem.

5.6.2 The Branching Scheme for the SDVRPTW.

The column generation gives the optimal solution of the linear relaxation of the set covering formulation (MP_3). If all the variables x_k have an integer value, the solution is also optimal for the SDVRPTW since we do not demand that the variables y_{ik} be also integer. In the case that the x_ks are fractional, a Branch & Price tree must be explored and additional columns might be generated at each node of the tree.

It may seem, at first, that the Branch & Price scheme described by Desrochers et al. (1992) is not reusable. This scheme involves branching at the first level on the number of vehicles and at the second level, on the arcs of the subproblem network. It means that if one branches on an arc (i, j), it imposes that in one branch $x_{ij} = 1$, i.e. if node i is served, one is forced to serve node j just after and in the other branch $x_{ij} = 0$, i.e. it is forbidden to serve node j just after node i. In each case, some arcs are deleted in the subproblem network.

Because split is allowed, several vehicles might visit a customer. So, it is possible, in a solution, that a vehicle k uses the link (i, j) and some other vehicle l uses the link (i, m) where j and m are two different customers. It is then impossible to delete links in the subproblem. Subsequently the branching rules of the second level are inappropriate in this case. For the same reason, it is inappropriate to use the rules described by Barnhart et al. (1998). They proposed that two customers r and s are covered by the same column on the first branch and by different columns on the second branch. It is clear that this rule cannot be used here for the same argument. These methods therefore cannot be used directly, but similar ideas can lead to a valid branching scheme.

First, it is clear that it is still possible to branch on the number of vehicles. If the number of vehicles used: $\sum_{k\in\Omega} x_k$ is fractional, say equal to v, we create two branches corresponding to $\sum_{k\in\Omega} x_k \leq \lfloor v \rfloor$ and $\sum_{k\in\Omega} x_k \geq \lfloor v + 1 \rfloor$. In each case, the dual d_{nv} variable associated to such a constraint is adequately transferred in the subproblem.

Secondly, we can introduce b_{ij}^k which is equal to 1 if vehicle k goes through the arc (i, j) and 0 otherwise. Then, $\sum_{k\in\Omega} b_{ij}^k x_k$ is equal to the number of time arc (i, j) is covered by all the vehicles. But, since 2 cycle splits do not appear in an optimal solution, it is possible to generate an

optimal solution for which this sum is equal to 0 or 1. Thus, if this value is fractional, then we create two branches, one where the sum is equal to 1, and one where it is equal to 0. And in each case, the dual variables γ_{ij} associated to the new constraints are appropriately transferred in the subproblem. But no arcs are deleted of the subproblem network.

It is very important to remark that it is not true for the arcs like $(0, i)$ and $(i, 0)$ for which $\sum_{k\in\Omega} b_{ij}^k x_k$ is still integer but might have a value higher than 1 specially if the demand of a client i is larger than the capacity of the vehicles.

The difference with the branching rules of Desrochers et al. (1992) is that here all the constraints in the Master Problem are introduced. Then, the reduced cost of a new column k_0 is equal to:

$$\bar{c}_{k_0} = c_{k_0} - \sum_{i=1}^{n} a_{ik_0}\alpha_i - \sum_{(i,j)\in A} b_{ij}^{k_0}\gamma_{ij} - d_{nv}$$

The rest of the subproblem does not change. What remains is to conduct computational testing of the above ideas for SDVRPTW.

6. CONCLUSION

This chapter is primarily about column generation solution methodology for vehicle routing problems with time windows. The motivation for this is that arc routing problems with time windows are "hard" and at the present time we do not know how to describe this kind of problems using traditional integer programming models. Thus, such problems have to be transformed into an appropriate node routing graphs and the solutions to these vehicle routing problems are then to be transformed back into their original arc routing settings.

The first three sections of this chapter address the issue of when a transformation of an arc routing problem into a node routing one is necessary and when, or, from a computational point of view, for what arc routing problems a transformation to node routing is appropriate. An additional motivation for such a transformation is the fact that for hard node routing problems, for instance the VRPTW, a numerous algorithms and extensive solution software have been developed over the past years which now could be used to solve hard arc routing problems in their node routing "image". The three sections, also attempt to provide a partial description and account of the different transformation schemes proposed over the years. In section 3, transformations of capacitated arc routing problems with and without split deliveries into an equivalent node routing problem are described in some detail.

The node routing problem analyzed in the second part of this chapter is the capacitated node routing problem with time windows. This is a very important problem in the node routing literature with numerous applications in real-life (Desrochers, et al. 1992). The solution methodology for this problem has been "traditionally" based on a set covering problem representation followed by an implementation of a suitable column generation solution procedure. For this problem, starting from section 4, in addition to the more traditional solutions based primarily on the work of Desrochers (1988), we describe a very recent solution approach based on the work of Guéguen et al. (1998), and Guéguen et al. (1999). In these two papers, Desrocher's (1988) dynamic programming scheme for solving the shortest path with resource constraints problem not necessarily as an elementary path has been modified to account for the fact that for some problems a solution has to be in the form of an elementary shortest path. We have reviewed these findings in detail. We have also examined the capacitated node routing problem with time windows and split deliveries and presented a column generation model for its solution. This new problem variant adds a number of nontrivial complications to an already hard problem. Following Mullaseril (1996) and Mullaseril and Dror (1997), by restricting the split options, a solution similar to the one developed for the non-split delivery problem is described.

In summary, this chapter describes solutions for node routing problems based on problems which originated in a real-life arc routing setting. In the process, we reviewed in detail the recent interesting developments in the column generation solution methodology.

References

[1] Assad, A.A. and B.L. Golden (1995). "Arc routing methods and applications", in M.O. Ball, T.L. Magnanti, C.L. Monma and G.L. Nemhauser (Eds.) *Network Routing.* Handbooks of Operations Research and Management Science 8. Amsterdam: North Holland, 375-483.

[2] Balinski, M.L. and Quandt, R.E., (1964). "On an integer program for a delivery problem", *Operations Research* 12, 300-304.

[3] Barnhart, C., Johnson, E.L., Nemhauser, G.L., Savelsbergh, M.W.P., and Vance, P.H., (1998). "Branch-and-price: Column generation for solving huge integer programs", *Operations Research*, 46, 316-329.

[4] Beasley, J.E. and Chrostofides, N., (1989). "An algorithm for the resource constrained shortest path problem", *Networks* 19, 379-394.

[5] Benavent, E., A. Corberán, and J.M. Sanchis (2000). "Linear programming based methods for solving arc routing problems", in Dror, M. ed., *Arc Routing: Theory, Solutions, and Applications*, Kluwer Academic Publishers (this book).

[6] Christofides, N., V. Campos, A. Corberán and E. Mota (1986) An algorithm for the rural postman problem on a directed graph. *Math. Prog. Study 26*, 155-166.

[7] Chvátal, V., (1983). Linear Programming, Freeman.

[8] Cook, S.A., (1971). "On the complexity of theorem-proving procedures", *Proc. 3rd Annual ACM Symp. Theory of Computing*, 151-158.

[9] Denardo, B.V. and B.L. Fox, (1979). "Shortest-route methods 1: Reaching, pruning, and buckets", *Operations Research* 27, 161-186.

[10] Derigs, U. (2000). "Matching: Arc routing and the solution connection", in M. Dror, ed., *Arc Routing: Theory, Solutions, and Applications*, Kluwer Academic Publishers (this book).

[11] Desrochers, M., (1988). "An algorithm for the shortest path problem with resource constraints", Cahiers du GERARD G-88-27, Ecole des Hautes Etudes Commerciales, Montreal, Canada H3T 1V6.

[12] Desrochers, M. and F. Soumis, (1988). "A generalized permanent labeling algorithm for shortest path problem with time windows", *INFOR* 26, 191-211.

[13] Desrochers, M., J. Desrosiers, and M.M. Solomon, (1992). "A new optimization algorithm for the vehicle routing with time windows", *Operations Research* 40, 342-354.

[14] Desrosiers, J., Y. Dumas, M.M. Solomon, and F. Soumis, (1995)."Time constrained routing and scheduling", in M.O. Ball, T.L. Magnanti, C.L. Monma and G.L. Nemhauser (Eds.) *Network Routing.* Handbooks of Operations Research and Management Science, 8. Amsterdam: North Holland, 35-139.

[15] Dror, M. (1994), "Note on the complexity of the shortest path models for column generation in the VRPTW", *Operations Research* 42, 977-978.

[16] Dror, M., (2000). "Arc Routing: Complexity and Approximability", in M. Dror, ed., *Arc Routing: Theory, Solutions, and Applications*, Kluwer Academic Publishers (this book).

[17] Dror M. and P. Trudeau, (1989). "Savings by split delivery routing", *Transportation Science* 23, 141-145.

[18] Dror, M. and P. Trudeau, (1990). "Split delivery routing",*Naval Research Logistics* 37, 383-402.

[19] Dror, M., G. Laporte, and P. Trudeau, (1994). "Exact solutions for split delivery routing", *Discrete Applied Mathematics* 50, 239-254.

[20] Dror, M. and J.M.Y. Leung, (1998). "Combinatorial optimization in a cattle yard: Feed distribution, vehicle scheduling, lot sizing, and dynamic pen assignment", in Gang Yu, ed., Industrial Applications of Combinatorial Optimization, Chapter 7, Kluwer Academic Publishers, 142-171.

[21] Dror, M., J.M.Y. Leung, and P.A. Mullaseril (2000). "Livestock Feed Distribution and Arc Traversal Problems", in M. Dror, ed., *Arc Routing: Theory, Solutions, and Applications*, Kluwer Academic Publishers (this book).

[22] Eilon, S., C.D.T. Watson-Gandy and N. Christofides, (1971). *Distribution Management: Mathematical Modeling and Practical Analysis*, Giffin, London.

[23] Eglese, R.W. and A.N. Letchford, (2000). "Polyhedral theory for arc routing problems", in M. Dror, ed., *Arc Routing: Theory, Solutions, and Applications*, Kluwer Academic Publishers (this book).

[24] Eiselt, H.A., M. Gendreau and G. Laporte (1995b). "Arc-routing problems, part 2: the rural postman problem", *Oper. Res.* 43, 399-414.

[25] Fleischner, H. (2000). "Traversing Graphs: The Eulerian and Hamiltonian Theme", in M. Dror, ed., *Arc Routing: Theory, Solutions, and Applications*, Kluwer Academic Publishers (this book).

[26] Frizzell, P.W. and J.W. Giffin, (1995). "The split delivery vehicle scheduling problem with time windows and grid network distances", *Computers & Operations Research* 22, 655-667.

[27] Garey, M.R. and D.S. Johnson (1979) *Computers and Intractability: a guide to the theory of NP-completeness.* San Fr.; Freeman.

[28] Ghiani, G. and G. Improta (2000). "An efficient transformation of the generalized vehicle routing problem", *European J. of Operational Research* 122, 11-17.

[29] Golden, B.L. and R.T. Wong (1981) Capacitated arc routing problems. *Networks 11*, 305-315.

[30] Guéguen, C., Dejax, P., M. Dror, and M. Gendreau, (1998). "An exact algorithm for the elementary shortest path problem with resource constraints", Technical Report 98-04-A, Laboratoire Productique Logistique, Ecole Central Paris, April 1998 (revised in July 1999, also with D. Feillet as a co-author).

[31] Guéguen, C., P. Dejax, and M. Gendreau, (1999). "The selective and prize collecting vehicle routing problems with time windows"", Technical Report, Laboratoire Productique Logistique.

[32] Haouari M., P. Dejax and M Desrochers (1991). "Modelling and solving complex vehicle routing problems using column generation", Working Paper, LEIS, Ecole Centrale, Paris.

[33] Hertz, A. and M. Mittaz, (2000). "Heuristic algorithms, in M. Dror, ed., *Arc Routing: Theory, Solutions, and Applications*, Kluwer Academic Publishers (this book).

[34] Johnson, D.S. and C.H. Papadimitriou (1985). "Computational complexity", in The Traveling Salesman Problem, E.L. Lawler, J.K. Lenstra, A.H.G. Rinnooy Kan, and D.B. Shmoys (eds.), John Wiley & Sons Ltd., 251-305.

[35] Johnson, E.L., (2000). "Chinese postman and Euler tour problems in bi-directional graphs", in M. Dror, ed., *Arc Routing: Theory, Solutions, and Applications*, Kluwer Academic Publishers (this book).

[36] Laporte G., (1997). "Modeling and solving several classes of arc routing problems as traveling salesman problems",*Comput. Ops. Res.* 24, 1057-1061.

[37] Lenstra, J.K. and A.H.G. Rinnooy-Kan (1976). "On general routing problems", *Networks* 6, 273-280.

[38] Mullaseril, P.A., (1996). "Capacitated Rural Postman Problem with Time Windows and Split Delivery", Ph.D. Thesis, MIS Department, University of Arizona, Tucson, Arizona 85721.

[39] Mullaseril, P.A. and M. Dror (1997). "A set covering approach for directed node and arc routing problems with split delivery and time windows", Working Paper, MIS Department, University of Arizona, Tucson, Arizona 85721 (submitted for publication).

[40] Mullaseril, P.A., M. Dror and J.Y.M. Leung (1997). "Split delivery routing heuristic in livestock feed distribution", *J. of Operational Research Society* 19, 61-72.

[41] Nobert, Y. and J.-C. Picard (1996). "An optimal algorithm for the mixed Chinese postman problem", *Networks 27*, 95-108.

[42] Noon, C.E. and J.C. Bean (1993). "An efficient transformation of the generalized traveling salesman problem", *INFOR* 31, 39-44.

[43] Papadimitriou, C.H. (1976) On the complexity of edge traversing. *J. of the A.C.M.* 23, 544-554.

[44] Pearn, W.L., A. Assad and B.L. Golden (1987). "Transforming arc routing into node routing problems", *Comput. Oper. Res.* 14, 285-288.

[45] Sierksma G. and G.A. Tijssen, (1998). "Routing helicopters for crew exchanges on off-shore locations", *Annals of Operations Research* 76, 261–286.

[46] Solomon, M.M., (1983). "Vehicle Routing and Scheduling with Time Window Constraints: Models and Algorithms", Ph.D. Thesis, Department of Decision Sciences, Wharton School, University of Pennsylvania, Philadelphia, PA., U.S.A.

Chapter 9

HEURISTIC ALGORITHMS

Alain Hertz
Ecole Polytechnique Fedéralé de Lausanne

Michel Mittaz
Ecole Polytechnique Fedéralé de Lausanne

1. Introduction 327
2. Heuristics for Uncapacitated Arc Routing Problems 329
   2.1 The Chinese Postman Problem 329
      2.1.1 The Undirected Chinese Postman Problem 329
      2.1.2 The Directed Chinese Postman Problem 331
      2.1.3 The Mixed Chinese Postman Problem 332
      2.1.4 The Windy Postman Problem 336
   2.2 The Rural Postman Problem 338
      2.2.1 The Undirected Rural Postman Problem 338
      2.2.2 The Directed Rural Postman Problem 340
      2.2.3 The Mixed Rural Postman Problem 344
      2.2.4 The Stacker Crane Problem 349
      2.2.5 Additional Algorithmic Tools 355
3. Heuristics for Capacitated Arc Routing Problems 361
   3.1 Simple Constructive Methods 361
   3.2 Two-Phase Constructive Methods 373
      3.2.1 Route First-Cluster Second Heuristics 373
      3.2.2 Cluster First-Route Second Heuristics 377
   3.3 Meta-Heuristics for the CARP 379
4. Conclusion 383

1. INTRODUCTION

Arc routing problems (ARPs) arise naturally in contexts where streets require treatments, or customers located along road must be served. Most ARPs are $\mathcal{NP}$-hard and of large scale. They can therefore rarely be tackled by means of exact solution methods. This certainly explains why numerous heuristic algorithms have been developed for the solutions

of ARPs. We will try in this chapter to give an overview of the main approaches and tools that are commonly used in heuristic procedures for ARPs. We do not aim to make an exhaustive survey of all articles that are devoted to this topic. We seek, however, to point out that the ingredients used in the heuristic methods strongly depend on the structure of the underlying graph.

ARPs are formally defined over a graph $G = (V, E \cup A)$, where V is the vertex set, E is the edge set, and A is the arc set. The traversal cost c_{ij} of an edge or arc (v_i, v_j) in $E \cup A$ is supposed to be non-negative and is also called *length* of (v_i, v_j). In case of an edge (v_i, v_j) in E, it is usually assumed that $c_{ij} = c_{ji}$. G is called *directed* if E is empty, *undirected* if A is empty, and *mixed* if both E and A are non-empty. A chain or path in G is represented by a vector of the form $(v_{i_1}, v_{i_2}, ..., v_{i_t})$, where $(v_{i_j}, v_{i_{j+1}})$ belongs to $E \cup A$ for $j = 1, ..., t-1$. A *tour* T, or cycle in G is a path $(v_{i_1}, v_{i_2}, ..., v_{i_t})$ with $v_{i_t} = v_{i_1}$.

The *in-degree* d_i^- of a vertex v_i is the number of arcs entering v_i, its *out-degree* d_i^+ is the number of out-going arcs, and its *degree* d_i is the total number of edges and arcs incident to v_i. A graph is called *even* if each vertex has an even degree. It is *symmetric* if the in-degree and the out-degree of each vertex are equal. A graph $G = (V, E \cup A)$ is *balanced* if, given any subset S of vertices, the difference between the number of directed arcs from S to $V \setminus S$ and the number of directed arcs from $V \setminus S$ to S is less than or equal to the number of undirected edges joining S and $V \setminus S$.

An *Eulerian* tour in a graph $G = (V, E \cup A)$ is a tour which contains every edge and arc exactly once. G is called Eulerian (or unicursal) if it possesses an Eulerian tour. A subset $R \subseteq E \cup A$ of edges and arcs are said to be *required*, i.e. they must be *serviced* or *covered* by a vehicle. A covering tour for R is a tour that traverses all edges and arcs of R at least once. In the Chinese Postman Problem (CPP), one seeks a minimum cost *covering tour* for $E \cup A$. When R is a proper subset of $E \cup A$, the problem of finding a minimum cost covering tour for R becomes a Rural Postman Problem (RPP). Extensions of these classical problems are obtained by imposing capacity constraints. In the Capacitated Arc Routing Problem (CARP), each required edge or arc (v_i, v_j) in R has a non-negative demand q_{ij}, and all arcs and edges in R must be covered by a fleet of identical vehicles of capacity Q.

This chapter is organized as follows. Section 2 is devoted to uncapacitated ARPs, while Section 3 deals with capacitated ones. We describe in Section 2.1 heuristics for the CPP. The techniques used to solve the

CPP depend on the structure of the underlying graph G which can be directed, undirected or mixed. We also analyze a related problem, called the windy postman problem (WPP), in which G is undirected although the cost of traversing an edge depends on the direction of traverse. In Section 2.2, we focus on the RPP and first analyze the directed, undirected and mixed cases. We then consider the Stacker Crane Problem (SCP) where some edges must be traversed at least once in a given direction, but can also be traversed as often as needed in the opposite direction. We conclude Section 2.2 by the description of recent algorithmic tools that have been developed for the solution of RPPs.

Section 3 is devoted to the CARP. We first describe simple constructive heuristics and then analyze two-phase constructive methods which belong to two different categories: cluster first-route second and route first-cluster second heuristics. We finally review adaptations of metaheuristics for the solution of the CARP (e.g., Simulated Annealing and Tabu Search). A conclusion follows in Section 4, where examples are given to show that the described classical models can easily be extended to deal with real-life problems.

2. HEURISTICS FOR UNCAPACITATED ARC ROUTING PROBLEMS

2.1. THE CHINESE POSTMAN PROBLEM

Consider a graph $G = (V, E \cup A)$. The chinese postman problem (CPP) is to determine a tour of minimum cost that covers all edges and arcs in $E \cup A$. The CPP is polynomially solvable if G is directed or undirected. However, both the mixed CPP and the WPP are $\mathcal{NP}$-hard. We describe in this section algorithms for the solution of these four problems. The procedures described for the solution of the directed and undirected CPP are exact methods that are often used as basic ingredients for more complex ARPs. We have therefore decided to include them in this chapter.

2.1.1 The undirected chinese postman problem.

An *Eulerian* tour in an undirected graph $G = (V, E)$ is a tour which contains every edge exactly once. Finding an Eulerian tour in an Eulerian undirected graph is an easy problem that can be solved, for example, by means of the $\mathcal{O}(|E|)$ algorithm described in [17]. This algorithm is outlined below. Other procedures are described in chapter 10 of [21].

Algorithm EULERIAN-CYCLE

INPUT: An Eulerian undirected graph $G = (V, E)$.

OUTPUT: An Eulerian tour T on G.

Step 1. Determine a tour T covering some edges in E. If T covers all edges in T then stop.

Step 2. Consider any vertex v on T incident to an edge not on T. Form a second tour T'containing v and such that T and T' are edge-disjoint.

Step 3. Let e_1, e_2 be two consecutive edges incident to v on T and let e_3, e_4 be two consecutive edges incident to v on T'. Merge T and T' into a tour T'', starting at v with e_1, following T until it meets v by way of e_2, then continue on T', starting with e_3 until v is met by way of e_4.

Step 4. Set $T := T''$. If T covers E then stop, else return to Step 2.

It is well-known that an undirected graph is Eulerian if and only if it is even. Assume that edges are added to a non Eulerian graph G in order to create an even graph G'. An Eulerian tour in G' corresponds to a chinese postman tour in G. In order to solve the CPP to optimality, the edges added to G must be of least total cost. As shown in [17], these edges can be determined by solving a perfect matching problem on a graph whose vertices are the odd-degree vertices of G. The algorithm that solves the CPP on an undirected graph can be described as follows.

Algorithm UCPP

INPUT: An undirected graph $G = (V, E)$.

OUTPUT: A minimum cost chinese postman tour T on G.

Step 1. If G is even then determine a chinese postman tour T by means of the EULERIAN-CYCLE algorithm, and stop.

Step 2. Determine the set V_0 of odd degree vertices in G. Construct a complete graph G_0 with vertex set V_0. Define the cost of an edge (v_i, v_j) in G_0 as the length of the shortest chain SP_{ij} between v_i and v_j in G.

Step 3. Find a minimum cost perfect matching in G_0.

Step 4. Set $G' := G$. For each edge (v_i, v_j) in the optimal matching, add to G' all edges that lie on SP_{ij}.

Step 5. Determine an Eulerian tour T on G' by means of the EULERIAN-CYCLE algorithm, and stop. T corresponds to an optimal chinese postman tour on G.

Step 3 of the above algorithm can be solved by means of the $\mathcal{O}(|V|^3)$ algorithm proposed in [32]. More elaborate algorithms of lower complexity can also be used (see, e.g., [24],[13]).

2.1.2 The directed chinese postman problem.

The directed CPP is also polynomially solvable. We first observe that a solution exists if and only if G is strongly connected. The techniques used to solve the directed CPP are similar to those describe for the undirected case. Assume first that G is symmetric, i.e., each vertex in G has equal in- and out-degrees. In such a case, the chinese postman tour correspond to an Eulerian circuit in G. Such a circuit can be obtained by means of the following algorithm described in [1] and [17].

Algorithm EULERIAN-CIRCUIT

INPUT: A symmetric directed graph $G = (V, A)$.

OUTPUT: An Eulerian tour T on G.

Step 1. Construct a spanning anti-arborescence rooted at a vertex v in G.

Step 2. Order and label the arcs out-going from v in an arbitrary fashion. Order and label the arcs out of any other vertex in an arbitrary fashion, as long as the last arc is the arc used in the arborescence.

Step 3. Construct a tour T that starts at v with the lowest labelled arc out-going from v. Then continue as follows: whenever a vertex is entered, leave it by the arc not yet traversed having the lowest label.

If G is not symmetric, then copies of some arcs must be added to G so that the augmented graph G' becomes symmetric. An Eulerian circuit in G' corresponds to a chinese postman tour in G. The symmetric graph G' can be obtained from G by solving a transportation problem described in [5]. This solution method for the directed CPP has a complexity $\mathcal{O}(|A||V|^2)$ and can be summarized as follows.

Algorithm DCPP

INPUT: A directed graph $G = (V, A)$.

OUTPUT: A minimum cost chinese postman tour T on G.

Step 1. If G is symmetric then determine a chinese postman tour T by means of the EULERIAN-CIRCUIT algorithm, and stop.

Step 2. Compute $b_i = d_i^- - d_i^+$ for each vertex v_i in V. Define $I = \{v_i \in V |\ b_i > 0\}$ and $J = \{v_j \in V |\ b_j < 0\}$.

Step 3. Solve the transportation problem in which each vertex $v_i \in I$ has supply b_i, each vertex $v_j \in J$ has demand $|b_j|$, and the cost of

the arc $(v_i, v_j) \in I \times J$ is equal to the length of the shortest path SP_{ij} linking v_i to v_j in G.

Step 4. Let x_{ij} be the flow on arc (v_i, v_j) in the optimal solution of the transportation problem. Set $G' := G$. For each pair $(v_i, v_j) \in I \times J$, add to G' x_{ij} copies of each arc that lies on SP_{ij}.

Step 5. Determine an Eulerian tour T in G' by means of the EULERIAN-CIRCUIT algorithm, and stop. T corresponds to an optimal chinese postman tour on G.

Other algorithms have been proposed for the solution of the directed CPP. For example, Edmonds and Johnson [17] propose to use an $\mathcal{O}(|V|^3)$ algorithm which determines a minimum cost flow on an extended graph. Lin and Zhao [34] have shown that the use of the Complementary Slackness Theorem makes it possible to solve the directed CPP directly on the original graph. Their algorithm requires $(|A| - |V| + 1)$ calls to an $\mathcal{O}(|V|^2)$ procedure. For sparse graphs, this complexity is better than the $\mathcal{O}(|V|^3)$ and the $\mathcal{O}(|A||V|^2)$ algorithms mentioned above.

2.1.3 The mixed chinese postman problem.

Consider a mixed graph $G = (V, E \cup A)$. It is well-known that G is Eulerian if and only if it is even and balanced. Notice that if G is even and symmetric, then it is also balanced. The mixed CPP is $\mathcal{NP}$-hard. It is however polynomially solvable if G is even. Consider first the case where G is even and balanced. An optimal tour can easily be determined by means of a three step algorithm that first directs some edges in E in order to get a symmetric graph G', then directs the edges of G' for getting a symmetric directed graph G'', and finally determines an Eulerian tour in G''. This algorithm can be described as follows.

Algorithm EVEN-BALANCED-MCPP

INPUT: An even balanced mixed graph $G = (V, E \cup A)$.

OUTPUT: An Eulerian tour T on G.

(*construction of a symmetric graph* G')

Step 1. Construct a directed graph $H = (V, A \cup A')$ by replacing each edge in E with a pair of opposite directed arcs. Assign to each arc of $A \cup A'$ an upper bound of 1. Also, assign to each arc of A a lower bound of 1 and to each arc of A' a lower bound of 0. Determine a compatible flow in H. Let x_{ij} be the flow on arc (v_i, v_j). Set $G' := G$ and consider each undirected edge (v_i, v_j): if $x_{ij} = 1$ and $x_{ji} = 0$, then orient this edge from v_i to v_j.

(*construction of a symmetric directed graph* G'')

Step 2. Set $G'' := G'$. If G'' is a directed graph, then go to Step 3. Construct an Eulerian directed tour on each connected component of the partial graph of G'' induced by its undirected edges. Orient the edges of G'' as in the Eulerian directed tours.

(*construction of an Eulerian tour*)

Step 3. Construct an Eulerian tour T on G' by means of the EULERIAN-CIRCUIT algorithm. T corresponds to an Eulerian tour on G.

If the mixed graph $G = (V, E \cup A)$ is even but not necessarily balanced, then the problem can still be solved in polynomial time by means of an algorithm proposed by Edmonds and Johnson [17] (see also [23],[37], [38]). This algorithm can be described as follows.

Algorithm EVEN-MCPP

INPUT: An even mixed graph $G = (V, E \cup A)$.

OUTPUT: A minimum cost chinese postman tour T on G.

Step 1. Construct a directed graph $G_1 = (V, A \cup A_1)$ from G by arbitrarily orienting each edge in E (i.e., replace the edge set E by an arc set A_1). Compute the difference b_i between the in-degree and the out-degree of each vertex v_i in G_1. If all $b_i = 0$, set $G_3 := G_1$ and go to Step 3.

Step 2. Construct a network $G_2 = (V, A \cup A_1 \cup A_2 \cup A_3,)$ such that $A_1 \cup A_2$ contains arcs of opposite direction and cost c_{ij} for each edge (v_i, v_j) in E, and A_3 contains an arc (v_j, v_i) with zero cost for each arc (v_i, v_j) in A_1. Assign an infinite capacity on each arc in $A \cup A_1 \cup A_2$ and a unit capacity on each arc in A_3. Attach a supply of b_i units to each vertex with $b_i > 0$ and a demand of $|b_i|$ units to each vertex with $b_i < 0$. Find a minimum cost flow in G_2 that satisfies the demands. Let x_{ij} be the flow on an arc (v_i, v_j) in $A \cup A_1 \cup A_2 \cup A_3$. Set $G_3 := G$. For each arc (v_i, v_j) in A_3 do the following: if $x_{ij} = 2$ then orient the edge (v_i, v_j) in G_3 from v_i to v_j; if $x_{ij} = 0$ then orient (v_i, v_j) from v_j to v_i. Augment G_3 by adding x_{ij} copies of each arc (v_i, v_j) in $A \cup A_1 \cup A_2$.

Step 3. Determine an Eulerian tour T on G_3 by means of the EULERIAN-CIRCUIT algorithm. T corresponds to an optimal chinese postman tour on G.

Different descriptions of this algorithm can be found in [17],[23],[37], [38]. It is not difficult to prove that, at Step 2, there is a minimum cost flow in G_2 that uses each arc an even number of times. Frederickson [23] has however pointed out that there may also exist an optimal flow in G_2

that uses some arcs an odd number of times. For example, it may happen that $x_{ij} = 1$ for an arc (v_i, v_j) in A_3, in which case the corresponding edge (v_i, v_j) in G is left non-oriented. With such an optimal flow in G_2, the graph G_3 obtained at the end of Step 2 is not necessarily even. The example depicted in Figures 9.1*(a)* and 9.1*(b)* illustrates this fact. Frederickson [23] has therefore developed a procedure, called EVENPARITY, for repairing this problem. One may also simply avoid such a situation by determining a minimum cost flow on a network G'_2 (instead of G_2) in which each arc in A_3 has a capacity of 1 (instead of 2), each vertex with $b_i > 0$ has a supply of $\frac{b_i}{2}$ units (instead of b_i), and each vertex with $b_i < 0$ has a demand of $\frac{|b_i|}{2}$ units (instead of $|b_i|$). Given any optimal flow in G'_2, an edge (v_i, v_j) is oriented from v_i to v_j if the flow x_{ij} on the arc (v_i, v_j) in A_3 is equal to 1, and it is oriented from v_j to v_i if $x_{ij} = 0$. Moveover, $2x_{ij}$ copies (instead of x_{ij}) of each arc (v_i, v_j) in $A \cup A_1 \cup A_2$ are added to G in order to obtain G_3. This is illustrated in Figure 9.1*(c)*.

We now study the mixed CPP for non-even mixed graphs G. In this case, the problem is $\mathcal{NP}$-hard [39]. We describe two possible heuristics that extend the techniques used for even graphs. Both heuristics have been developed by Frederickson [23]. The first heuristic, called MCPP1 transforms G into an even graph G' by replicating some edges, and then uses the EVEN-MCPP algorithm on G' in order to get a chinese postman tour in G. The second heuristic, called MCPP2, transforms G into a symmetric graph G' and then augment G' by adding some edges in order to get an even symmetric graph. A chinese postman tour in G is determined by applying Steps 2 and 3 of the EVEN-BALANCED-MCPP algorithm on G'.

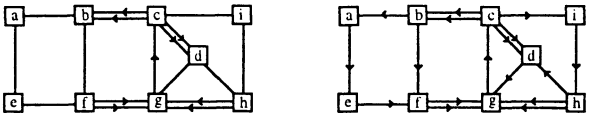
Algorithm MCPP1

INPUT: A mixed graph $G = (V, E \cup A)$.

OUTPUT: A chinese postman tour T on G.

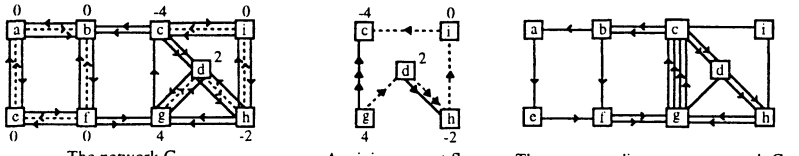
Step 1. Determine the set V_0 of odd degree vertices in G. Construct a complete graph G_0 with vertex set V_0. Define the cost of an edge (v_i, v_j) in G_0 as the length of the shortest chain SP_{ij} between v_i and v_j in G (irrespectively of arc directions). Find a minimum cost perfect matching in G_0, and set $G' := G$. For each edge (v_i, v_j) in the optimal matching, augment G' by adding a copy of each edge and arc that lies on SP_{ij}.

Step 2. Determine an optimal chinese postman tour T on G' by means of the EVEN-MCPP algorithm. T corresponds to a chinese postman tour on G.



The original even non-balanced graph G. All arcs have unit cost. Each edge has been arbitrarily oriented.

(a)
Step 1 of the EVEN-MCPP algorithm.

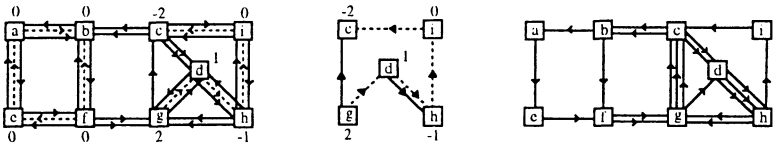


The network G_2. The arcs in A_3 have zero cost, a capacity of 2, and are represented by dotted lines. The other arcs have unit cost and infinite capacity. Demands and supplies are shown on the vertices.

A minimum cost flow in G_2.

The corresponding non-even graph G_3.

(b)
A problem raised by Frederickson [Fre79] at Step 2 of the EVEN-MCPP algorithm.



The network G_2'. The arcs in A_3 have zero cost, unit capacity, and are represented by dotted lines. The other arcs have unit cost and infinite capacity. Demands and supplies are shown on the vertices.

A minimum cost flow in G_2'.

The corresponding even and symmetric graph G_3.

(c)
A modified version of Step 2 of the EVEN-MCPP algorithm.

Figure 9.1

Algorithm MCPP2

INPUT: A mixed graph $G = (V, E \cup A)$.

OUTPUT: A chinese postman tour T on G.

Step 1. Apply steps 1 and 2 of the EVEN-MCPP algorithm and let $G' = (V, E' \cup A')$ be the resulting symmetric graph.

Step 2. Determine the set V_0 of odd degree vertices in the partial subgraph $H' = (V, E')$ of G'. Construct a complete graph G_0 with vertex set V_0. Define the cost of an edge (v_i, v_j) in G_0 as the length of the shortest chain SP_{ij} between v_i and v_j in the subgraph $H = (V, E)$ of G. Find a minimum cost perfect matching

in G_0, and set $G'' := G'$. For each edge (v_i, v_j) in the optimal matching, augment G'' by adding a copy of each edge that lies on SP_{ij}.

Step 3. Determine an optimal chinese postman tour T on G'' by means of Steps 2 and 3 of the EVEN-BALANCED-MCPP algorithm. T corresponds to a chinese postman tour on G.

As shown in Frederickson [23], both algorithms MCPP1 and MCPP2 have a worst case ratio of 2, and these bounds are sharp. However, if the two heuristics are applied in succession, the worst case ratio goes down to 5/3. To our knowledge, it has not been proved that this better bound is sharp. The combined algorithm can simply be described as follows.

Algorithm COMBINED-MCPP

INPUT: A mixed graph $G = (V, E \cup A)$.

OUTPUT: A chinese postman tour on G.

Step 1. Determine two chinese postman tours T and T' on G by means of the MCPP1 and MCPP2 algorithms, respectively.

Step 2. Select the tour of smallest cost among T and T'.

2.1.4 The windy postman problem.

The windy postman problem (WPP) consists of determining a least cost traversal of all edges of an undirected graph $G = (V, E)$ in which the cost of traversing an edge depends on the direction of travel. This problem was first introduced by Minieka [38], and Guan [29] has shown that it is $\mathcal{NP}$-hard except for some special cases which can be solved in polynomial time. For example, Guan [29] has shown that if the two possible orientations of each cycle in G have the same length, then the WPP can be solved by means of the following algorithm.

Algorithm GUAN-WPP

INPUT: An undirected graph $G = (V, E)$ where c_{ij} may differ from c_{ji}, and in which the two possible orientations of each cycle have the same length.

OUTPUT: A minimum cost windy postman tour on G.

Step 1. Consider the graph $G' = (V, E)$ in which the cost c'_{ij} of an edge (v_i, v_j) is set equal to $\frac{c_{ij}+c_{ji}}{2}$.

Step 2. Determine an optimal chinese postman tour T on G' by means of the UCPP algorithm described in Section 2.1.1. T corresponds to an optimal windy postman tour on G.

Another sufficient condition for polynomial solvability is for G to be Eulerian. For this case, [49] has proved that the EVEN-MCPP described in Section 2.1.3 can be slightly modified in order to solve the WPP to optimality. This algorithm can be described as follows.

Algorithm EULERIAN-WPP

INPUT: An Eulerian undirected graph $G = (V, E)$ in which c_{ij} may differ from c_{ji}.

OUTPUT: A minimum cost windy postman tour on G.

Step 1. Apply Step 1 of EVEN-MCPP (see Section 2.1.3) by orienting an edge (v_i, v_j) in E from v_i to v_j if $c_{ij} < c_{ji}$, and from v_j to v_i otherwise.

Step 2. Apply Step 2 of EVEN-MCPP (see Section 2.1.3) by assigning a cost $\frac{c_{ji}-c_{ij}}{2}$ (instead of zero) to each arc (v_j, v_i) in A_3.

Step 3. Apply Step 3 of EVEN-MCPP in order to get a tour T. This tour corresponds to an optimal windy postman tour on G.

Win [49] has also suggested to take advantage of the easiness of the WPP on Eulerian graphs for the design of a heuristic algorithm for the general WPP. This algorithm can be described as follows.

Algorithm WPP

INPUT: An undirected graph $G = (V, E)$ in which c_{ij} may differ from c_{ji}.

OUTPUT: A windy postman tour in G.

Step 1. If G is Eulerian then determine an optimal windy postman tour by means of the EULERIAN-WPP algorithm, and stop.

Step 2. Consider the graph $G' = (V, E)$ in which the cost c'_{ij} of an edge (v_i, v_j) is set equal to $\frac{c_{ij}+c_{ji}}{2}$.

Step 3. Determine the set V_0 of odd degree vertices in G'. Construct a complete graph G_0 with vertex set V_0. Define the cost of an edge (v_i, v_j) in G_0 as the length of the shortest chain SP_{ij} between v_i and v_j in G'. Find a minimum cost perfect matching in G_0, and set $G'' := G$. For each edge (v_i, v_j) in the optimal matching, augment G'' by adding a copy of each edge that lies on SP_{ij}. (The resulting graph G'' is Eulerian and has the original costs as in G).

Step 4. Determine an optimal windy postman tour T on G'' by means of the EULERIAN-WPP algorithm. T corresponds to a windy postman tour on G.

Win [49] has shown that the above algorithm has a worst case ratio of 2, and that this bound is sharp.

2.2. THE RURAL POSTMAN PROBLEM

Consider a graph $G = (V, E \cup A)$, and let R be a subset of *required* arcs and edges in G. The rural postman problem (RPP) is to determine a minimum cost covering tour for R. The set of vertices in G incident to at least one arc or edge in R is denoted V_R, and each vertex in V_R is said to be *required.* The *required subgraph* $G_R = (V_R, R)$ is defined as the partial graph of G induced by R. The RPP is polynomially solvable if G is directed or undirected and G_R is connected. However, in general, the RPP is $\mathcal{NP}$-hard [33]. We describe in this section algorithms for the solution of the directed, undirected and mixed RPP. We also study the stacker crane problem (SCP) which can be viewed as a particular directed RPP. Finally, we describe new tools that have recently been developed for the solution of RPPs.

2.2.1 The undirected rural postman problem.

Consider an undirected graph $G = (V, E)$, and let R be its subset of required edges. Since G may contain many non-required vertices, the RPP is often solved on a simplified graph $G_S = (V_R, R \cup E_S)$ derived from G_R as follows. An edge (v_i, v_j) is first included in E_S for each $v_i, v_j \in V_R$. The cost of an added edge (v_i, v_j) in E_S is set equal to the length of a shortest chain between v_i and v_j in G (while the edges in R have the same cost as in G). The set E_S of added edges is then reduced by eliminating

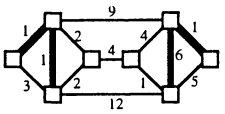
(*a*) all edges $(v_i, v_j) \in E_S$ for which $c_{ij} = c_{ik} + c_{kj}$ for some $v_k \in V_R$, and

(*b*) one of two parallel edges if they have the same cost.

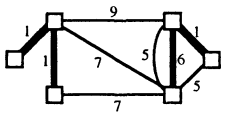
A rural postman tour T is then determined on G_S, and a tour on G is obtained by replacing each non-required edge on T by its corresponding shortest chain in G.

To illustrate, consider the graph G shown in Figure 9.2*(a)*, where the edges of R are shown in bold lines and the numbers correspond to edge costs. The simplified graph G_S is represented in Figure 9.2*(b)*, and a rural postman tour covering R is shown on G_S and on G in Figures 9.2*(c)* and 9.2*(d)*, respectively.

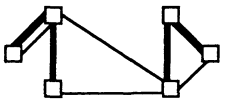
From now on, we will assume that the considered graph G has already been simplified (hence $V = V_R$). Consider first the case where the required subgraph G_R is connected. An optimal postman tour in G can easily determined by means of a procedure that works along the



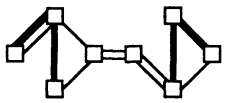
(a)
A graph G.



(b)
Its corresponding simplified graph G_S.



(c)
A rural postman tour on G_S.



(d)
The corresponding rural postman tour on G.

Figure 9.2

lines of the UCPP procedure described in Section 2.1.1. A polynomial algorithm is described below.

Algorithm CONNECTED-URPP

INPUT: An undirected graph $G = (V, E)$, and a subset R of required edges such that $G_R = (V, R)$ is connected.

OUTPUT: A minimum cost rural postman tour on G.

Step 1. Set $G' := G_R$. If G' is even then go to Step 3.

Step 2. Determine the set V_0 of odd degree vertices in G'. Construct a complete graph G_0 with vertex set V_0. Define the cost of an edge (v_i, v_j) in G_0 as the length of the shortest chain SP_{ij} between v_i and v_j in G. Find a minimum cost perfect matching in G_0. For each edge (v_i, v_j) in the optimal matching, add to G' all edges that lie on SP_{ij}.

Step 3. Determine an Eulerian tour T on G' by means of the EULERIAN-CYCLE algorithm (see Section 2.1.1). T is an optimal rural postman tour on G.

Consider now the case where the required subgraph G_R is not connected. Frederickson [23] has suggested to solve the undirected RPP by means of a procedure that works along the lines of the Christofides heuristic for the undirected traveling salesman problem [8]. This procedure, which has a worst case ratio of 3/2 [23], can be described as follows.

Algorithm URPP

INPUT: An undirected graph $G = (V, E)$ and a set R of required edges.

OUTPUT: A rural postman tour on G.

Step 1. Set $G' := G_R$, and let $C_1, ..., C_c$ denote the connected components of G_R. If $c = 1$ then go to Step 3. Construct a complete undirected graph G'' with vertex set $\{1, ..., c\}$, and in which the cost of an edge (p, q) is equal to the length of the shortest chain between a vertex of C_p and a vertex of C_q in G.

Step 2. Determine a minimum cost spanning tree in G''. For each edge (p, q) in the optimal spanning tree, add to G' all edges that lie on the shortest chain between a vertex of C_p and a vertex of C_q in G.

Step 3. Apply Steps 2 and 3 of the CONNECTED-URPP algorithm in order to obtain an Eulerian tour T on G'. T corresponds to a rural postman tour on G.

2.2.2 The directed rural postman problem.

Consider a directed graph $G = (V, A)$. As for the undirected case, the directed RPP can be solved on a simplified graph $G_S = (V_R, R \cup A_S)$ derived from G_R as follows. Two arcs (v_i, v_j) and (v_j, v_i) are first included in A_S for each $v_i, v_j \in V_R$. The cost of an added arc (v_i, v_j) in A_S is set equal to the length of a shortest path linking v_i to v_j in G (while the arcs in R have the same cost as in G). The set A_S of added arc is then reduced by eliminating

(a) all arcs $(v_i, v_j) \in A_S$ for which $c_{ij} = c_{ik} + c_{kj}$ for some $v_k \in V_R$, and

(b) one of two parallel arcs if they have the same cost and the same orientation.

A rural postman tour T is then determined on G_S, and a tour on G is obtained by replacing each non-required arc on T by its corresponding shortest path in G.

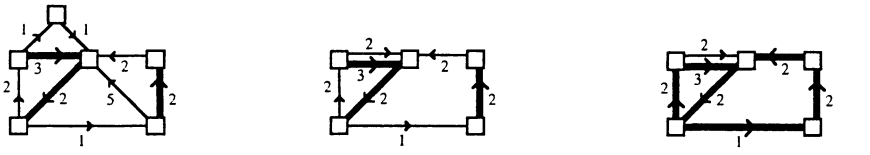
Christofides et al. [9] have observed that some non-required arcs in A_S can be transferred into the set R of required arcs without modifying the optimal value of the directed RPP. Such a transfer of an arc $(v_i, v_j) \in A_S$ can be performed if at least one of the three following properties holds:

(i) (v_i, v_j) is the unique arc out-going vertex v_i in G_S;

(ii) (v_i, v_j) is the unique arc entering vertex v_j in G_S;

(iii) there is a partition of V_R into two sets W_1 and W_2, such that (v_i, v_j) is the unique arc directed from W_1 towards W_2.

To illustrate, consider the graph G shown in Figure 9.3*(a)*, where the arcs of R are shown in bold lines and the numbers correspond to arc costs. A first simplification leads to the graph G_S represented in Figure 9.3*(b)*. This graph can be further modified by transferring three arcs from A_S to R. The resulting graph is shown in Figure 9.3*(c)*. Notice that the required subgraph of this latter graph is connected, and an optimal rural postman tour can therefore be obtained in polynomial time. Such a situation was not apparent when dealing with the original graph G since G_R is not connected.



(a)
A graph G.

(b)
Its corresponding simplified graph G_S.

(c)
The graph obtained by transferring non-required arcs into R.

Figure 9.3

From now on, we will assume that the considered graph G has already been simplified, using the above mentioned tools. Suppose first that the required subgraph G_R is connected. In this case, the directed RPP can be solved in polynomial time by means of an algorithm that works along the lines of the DCPP algorithm described in Section 2.1.2 for the directed Chinese Postman Problem. The algorithm can be described as follows.

Algorithm CONNECTED-DRPP

INPUT: A directed graph $G = (V, A)$ and a set R of required arcs such that $G_R = (V, R)$ is connected.

OUTPUT: A minimum cost rural postman tour T on G.

Step 1. Set $G' := G_R$. If G' is symmetric then go to Step 5.

Step 2. Compute $b_i = d_i^- - d_i^+$ for each vertex v_i in G'. Define $I = \{v_i \in V |\ b_i > 0\}$ and $J = \{v_j \in V |\ b_j < 0\}$.

Step 3. Solve the transportation problem in which each vertex $v_i \in I$ has supply b_i, each vertex $v_j \in J$ has demand $|b_j|$, and the cost of the arc $(v_i, v_j) \in I \times J$ is equal to the length of the shortest path SP_{ij} linking v_i to v_j in G.

Step 4. Let x_{ij} be the flow on arc (v_i, v_j) in the solution of the transportation problem. For each pair $(v_i, v_j) \in I \times J$, add to $G' \in x_{ij}$ copies of each arc that lies on SP_{ij}.

Step 5. Determine an Eulerian tour T on G' by means of the EULERIAN-CIRCUIT algorithm (see Section 2.1.2). T is an optimal rural postman tour on G.

Consider now the case where the required subgraph G_R is not connected. Two main heuristic approaches have been proposed to solve the directed RPP. The first approach, which will be denoted S&C, starts by extending G_R to a symmetric graph G', and then add arcs to G' in order to get a symmetric, strongly connected graph. The second approach, called C&S, first extends G_R to a connected graph G', and then add arcs to G' in order to get a symmetric graph. An algorithm following the S&C approach has been suggested by Ball and Magazine [3] and is described below.

Algorithm S&C-DRPP

INPUT: A directed graph $G = (V, A)$, and a set R of required arcs.

OUTPUT: A rural postman tour T on G.

Step 1. Set $G' := G_R$. If G' is symmetric then go to Step 3.

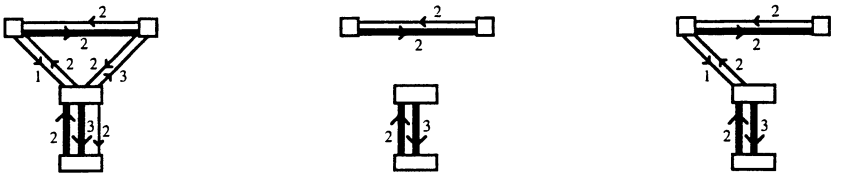
Step 2. Apply Steps 2, 3 and 4 of the CONNECTED-DRPP algorithm in order to transform G' into a symmetric graph.

Step 3. If G' is connected then go to Step 4. Let $C_1, ..., C_c$ be the connected components of G', and let L_{ij} denote the length of a shortest path linking v_i to v_j in G. Construct a complete undirected graph G' with vertex set $\{1, ..., c\}$, and with edge costs $c'_{pq} = \min\{L_{ij} + L_{ji} |\ v_i \text{ is in } C_p \text{ and } v_j \text{ is in } C_q\}$. Determine a minimum cost spanning tree in G''. For each edge (p, q) in the optimal spanning tree, consider the vertices v_i in C_p and v_j in C_q such that $c'_{pq} = L_{ij} + L_{ji}$, and add to G' all arcs that lie on the shortest paths linking v_i to v_j and v_j to v_i in G.

Step 4. Determine an Eulerian tour T in G' by means of the EULERIAN-CIRCUIT algorithm (see Section 2.1.2). T corresponds to a rural postman tour on G.

This algorithm is illustrated in Figure 9.4. In this example, the complete undirected graph G'' contains two vertices linked by an edge of cost 3.

The C&S approach has been studied in several papers (see for example [3] and [9]). We describe below the algorithm proposed by Christofides et al. [9].



(a)
A graph G.

(b)
The symmetric graph produced in Step 2.

(c)
The strongly connected and symmetric graph produced in Step 3.

Figure 9.4

Algorithm C&S-DRPP

INPUT: A directed graph $G = (V, A)$, and a set R of required arcs.

OUTPUT: A rural postman tour T on G.

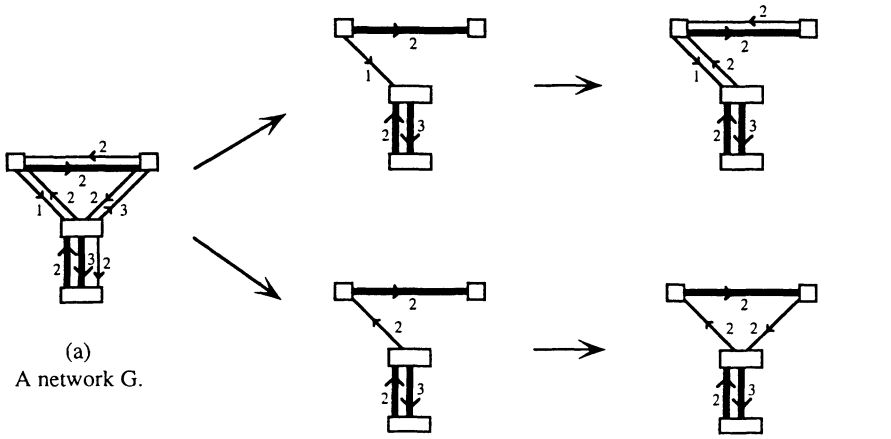
Step 1. Set $G' := G_R$. If G' is connected then go to Step 3.

Step 2. Let $C_1, ..., C_c$ be the connected components of G'. Construct a complete directed graph G'' with vertex set $\{1, ..., c\}$, and with edge costs $c''_{pq} = \min \{c_{ij} \mid v_i \text{ is in } C_p \text{ and } v_j \text{ is in } C_q\}$. Determine a minimum cost spanning arborescence in G'' (see for example [16]), rooted at any arbitrary vertex. For each arc (p, q) in the optimal spanning arborescence, consider the vertices v_i in C_p and v_j in C_q such that $c''_{pq} = c_{ij}$, and add to G' all arcs that lie on the shortest path linking v_i to v_j in G.

Step 3. Apply Steps 2, 3 and 4 of the CONNECTED-DRPP algorithm in order to transform G' into a symmetric graph.

Step 4. Determine an Eulerian tour T in G' by means of the EULERIAN-CIRCUIT algorithm (see Section 2.1.2). T corresponds to a rural postman tour on G.

The above algorithm can be repeated by considering in turn all vertices in G'' as root of the spanning arborescence (see Step 2), and then selecting the best solution. To illustrate, consider the graph G shown in Figure 9.5*(a)*. The complete graph G'' contains 2 vertices and two spanning arborescence can therefore be determined for getting a connected graph G'. These two graphs are shown in Figure 9.5*(b)* and their symmetric extension is represented in Figure 9.5*(c)*. Notice that the second solution is better than the first one.



(a)
A network G.

(b)
Two possible
connected graphs G'.

(c)
Two symmetric and
strongly connected graphs.

Figure 9.5

As observed by Christofides et al. [9], any rural postman tour can be improved by replacing two consecutive non-required arcs (v_i, v_j) and (v_j, v_k) on the tour, by the arc (v_i, v_k) if $c_{ik} < c_{ij} + c_{jk}$. For example, the first solution in Figure 9.5*(c)* can be transformed into the second better one by replacing two consecutive arcs of cost 2 and 1, respectively, by an arc of cost 2.

2.2.3 The mixed rural postman problem.

The mixed RPP is defined on a mixed graph $G = (V, E \cup A)$. It can be assumed that each vertex in V is incident to a required edge or arc, and that each edge in E is required. Indeed, if this is not the case, then the original graph can be simplified in a similar way as in the previous sections. More precisely, consider the graph G' obtained from G by replacing each edge (v_i, v_j) in E by two arcs (v_i, v_j) and (v_j, v_i) of cost c_{ij}. The simplified graph $G_S(V_R, R \cup A_S)$ is obtained from $G_R(V_R, R)$ by first including in A_S two arcs (v_i, v_j) and (v_j, v_i) for each v_i, v_j in V_R, the cost of (v_i, v_j) being set equal to the length of the shortest path linking v_i to v_j in G' (while the arcs and edges in R have the same cost as in G_R. The set A_S of added arcs is then reduced by eliminating

(*a*) all arcs $(v_i, v_j) \in A_S$ for which $c_{ij} = c_{ik} + c_{kj}$ for some $v_k \in V_R$,

(*b*) all arcs $(v_i, v_j) \in A_S$ which are parallel to a required edge with the same cost, and

(*c*) all arcs $(v_i, v_j) \in A_S$ which are parallel to a required arc with the same cost and the same orientation.

Notice that all edges in G_S are required. This construction is illustrated in Figure 9.6.

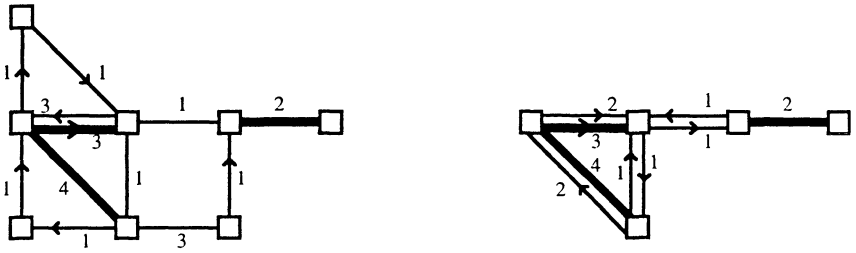


Figure 9.6

From now on, we will assume that the considered graph $G = (V, E \cup A)$ has been simplified, which means that $E \subset R$. We denote $A_R = A \cap R$ the set of required arcs in G. Very few papers have been devoted to the design of heuristic solution methods for the mixed RPP. The main reason is probably that this problem can be tackled by means of tools similar to those used for the solution of the directed or undirected RPP. The construction of a solution is done by first choosing an orientation for each required edge, then connecting the required subgraph by means of a minimum cost spanning tree, and finally adding arcs in order to get an even symmetric graph on which an Eulerian circuit can be determined. As for the mixed CPP, when arcs are added to the graph in order to make it symmetric, a possibility is given to reverse the chosen orientation of a required edge.

We describe below a heuristic method proposed by Corberan et al. [11] that works along these lines. In addition to the above mentioned strategy, these authors propose to orient the required edges in such a way that each vertex in the resulting directed graph has its out-degree almost equal to its in-degree. This is done as follows.

Algorithm MRPP

INPUT: A mixed graph $G = (V, E \cup A)$ and a set R of required edges and arcs.

OUTPUT: A rural postman tour on G.

(*Orientation of the edges*)

Step 1. Construct a directed graph $G' = (V, A_E \cup A_R)$ by orienting in turn each required edge (v_i, v_j) in E as follows. First set A_E equal to the empty set, and let b'_i denote the difference between the in-degree and the out-degree of a vertex v_i in G'. A required edge (v_i, v_j) is oriented from v_i to v_j if $b'_i \geq b'_j$, and from v_j to v_i otherwise. Once an edge has been oriented, it is introduced into A_E and the next required edge is considered.

(*Construction of a connected graph G'*)

Step 2. Let $C_1, ..., C_c$ denote the connected components of G_R. Define $\delta_i^- = \min \{c_{ji} | \ (v_j, v_i) \in E \cup A\}$ and $\delta_i^+ = \min \{c_{ij} | \ (v_i, v_j) \in E \cup A\}$. Assign the following cost c'_{ij} to each arc (v_i, v_j) linking two connected components of G_R:

$$c'_{ij} = \begin{cases} c_{ij} + \delta_i^- + \delta_j^+ & \text{if } b'_i \leq 0 \text{ and } b'_j \geq 0 \\ c_{ij} - \delta_i^+ + \delta_j^+ & \text{if } b'_i > 0 \text{ and } b'_j \geq 0 \\ c_{ij} + \delta_i^- - \delta_j^- & \text{if } b'_i \leq 0 \text{ and } b'_j < 0 \\ c_{ij} - \delta_i^+ - \delta_j^- & \text{if } b'_i > 0 \text{ and } b'_j < 0 \end{cases}$$

Construct a complete undirected connected graph G_c with vertex set $\{1, ..., c\}$ and edge costs $c''_{pq} = \min\{c'_{ij} | \ v_i \in C_p \text{ and } v_j \in C_q \text{ or } v_i \in C_q \text{ and } v_j \in C_p\}$. Determine a spanning tree S of minimum cost in G_c and introduce the corresponding arcs into G'. If G' is symmetric then go to Step 5.

(*Extension of G' to a symmetric graph*)

Step 3. Construct a network $H = (V, A_E \cup A'_E \cup A''_E \cup A)$ such that $A_E \cup A'_E$ contains directed arcs of opposite orientation and cost c_{ij} for each edge (v_i, v_j) in E, and A''_E contains a directed arc (v_j, v_i) with zero cost for each (v_i, v_j) in A_E. Assign an infinite capacity to each arc in $A_E \cup A'_E \cup A$ and a capacity of 2 to each arc in A''_E . As in Step 1, let b'_i denote the difference between the in-degree and the out-degree of a vertex v_i in G'. Attach a supply of b'_i to each vertex of H with $b'_i > 0$, and a demand of $|b'_i|$ to each vertex with $b'_i < 0$. Find a minimum cost flow in H that satisfies the demands.

Let x_{ij} be the flow on arc (v_i, v_j). For each arc (v_j, v_i) in A''_E do the following: if $x_{ji} = 2$ then change the orientation of (v_i, v_j) in G' (i.e., orient it from v_j to v_i instead of from v_i to v_j), and if $x_{ji} = 1$ then remove the orientation of the arc (v_i, v_j) in G'. Augment G' by adding x_{ij} copies of each arc (v_i, v_j) in $A_E \cup A'_E \cup A$. If G' does not contain any undirected edge then go to Step 5.

(*Extension of G' to a directed symmetric graph*)

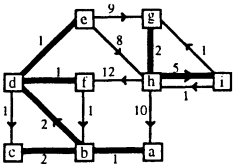
Step 4. Let E' be the set of undirected edges in G'. Determine the set V_0 of vertices incident to an odd number of edges in E', and construct a complete graph G_0 with vertex set V_0. Let L_{ij} denote the length of a shortest path linking v_i to v_j, using arcs in $A_E \cup A'_E \cup A$. Assign a cost $c^0_{ij} = \min\{L_{ij}, L_{ji}\}$ to each edge of G_0. Find a perfect matching M in G_0 of minimum cost and orient each cycle induced by $E' \cup M$ so that the corresponding circuits are of minimum cost in G. Introduce these circuits into G', in replacement of E'.

(*Construction of an Eulerian tour in G''*)

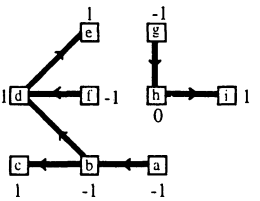
Step 5. Determine an Eulerian tour T in G' by means of the EULERIAN-CIRCUIT algorithm (see Section 2.1.2). T corresponds to a rural postman tour on G.

The above algorithm is illustrated in Figure 9.7. The graph G' is first constructed in Step 1 by successively orienting the edges (a, b), (b, c), (d, e), (f, d) and (g, h). The costs c'_{ij} of the edges (v_i, v_j) linking the connected components of G' are then computed. For example, there is one arc entering vertex e and no out-going ones, while vertex g has one out-going arc and no in-coming ones. Hence, the cost of the edge (e, g) is obtained from its cost in G by subtracting the cost of the shortest edge or arc out-going e, as well as the cost of the shortest edge or arc entering g. The new cost of the edge (e, g) is therefore equal to $9 - 1 - 1 = 7$. All costs c'_{ij}, as well as the undirected graph G_c are represented in Figure 9.7*(c)*.

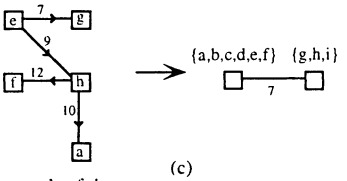
The unique edge of G_c is a minimum cost spanning tree. It corresponds to the edge (e, g) of G which is added to G' in order to get a connected graph (see Figure 9.7*(d)*). Notice that we could have determined a minimum cost spanning tree S according to the original costs c_{ij} instead of c'_{ij}. We try however to build a graph G' which is as close as possible to a symmetric graph. We therefore favor the choice of edges (v_i, v_j) where v_i has more in-coming arcs than out-going ones, and where v_j has more out-going arcs than in-coming ones. By using c_{ij} instead of c'_{ij}, we would have chosen the edge (e, h) instead of (e, g), and the connected graph G' would have a total demand of five units instead of three.



(a)
The original network G

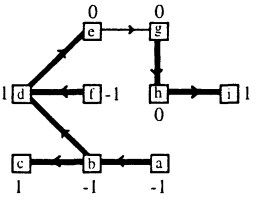


(b)
The directed graph G' obtained by successively orienting the edges (a,b), (b,c), (d,e), (f,d) and (g,h).
The b_i values (demands and supplies) are shown on the vertices.

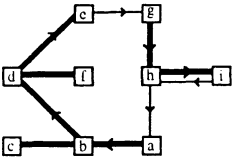


(c)
The costs c'_{ij} of the arcs connecting the connected components of G_R.

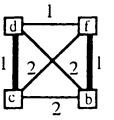
The complete undirected graph G_c.



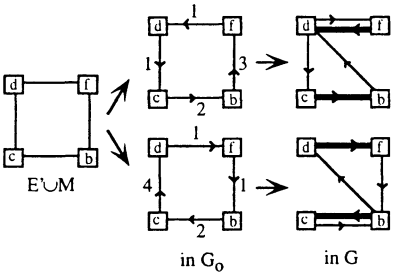
(d)
The connected directed graph G', at the end of Step 2.
Demands and supplies are shown on the vertices.



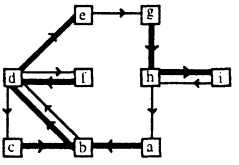
(e)
The symmetric graph G' at the end of Step 3.
Two required edges are undirected.



(f)
The complete undirected graph G_o
with an optimal matching M in bold line.



(g)
The two possible orientations of the cycle induced by E'∪M.



(h)
The final directed
and symmetric graph G'.

Figure 9.7

G' is then transformed into a symmetric graph by solving a minimum cost flow problem. The resulting graph is represented in Figure 9.7*(e)*. The required edges (d, f) and (b, c) are undirected and we therefore enter Step 4. The complete graph G_0 is shown in Figure 9.7*(f)*, with its optimal matching M in bold lines. The cycle induced by the union of E' and M has two possible orientations of cost 7 and 8, respectively (see Figure 9.7*(g)*). The best circuit is added to G', and the resulting symmetric graph is shown in Figure 9.7*(h)*.

Notice that in Step 1, we have not specified the order in which the edges of E should be considered for their orientation. Corberan et al. [11] suggest to use a particular ordering of the edges and also propose some variations for the construction of a spanning tree at Step 2. The reader interested in these developments is referred to [11].

2.2.4 The stacker crane problem.

Consider an undirected graph G in which some edges must be traversed at least once in a given direction, but can also be traversed as often as needed in the opposite direction. Such a situation can be represented by a mixed graph $G = (V, E \cup A)$ in which each arc in A is required and parallel to an edge in E with the same cost. The stacker crane problem (SCP) is to determine a covering tour for A of minimum cost in G.

Obviously, the SCP is a particular case of the MRPP. It can also be viewed as a DRPP by replacing each edge with two parallel arcs of opposite direction. Hence, all heuristic methods mentioned in Sections 2.2.2 and 2.2.3 can be used to solve the SCP. Frederickson et al. [22] have however developed several specific algorithms for which a worst case ratio can be obtained. These algorithms are described below.

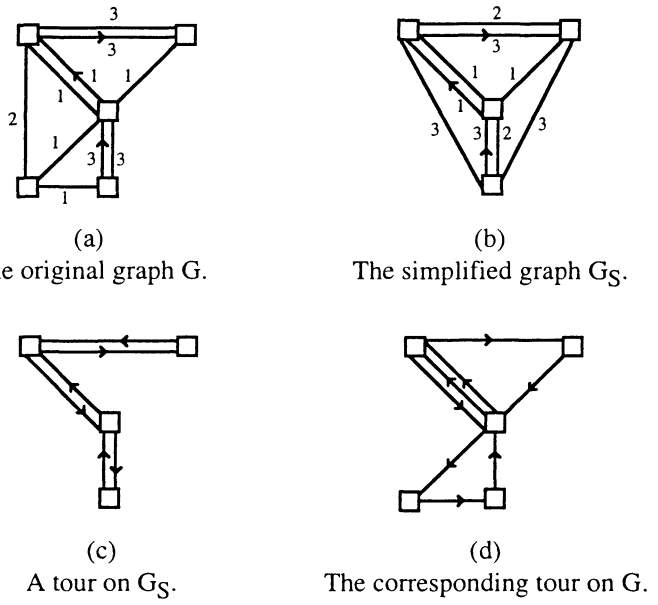
For solving the SCP, it is convenient to work on a simplified graph $G_S = (V_R, E_S \cup A)$ where V_R is the set of vertices in G incident to at least one arc in A, and E_S contains an edge (v_i, v_j) for each v_i, v_j in V_R . The cost of an edge (v_i, v_j) in E_S is set equal to the length of the shortest chain between v_i and v_j in G. A tour T on G_S can easily be transformed into a tour on G by replacing each edge of T by its corresponding shortest chain in G.

Figure 9.8 illustrates these concepts. From now on, we will assume that the given graph G has already been transformed. Notice that this means that

(*i*) each arc is required and parallel to an edge of no greater cost,

(*ii*) each pair of vertices is linked by an edge, and

(*iii*) the cost function defined on the edges satisfies the triangle inequalities.



(a)
The original graph G.

(b)
The simplified graph G_S.

(c)
A tour on G_S.

(d)
The corresponding tour on G.

Figure 9.8

The following simple heuristic has a worst-case ratio of 2. It is illustrated in Figure 9.9.

Algorithm SIMPLE-SCP

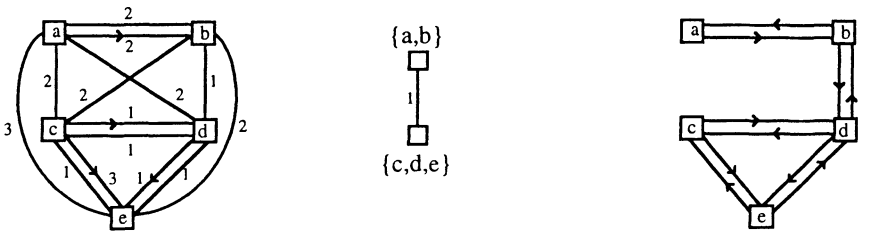
INPUT: A mixed graph $G = (V, E \cup A)$.

OUTPUT: A stacker crane tour T on G.

Step 1. Set $G' = (V, A \cup A')$ where $A \cup A'$ contains directed arcs (v_i, v_j) and (v_j, v_i) of opposite direction for each arc (v_i, v_j) in A.

Step 2. Let $C_1, ..., C_c$ be the connected components of G'. Construct a complete undirected graph G_c with vertex set $\{1, ..., c\}$, and in which the cost of an edge (p, q) is set equal to the smallest cost of an edge between C_p and C_q in G. Find a minimum cost spanning tree in G_c and let S be its corresponding set of edges in G. For each edge (v_i, v_j) in S, add two arcs (v_i, v_j) and (v_j, v_i) of opposite direction into G'.

Step 3. Determine an Eulerian circuit T in G' by means of the EULERIAN-CIRCUIT algorithm (see Section 2.1.2). T corresponds to a covering tour for A in G.



(a) A graph G. (b) The graph G_C. (c) A covering tour for A.

Figure 9.9

Let C_A and C_S denote the total cost of the arcs in A and S, respectively, and let C^* be the optimal value of the SCP. The above algorithm determines a solution of cost $2C_A + 2C_S$. Since $C_S \leq C^* - C_A$, we have $2C_A + 2C_S \leq 2C^*$, which means that the SIMPLE-SCP algorithm has a worst case ratio of 2. Frederickson et al. have proposed a more elaborate procedure which has a worst case ratio of 9/5. It combines the two following complementary strategies, called LARGE-ARCS and SMALL-ARCS.

Algorithm LARGE-ARCS

INPUT: A mixed graph $G = (V, E \cup A)$.

OUTPUT: A stacker crane tour T on G.

Step 1. Let H and T be the multisets of heads and tails of the arcs in A. Construct a complete bipartite graph B in which each v_i in H is linked to each v_j in T by an edge having the same cost as the edge (v_i, v_j) in E. Determine a minimum cost perfect matching M in B. Construct a directed graph $G' = (V, A \cup A_B)$ where A_B is obtained by orienting each edge (v_i, v_j) in M from H to T.

Step 2. Apply Steps 2 and 3 of SIMPLE-SCP to G' in order to determine a covering tour for A.

Algorithm SMALL-ARCS

INPUT: A mixed graph $G = (V, E \cup A)$.

OUTPUT: A stacker crane tour T on G.

Step 0. If the arcs in A are not vertex disjoint, then consider each vertex v of V which is adjacent to a set $\{a_1, ..., a_k\}$ of arcs, with $k > 1$:

- replace v by a clique C_v of size k in which each edge has a zero cost,
- replace each arc a_i $(i = 1, ..., k)$ by an arc incident to the i-th element in C_v,
- replace each edge in G that was originally incident to v by an edge incident to any vertex in C_v.

All arcs are now vertex disjoint.

Step 1. Construct a complete undirected graph G_c with vertex set A (i.e., each arc in G has is corresponding vertex in G_c). Define the cost of an edge (x, y) in G_c as the cost of the smallest edge between one endpoint of x and one endpoint of y. Let L_{xy} denote the length of shortest chain between x and y in G_c.

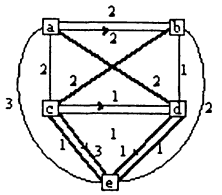
Step 2. Determine a minimum cost spanning tree S in G_c, using the distance function L. Identify the odd-degree vertices in S and determine a minimum cost matching M on these vertices, using the distance function L. Set $G' = (V, A)$, and add to G' all edges of G induced by $S \cup M$.

Step 3. Let A_{odd} be the subset of arcs in A whose two endpoints have an odd degree in G', and let $A_{even} = A \setminus A_{odd}$. For each arc (v_i, v_j) in A_{odd}, add an arc (v_j, v_i) in G'. Replace all cliques C_v defined in Step 0 by their original single vertex v.

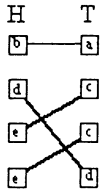
Step 4. Find an Eulerian cycle ζ in G' (ignoring even arc directions) by means of the EULERIAN-CYCLE algorithm (see Section 2.1.1). Choose an orientation for ζ so that the total length of the arcs in A_{even} which are traversed backward is at most equal to half of the total length of the arcs in A_{even}. Orient all edges in G' according to the chosen orientation of ζ. If an arc (v_i, v_j) in A_{even} is traversed in the wrong direction, then add two arcs (v_i, v_j) and (v_j, v_i) into G'.

Step 5. Determine an Eulerian circuit T in G' by means of the EULERIAN-CIRCUIT algorithm (see Section 2.1.2). T corresponds to covering tour for A.

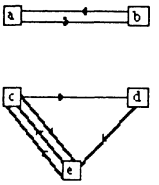
These two procedures are illustrated in Figure 9.10, on the same example as in Figure 9.9. Both LARGE-ARCS and SMALL-ARCS have an $\mathcal{O}(\max\{|V|^3, |A|^3\})$ complexity. Notice that the original SMALL-ARCS algorithm described in [22] does not contain Step 0. This Step is however crucial if the arc set is not vertex disjoint. Indeed, if Step 0 is not



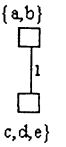
(a)
The original graph G.



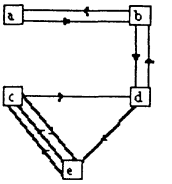
An optimal matching in B.



The graph G' at the end of Step 1.

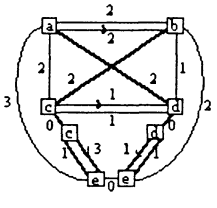


The graph G_c.

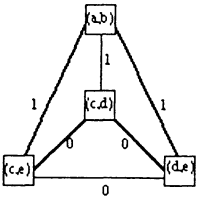


A covering tour for A.

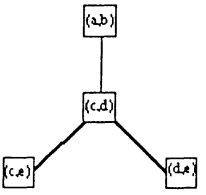
(b)
Illustration of procedure LARGE-ARCS.



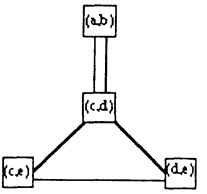
The transformed graph at the end of Step 0.



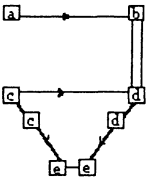
The complete graph G_c with L distances.



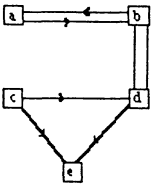
The minimum cost spanning tree S in G_c.



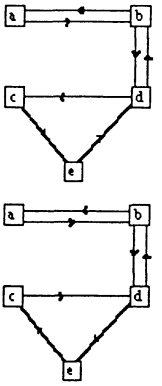
S∪M in G_c.

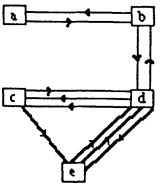


The graph G' at the end of Step 2.



The graph G' at the end of Step 3.





A covering tour for A.

Two possible orientations of G'.

(c)
Illustration of procedure SMALL-ARCS.

Figure 9.10

applied while A is not vertex disjoint, then it may happen, at the end of Step 2, that the endpoints of the arcs in G' do not have the same parity (hence, A_{odd} and A_{even} cannot be defined). Assume for example that G contains three vertices a, b, c, two arcs (a, b) and (b, c) of cost 1, and three edges (a, b), (b, c) and (a, c) of cost 1. In such a case, the complete undirected graph G_c has two vertices linked by an edge of zero cost. Hence, the graph G' at the end of Step 2 contains only two arcs (a, b) and (b, c), and these two arcs have one endpoint of degree 1, and one of degree 2.

Let us first analyze the worst-case behavior of the LARGE-ARCS algorithm. Let C_A, C_M and C_S denote the total cost of the arcs in A, M and S, respectively, and let C^* be the optimal value of the SCP. Both C_M and C_S are smaller or equal to $C^* - C_A$. It follows that LARGE-ARCS provides a solution of cost at most equal to $C_A + (C^* - C_A) + 2(C^* - C_A) = 3C^* - 2C_A$.

Let C_S and C_M denote the total cost of the spanning tree and the matching in Step 2 of SMALL-ARCS, and let C_{odd} and C_{even} denote the total cost of the arcs in A_{odd} and A_{even}, respectively. SMALL-ARCS delivers a solution of cost smaller or equal to $C_A + C_S + C_M + C_{odd} + 2(\frac{1}{2}C_{even})$. Following Christofides [8], we have $C_S + C_M \leq \frac{3}{2}(C^* - C_A)$. Since $C_{even} + C_{odd} = C_A$, it follows that SMALL-ARCS provides a solution of cost at most equal to $\frac{1}{2}C_A + \frac{3}{2}C^*$.

The above analysis shows that LARGE-ARCS is more efficient than SMALL-ARCS when C_A is large relative to C^*, while SMALL-ARCS should be preferred when C_A is small relative to C^*. By combining these two heuristics, one gets an algorithm with worst case ratio of $\frac{9}{5}$. Indeed, consider the following algorithm.

Algorithm COMBINED-SCP

INPUT: A mixed graph $G = (V, E \cup A)$.

OUTPUT: A stacker crane tour T on G.

Step 1. Determine two tours T_L and T_S by means of LARGE-ARCS and SMALL-ARCS, respectively.

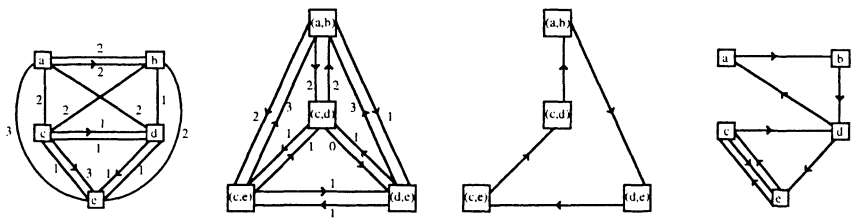
Step 2. Select the tour of smallest cost among T_L and T_S.

If $C_A < \frac{3}{5}C^*$, then SMALL-ARCS provides a solution of cost no larger than $\frac{1}{2}\left(\frac{3}{5}C^*\right) + \frac{3}{2}C^* = \frac{9}{5}C^*$.

If $C_A \geq \frac{3}{5}C^*$, then LARGE-ARCS provides a solution of cost smaller or equal to $3C^* - 2\left(\frac{3}{5}C^*\right) = \frac{9}{5}C^*$. It follows that COMBINED-SCP has a

worst case ratio of $\frac{9}{5}$.

The SCP can be viewed as a particular case of the directed Traveling Salesman Problem (TSP). Indeed, given a graph $G = (V, E \cup A)$, consider the complete directed graph G_{TSP} with vertex set A and in which the cost of an arc linking a vertex (v_i, v_j) to a vertex (v_k, v_h) is equal to the smallest length of a chain linking v_j to v_k in G. Any traveling salesman tour of cost C_{TSP} in G_{TSP} can be transformed into a stacker crane tour of cost $C_{TSP} + C_A$ in G (see Figure 9.11). Heuristic algorithms for the TSP can therefore be adapted in order to solve the SCP directly on G. For example, Lukka and Salminen [36] have developed an arc insertion heuristic for the SCP which is an adaptation of a TSP insertion heuristic proposed by Rosenkrantz et al. [44].



(a) A graph G.

(b) The corresponding graph G_{TSP}.

(c) An optimal tour in G_{TSP}.

(d) The corresponding optimal tour in G.

Figure 9.11

2.2.5 Additional algorithmic tools.

The heuristic algorithms described in sections 2.2.1 to 2.2.4 use the same basic ingredients. A minimum cost matching or flow is determined in order to get an even or symmetric graph, while a minimum cost spanning tree or arborescence is found for connecting the components of the required subgraph G_R. Recently, additional basic algorithmic tools have been developed for the solution of uncapacitated arc routing problems [30]. While these tools are described here for the undirected rural postman problem, they can easily be adapted and extended in order to deal with directed or mixed graphs.

It has been observed by many authors that if a tour contains a chain C of non-required edges, then it can eventually be reduced by replacing C by a shortest chain linking the endpoints of C. This simple idea has been extended by Hertz et al. [30] who have designed a procedure, called SHORTEN, that attempts to reduce the length of a given tour T by

covering the same set of required edges, but not necessarily in the same order. This procedure can be described as follows, and is illustrated in Figure 9.12.

Algorithm SHORTEN

INPUT: A rural postman tour T on an undirected graph $G = (V, E)$.

OUTPUT: A possibly better rural postman tour on G.

Step 1. Choose an orientation for T and let v_i be any vertex on the tour.

Step 2. Determine a vertex v_j on T such that the length of the path linking v_i to v_j is as long as possible, while the path linking v_j to v_i covers all required edges. Let P denote the path from v_i to v_j, and Q the path from v_j to v_i.

Step 3. If all arcs entering v_j on Q are serviced arcs, then go to Step 4. Otherwise, let (v_k, v_j) be a non-serviced arc on Q. The arcs on Q up to (v_k, v_j) induce a circuit $C = (v_j, ..., v_k, v_j)$. Reverse the orientation of C and go to Step 2.

Step 4. If the length of the shortest chain SP_{ij} linking v_i to v_j in G is shorter than the length of P, then replace P by SP_{ij}.

Step 5. Repeatedly apply Steps 2 to 4, considering the two possible orientations of T, and each possible starting vertex v_i on T, until no improvement can be obtained.

Given a required edge (v_i, v_j) in a covering tour T for R, the second procedure, called DROP, builds a new tour T' that covers all edges in $R \setminus \{(v_i, v_j)\}$. It can simply be described as follows.

Algorithm DROP

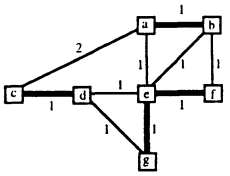
INPUT: A covering tour T for R in an undirected graph $G = (V, E)$.

OUTPUT: A covering tour T' for $R \setminus \{(v_i, v_j)\}$.

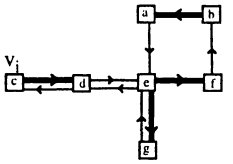
Step 1. Set $R := R \setminus \{(v_i, v_j)\}$.

Step 2. Try to get a shorter tour T' by means of the SHORTEN algorithm.

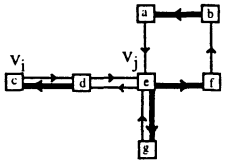
Figure 9.13 illustrates the above procedure. Inversely, given a covering tour for a set R of edges, and given an edge $(v_i, v_j) \notin R$, the next procedure, called ADD, constructs a covering tour for $R \cup \{(v_i, v_j)\}$. In



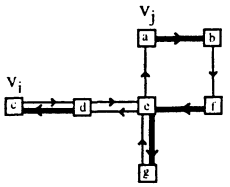
(a)
A graph G.



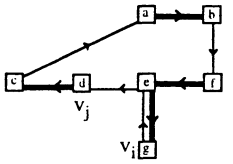
(b)
A tour (c,d,e,f,b,a,e,g,e,d,c) on G,
and a starting vertex v_i.



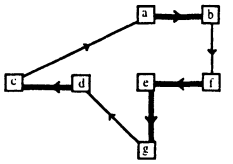
(c)
The situation
when entering Step 3.
P=(c,d,e), Q=(e,f,b,a,e,g,e,d,c),
and (a,e) is a non-serviced arc on Q.



(d)
The situation
when entering Step 4.
The orientation of the circuit
(e,f,b,a,e) has been reversed.

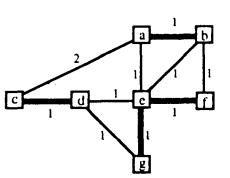


(e)
A shorter tour covering R,
and a new pair (v_i,v_j)
of vertices on T.

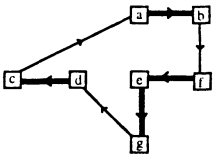


(f)
The final shorter tour.

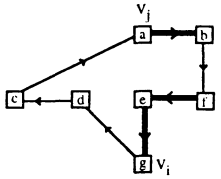
Figure 9.12



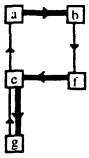
(a)
A graph G.



(b)
A covering tour for R.



(c)
A pair (v_i,v_j) of vertices on T,
when (c,d) is removed from R.



(d)
A shorter tour.

Figure 9.13

what follows, SP_{ij} denotes the shortest chain between v_i and v_j in G, while L_{ij} is the length of SP_{ij}.

Algorithm ADD

INPUT: A covering tour T for R in an undirected graph $G = (V, E)$, and an edge $(v_i, v_j) \notin R$.

OUTPUT: A covering tour for $R \cup \{(v_i, v_j)\}$.

Step 1. If neither v_i nor v_j appears on T, then identify a vertex v_k on T yielding the minimum value of $L_{ki} + L_{jk}$, and add the circuit

$SP_{ki} \cup \{(v_i, v_j)\} \cup SP_{jk}$ on T. Otherwise, if only one of v_i and v_j (say v_i), or both of them appear on T, but not consecutively, add the circuit (v_i, v_j, v_i) on T.

Step 2. Set $R := R \cup \{(v_i, v_j)\}$ and determine a possibly shorter tour by means of the SHORTEN algorithm.

Step 1 of the above procedure is illustrated in Figure 9.14. Notice that the procedures described in this section can also be used as basic tools for the design of constructive algorithms for the RPP. As an example, a solution to the undirected RPP can easily be obtained by means of the following constructive algorithm.

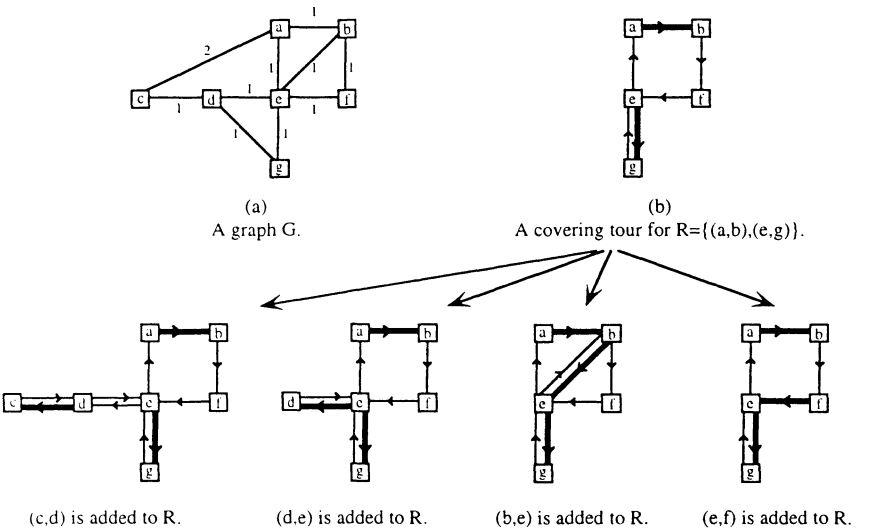


Figure 9.14

Algorithm CONSTRUCT-URPP

INPUT: An undirected graph $G = (V, E)$, and a set R of required edges.

OUTPUT: A rural postman tour on G.

Step 1. Choose a required edge (v_i, v_j). Set $T := (v_i, v_j, v_i)$ and $R' := \{(v_i, v_j)\}$.

Step 2. If $R' = R$ then stop. Otherwise, choose a required edge (v_i, v_j) in $R \setminus R'$ and determine a tour T' covering $R' \cup \{(v_i, v_j)\}$ by means of the ADD procedure. Set $R' := R' \cup \{(v_i, v_j)\}$, $T := T'$, and repeat Step 2.

Post-optimization procedures can be designed on the basis of procedures SHORTEN, DROP and ADD. As an example, one can try to improve any given tour T by removing a required edge and reinserting it into the tour, by means of the DROP and ADD algorithms, respectively. This procedure, called DROP-ADD, is analogous of the Unstringing-Stringing (US) algorithm for the TSP [25]. More precisely, it can be described as follows.

Algorithm DROP-ADD

INPUT: A rural postman tour on an undirected graph $G = (V, E)$.

OUTPUT: A possibly better rural postman tour on G.

Step 1. Choose a required edge (v_i, v_j) and construct a covering tour T' for $R \setminus \{(v_i, v_j)\}$ by means of the DROP algorithm. Construct a covering tour T'' for R by applying the ADD algorithm on T'.

Step 2. If T'' is shorter than T, then set $T := T''$.

Step 3. Repeat Steps 1 and 2 with all possible edges (v_i, v_j) in R, until no additional improvement can be obtained.

An algorithm, similar to the 2-opt procedure [12] for the undirected TSP, can also be designed for the RPP, as shown below.

Algorithm 2-OPT-RPP

INPUT: A rural postman tour T on an undirected graph $G = (V, E)$.

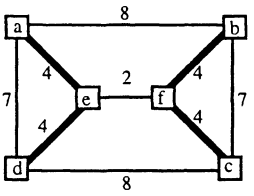
OUTPUT: A possibly better rural postman tour on G.

Step 1. Choose an orientation for T and select two arcs (v_i, v_j) and (v_k, v_h) on T. Replace (v_i, v_j) and (v_k, v_h) by the two shortest chains SP_{ik} and SP_{jh} in G, respectively. Reverse the orientation of the path linking v_j to v_k on T. Let T_1 be the resulting tour, and let R' be the set of required edges covered by T_1. Determine a possibly shorter covering tour T_2 for R', by applying the SHORTEN algorithm on T_1.

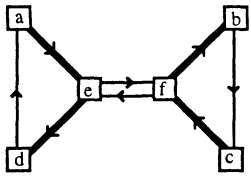
Step 2. If (v_i, v_j) and/or (v_k, v_h) is a required edge that is not covered by T_2, then determine a tour T_3 covering R by means of the ADD algorithm. If T_3 is shorter than T, then set $T := T_3$.

Step 3. Repeat Steps 1 and 2 with the two possible orientations of T, and with all possible pairs of arcs (v_i, v_j) and (v_k, v_h) on T, until no additional improvement can be obtained.

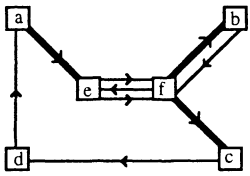
In spite of its apparent simplicity, the 2-OPT-RPP procedure can in fact be quite involved and its application to a tour T can lead to a



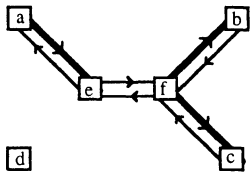
(a)
A network G.



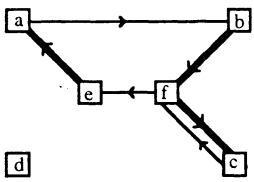
(b)
A directed rural postman tour on G.



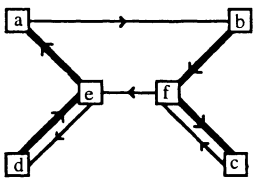
(c)
The arcs (b,c) and (e,d) are replaced
by the paths (b,f,e) and (c,d),
and the orientation of the path (c,f,e) is reversed.
Notice that the tour does not cover (d,e).



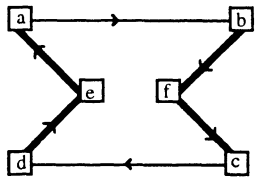
(d)
SHORTEN has removed
the circuit (f,e,f),
and has replaced the path (c,d,a)
by the shorter path (c,f,e,a).



(e)
SHORTEN has replaced
the path (a,e,f,b)
by the shorter path (a,b).



(f)
ADD has introduced
the circuit (e,d,e)
in order to cover (d,e).



(g)
SHORTEN has replaced
the path (c,f,e,d) by (c,d).

Figure 9.15

complicated sequence of intermediate steps before reaching a shorter tour. This is very different from what happens in the TSP, where a 2-opt exchange simply means that two edges are replaced by two other edges. This is illustrated in Figure 9.15.

Such basic tools can also be used as ingredients in adaptations of metaheuristic algorithms for the RPP. For example, Corberan et al. [11] have

recently developed a tabu search technique for the solution of the mixed RPP. They define a tour T' as a neighbor of a given tour T, if T' can be obtained from T by means of a procedure similar to the SHORTEN algorithm. For more details, the reader is referred to [11].

3. HEURISTICS FOR CAPACITATED ARC ROUTING PROBLEMS

This section is devoted to the capacitated arc routing problem (CARP). Consider a graph $G = (V, E \cup A)$ with vertex set $V = \{v_0, v_1, ..., v_n\}$, edge set E and arc set A. Vertex v_0 represents a depot at which are based identical vehicles of capacity Q. A subset R of edges and arcs are said to be required, i.e., they must be *serviced* or *covered* by a vehicle. The number of vehicles can be a decision variable or a fixed parameter. Each edge and arc (v_i, v_j) of R has a non-negative weight or demand q_{ij}. Each required edge and arc can be *traversed* any number of times by any number of vehicles. A vehicle route is *feasible* if it starts and ends at the depot, and the total weight of the edges and arcs serviced by the vehicle does not exceed Q. The CARP consists of designing a set of feasible vehicle routes of least total cost, such that each required edge and arc appears in at least one route and is serviced by exactly one vehicle.

The CARP is $\mathcal{NP}$-hard since it includes the RPP as special case. Even finding a 0.5-approximation to the CARP is $\mathcal{NP}$-hard, as shown by Golden and Wong [26]. We describe in this section heuristic methods that have been developed for the solution of the CARP. These heuristics can be broadly classified into three categories: simple constructive methods, two-phase constructive methods, and adaptations of meta-heuristics. In the next sections, we give examples of heuristics for each of these categories.

3.1. SIMPLE CONSTRUCTIVE METHODS FOR THE CARP

The capacitated Chinese Postman Problem (CCPP) is a special case of the CARP where $R = E \cup A$. The CCPP can be considered as the counterpart of the well-known node-oriented standard Vehicle Routing Problem (VRP). Most simple constructive methods for the CARP have been developed in a undirected CCPP context, but can easily be adapted to deal with more general CARPs. We give in this section the original description of these algorithms.

We will use the following notation. Given two (possibly closed) chains $P = (x, ..., y)$ and $P' = (y, ..., z)$ having a common endpoint y, the union of the edges of these two chains is a longer (possibly closed) chain

$P'' = (x, ..., y, ..., z)$ which is denoted $P + P'$.

In 1973, Christofides [7] has developed the CONSTRUCT-STRIKE algorithm for the solution of the undirected CCPP. The basic idea of this algorithm is to construct feasible vehicle routes which, when removed, do not separate the graph into disconnected components (not counting isolated vertices). When such a route C is constructed, all required edges on C are removed from the original graph, and this construct-then-remove procedure is repeated until no more feasible vehicle routes can be determined. If edges have not been covered, then artificial non-required edges are added to the remaining graph and new feasible vehicle routes are constructed. This process is repeated until a feasible solution of the undirected CCPP is obtained. A more formal description of this algorithm is given below.

Algorithm CONSTRUCT-STRIKE

INPUT: An undirected graph $G = (V, E)$.

OUTPUT: A solution of the undirected CCPP.

Step 0. Set $G' := G$.

Step 1. Attempt to determine a feasible vehicle route such that when its edges are removed from G', the remaining graph is still connected (not counting isolated vertices). If a feasible route has been determined, then remove all required edges on this route from G' and repeat Step 1.

Step 2. Remove all artificial edges (if any) from G'. If all edges of G have been covered (i.e., G' is empty), then stop. If the depot v_0 has degree zero, then add a copy v_0' of v_0 in G' and link v_0 to v_0' by an artificial non-required edge of infinity cost. If G' contains odd degree vertices then go to Step 3, else go to Step 4.

Step 3. Solve a minimum cost perfect matching problem in order to transform G' into an even graph (see Section 2.1.1). All edges added to G' are artificial and non-required. If a copy of the depot has been created at Step 2, then merge v_0 and v_0'. Return to Step 1.

Step 4. Add to G' two shortest chains of artificial non-required edges between v_0 and the vertex nearest v_0, and return to Step 1. If feasible vehicle routes can still not be determined, then add two more shortest chains of artificial non-required edges between v_0 and its second nearest vertex, etc., until a feasible vehicle route is found in Step 1.

Christofides does not suggest any particular method for constructing the feasible vehicle routes in Step 1. He uses visual inspection. The above algorithm is illustrated in Figure 9.16. In this example, all edges have a unit demand, and the vehicle capacity Q is equal to 3. The values on the edges correspond to their length. A first vehicle route $(depot, a, b, depot)$ is determined in Step 1. The remaining cycle on G' is not a feasible vehicle route since its demand exceeds the vehicle capacity. Since G' is even, two shortest chains of artificial non-required edges are added between the depot and its nearest vertex c (the artificial edges are represented by dotted lines). The feasible vehicle route $(depot, c, depot)$ is then detected and removed from G'. Since the remaining graph contains odd degree vertices, an optimal matching is added to G' which is now a cycle with a total demand of 4 units. Two shortest chains are added from the depot to c, and from the depot to f, before a feasible vehicle route $(depot, f, depot)$ can be obtained and removed from G'. Since the depot is now an isolated vertex, a copy of it is introduced into G'. Once the optimal matching determined in Step 3 has been added to G', one gets a feasible vehicle route $(depot, c, d, e, f, depot)$ on which only edges (c, d), (d, e) and (e, f) are serviced.

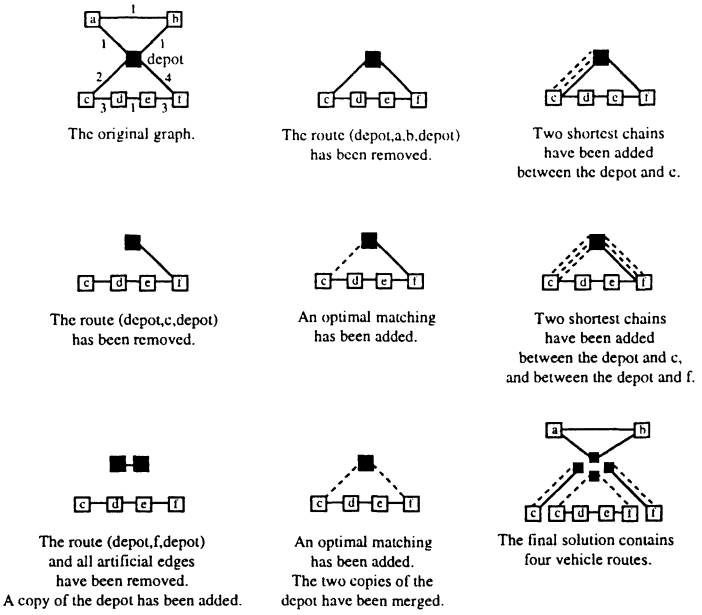


Figure 9.16

Pearn [40] has proposed a modified version of the above algorithm. He does not impose the restriction that the graph obtained when removing

feasible vehicle routes remains connected. Pearn's algorithm is outlined below.

Algorithm MODIFIED-CONSTRUCT-STRIKE

INPUT: An undirected graph $G = (V, E)$.

OUTPUT: A solution of the undirected CCPP.

Step 0. Set $G' := G$ and $G'' := G$.

Step 1. Choose an edge (v_0, v_i) incident to the depot in G'' and remove it from G''. Set $P = (v_0, v_i)$ and $v_{end} := v_i$ (v_{end} denotes the endpoint of P different from the depot).

Step 2. For each edge (v_{end}, v_j) in G'', determine the least quantity chain S_j in G'' that starts with (v_{end}, v_j) and links vend to the depot. Remove from G'' all edges (v_{end}, v_j) such that the total demand on $P + S_j$ is larger than the vehicle capacity Q. If v_{end} is an isolated vertex in G'', then set $G'' := G'$ and go to Step 4. Otherwise, determine the edge (v_{end}, v_j) in G'' that maximizes the total demand on S_j. Set $P := P + (v_{end}, v_j)$ and $v_{end} = v_j$. If v_{end} is not equal to the depot then repeat Step 2. Otherwise, remove all non-artificial edges of the feasible vehicle route P from G', and set $G'' := G'$.

Step 3. If all edges of G have been covered (i.e., G' is empty), then stop. If the depot is connected to all edges of G'' then return to Step 1. If the depot is an isolated vertex in G'', then add to G'' a shortest chain of artificial non-required edges between v_0 and the nearest non-isolated vertex in G''.

Step 4. If G'' is even and connected (not counting isolated vertices), then go to Step 5. Otherwise, transform G'' into an even connected graph by solving a minimum cost spanning tree problem and a minimum cost matching problem (see Section 2.2.1). All edges added to G'' are artificial and non-required. Return to Step 1.

Step 5. Choose an edge (v_0, v_i) incident to the depot in G'' and remove it from G''. Set $P = (v_0, v_i)$ and $v_{end} = v_i$.

Step 6. Remove from G'' all edges (v_{end}, v_j) such that the total demand on $P + (v_{end}, v_j)$ is larger than Q. If v_{end} is an isolated vertex in G'', then go to Step 7. Choose any edge (v_{end}, v_j) in G'', remove it from G'', set $P := P + (v_{end}, v_j)$, $v_{end} := v_j$, and repeat Step 6.

Step 7. Add to P the shortest chain C between v_{end} and v_0 in G. $P+C$ is a feasible vehicle route in which all non-artificial edges of P are serviced, while the artificial edges of P and all edges of C are

only traversed. Remove all non-artificial edges of P from G', set $G'' := G'$, and return to Step 3.

The choice, at Step 1, of an edge incident to the depot may considerably affect the quality of the final solution. For this reason, Pearn suggests to generate a set of solutions by considering each edge incident to the depot as the first edge of the first vehicle route. The final solution is then the best among all these solutions. The complexity of his algorithm is $\mathcal{O}(|E||V|^4)$ while the complexity of the original heuristic proposed by Christofides is $\mathcal{O}(|E||V|^3)$.

Figure 9.17 illustrates Pearn's algorithm on the same example as in Figure 9.16. There are four possible choices for the first edge of the first vehicle route. By choosing $(depot, a)$, one gets a first feasible vehicle route $(depot, a, b, depot)$. The remaining graph G'' contains a cycle with total demand $5 > Q$. Since G'' is even and connected, we move to Step 5. By choosing the edge $(depot, f)$ as first edge for the second vehicle route, Step 6 determines the path $P = (depot, f, e, d)$ and no more edges can be serviced without exceeding the vehicle capacity. The chain $C = (d, c, depot)$ of non-serviced edges is added to P in order to get a second feasible vehicle route. The remaining graph G'' is now connected but not even. Step 4 adds the edges $(depot, c)$ and (c, d) to G' and a third feasible vehicle route $(depot, c, d, c, depot)$ is determined in Steps 1 and 2. The total cost of this solution is two units better than the one of Figure 9.16 (26 instead of 28).

Pearn uses a special rule in Step 2 for attempting to construct a feasible vehicle route. He always chooses the edge that maximizes the total demand on the least return path to the depot. Other edge-selection rules can be preferred. Golden et al. [27] describe five such rules. Given a chain P that ends at vertex v_i, one can choose the edge (v_i, v_j) in the following ways:

rule 1: minimize the distance, c_{ij}, per unit remaining demand;

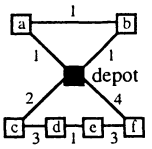
rule 2: maximize the distance, c_{ij}, per unit remaining demand;

rule 3: minimize the distance from v_j to the depot;

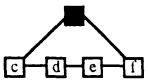
rule 4: maximize the distance from v_j to the depot;

rule 5: use rule 4 if the vehicle is less than half-full, and rule 3 otherwise.

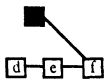
Golden et al. [27] have developed a heuristic method, called PATH-SCANNING, that uses these edge-selection rules. Their algorithm has an $\mathcal{O}(|V|^3)$ complexity and can be described as follows.



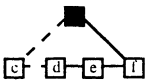
The original graph.



The first feasible vehicle route (depot,a,b,depot) has been removed from G'.



This chain P=(depot,f,e,d) cannot be extended without violating the vehicle capacity.



The chain C=(d,c,depot) has been added to P in order to get a second feasible vehicle route.

The chain P=(depot,f,e,d) has been removed from G'.

The chain (d,c,depot) has been added at Step 4 in order to get an even graph G", which now corresponds to the third feasible vehicle route.

Figure 9.17

Algorithm PATH-SCANNING

INPUT: An undirected graph $G = (V, E)$.

OUTPUT: A solution of the undirected CCPP.

Step 0. Set $G' := G$ and $G'' := G'$.

Step 1. If the depot is an isolated vertex in G'', then add to G'' a shortest chain $(v_0, ..., v_i)$ of artificial non-required edges between v_0 and the nearest non-isolated vertex v_i in G''. Set $P = (v_0, ..., v_i)$ and $v_{end} := v_i$. Otherwise, choose an edge (v_0, v_i) incident to the depot in G'' and remove it from G''. Set $P = (v_0, v_i)$, and $v_{end} := v_i$.

Step 2. Remove from G'' all edges (v_{end}, v_j) such that the total demand on $P + (v_{end}, v_j)$ is larger than Q. If v_{end} is an isolated vertex in G'', then go to Step 3. Choose an edge (v_{end}, v_j) in G'', according to one of the above mentioned edge-selection rules. Remove this edge from G'', set $P := P + (v_{end}, v_j)$, $v_{end} := v_j$, and repeat Step 2.

Step 3. Add to P the shortest chain C from v_{end} to the depot in G. $P + C$ is a feasible vehicle route in which all non-artificial edges of P are serviced, while the artificial edges of P and all edges of C are only traversed. Remove all non-artificial edges of P from G', and set $G'' := G'$. If all edges of G are covered, then stop, else return to Step 1.

The PATH-SCANNING algorithm generates a complete solution with each of the five edge-selection rules, and the final solution is the best among these five solutions. Pearn [40] has proposed a modified version of this algorithm where the edge-selection rule is chosen randomly at each step, following a given probability distribution. He then selects the best of several solutions generated by this process.

Golden et al. [27] have developed a solution method, called AUGMENT-MERGE, which is inspired by the Clarke and Wright algorithm [10] for the Vehicle Routing Problem. The AUGMENT-MERGE algorithm was originally proposed by Golden and Wong [26] and then modified and improved in [27]. We describe here the improved version.

Given a feasible vehicle route C, let P_C denote one of the two maximal chains of non-serviced edges on C that starts at the depot, and let v_C denote the second endpoint of P_C (see Figure 9.18). It follows from this definition that $P_C = (v_0, ..., v_C)$, and it may happen that $v_0 = v_C$ (if the depot is incident to a serviced edge).

A *merge* of two feasible vehicle routes C and C' is obtained by replacing $P_C + P_{C'}$ in $C + C'$ by a shortest chain P between v_C and $v_{C'}$ in G. Let L_C, $L_{C'}$ and L denote the total length of P_C, $P_{C'}$ and P, respectively. The saving induced by such a merge is equal to $L_C + L_{C'} - L$.

The two possible choices for P_C on C, and the two for $P_{C'}$ on C', induce four possible merges of C and C'. We only consider the merge that yields the largest saving (see Figure 9.18). Golden et al. [27] only consider merges where $v_C = v_{C'}$, while they do not impose that P_C and $P_{C'}$ are maximal chains. The AUGMENT-MERGE algorithm works as follows.

Algorithm AUGMENT-MERGE

INPUT: An undirected graph $G = (V, E)$.

OUTPUT: A solution of the undirected CCPP.

Step 1. For each edge (v_i, v_j) in G, construct a feasible vehicle route C_{ij} made of a shortest chain of non-serviced edges between the depot and v_i, the serviced edge (v_i, v_j), and a shortest chain of non-serviced edges between v_j and the depot.

(*Augment phase*)

Step 2. Starting with the longest vehicle route C_{ij}, and as long as vehicle capacity permits, change the status of traversed edges, from

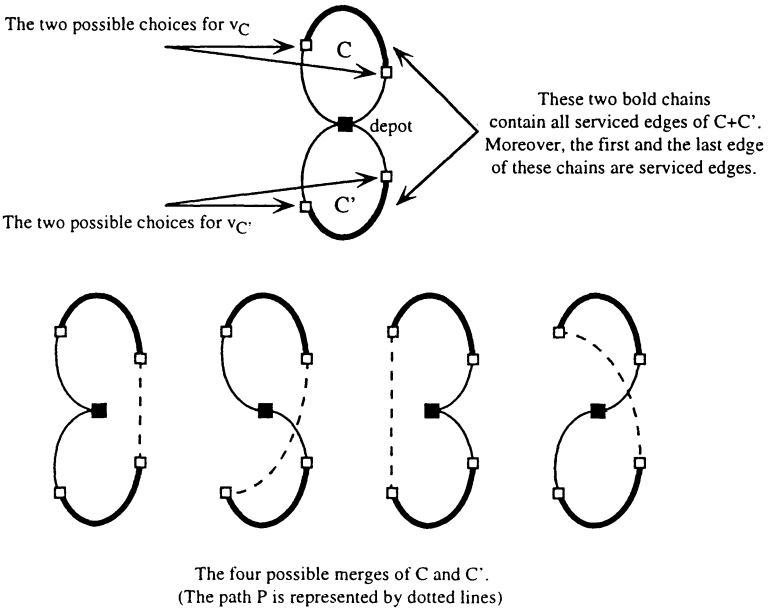


Figure 9.18

non-serviced to serviced, if these edges are covered by shorter vehicle routes. Remove the shorter vehicle routes whose unique serviced edge is now covered by a longer route.

(*Merge phase*)

Step 3. Subject to capacity constraints, evaluate the merge of any two vehicle routes. If the largest saving is positive then merge the two vehicle routes which yield the largest positive saving, and repeat Step 3. Else stop.

Chapleau et al. [6] have developed a heuristic method, called PARALLEL-INSERT, which is inspired by insertion procedures for the Traveling Salesman Problem. They use two complementary insertion strategies:

- given an edge, they determine the existing vehicle route into which this edge should be inserted in order to minimize the detour incurred;
- given a route, they determine which uncovered edge should be inserted next.

In addition to the capacity constraints, Chapleau et al. consider a limit L on the length of each vehicle route, and a limit M on the number of vehicle routes. Their algorithm can be outlined as follows.

Algorithm PARALLEL-INSERT

INPUT: An undirected graph $G = (V, E)$.

OUTPUT: A solution of the undirected CCPP.

Step 1. Determine the farthest edge (v_i, v_j) from the depot and create a feasible vehicle route servicing (v_i, v_j) (i.e., a route made of a shortest chain of non-serviced edges between the depot and v_i, the serviced edge (v_i, v_j), and a shortest chain of non-serviced edges between v_j and the depot).

(*First insertion strategy*)

Step 2. Select the farthest uncovered edge (v_a, v_b) from the depot. Among the existing routes with sufficient capacity, determine the route C for which the insertion of (v_a, v_b) creates the smaller detour and of which the total length after inserting (v_a, v_b) does not exceed the fixed upper bound L. If such a route has been determined, then insert (v_a, v_b) into C and repeat Step 2. If no such route exists, while the current number of routes is smaller than M, then create a new feasible vehicle route servicing edge (v_a, v_b), and repeat Step 2. Declare a route C "closed" if no additional uncovered edge can be serviced by C, without violating the vehicle capacity. All other existing routes are "open".

(*Second insertion strategy*)

Step 3. Select the open route C with the lowest load. Let E' be set of uncovered edges that can be added to C without exceeding the vehicle capacity. Select the edge (v_a, v_b) in E' that induces the smallest detour. If no such edge exists, or if the total length of C after inserting (v_a, v_b) exceeds the upper bound L, then declare route C "closed", else insert (v_a, v_b) into C. Repeat Step 3 until all edges are covered, or all routes are closed.

(*Termination check*)

Step 4. If all edges are covered then stop. Else select the farthest uncovered edge (v_a, v_b) from the depot, and create a new vehicle route servicing edge (v_a, v_b), and return to Step 2.

Chapleau et al. [6] do not describe how to insert an edge into a given vehicle route. Such an insertion can be performed, for example, by means of the ADD algorithm described in Section 2.2.5. One may prefer to use only Step 1 of the ADD algorithm, since the call to SHORTEN at Step 2 may be time consuming. The PARALLEL-INSERT algorithm has been applied to a school bus routing problem, and it has been observed that the use of additional ingredients may help finding better solutions. For

example, when entering Step 4, Chapleau et al. apply improvement procedures for possibly reducing the length of each vehicle route. Examples of such procedures are given in Section 2.2.5 (e.g., SHORTEN, DROP-ADD, 2-OPT-RPP). Notice that these improvement procedures can in fact be applied to each feasible vehicle route produced by any algorithm presented in this section.

In order to try to reduce the number of vehicle routes, Chapleau et al. propose to relax the length constraints by a fixed percentage (i.e., to slightly increase the upper bound L). Vehicle routes of length larger than L can then possibly be shortened by means of the above mentioned improvement procedures.

As other ingredient, Chapleau et al. suggest to eliminate the vehicle route with lowest load and to consider the edges it covered as non-covered. These edges are then re-inserted in the existing vehicle routes by means of Steps 2, 3 and 4 of the above algorithm.

Finally, Chapleau et al. describe a strategy that aims to avoid zigzag routes. For this purpose, they define a threshold length L' which is used as follows. Let C be a vehicle route and (v_a, v_b) an edge to be inserted into C. If the length of C is smaller than L', then (v_a, v_b) can be inserted anywhere on C. Otherwise, (v_a, v_b) can only be inserted between the depot and the last vertex incident to a serviced edge on C.

The last heuristic procedure described in this section is due to Pearn [41] and combines the ideas contained in the AUGMENT-MERGE and the PARALLEL-INSERT algorithms. In a first phase, feasible vehicle routes are generated following the approach used in the "Augment" phase of the AUGMENT-MERGE algorithm. In a second phase, the remaining non-covered edges are sequentially inserted into feasible vehicle routes by means of a procedure similar to the first insertion strategy used in the PARALLEL-INSERT algorithm. Pearn's algorithm, called AUGMENT-INSERT, can be described as follows.

Algorithm AUGMENT-INSERT

INPUT: An undirected graph $G = (V, E)$.

OUTPUT: A solution of the undirected CCPP.

Step 1. Let SP_{ij} denote the shortest chain between v_i and v_j in G, and let L_{ij} be its length. For each edge (v_i, v_j) in G, define $D_{ij} = L_{0i} + L_{j0}$. Set $G' := G$.

(*Augment phase*)

Step 2. Determine the edge (v_i, v_j) in G' with largest value D_{ij} and such that there is a cycle in G' containing (v_i, v_j) and the depot. If no such edge is found then go to Step 3. Else, determine the least cost cycle C in G' containing (v_i, v_j) and the depot, and consider C as a vehicle route on which (v_i, v_j) is the unique serviced edge. As long as vehicle capacity permits, change the status of traversed edges (v_a, v_b) on C from non-serviced to serviced, in descending order of D_{ab}. Let v_p and v_q be the first and last vertex on C incident to a serviced edge. Replace the chains of non-serviced edges linking the depot to v_p and the depot to v_q by SP_{0p} and SP_{q0}. Remove the serviced edges from G'. If all edges of G are covered (i.e., G' is empty), then stop. Else, repeat Step 2.

(*Insert phase*)

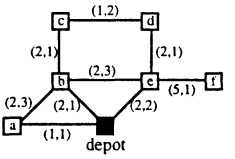
Step 3. Determine the non-covered edge (v_i, v_j) with largest value D_{ij}, and create the vehicle route $C = SP_{0i} + (v_i, v_j) + SP_{j0}$ on which (v_i, v_j) is the unique serviced edge.

Step 4. Let E' be the set of non-covered edges that can be added to C without violating the capacity constraints, and let (v_a, v_b) be the edge in E' with maximum value D_{ab}. Compute the detour incurred by the insertion of (v_a, v_b) between the depot and the first or last vertex incident to a serviced edge on C. If this insertion cost is larger than a fixed upper bound B, then remove (v_a, v_b) from E'. Else, insert (v_a, v_b) into C. Repeat Step 4 until E' is empty or both edges incident to the depot are serviced edges.

Step 5. Remove the serviced edges from G'. If G' is empty, then stop. Else, return to Step 3.

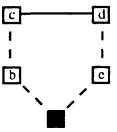
Pearn has proposed two versions of this algorithm. The above is Version I. Version II uses, at Step 2, the least quantity cycle (with respect to edge demand) instead of the least cost cycle. The upper bound B used at Step 4 is an input parameter that controls the length of the vehicle routes. Notice that the above algorithm considers the edges in decreasing order of D_{ij}, the idea being to favor servicing far-away edges.

The AUGMENT-INSERT algorithm is illustrated in Figure 9.19. In this example, all edge costs c_{ij} and demands q_{ij} are indicated as (c_{ij}, q_{ij}), and the vehicle capacity Q is equal to 5. Doted lines represent traversed but non-serviced edges. The farthest edge (i.e., the edge with maximum value D_{ij}) is (e, f), but there is no cycle in G containing (e, f) and the depot. Hence, we choose to serve (c, d) and build an initial cycle $(depot, b, c, d, e, depot)$ on which (c, d) is the unique serviced edge. The service is then extended on edges (b, c), (d, e) and $(b, depot)$, and one gets the first feasible vehicle route. The farthest uncovered edge which

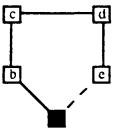


(v_i,v_j)	D_{ij}
(e,f)	9
(c,d)	8
(b,c)	6
(d,e)	6
(b,e)	4
(a,b)	3
(depot,b)	2
(depot,e)	2
(depot,a)	1

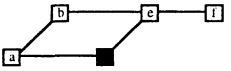
The original network and the edges in decreasing order of D_{ij}.



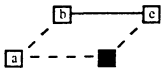
The cycle generated at the beginning of Step 2.



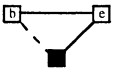
The first feasible vehicle route.



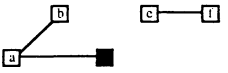
The remaining graph G'.



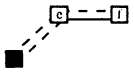
The cycle generated at the beginning of Step 2.



The second feasible vehicle route obtained by covering (depot,e) and by replacing (depot,a,b) by (depot,b).

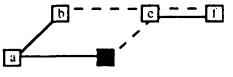


The remaining graph G'.



The cycle generated at the beginning of Step 3.

(a,b) has been inserted assuming that B≥3.



The third feasible vehicle route.

Figure 9.19

is in a cycle of G' going through the depot is (b,e). Hence Step 2 builds the cycle $(depot,a,b,e,depot)$. The service is then extended on edge $(depot,e)$, and the second feasible vehicle route is obtained by replacing the chain $(depot,a,b)$ by the shorter chain $(depot,b)$. We then enter the "Insert" phase. The farthest uncovered edge is (e,f), and Step 3 builds the cycle $(depot,e,f,e,depot)$, servicing edge (e,f). Then, assuming that the upper bound B on the insertion cost is at least equal to 3, the edge (a,b) is inserted and one gets the cycle $(depot,a,b,e,f,e,depot)$. By extending the service on edge $(depot,a)$ one gets the third feasible vehicle route. then extended on edge $(depot,e)$, and the second feasible vehicle route is obtained by replacing the chain $(depot,a,b)$ by the shorter chain $(depot,b)$. We then enter the "Insert" phase. The farthest uncovered edge is (e,f), and Step 3 builds the cycle $(depot,e,f,e,depot)$, servicing edge (e,f). Then, assuming that the upper bound B on the insertion cost is at least equal to 3, the edge (a,b) is inserted and one gets the cycle

$(depot, a, b, e, f, e, depot)$. By extending the service on edge $(depot, a)$ one gets the third feasible vehicle route.

3.2. TWO-PHASE CONSTRUCTIVE METHODS FOR THE CARP

Two-phase constructive methods belong to two different categories. The "route first-cluster second" strategy first constructs a giant Eulerian tour covering all required edges or arcs. Then, in a second phase, the tour is partitioned into feasible vehicle routes. On the opposite, "cluster first-route second" heuristics first determine a partition of the required edges and arcs into clusters, each having a total weight not exceeding the vehicle capacity Q. An uncapacitated RPP is then solved on each cluster. Both strategies are illustrated in the next sections in an undirected context. The algorithms described below can however be extended in order to deal with directed and mixed graphs.

3.2.1 Route first-cluster second algorithms.

The first phase of route first-cluster second algorithms is the construction of a giant Eulerian tour. All algorithms presented in Section 2 can be used to this end. We describe here several techniques that have been proposed for the "cluster" phase, that is for the partition of the Eulerian tour into feasible vehicle routes. An interesting algorithm has been proposed by Ulusoy in 1985 [47] who transforms the original graph into another one on which a shortest path problem has to be solved. Ulusoy's algorithm can be described as follows.

Algorithm ULUSOY-PARTITIONING

INPUT: An undirected graph G, and a covering tour T for R.

OUTPUT: A set of feasible vehicle routes covering R.

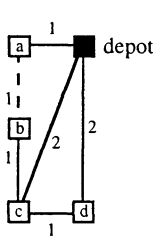
Step 0. If a required edge is traversed several times, then service it the first time it is traversed.

Step 1. Relabel the vertices in G so that the given tour T is equal to $(v_0, v_1, v_2, ..., v_t = v_0)$, where v_0 is the depot. Let r be the largest index of a vertex incident to a serviced edge on T. Construct a directed graph $G' = (V', A)$ with vertex set $V' = \{0, 1, ..., r\}$ and introduce an arc (a, b) in A $(b > a)$ if and only if the total load on the path $(v_a, ..., v_b)$ does not exceed Q. Remove all arcs (a, b) such that $b > a + 1$ and $(v_a, v_a + 1)$ or $(v_b - 1, v_b)$ is not a serviced edge on T. Define the cost c'_{ab} of (a, b) in G' as follows.

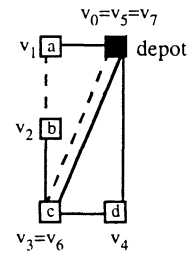
- If $b = a + 1$ and $(v_a, v_a + 1)$ is not a serviced edge on T, then set $c'_{ab} = 0$.

- If $b > a + 1$ or $(v_a, v_a + 1)$ is a serviced edge on T, then consider the chain $P_{ab} = (v_a, ...v_b)$ on T. If P_{ab} contains the depot, then add to P_{ab} a shortest chain between v_b and v_a in G; else, add to P_{ab} a shortest chain between the depot and a, and a shortest chain between the depot and b. In both cases, one gets a cycle C_{ab}, and the cost c'_{ab} of (a, b) in G' is defined as the total distance of C_{ab} in G.

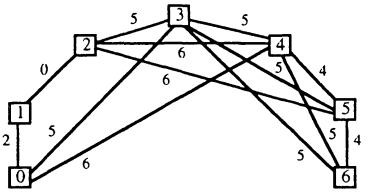
Step 2. Solve a shortest path problem from 0 to r in G'. Each arc (a, b) used in the shortest path corresponds to a feasible vehicle route on G.



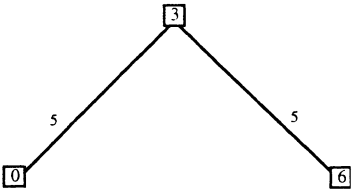
The original graph G.



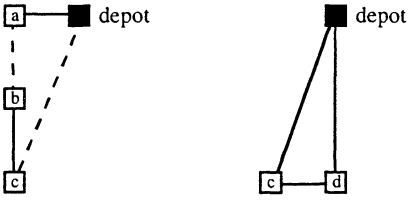
An Eulerian tour T on G.



The associated graph G'.
(All arcs (a,b) with a<b are directed from a to b)



A shortest path from 0 to 6 in G'.



The two corresponding feasible vehicle routes.

Figure 9.20

Ulusoy's algorithm is illustrated in Figure 9.20. In this example, all edges have a unit demand, and the vehicle capacity Q is equal to 3. The

values on the edges correspond to their length. Doted lines represent non-required or traversed but non-serviced edges. Since v_6 is the last vertex on T incident to a serviced edge, the auxiliary graph G' has 7 vertices. There is no arc $(0,2)$, $(1,3)$, $(1,4)$, $(1,5)$ and $(1,6)$ in G' since (v_1, v_2) is a non-serviced edge on T. Also, there is no arc $(0,5)$, $(0,6)$ and $(2,6)$ in G' since the corresponding chains on T have a total weight larger than $Q = 3$. As an example of computation of cost c'_{ab} , consider the arc $(4,6)$ in G': the corresponding path (v_4, v_5, v_6) on T contains the depot, and the cycle C_{46} is therefore obtained by adding to P_{46} the shortest chain (v_4, v_6) between v_4 and v_6. Hence, we get $c'_{46} = 5$. As other example, the cycle C_{23} induced by $(2,3)$ is obtained by adding the chains (v_0, v_1, v_2) and (v_3, v_0) to $P_{23} = (v_2, v_3)$, which implies that $c'_{23} = 5$. The shortest path from 0 to 6 in G' is the path $(0,3,6)$, and we therefore get two feasible vehicle routes: the arc $(0,3)$ corresponds to the vehicle route $(depot, a, b, c, depot)$ on which $(depot, a)$ and (b, c) are the unique serviced edges, while the arc $(3,6)$ corresponds to the vehicle route $(depot, c, d, depot)$ on which all edges are serviced.

A completely different procedure has been proposed by Hertz et al. [31]. Consider a tour $T = (v_0, v_1, ..., v_t = v_0)$ with total demand D larger than Q. In order to partition T into feasible vehicle routes, Hertz et al. first determine a vertex v_r such that the total demand on the chain $(v_0, v_1, ..., v_r)$ does not exceed the vehicle capacity Q, and the total demand on the chain $(v_r, ..., v_t = v_0)$ does not exceed $Q(\lceil D/Q \rceil - 1)$. Once v_r has been determined, a feasible vehicle route is constructed by adding to $(v_0, v_1, ..., v_r)$ the shortest chain from v_r to the depot. Let $r' \geq r$ be the smallest index such that $(v_{r'}, v_{r'+1})$ is a serviced edge on T. The procedure is reapplied on the tour T' obtained from T by replacing the chain $(v_0, v_1, ..., v_{r'})$ by a shortest chain of non-serviced edges between the depot and $v_{r'}$. This algorithm, called CUT, can be described more precisely as follows.

Algorithm CUT

INPUT: An undirected graph G, and a covering tour T for R.

OUTPUT: A set of feasible vehicle routes covering R.

Step 0. If a required edge is traversed several times, then service it the last time it is traversed.

Step 1. Relabel the vertices in G so that the given tour T is equal to $(v_0, v_1, v_2, ..., v_t = v_0)$, where v_0 is the depot. Let D be the total demand on T. If $D \leq Q$ then stop (T is a feasible vehicle route).

Step 2. Determine the largest index s such that (v_{s-1}, v_s) is a serviced edge, and the total demand on $(v_0, v_1, ..., v_s)$ does not exceed Q.

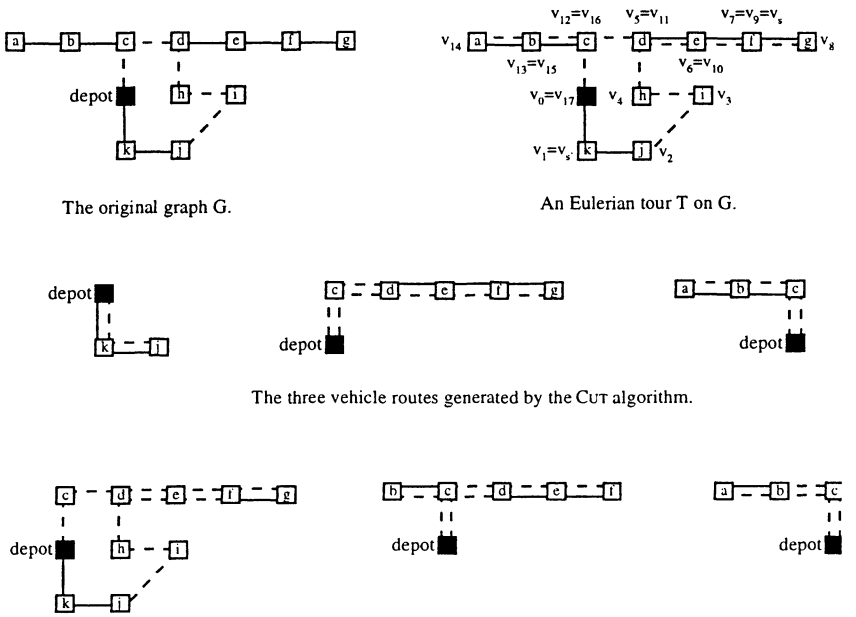


Figure 9.21

Determine the smallest index s' such that $(v_{s'-1}, v_{s'})$ is a serviced edge, and the total demand on $(v_{s'}, ..., v_t = v_0)$ does not exceed $Q(\lceil D/Q \rceil - 1)$. If $s' \geq s$ then set $r = s$ and go to Step 3. For each r in $[s', s]$ such that (v_{r-1}, v_r) is a serviced edge, do the following:

- determine the smallest index $r' \geq r$ such that $(v_{r'}, v_{r'+1})$ is a serviced edge on T;
- consider the path $P_r = SP_{r0} + SP_{0r'}$ (where SP_{ij} denotes the shortest chain between v_i and v_j in G);
- compute δ_r equal to the difference between the length of P_r and the length of the path $(v_r, ..., v_{r'})$ on T. Select the index r in $[s', s]$ with smallest value δ_r.

Step 3. Construct the feasible vehicle route $(v_0, ..., v_r) + SP_{r0}$, on which the serviced edges are those that are serviced on the chain $(v_0, ..., v_r)$ of T. Determine the smallest index $r' \geq r$ such that $(v_{r'}, v_{r'+1})$ is a serviced edge on T. Replace $(v_0, ..., v_{r'})$ by $SP_{0r'}$ on T, and return to Step 1.

The CUT algorithm is in fact a generalization of a procedure proposed by Greistorfer in 1994 [28] who always sets $r = s$ in Step 2. Figure 9.21 illustrates the improvement that can be obtained by allowing other

choices for index r. In this example, all edges have a unit demand and a unit cost, and the vehicle capacity is equal to 3. Since the total demand D on T is equal to 7, at least 3 vehicles are needed. The first vehicle must at least reach vertex $v_{s'} = v_1$ before returning to the depot, else 4 vehicles will be needed. Also, the capacity constraints impose that the first vehicle cannot cover more than three edges, which means that $v_s = v_9$. Hence, r is chosen in the interval $[1, 9]$. If the first vehicle return to the depot at vertex v_1, then we compute $\delta_1 = 2$. Similarly, we compute $\delta_2 = 1$ and $\delta_9 = 8$, and we therefore choose $r = 2$ (while Greistorfer's algorithm set $r = 9$). The total length of the vehicle routes generated by the CUT algorithm is equal to 20, while Greistorfer's algorithm generates a solution of total length 29.

Notice that it may happen that the index s' is strictly larger than s. Indeed, consider a tour $T = (v_0, v_1, v_2, v_0)$ on which each edge is serviced and has a demand of 2 units, while the vehicle capacity Q is equal to 3. A vehicle cannot service more than one edge, which means that $v_s = v_1$. On the other hand, $Q(\lceil D/Q \rceil - 1)$ is equal to 3, which means that $v_{s'} = v_2$. Hence, $s' = 2 > 1 = s$. Such a situation occurs because there is no solution with $\lceil D/Q \rceil = 2$ feasible vehicle routes. In such a case, the above algorithm set $r = s = 1$, and a first vehicle covers the edge (v_0, v_1). Two additional vehicles are needed to cover (v_1, v_2) and (v_2, v_0).

Improvement on the CUT algorithm have been proposed by Hertz et al. [31]. For example, procedure SHORTEN, described in Section 2.2.5, is applied on each feasible vehicle route generated at Step 3. For more details, the reader is referred to [31].

To conclude this section, let us mention that other partitioning procedures have been proposed for performing the "cluster" phase of route-first-cluster second algorithms. For example, Win [48] suggests to use a next fit bin packing heuristic.

3.2.2 Cluster first-route second algorithms.

Cluster first-route second strategies first determine a partition of the required edges and arcs into clusters. Vehicle routes are then constructed in each cluster. Any algorithm of Section 2 can be applied for the second phase of these algorithms. We here describe a technique proposed by Benavent et al.[4] for partitioning the required edges into clusters. This technique, called CYCLE-ASSIGNMENT, determines the assignment of the required edges to the vehicles by solving a generalized assignment problem (GAP). It is inspired by the famous algorithm proposed by Fisher and Jaikumar for the VRP [20], and seems to be more efficient than the simple greedy approach proposed by Win [48].

Algorithm CYCLE-ASSIGNMENT

INPUT: An undirected graph G, and a given number K of available vehicles.

OUTPUT: An assignment of the required edges to the vehicles.

Step 1. Determine a set $\{s_1, ..., s_K\}$ of K seed vertices, selecting for each $k = 1, ..., K$ a vertex s_k such that the product of the distances from s_k to the seed vertices $s_1, ..., s_{k-1}$ and to the depot is maximum. Perform interchanges between seed vertices and non-seed vertices as long as such interchanges increase the product of the distances among all seed vertices and the distances between the seed vertices and the depot.

Step 2. Solve a minimum cost spanning tree problem and a minimum cost matching problem in order to transform G_R into an even connected graph G' (see Section 2.2.1). For each $k = 1, ..., K$ construct the graph G'_k containing only vertex s_k. Declare all vehicles "open".

Step 3. Select the open vehicle k with largest residual capacity, and determine in G' the minimum load cycle containing at least one vertex of G'_k. If no such cycle exists, or if its load exceeds the available residual capacity on vehicle k, then declare vehicle k "closed". Otherwise, add this cycle to G'_k, and remove it from G'. Repeat Step 3 until all vehicles are closed. Let D_k denote the shortest distance from the depot to s_k in G.

Step 4. Let $P_{k,ij}$ denote the minimum load chain between v_i and v_j in G'_k , and let $L_{k,ij}$ be its load. Moreover, let r_k denote the residual capacity on vehicle k. If two vertices v_i and v_j are linked by a chain P in G' of load L, and if there exists a vehicle k, visiting v_i and v_j, and such that $0 < L - L_{k,ij} < r_k$, then interchange the edges of P and $P_{k,ij}$ in G' and G'_k. Perform such interchanges as long as the total demand of the required edges in G' can be reduced.

Step 5. For each vehicle k, and each required edge e not in G'_k, determine the shortest chain $C_{k,e}$ in G that contains edge e and links the depot to s_k. Let $\delta_{k,e}$ denote the difference between the length of $C_{k,e}$ and D_k, and let $q_e = q_{ij}$ denote the demand on edge $e = (v_i, v_j)$. For all required edges e in G'_k, set $\delta_{k,e} = 0$. For each vehicle k, and each required edge e in G, consider the Boolean variable $x_{k,e}$ defined as 1 if the edge e is assigned to vehicle k, and 0 otherwise.

Solve the following generalized assignment problem (GAP) (by means of Ross and Soland's algorithm [43], for example):

$$\text{minimize} \quad \sum_{k=1}^{K} \sum_{e \in R} \delta_{k,e} x_{k,e}$$

subject to

$$\sum_{k=1}^{K} x_{k,e} = 1, \quad \text{for all } e \in R$$

$$\sum_{e \in R} q_e x_{k,e} \leq Q, \quad \text{for all } k = 1, ..., K$$

$$x_{k,e} = [0, 1], \quad \text{for all } k = 1, ..., K \text{ and } e \text{ in } R$$

The above heuristic has an interesting feature. A partial clustering is first determined in Steps 1 to 4. The information contained in this partial solution passes to Step 5 through the costs $\delta_{k,e}$. Indeed, $\delta_{k,e} = 0$ for all edges attached to vehicle k at the end of Step 4. The other values of $\delta_{k,e}$ are positive and approximate the insertion cost of edge e into the k-th vehicle route. These costs $\delta_{k,e}$ provide an incentive for the GAP to retain the clusters obtained in the first steps.

3.3. META-HEURISTICS FOR THE CARP

In recent years, several meta-heuristics have proved highly efficient for the solution of combinatorial optimization problems [2] [42]. In particular, local search techniques (e.g., simulated annealing and tabu search) appear to be quite successful when applied to a broad range of practical problems.

Consider a set S of feasible solutions of an optimization problem, and assume that a *neighborhood* $N(s) \subseteq S$ is defined for each solution s of S. Local search techniques are iterative procedures that aim to find a solution s in S which minimizes a real-valued objective function $f : S \rightarrow \mathbb{R}$. The iterative process starts from an initial solution in S, and given any solution s, the next solution s' is chosen in $N(s)$. Stopping rules are defined for interrupting the optimization process. Local search techniques can be outlined as follows.

Local search techniques

Step 0. Choose an initial solution s in S. Set Best_Solution := s.

Step 1. Find a "good" solution s' in $N(s)$. If $f(s') < f$ (Best_Solution) then set Best_Solution $:= s'$. Set $s := s'$. If a stopping condition is met, then stop. Else repeat Step 1.

Finding the best solution s' in $N(s)$ (i.e., such that $f(s') \leq f(s'')$ for any s'' in $N(s)$) may be a very difficult problem, perhaps not simpler than the original optimization problem. In such cases, heuristic procedures are used, at each iteration, in order to generate a "good" (but not necessarily "best") solution in $N(s)$.

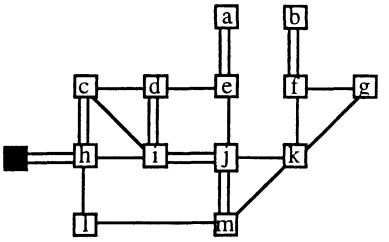
In this spirit, Li [35] has applied with limited success simulated annealing and tabu search methods to a road gritting problem. In 1994, Eglese [18] has developed a simulated annealing approach that can deal with multiple depot locations and several side constraints. Eglese uses an interesting concept, called *cyclenode graph* which can be described as follows.

Let G be an even connected graph G. Eglese suggests to decompose G into cycles, according to a checkerboard pattern (see Figure 9.22*(b)*). The cyclenode graph N is then created as follows. A vertex in N represents a cycle in the above mentioned decomposition. An edge is included between two vertices in N if the corresponding cycles have a common vertex in the original graph G. A vertex representing the depot is then added to N and linked to each other vertex in N. The decomposition procedure as well as the construction of N are illustrated in Figure 9.22 which reproduces the example given in [18].

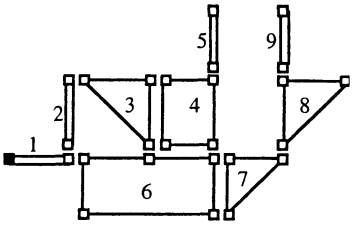
Each tree containing the depot vertex in the cyclenode graph corresponds to a tour in the original graph G. An example is depicted in Figure 9.22*(d)*.

Eglese [18] has developed a simulated annealing algorithm that works directly on the cyclenode graph. A solution of the CARP is defined as a set of trees in N rooted at the depot node. Neighbor solutions are obtained by performing simple modifications on at most two trees. Eglese mentions that the main difficulty encountered with this model is that not all possible routes in G can be represented by a tree in the cyclenode graph N. For example, in Figure 9.22, the route $(depot, h, c, i, h, depot)$ has no corresponding tree in N.

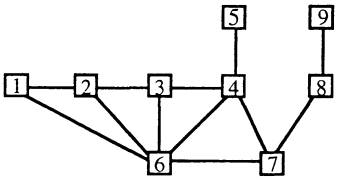
Recently, Hertz et al. [31] have developed a tabu search algorithm, called CARPET, for the solution of the CARP. The CARPET algorithm is based on several procedures described in the previous sections. For example, an initial solution is obtained by means of a route first-cluster second heuristic a giant tour T covering all edges and arcs is first con-



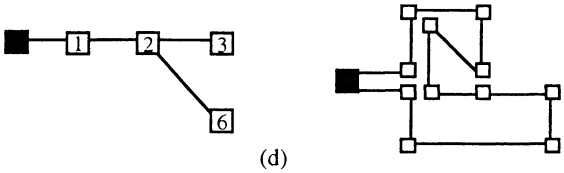
(a)
An even connected graph G.



(b)
A decomposition of G into cycles, according to a checkerboard pattern.



(c)
The associated cyclenode network, without the depot vertex.



(d)
A tree in N and its corresponding tour in G.

Figure 9.22

structed by means of algorithms presented in Section 2; this tour T is then partitioned into feasible vehicle routes by means of the CUT algorithm.

Solutions violating the capacity constraints can be visited during the search process. These infeasible solutions are however penalized by the objective function f. More precisely, let S be a solution, $L(S)$ the total length of the vehicle routes in S, and $E(S)$ the total excess demand with respect to vehicle capacity. The value $f(S)$ of a solution S is set equal to $L(S) + \alpha E(S)$, where α is a self-adjusting penalty parameter. Initially, α is set equal to the average length of a shortest chain between the depot and the extremity of all required edges. Every 5 iterations of the CARPET algorithm, α is halved if all previous 5 solutions were feasible, and doubled if they were all infeasible.

A neighbor solution S' is obtained from a solution S by moving the service of an edge e from a route T to another route T' of S. The service of e is removed from T by means of the DROP algorithm while it is introduced into T' by means of the ADD algorithm (see Section 2.2.5).

In order to avoid cycling and being trapped in local optima, a tabu list $\mathcal{T}$ registers the most recent moves that have been performed during the search process. If the service of an edge has been moved from a route T to a route T', then the pair (e, T) enters the list $\mathcal{T}$, and the service of e cannot be reintroduced into T during a given number of iterations.

The main structure of the CARPET algorithm can be outlined as follows.

Algorithm CARPET

INPUT: An undirected graph $G = (V, E)$.

OUTPUT: A solution of the undirected CARP.

Step 1. Construct a covering tour T for R by means of one of the heuristics described in Section 2.2.1. Partition T into a set S of feasible vehicle routes by means of the CUT algorithm. Set the penalty parameter α equal to the average length of a shortest chain between the depot and the extremity of all required edges. Set $\mathcal{T}$ equal to the empty set. Set Best_L := $L(S)$, Best_f := $L(S)$ and Best_Solution := S.

Step 2. Determine the neighborhood $N(S)$ of S as follows. For each required edge, consider the route T servicing it, as well all other routes T' in S. Remove the service from T by means of the DROP algorithm, and introduce it into T' by means of the ADD algorithm.

Step 3. Choose the non-tabu neighbor S' in $N(S)$ which minimizes the objective function f(i.e., $f(S') \leq f(S'')$ for all S'' in $N(S) \setminus \mathcal{T}$). If S' satisfies the capacity constraints (i.e., $E(S') = 0$) and $L(S') <$ Best_L, then set Best_L := $L(S\prime)$ and Best_Solution := S'. If $f(S') <$ Best_f then set Best_f := $f(S')$. Set S' := S, and update the tabu list $\mathcal{T}$ and the penalty parameter α.

Step 4. If no stopping condition is met, then return to Step 2. Else stop, Best_Solution is the best solution encountered during the search process.

Several additional ingredients are described in [31]. For example, diversification of the search process is obtained by performing a three steps procedure. A set of routes is first merged in order to obtain a giant (possibly not feasible) route T. This giant route is then shortened, using

procedure SHORTEN (see Section 2.2.5). Finally T is decomposed into feasible vehicle routes by means of the CUT algorithm.

The search process can also be intensified in some promising regions of the search space. This is done by applying improvement procedures such as DROP-ADD or 2-OPT-RPP on each vehicle route of the best solutions encountered during the search process. For more details, the reader is referred to [31].

One interesting line of research would be the development of hybrid meta-heuristics for the solution of the CARP. Such techniques combine population based methods (e.g., genetic algorithms, scatter search) with local search techniques, and have already proved highly successful in other areas of combinatorial optimization.

4. CONCLUSION

Arc routing problems occur in a wide variety of practical problems with different constraints and objectives. We have described in this chapter algorithms that can be used to solve basic problems. When additional constraints must be taken into account, the known heuristic methods must be adapted or extended. The algorithms described in this chapter should therefore be considered as skeletons of more specialized algorithms to be designed for each particular real life problem. Such adaptations or extensions have already been described in several papers. For example, Dror et al. [14] [15] have solved a problem in which the edges and arcs must be serviced in an order that respects a given precedence relation. Multiple depot locations [18] have been handled by using the cyclenode graph model described in Section 3.3. Time windows as well as time limits on the routes can also be taken into account as shown, for example in [19] [45]. Such real-life applications are described in more details in another chapter of this book.

The reader interested in developing its own heuristic method for the solution of a given arc routing problem should now have the basic ingredients into his hands which should help him designing the most adequate heuristic method.

References

[1] van AARDENNE-EHRENFEST, T. and N.G. de BRUIJN. 1951. Circuits and trees in oriented linear graphs. Simon Stevin 28, 203-217.

[2] AARTS, E. and J.K. LENSTRA (ED.). 1997. Local Search in Combinatorial Optimization. Wiley, West Sussex.

[3] BALL, M.O. and M.J. MAGAZINE. 1988. Sequencing of Insertions in Printed Circuit Board Assembly. Operations Research 36 (2), 192-201.

[4] BENAVENT, E., V. CAMPOS, A. CORBERAN, and E. MOTA. 1990. The Capacitated Arc Routing Problem: A Heuristic Algorithm. Qüestiió 14 (1-3), 107-122.

[5] BELTRAMI, E. and L. BODIN. 1974. Networks and Vehicles Routing for Municipal Waste Collection. Networks 4 (1), 65-94.

[6] CHAPLEAU, L., J.A. FERLAND, G. LAPALME, and J.-M. ROUSSEAU. 1984. A Parallel Insert Method for the Capacitated Arc Routing Problem. Operations Research Letters 3 (2), 95-99.

[7] CHRISTOFIDES, N. 1973. The Optimum Traversal of a Graph. Omega 1 (6), 719-732.

[8] CHRISTOFIDES, N. 1976. Worst-Case Analysis of a New Heuristic for the Traveling Salesman Problem. Report N°388, Graduate School of Industrial Administration, Carnegie Mellon University, Pittsburgh.

[9] CHRISTOFIDES, N., V. CAMPOS, A. CORBERAN, and E. MOTA. 1986. An Algorithm for the Rural Postman Problem on a Directed Graph. Math. Program. Study 26, 155-166.

[10] CLARKE, G. and J. WRIGHT. 1964. Scheduling of Vehicles from a Central Depot to a Number of Delivery Points. Operations Research 12, 568-581.

[11] CORBERAN, A., R. MARTI, and A. ROMERO. 1997. A Tabu Search Algorithm for the Mixed Rural Postman Problem. Research Report. Dept. of Statistics and OR. University of Valencia. Spain.

[12] CROES, G.A. 1958. A Method for Solving Traveling-Salesman Problems. Operations Research 6, 791-812.

[13] DERIGS, U. and A. METZ. 1991. Solving (Large Scale) Matching Problems. Mathematical Programming 50, 113-121.

[14] DROR, M., H. STERN, and P. TRUDEAU. 1987. Postman Tour on a Graph with Precedence Relation on Arcs. Networks 17 (3), 283-294.

[15] DROR, M. and A. LANGEVIN. 1997. A Generalized Traveling Salesman Problem Approach to the Directed Clustered Rural Postman Problem. Transportation Science 31 (2), 178-192.

[16] EDMONDS, J. 1967. Optimum Branching. J. Res. Natl. Bur. Stand., section B. 71, 233-240.

[17] EDMONDS, J. and E.L. JOHNSON. 1973. Matching, Euler Tours and the Chinese Postman. Mathematical Programming 5, 88-124.

[18] EGLESE, R.W. 1994. Routing Winter Gritting Vehicles. Discrete Appl. Math. 48 (3), 231-244.

[19] EGLESE, R.W. and L.Y.O. LI. 1996. A Tabu Search based Heuristic for Arc Routing with a Capacity Constraint and Time Deadline. Meta-Heuristics: Theory and Applications, 633-649.

[20] FISHER, M. and R. JAIKUMAR. 1981. A Generalized Assignment Heuristic for Vehicle Routing. Networks 11, 109-124.

[21] FLEISCHNER, H. 1990. Eulerian Graphs and Related Topics, Part 1, Volume 1. Annals of Discrete Mathematics 45, North-Holland, Amsterdam.

[22] FREDERICKSON, G.N., M.S. HECHT, and C.E. KIM. 1978. Approximation Algorithms for some Routing Problems. SIAM J. Comput. 7 (2), 178-193.

[23] FREDERICKSON, G.N. 1979. Approximation Algorithms for some Postman Problems. J. ACM 26 (3), 538-554.

[24] GALIL, Z., S. MICALI, and H. GABOW. 1986. An $\mathcal{O}$(EV logV) Algorithm for Finding a Maximal Weighted Matching in General Graphs. SIAM J. Comp. 15, 120-130.

[25] GENDREAU, M., A. HERTZ, and G. LAPORTE. 1992. New Insertion and Postoptimization Procedures for the Traveling Salesman Problem. Operations Research 40. 1086-1094.

[26] GOLDEN, B.L. and R.T. WONG. 1981. Capacitated Arc Routing Problems. Networks 11 (3), 305-315.

[27] GOLDEN, B.L., J.S. DEARMON, and E. BAKER. 1983. Computational Experiments with Algorithms for a Class of Routing Problems. Comput. Oper. Res. 10 (1), 47-59.

[28] GREISTORFER, P. 1994. Algorithms and Implementations for the Mixed Capacitated Chinese Postman Problem. Working Paper 33, Department of Business, University of Graz.

[29] GUAN, M. 1984. On the Windy Postman Problem. Discrete Applied Mathematics. 9, 41-46.

[30] HERTZ, A., G. LAPORTE, and P. NANCHEN. 1999. Improvement Procedures for the Undirected Rural Postman Problem. INFORMS Journal on Computing. 11, 53-62.

[31] HERTZ, A., G. LAPORTE, and M. MITTAZ. 2000. A Tabu Search Heuristic for The Capacitated Arc Routing Problem. Operations Research. 48(A).

[32] LAWLER, E.L. 1976. Combinatorial Optimization: Networks and Matroïds. Holt, Rinehart & Winston, New-York.

[33] LENSTRA, J.K. and A.H.G. RINNOOY KAN. 1976. On General Routing Problems. Networks 6 (3), 273-280.

[34] LIN, Y. and Y. ZAO. 1988. A new Algorithm for the Directed Chinese Postman Problem. Comput. Oper. Res. 15 (6), 577-584.

[35] LI, L.Y.O. 1992. Vehicle Routeing for Winter Gritting. Ph.D. Dissertation, Department of OR & OM, Lancaster University.

[36] LUKKA, A. and L. SALMINEN. 1987. Comparison of Three Heuristics for an Arc Routing Problem. Research Report 6/1987, Department of Information Technology, Lappeenranta University of Technology, Finland.

[37] MINIEKA, E. 1978. Optimization Algorithms for Networks and Graphs. Marcel Dekker, Inc., New York.

[38] MINIEKA, E. 1979. The Chinese Postman Problem for Mixed Networks. Management Science 25 (7), 643-648.

[39] PAPADIMITRIOU, C.H. 1976. On the Complexity of Edge Traversing. J. ACM 23 (3), 544-554.

[40] PEARN, W.L. 1989. Approximate Solutions for the Capacitated Arc Routing Problem. Comput. Oper. Res. 16 (6), 589-600.

[41] PEARN, W.L. 1991. Augment-Insert Algorithms for the Capacitated Arc Routing Problem. Comput. Oper. Res. 18 (2), 189-198.

[42] REEVES, C.R. (ED.). 1993. Modern Heuristic Techniques for Combinatorial Optimization. Blackwell, Oxford.

[43] ROSS, G. and R. SOLAND. 1975. A Branch and Bound Algorithm for the Generalized Assignment Problem. Mathematical Programming 8, 91-103.

[44] ROSENKRANTZ, D.J., R.E. STEARNS, and P.M. LEWIS III. 1977. An Analysis of Several Heuristics for the Traveling Salesman Problem. SIAM Journal on Computing 6, 563-581.

[45] ROY, S. and J.M. ROUSSEAU. 1989. The Capacitated Canadian Postman Problem. INFOR 27 (1), 58-73.

[46] SU, S.I. 1992. The General Postman Problem - Models and Algorithms. Ph.D. Dissertation, College of Business & Management, University of Maryland, Md.

[47] ULUSOY, G. 1985. The Fleet Size and Mix Problem for Capacitated Arc Routing. Eur. J. Oper. Res. 22 (3), 329-337.

[48] WIN, Z. 1988. Contribution to Routing Problems. Doctoral Dissertation, University of Augsburg.

[49] WIN, Z. 1989. On the Windy Postman Problem on Eulerian Graphs. Mathematical Programming 44, 97-112.

III

APPLICATIONS

Chapter 10

ROADWAY SNOW AND ICE CONTROL

James F. Campbell
University of Missouri - St. Louis

André Langevin
GERAD and École Polytechnique

1. Introduction 389
2. Brief History of RSIC 390
3. Characteristics of Arc Routing for RSIC 392
4. Solution Approaches 395
5. Early Work 396
6. Recent Work 400
   6.1 CASPER 400
   6.2 GeoRoute 406
   6.3 Other Recent Work 411
7. Future 413

1. INTRODUCTION

Roadway snow and ice control (RSIC) is one of the most complex and fascinating venues for arc routing applications. Arc routing problems occur in several different aspects of RSIC, including plowing roadways and sidewalks, spreading chemicals (salt and other de-icers) and abrasives (sand and stone), and in using snowblowers to load snow into trucks for hauling to disposal sites. The limited progress of computerized routing packages in this area (Office of the Legislative Auditor, 1995) highlights the difficulty of the problems and the weaknesses of theoretical arc routing models for snow and ice control. Part of the difficulty is due to the complex operational and infrastructure constraints, especially in urban areas. The dynamic nature of the problems also adds complexity, as conditions can vary dramatically over time and space. Furthermore, a wide variety of equipment with different operating characteristics can be deployed for RSIC.

This chapter opens with a brief history of roadway snow and ice control and then details the dynamic and complex characteristics of arc routing in this arena. Some early work on analytical approaches to winter arc routing beginning in the 1970s are discussed next. Then two major successful applications are described in more detail. The first is the CASPER system, which is currently used, and has been extensively tested, in the State of Indiana (U.S.A.). The second system is based on the GeoRoute software and it has been implemented in a variety of locations in Canada and the U.K. The chapter also includes a brief discussion of other more recent works, and it closes with a look to the future.

2. BRIEF HISTORY OF RSIC

Roadway snow and ice control has a surprisingly long history that even predates the automobile. Minsk (1970) and Gray and Male (1981) provide some historical perspectives on snow removal, and McKelvey (1995) provides considerable details on early urban snow and ice control, especially in the northeastern United States. This section highlights some of the history of RSIC in the U.S. In the last half of the 19th century, as cities developed into centers of commerce, safe winter travel by horse drawn vehicles and pedestrians on city streets became a pressing need. Horse drawn snow plows were used as early as 1862 in Milwaukee, and in the 1870s private street car companies plowed the snow off their tracks, but often left it in large piles and ridges that interfered with pedestrian travel. Winter travel problems were so severe that in the winter of 1873-74 the New York Times called for the construction of elevated lines or subways to avoid the snow.

Because private snow removal efforts often led to conflicts as snow removed by one business might block another, urban snow and ice control came to be viewed as a public service. (New York City established a Department of Street Cleaning with responsibilities including snow removal in 1881.) Before motor vehicles, horse-power and manpower were used to clear snow, load it into carts, and haul it to disposal sites, which included rivers, lakes and the city sewer system. The limitations of this arrangement were well demonstrated in the "Blizzard of 1888", which dropped up to four feet of snow in the northeastern U.S. and caused over 400 deaths. This storm paralyzed New York City (population 2 million) with 20 inches of snow, stopped the elevated trains, knocked out communication and electricity, and prompted the New York Times to again urge building of a subway and underground conduits for utilities. From this one storm, the City hauled 76,000 loads of snow (primarily to the East River), and private street car companies dumped even more. Following the Blizzard of 1888, many cities became more systematic about snow and ice control, including dividing their territory into sectors and

assigning plows to each sector. Many cities, including New York, Boston and Chicago, also sought relief by building elevated trains and subways.

As automobiles and trucks rose to prominence in the 20th century, they focused greater attention on needs for roadway snow and ice control, and they also provided powerful new tools for snowfighting. Motorized trucks, road scrapers, plows, snow loaders and caterpillar tractors were used prior to 1920, and subsequent decades saw mechanized snow fighting quickly replace manual labor. The use of salt for snow and ice control (though common in European cities) was slower to catch on in North America (in part due to its hazards for horses!). Salt use became widespread with the development of practical mechanical spreaders in the 1940s, and with the greater demand for safe driving conditions. Salt was readily adopted because it both cut costs for snow and ice control, and reduced the need for hauling snow. The widespread use of the automobile in the last half of the 20th century, along with population shifts, made snow and ice control a crucial and complex winter activity in urban and rural areas. The high levels of service demanded led to greater usage of salt, which in turn led to a variety of difficult issues involving environmental hazards, as well as corrosion to vehicles and infrastructure. Further obstacles to efficient snowfighting have been added by large numbers of parked cars, and roadway geometries in suburbs.

The advent of satellite based weather forecasting in the 1960s and better public awareness of snow and ice hazards have helped in snowfighting efforts, but large storms continue to overwhelm available resources. For example, storms in Chicago in 1967 paralyzed the metropolitan area, causing shortages of food and medical supplies. Looting and rioting resulted in some parts of the city, and helicopters were used to reach otherwise inaccessible hospitals. The snow removal bill for cleanup was ten times the annual budget for snow and ice control. More recently, the "Blizzard of 1996" (January 1996) in the northeastern United States cost $20 billion (Byron Lord, 1996) and forced the closure of U.S. federal government offices due to the inability of Washington D.C. public works agencies to maintain passable roadways.

Today, Sapporo, Japan has perhaps the most sophisticated snowfighting and snow disposal system. Sapporo faces special problems as a large city (population of 1.7 million) which receives an average of five meters of snow each year! In addition to spreading abrasives and chemicals, and plowing snow, Sapporo also uses road heating systems, snow melting tanks, and snow flowing gutters along streets (using river water or treated sewage) for dumping snow, rather than hauling it to disposal sites (City of Sapporo 1991).

3. CHARACTERISTICS OF ARC ROUTING FOR RSIC

Arc routing problems for roadway snow and ice control (RSIC) are especially difficult because of the dynamic environment and the complexities of equipment, infrastructure and operations. The two major roadway snow and ice control operations are plowing, and spreading chemicals and abrasives, both of which require solving arc routing problems to direct vehicles through a network of roadways. Arc routing is also required when using snowblowers to load snow into trucks to be hauled to disposal sites, as is common in urban areas with heavy snowfall. Major storms that cause substantial accumulations of snow and ice may require plowing, spreading and snow loading operations over several days. In contrast, rain followed by freezing temperatures may only require precautionary spreading operations.

RSIC arc routing problems differ from many other arc routing applications in a number of important ways, and yet there is also tremendous variation in RSIC operations themselves, due to vast differences in climate, levels of service, network complexity and size. However, in all cases service should be provided efficiently, rapidly, equitably and simultaneously across a large and complex network in the presence of other vehicles and adverse weather conditions. This section highlights some of the key characteristics of RSIC that affect arc routing.

Roadway snow and ice control is a very dynamic environment both spatially and temporally. Weather conditions (temperature, wind, precipitation type and amount, accumulation rate, total accumulation, etc.) can vary significantly even over small distances due to topographic features and water sources, and can vary dramatically over larger distances. Some spatial variations are persistent, such as the large "lake-effect" snows from the Great Lakes in North America, while others depend on the paths of particular storms. Weather conditions also vary over time, and as conditions evolve the appropriate snow and ice control actions will change. Light dry snow, heavy wet snow and freezing rain all require different responses, yet all may occur in a matter of hours. Though each precipitation event is unique in duration, intensity, and composition, vehicle routes are generally fixed at the beginning of winter, so the routes must be robust enough to provide an appropriate level of service over a wide range of conditions. Traffic conditions are also dynamic and can have very strong effects on snow removal operations.

The complexity of roadway snow and ice control results from the different environments in which the problems occur in terms of location, equipment, infrastructure, policies and operations. Rural problems are

often simpler due to the sparser road networks and the service requirement, in many cases, to only remove snow from the roadways. For many storms, plowing snow off of the roadways, or spreading chemicals and abrasives is sufficient. Heavier storms may require heavier equipment, but the availability of open space adjacent to rural roadways usually means that snow can be piled beside the road to accumulate over the winter. Snow and ice control in urban areas is a more complex problem due to the needs to not only clear roadways, but to also clear sidewalks, crosswalks, fire hydrants, public transit stops, and intersections. Further complexity in snow plowing is introduced by parked cars, the desire not to block driveways or sidewalks with plowed snow, and turn restrictions for u-turns and turns across traffic lanes. (For example, a snow plow driving on the right-hand side of a roadway is designed to move snow toward the right edge of the roadway, so a left turn at an intersection results in a row of snow being left in the middle of the intersection.) In areas where plowed snow accumulates beside the roadway and impedes pedestrian or vehicle traffic, it must be physically removed to a disposal site. Snow disposal operations generally involve plowing snow into a windrow in the middle of the street and then loading it into trucks with snow blowers (rotary plows), snow loaders, or other equipment. This is common following large storms in many urban areas, and can be quite expensive. For example, Montreal hauls an average of seven million cubic meters of snow each year (plowed from the streets and sidewalks) to a variety of disposal sites around the city.

Complexity in roadway snow and ice control is also introduced by the wide range of equipment available. Minsk (1981), Keyser (1981), and the Winter Urban Environment Research Subcommittee (1994) provide a wealth of details of snow and ice control equipment and use of chemicals and abrasives. Each type of equipment (plows, spreader vehicles, snow loaders) occurs in multiple variations and a single agency's fleet may include a diverse mix of snowfighting vehicles. Snow plows may be divided into displacement or blade plows, and rotary plows. Displacement plows are the most common and may have blades of different lengths, compositions, and angles, which can be straight, articulated or V shaped. Plows may be mounted on the front of vehicles, under the body, at the side, or on the rear. Road graders are also used for snow plowing. Rotary plows also come in several different varieties. Special equipment is also available for heavy conditions and ice removal. Spreader trucks have different carrying capacities for materials, and can distribute a variety of different materials, at different rates, in dry or wet form (wet materials can increase adhesion to the roadway and speed melting). The rate of application and the speed of spreader trucks depends on a wide range of factors including weather conditions, road conditions (wet or

dry, pavement temperature, etc.), traffic, material being spread, and characteristics of the spreader mechanism. Trucks can also spread one or two lanes in a single pass. Snowblowers and snow loaders can also operate at different rates. In addition to the multiple types of equipment that may be available, some equipment is reconfigurable during and between storms. For example, plow angles can be changed to allow wider or narrower swaths to be cleared depending on the width of the roadway. Also, spreader trucks may change rates of application or materials.

The existing infrastructure in a jurisdiction provides the framework for the roadway snow and ice control operations. The wide range of roadways encountered lead to different arc routing problems. A typical agency may treat a mix of road types and surfaces including single lane roads, two lane roads, multi-lane roads, alleys, cul-de-sacs, one way roads, gravel roads, etc. In addition, some roadways may be available for travel but need not be cleared, generally if they are treated by another jurisdiction. In urban regions the problems are compounded by the need to keep clear a variety of other areas such as sidewalks, crosswalks, fire hydrants, transit stops, intersections, etc. The locations of depots for chemicals and abrasives, and of vehicle garages also provide constraints on RSIC operations. Several depots may be located across a region to provide opportunities for spreader vehicles to refill with materials without returning to the original starting point. In snow loading operations, the linkage between snow disposal sites and vehicle routes also imposes restrictions on the arc routing problems (Campbell and Langevin, 1995a, 1995b).

Policy decisions and operational decisions for roadway snow and ice control contribute to the problem complexity in several ways. Most public works agencies assign priorities for treatment to certain roadways to ensure essential travel routes remain safe. A recent survey in Minnesota showed that 90% of the counties and 84% of the cities employed priority routes (Office of the Legislative Auditor 1995). These roadways may be treated first, and with greater frequency than other roadways. Service levels for roadways may be specified in terms of route length or frequency of treatment (e.g., each route should be covered every two hours) and in terms of end result (e.g., bare pavement of a specified width). Shift lengths for employees can also affect vehicle routing and scheduling.

Numerous roadway snow and ice control operational decisions affect the complexity of the arc routing problems. For multi-lane roadways, two (or more) plows may operate in tandem clearing two (or more) lanes in one pass. Spreader trucks may similarly operate in tandem or alone. A spreader truck can also spread materials in swaths of different widths,

allowing a single pass to treat either one lane or two lanes. Thus, a two-lane road may be modeled as two directed arcs, or a single undirected arc, depending on operational decisions. Each road segment of two or more lanes may also be assigned to a single route or to two (or more) routes, since each lane (e.g., one direction of travel) could be assigned to a separate route. Operational considerations in snow plowing include special considerations to not block intersections or driveways. Since snow plows turning across traffic lanes (e.g., a left turn from the right side of a roadway) typically leave a large row of snow through the middle of an intersection, such turns by plows may be prohibited or discouraged in plow routes. Alternatively, plows could make these turns, but take additional time and effort to clear snow from the intersection. Similarly, plows may leave large rows or piles of snow blocking driveways if care is not taken to tuck the snow into snowbanks that leave the driveways clear.

In summary, the complex and dynamic practical issues raised in this section contribute to making arc routing for snow and ice control an extremely challenging proposition. Haslam and Wright (1991) state that "design of routes for ... snow and ice control is perhaps the most difficult and complex of all public service routing problems". Some of the elements that add difficulty are fixed, such as the roadway network and infrastructure, while other elements involve long term decisions and policies such as fleet mix and service levels, and still other elements involve shorter term operational decisions in conducting roadway snow and ice control. Arc routing models and solutions that ignore the practical complexities of the problem are likely to provide little real benefits in the field.

4. SOLUTION APPROACHES

In spite of the difficulty of roadway snow and ice control arc routing problems, public works agencies have been designing routes every winter for many decades. Routes developed and modified based on field experiences, usually with considerable input from the drivers, represent perhaps the earliest solution approach, and one that is still used in a great many jurisdictions today. In a stable environment with no changes in the road network, equipment, or service levels, such an approach may over time produce very good routes, as experienced drivers and public works officials undoubtedly accumulate a great deal of knowledge and expertise in RSIC. However, in a more realistic dynamic environment where infrastructure, equipment, and service levels may change over time, route design is much more complex, and periodic redesign is essential for efficient operations. Design of new routes is also required as jurisdictions and responsibilities for RSIC change. A 1995 survey in Minnesota reported that 62% of counties and 55% of cities re-evaluate snow routes

every year; yet only one agency used a computerized routing package (Office of the Legislative Auditor 1995). (In the same survey, 20% of cities and 45% of townships reported never re-evaluating their routes!)

Analytical approaches to route design offer great promise for improving roadway snow and ice control. A simple analytical approach to RSIC arc routing involves manual analysis based on maps and statistical route summaries. Careful visual analysis of a route map combined with the ability to quickly generate route summary statistics (length, travel time, cost, lengths of different classes of roads, etc.) may lead to manual design of routes that improve those developed in the field. The development of a variety of heuristic algorithms for theoretical arc routing problems has led to applications in snow and ice control, though early results were less promising than hoped. Idealized models may treat snow plow routing as a variant of the Chinese Postman Problem (CPP) or the Rural Postman Problem (RPP), and spreader vehicle routing as a type of Capacitated Arc Routing Problem (CARP). (Some RSIC routing problems can also be modeled as node routing, rather than arc routing, problems. See Laporte 1997.) However, theoretical models for arc routing based on the CPP, RPP or CARP (see Eiselt et al. 1995a, 1995b) generally fall short of the requirements for RSIC. The development of digital road networks and geographic information systems (GIS) with sophisticated abilities to maintain and analyze locational and attribute data, provides new opportunities for "optimization" in arc routing. The following two sections discuss some important developments in RSIC arc routing, and concentrate on work that has been implemented in the field, rather than theoretical models that may have been tested with data from the field.

5. EARLY WORK

Scientific and analytical research on roadway snow and ice control began to attract considerable attention in the 1960s, and the Highway Research Board (now Transportation Research Board) of the United States held its first international symposium on "snow removal and ice control research" in 1970 (Highway Research Board, 1970). This included no papers specifically on arc routing, but did cover topics such as a regression model for snow removal costs, chemicals for RSIC, pavement heating, ice-pavement bonds, skid resistance, snow fences, and engineering studies. By the time of the second such symposium in 1979, a variety of authors had begun to address RSIC arc routing problems. These early works were generally based on rather straightforward heuristics for arc routing problems, and used simulation models to evaluate benefits. Implementation details and operational constraints were rarely considered.

Marks and Stricker (1971) discuss routing of public service vehicles involved in urban trash collection and snow plowing. They identify parked cars, cars stuck in a roadway, and priorities for plowing as significant problems in modeling RSIC. They describe a decomposition algorithm for arc routing that relies on manual solution of a series of CPPs. They present an application for trash collection, but not snow plowing, in Cambridge, Massachusetts.

Liebling (1970 and 1973) presents a study to aid municipal authorities in Zurich, Switzerland in the planning of the following tasks: choice of suitable equipment for street cleaning; determination of sites for depots; planning of individual routes for snow removal and salting. He uses a graph-theoric formulation to partition the city into sectors, each serviced by a single vehicle. Then, a standard Chinese Postman Problem is solved in each sector.

Cook and Alprin (1976) describe a closest street heuristic for routing of salt spreader trucks. Street segments are defined to require exactly one load of salt, so this is not a typical arc routing application where routes include multiple street segments. In this paper street segments are assigned to trucks dynamically so that as each truck is filled at the depot it is assigned to the closest untreated street segment. Thus a single vehicle can be refilled multiple times to serve multiple street segments. However, each trip from the salt depot is to serve a single street segment. The objective was to minimize the time for a given fleet of spreader trucks to cover all streets in a network, and the heuristic was tested with data from Tulsa, Oklahoma.

Tucker and Clohan (1979) use the then new technology of computer graphics to help manually design a new route for a snow plow in Farmington, Connecticut. Their design guidelines included minimizing u-turns at intersections, left turns and deadhead travel (i.e., travel without providing service). They simulated five storms and conducted sensitivity analyses to evaluate the role of the rate of snowfall and the initial depth of snow when plowing begins. They summarized their study by saying that the current route is "near-optimum". They acknowledged the practical complexities of multiple passes, tandem plowing, delays at intersections, different road widths and varying snow accumulation rates, but they did not address any of these issues in their new route. Interestingly, they report that

> "Snow removal equipment routing is that phase of the operation which can potentially save the most money. Although certain methodology in the fields of graph theory and network analysis could aid in the routing problem, the techniques are somewhat dif-

ficult to grasp and most city officials are out of touch with this type of technology."

They also summarize the situation at that time as follows: "With the exception of a few cities that publish procedural manuals or plans, few attempts at routing improvement are made."

Lemieux and Campagna (1984) present a simple model for snow plow routing with roadway priority classes. The approach is to first find an Eulerian circuit on a directed graph, and then select higher priority roads when possible. An interactive computer program is described and tested on a very small network. This model does not consider one-way or multi-lane streets.

Gilbert (1989) models the snow removal problem for a single snow-blower as a node covering problem where the nodes correspond to the segments of streets to be cleared. Several precedence constraints are taken into account. The resulting model is a large scale non-linear 0-1 programming problem that is not solved. A heuristic approach is developed based on two versions of an insertion method that adds nodes iteratively to a given work shift. The first version inserts higher priority nodes in the first shifts. The second version balances the work load, mainly in the last days of work. The heuristic is tested with data from one district in the City of Montreal, Canada.

Gélinas (1992) presents an optimal approach for the snow removal problem. The problem is modeled as a single vehicle arc routing problem with precedence constraints. The solution approach involves constructing Eulerian subgraphs, finding upper and lower bounds by dynamic programming, and solving a traveling salesman problem by branch-and-bound. Data from the same district of the City of Montreal as in Gilbert (1989) was used to test the algorithm.

As both the technical and organizational difficulties of producing a useful solution became better understood, and as computing power became more widely available, decision support systems became the natural approach for RSIC arc routing problems. The SnowMaster system (Evans 1990, Evans and Weant 1990) is an early example of an arc routing DSS for RSIC. This system was designed to "assist...engineers and fleet managers to route snow and ice control vehicles efficiently for improved service, higher equipment utilization, and lower capital and operating costs." The route design component of the system included five different path scanning rules that could be used at each node in the street network: (1) select the shortest arc, (2) select the longest arc, (3)

select the arc that maximizes the distance back to the depot, (4) select the arc that minimizes the distance back to the depot, and (5) if the vehicle is less than half full, select according to rule 3; if the vehicle is more than half full, select according to rule 4. A further modification was included to give priority to adjacent nodes with exactly one unserviced arc. Route summaries were provided for all five rules and the user could select the most appropriate routes.

This system addressed capital costs of equipment and helped to identify the appropriate mix of equipment required. Simulation tests using data from Butler County, Ohio showed savings of $251,000 over three years from reducing the size of the fleet required through improved routes. Routes designed with SnowMaster were used in the field in November 1989 and the route completion times from the DSS were "accurate to within only a few percent". Waddell (1994) provides some sketchy details on several applications of SnowMaster in different U.S. counties and highlights the linkage between route efficiency and both storage depot locations and fleet size and composition. He also points out how "additional trucks are often employed in a misguided effort to overcome inefficient routes with more resources". Evans and Weant (1990) conclude: "Computer-based routing software can provide substantial benefits for snow and ice control decisions over existing manual methods by allowing fast scenario analysis and efficient optimization of routes that improve service and reduce costs." (A recent report of SnowMaster being used in an urban county was discouraging, as one route included excessive u-turns. Use of the software was quickly discontinued according to Simcox 1998).

In spite of the recognition of the value of route "optimization" and the availability of heuristic solution methods based on arc routing research, progress in arc routing for roadway snow and ice control was very limited. Early attempts to apply simple heuristics produced nice results from simulation studies, but were rarely implemented in the field. Much early work adapted theoretical arc routing models with little consideration of practical RSIC operations. The promise from these early computer implementations of heuristics was often derailed by poor implementation and user-unfriendly software, distrust of computer-based (black-box) approaches by public works personnel, and unappreciated operational complexities. However, these early systems did focus attention on savings from better routes, and perhaps more importantly the savings from a more rational and analytical approach to fleet size and service level (time limits for routes). The SnowMaster system provided a major advance in the trend towards route "optimization" by encouraging user involvement in route design. However, the inability to consider

all the nuances of real world operations may have helped to mask some of the benefits being achieved.

6. RECENT WORK

While the promise of applying analytical techniques of operations research and management science to roadway snow and ice control has been recognized for several decades, proof of the promise has been slow in coming. General theoretical arc routing algorithms showed substantial benefits in simulation studies, but actual implementation for RSIC was a very different matter. The State of Minnesota, a leader in RSIC, conducted a state-wide survey in 1995 to collect the "best practices" in snow and ice control, and reported that only a single agency out of the 414 jurisdictions (counties, cities and townships) surveyed, used computerized routing software (Office of the Legislative Auditor, 1995). Thus, early enthusiasm from researchers for RSIC arc routing "optimization" gave way to a more pragmatic understanding that "...the application of analytical and numerical methods, and computer technologies to the problem of designing winter service routes is in its infancy" (Wang and Wright 1994).

In spite of the slow progress, from a practical perspective, researchers and consultants have continued efforts to develop and implement computerized packages for RSIC. The new generation of software packages for RSIC combine arc routing heuristic algorithms, interactive route design functionality, user friendly interfaces, and ability to work with digital network representations. This section focuses on two rather sophisticated and successful examples: the CASPER system developed for rural RSIC in Indiana, and the GeoRoute Municipal package. CASPER was developed in a joint project involving researchers at Purdue University and personnel of the Indiana Department of Transportation. GeoRoute Municipal has been implemented in several urban and rural regions by PSR Group Ltd. (a consulting company in Nepean, Canada) which licenses the software from its developer, GIRO Enterprises, Inc. (of Montreal, Canada).

6.1. CASPER

CASPER (Computer Aided System for Planning Efficient Routes) is a true interactive decision support system for design of RSIC routes. It has evolved over a number of years incorporating a variety of improvements, and is currently used to design routes for the State of Indiana. The information in this section is derived from a number of sources (Haslam and Wright 1991, Wang 1992, Wang and Wright 1992, Wright 1993, Wang

and Wright 1994, Wang et al. 1995, Goode and Nantung 1995, Goode 1997, Wright 1997). This section first describes the operating environment for rural RSIC in Indiana, then the routing algorithms in CASPER and finally results from the field implementation which began in 1993.

The Indiana Department of Transportation (INDOT) maintains over 29,000 lane-miles in over 11,400 miles of roadway. RSIC involves over 1500 trained personnel and approximately 1100 vehicles. The annual budgets for RSIC since 1992-93 have ranged from \$20-\$28 million. The last formal evaluation of snow routes in Indiana, prior to CASPER, occurred in 1970. RSIC vehicles are based in depots across Indiana with each depot supporting between 5 and 15 routes. Vehicles used for RSIC include trucks with 8-ton and 12-ton capacities. A typical service vehicle outfitted for snow and ice control, including the plow and spreading attachments, costs between \$70,000 and \$120,000. Trucks travel approximately 20 mph while servicing and 40 mph while deadheading (not servicing). Because of the relatively large capacities of the trucks, the route lengths are limited by the level of service (specified as a time limit for traversing the route), rather than the capacity for carrying materials. Personnel turnover has averaged approximately 30% from one winter to the next, therefore requiring annual driver training and making experienced personnel especially valuable.

Roadways in Indiana are divided into three classes for RSIC based on average daily traffic (ADT). Class 1 roadways have ADT of 5000 or greater and receive continuous service to keep the road surface wet and bare. Class 2 roadways have ADT between 1000 and 5000 and receive continuous service to keep bare wet pavement in the center of the roadway. Class 3 roadways have ADT of less than 1000 and receive continuous service to keep the road passable, with chemical treatment only for hills, curves, and intersections. Route lengths are limited by specified service levels for each class of roads: class 1 roads should be serviced every 2 hours, while class 2 and class 3 roads should be serviced every 3 hours.

The goal in creating vehicle routes for RSIC in Indiana is to minimize cost, both by reducing the number of routes and reducing the number of deadhead miles traveled, while meeting specified service levels and maintaining class continuity. Costs are reduced by eliminating routes, since each route requires a vehicle, and by reducing the amount of unproductive deadhead travel. Deadhead travel is of particular concern, because "traditionally, the most intense and frequent of all citizen complaints directed at winter operations relates directly to the amount of deadhead travel" (Wright 1993). The class continuity condition reflects the inefficiencies of providing a higher than necessary level of service.

Because all road segments in a route receive the same treatment, the route must be serviced according to the highest level of service for any segment of the route. Although providing a higher than needed level of service may indicate unnecessary expense, it does provide the citizens with the benefits of safer travel from the better level of service.

CASPER designs multiple routes from a specified depot location (where vehicles begin and end their routes) to serve a specified network of roadways. The route design procedure is repeated for each depot across the region. Detailed data is required for each arc in the road network, including length, width, number of lanes, and ADT. Intersection details, such as turn lanes, and traffic control devices, are also required to capture the complex requirements at road intersections and interchanges. In addition, turn permissions (e.g., safe u-turn locations) can be specified at different locations. CASPER was originally developed before the widespread availability of detailed digital road networks, and considerable work was required to extract the necessary roadway data from the available sources, primarily U.S. Geological Survey digital line graph data (Wang and Wright 1992). This process is now made easier as public service agencies acquire detailed digital roadway data, generally with a geographic information system (GIS). However, considerable work may still be required to ensure an accurate network representation of complex intersections and interchanges.

CASPER incorporates multiple objectives in designing routes via a penalty function that incorporates the conflicting desires to meet specified service levels (time limits for routes), minimize deadhead travel, and maintain class continuity. The penalty function is calculated as a sum of three terms and was developed in consultation with INDOT personnel. The penalty is calculated for each route separately, and values can then be added for a summary performance measure. The service level constraint is incorporated in the penalty as the deviation in route time from a target time measured in minutes, and then cubed. The target time is specified as a range, and the current values are 90-120 minutes for class 1 routes, and 150-165 minutes for class 2 and class 3 routes. Thus, a class 1 route of 80 minutes (or of 130) minutes incurs a penalty of 1000 (ten cubed). Penalizing routes that are too short encourages using fewer routes, which reduces costs; penalizing routes that are too long encourages achieving the desired frequency of service. Deadhead travel is incorporated in the penalty as the sum of the deadhead miles. Class continuity is reflected in the penalty as one-third of the number of off-class miles, i.e., miles in a route of a class other than that desired (for example, a class 2 road included in a class 1 route). The one-third weight for off-class miles relative to the unit weight for deadhead miles indicates

a greater importance for deadhead miles. However, the time limits are much more important as reflected in the exponential (cubic) function. Thus, the penalty component for a large time deviation will far exceed the other components. However, when the route length penalty (time deviation) is small, then the deadhead and off-class miles will be important.

Arc routing in CASPER is accomplished with a heuristic route generation algorithm and an interactive tabu search based route improvement heuristic. The route generation heuristic grows routes from the depot using the penalty function to guide development. (An earlier greedy seed-node based route generation approach is described in Haslam and Wright 1991). CASPER also allows user specified (e.g., existing) routes to be treated as input to the route improvement heuristic. The route improvement heuristic is also guided by the penalty function, and is a refinement of that in Haslam and Wright (1991).

Before generating routes, CASPER estimates the number of routes required for each class of roadways based on the lane-miles of that class of roadways, vehicle speeds and the time limit (service level) for that class. Each route is started from the depot by selecting the closest unserviced arc of the appropriate class. Each route is then extended using a "two arc look-ahead procedure" that selects the next arc in a route based on maximizing the number of arcs of the appropriate class incident to the far end of the next arc. Thus, if the end of a route of class n is node i, then the next arc in the route will be the arc of class n incident to node i, that is adjacent to the greatest number of arcs of class n. Arcs are only added to a route if the shortest path from the endpoint back to the depot keeps the total route length within the time limit for that class.

Once all routes stop "growing" by this two arc look-ahead procedure, routes are expanded in three steps. In each step routes are only expanded if their total service time is within the specified limit. The first route expansion step takes any unserviced arc incident to a route of the same class and adds that arc to the route. This step maintains class continuity. The second route expansion step takes any unserviced arc incident to a route of higher service level class and adds that arc to the route. The arcs added in this step will receive a higher level of service than necessary. The third route expansion step adds each unserviced arc to a route of the same class. This involves deadhead travel to and from the added arc, since arcs incident to a route of the same class would have been added earlier. The selection of the route for each arc is based on minimizing the penalty function.

The final step in generating routes is to add any remaining unserviced arcs to a route. This is accomplished by ignoring both the classes of arcs and the time limits for routes. The addition of an arc to a route is again based on the penalty function calculation. At the end of this step, each arc will be serviced on exactly one route. Arcs may also be used for deadhead travel in one or more routes. The penalty function can be calculated for each route, and these can be summed to produce a single performance measure for each depot. The routes resulting from this generation procedure may not be very good, but they do provide a feasible starting point for the route improvement heuristic.

Once an initial set of routes is specified (whether from the route generation heuristic or some other procedure), CASPER invokes an interactive arc swapping route improvement heuristic. Swaps of a single arc or of two arcs are considered, and the penalty values are used to evaluate the improvement. The improvement heuristic is guided by a tabu search procedure with a tabu list that can be specified to contain either the arcs and routes involved in the swap, or just the arcs (a more restrictive option). A second "permanent tabu list" is used to retain user specified moves that can not be reversed. This allows the route designer to lock in desirable features of a route. Candidates arcs for swapping are generated from the routes with the greatest penalties to increase the chances of finding good swaps. User specified parameters allow considerable control over the route improvement heuristic. See Wang (1992) for details. Users are also provided with tools to specify arcs to swap, and to "freeze" portions of routes. Furthermore, users can specify arcs for class upgrades (service level improvements) as may be needed to address a hazardous spot (e.g., a sharp curve or steep hill). In practice, CASPER users have spent considerable time in the route improvement phase in an interactive effort to understand the routes being generated by CASPER, and to suggest improvements. CASPER is designed to be used by someone knowledgeable about snow and ice control, and about the transportation network being analyzed. The interactive nature of CASPER allows great flexibility in accommodating features not reflected in the data when designing routes.

CASPER was initially field tested in Winter 1992-93 following redesign of routes in about 40% of Indiana. A summary of the results from four northern INDOT districts is shown in Table 1.

	Before CASPER			After CASPER			
District	# of routes	Deadhead miles	Off-class miles	# of routes	Deadhead miles	Off-class miles	Penalty change
Greenfield	57	454	423	58	533	478	-98.8%
La Porte	113	1436	253	99	1039	761	-97.5%
Ft. Wayne	116	1174	606	108	1271	1021	-97.9%
Crawfordsville	144	1286	536	131	1322	1000	-94.8%
Total	430	4350	1818	396	4165	3260	-96.3%
Percent change				-7.9%	-4.3%	+79.3%	-96.3%

from Wright(1993) and Wang et al. (1995)

Table 1: CASPER Summary

The two most important changes are the decrease in the number of routes required of 34 (7.9%), and the very large decrease in the penalty values. The large decreases in penalty values (between 95% and 99% for each district and overall) result mainly from the CASPER routes better meeting the specified service levels. Note however, that in the routes designed with CASPER the deadhead miles and off-class miles may show an increase in particular districts, or even overall. The general pattern is that CASPER designs routes that meet the target service times, but may increase deadhead miles and off-class miles to achieve this. One interesting finding from consultations with INDOT was that designing routes that satisfy the target service times makes deadhead travel less offensive. Public complaints actually decreased with the CASPER routes, in spite of the reduction in the number of routes. The real problem from the field is when service level is not being met and citizens see vehicles deadheading. (Class continuity is also not a significant issue when service levels are met.)

Evaluation of CASPER included comprehensive field evaluation of new routes, based on forms completed by drivers following each traversal of a route during a storm. Field experience with the routes generated by CASPER showed that 72% of the new routes were deemed acceptable. Route acceptance ranged from 23% to 100% in different subdistricts. The general procedure was to use the routes for two winters and modify them when necessary, even adding new routes if needed. Thus, routes were not finalized until several winters worth of successful operation.

From the field evaluation in approximately 40% of Indiana, INDOT estimated savings of $2.2 million in the first year and $4.8 million over 10 years. Initial estimates from extrapolating the results from these field tests to the entire state of Indiana produced a total savings of about $10 million state-wide, from elimination of 50 routes (50 vehicles). A more recent estimate from INDOT of the net present worth of savings from CASPER routes for the period 1993-2016 is $4 million (Goode 1997).

This includes the cost savings for not replacing trucks (10 year life) at $75,000, plus annual savings for fuel, labor and maintenance on the truck, plow and spreader. The total cost for the project to develop CASPER, including INDOT and university contributions, has been estimated at $172,000 (Wright, 1997).

Following the initial route design efforts and field evaluations, CASPER was used to design additional routes in 1995 using INDOTS's new GIS basemap. INDOT also contracted with a consulting firm to complete a variety of modifications to CASPER. Delays in completing the software modifications slowed state-wide implementation of new routes. This arrangement also led to a dispute over ownership of the CASPER product (Goode 1997, Gini and Zhao 1997), with some attempts being made by the consulting firm to sell a product derived from CASPER. Currently CASPER is not a commercially available product, though there are plans to make it more widely available in the future (Wright 1997).

One interesting finding from the CASPER experience was the importance of depot location in achieving good vehicle routes. A centrally located depot (relative to the specified network) generally produced better routes than a depot located near the edge of the network. Depot locations adjacent to multi-lane highways also led to good routes. Because CASPER designs networks for a specified depot and associated network, reallocation of the road segments between depots is not considered. Kandula and Wright (1995) have thus developed arc partitioning models to optimally assign arcs to depots.

6.2. GEOROUTE

GeoRoute is a commercially available software package for vehicle routing developed by GIRO Enterprises, Inc. of Montreal, Canada. GeoRoute Municipal is the name of the relevant software for arc routing. PSR Group Ltd. of Nepean, Canada has incorporated GeoRoute Municipal in a system named MOPS (Municipal Operations Performance System) for roadway snow and ice control in urban regions. PSR Group has done considerable work implementing systems for snow and ice control, especially in urban regions, based on the GeoRoute arc routing software. PSR Group has probably accumulated the greatest amount of experience with "optimized" arc routing for snow and ice control in North America. In this section we will use "GeoRoute" to refer to these various related RSIC software packages.

GeoRoute is a complete interactive decision support system for design of roadway snow and ice control routes. It consists of four modules: network manager, route manager, site editor, and map generator. The

network manager is a geographical data base with map display capabilities. Each street segment on the map is linked with a data base containing information such as the intersections (nodes), address ranges, street names of the adjacent streets, etc. The network manager allows users to edit the network and its attributes. The route manager is the optimization system. It works hierarchically and allows for each type of operation: spreading materials, plowing snow, snowblowing (for loading and disposal of snow). The approach is to divide the region into sectors, and then for each sector to optimize the routes for a number of scenarios. The site editor allows users to locate various facilities on the network, such as garages, depots, etc. The map generator facilitates creation and printing of the required maps.

GeoRoute incorporates routines to consider a variety of different RSIC environments, including urban and rural operations, and thus is less narrowly focused than CASPER. The following characteristics of snow removal are taken into account by GeoRoute:

- Variable criteria are used for sequencing of the road segments within a route. For instance right turns are favored for plowing operations, while they are limited as much as possible for snowblowing. A user-definable multi-criteria evaluation function can be calibrated according to needs.

- In urban networks, both sides of each street are assigned to the same route. In rural road networks this is not the case, as it could lead to very inefficient routes.

- Repetitive treatments of the same road are allowed. The software takes into account the frequency of service, which depends on the level of precipitation and the need to balance the services over time.

- For the spreading operations, the number of passes needed to serve each road segment is determined automatically given the width of the street and the type of equipment. The vehicle capacity and the spreading rate, which varies according to the type of street, is also taken into account. Moreover, if desired the spreading can be done only on a portion of the street (e.g., at the intersections).

GeoRoute is a proprietary software package, and thus details on its "optimization" algorithms are not publicly available. GeoRoute designs routes one at a time combining clustering and sequencing. For clustering, a seed is selected first, and then the algorithm considers its nearest neighbors. Precautions are taken to keep the routes as compact as possible and to prevent isolating a sub-region that would lead to an inefficient route. A temporal dimension can be taken into account in selecting the

cluster when in the presence of time windows.

For constructing routes in GeoRoute, the GENIUS algorithm (Gendreau et al., 1992) has been adapted for arc routing. GENIUS is a generalized insertion algorithm for the traveling salesman problem. With generalized insertions, a "localized" re-optimization is done simultaneously with the insertion. The algorithm uses a multi-criteria evaluation function defined by the user. It takes into account time windows and determines if it is better to serve a street segment in a single pass or in two passes (e.g., for spreading two-way streets). Both clustering and sequencing work with subsets of arcs that the user specified to be on the same route.

GeoRoute has been used to optimize snow removal activities in many settings in Canada (including the cities of Laval, Charlesbourg, and Nepean, and the Ottawa-Carlton Regional Government) and in the United Kingdom. Case studies with GeoRoute in Ottawa, Canada and Suffolk County, U.K. are presented next to illustrate how the software was used.

6.2.1 Ottawa, Canada.

Ottawa, Canada has a population of 314,000 and an annual snowfall of 220 cm. The budget for winter maintenance is approximately $16 million for treating 2400 lane-km. The material presented here has been extracted mostly from Miner and Brethertan (1996), Miner (1997) and Bretherton (1997), and describes the development of plow routes for approximately one-third (800 lane-km) of the city using GeoRoute. The experience in Ottawa includes both computer simulation tests, as well as field tests in Winter 1995-96. This case study reports the analysis performed by the PSR Group, and the city-wide costs and savings identified from the implementation of a winter-long field trial.

Ottawa used 60 predefined roadway routes for plowing, 22 routes for salting, and over 70 sidewalk plow routes. Snow disposal operations are conducted on over 550 km of streets. Ottawa contracted with PSR Group Ltd. to conduct a pilot project involving routes for snow plows. A trial area corresponding to one district of Ottawa was selected. It contains 800 lane-km of streets within the City. The consultants were asked to undertake simulation tests to evaluate three sets of routes:

1 The actual plow routes.

2 Revised plow routes constructed by resequencing the streets in the actual routes.

3 Optimized plow routes.

Each revised route (#2) included the same streets in the corresponding actual route (#1), but possibly serviced in a different sequence. The optimized routes (#3) were developed by allowing the GeoRoute software to determine which streets are on each route and in what sequence they are visited.

For the computer simulation tests, the region of interest was originally served by 24 single plow routes and 9 tandem plow routes (two plows operating together). Table 2 provides results of the computer simulation tests.

		Time (hrs)		
	Number of routes	**Minimum**	**Average**	**Maximum**
Original	24 single	3.7	**6.9**	11.9
Revised	24 single	2.5	**5.0**	8.0
Optimized	19 single	5.8	**6.0**	6.5
Original	9 tandem	2.1	**4.2**	6.5
Revised	9 tandem	2.0	**3.5**	5.5
Optimized	8 tandem	3.0	**4.1**	4.5

Table 2: Computer Simulation Tests in Ottawa, Canada.

The row for "Original" refers to the actual routes prior to optimization with GeoRoute. The range of route times in this row show that the City did not provide a consistent level of service, with the maximum route time being over three times greater than the minimum route time.

Productivity gains and a more consistent level of service were obtained by simply resequencing the streets in an existing plow route. For the revised routes, both the average length of a route and the variation in route length are reduced. Thus, the routes are shorter (on average) and the workload is better balanced among the routes. This is true for both single and tandem routes. The optimized routes show further improvements. Although the average route time increased relative to the revised routes, for both single and tandem routes, optimization eliminates routes: five single routes and one tandem route are eliminated. The result is that the total plowing time (number of routes $\times$ average route length) decreases from 165.6 hours for the 24 single original routes to 114 hours for the 19 single optimized routes. The total time for the tandem routes decreased from 37.8 hours for the 9 tandem original routes to 32.8 for the 8 tandem optimized routes. The variation in route length was also greatly reduced by optimization. The results of the optimization produced pro-

jected savings of 17% of the hired plow hours per storm, which equated to $15,000 per storm, or $150,000 per year.

Based on the simulation results, it was recommended that a winter-long field trial be conducted to verify savings and identify potentially unexpected additional costs. For the field tests, only the optimized single routes were considered. Drivers were given a map and text description of their new optimized route. A number of problems became apparent after the first storm (on November 14, 1995), due to drivers not adequately understanding the new routes. Consequently, for the remainder of the winter, the field tests were scaled back to include only four routes and the drivers on these routes were trained to be familiarized with their routes. The remaining portion of the city was served by its original (current) routes. In the following eight storms that winter, the optimized routes fared better than the non-optimized (current) routes. Productivity on the four optimized routes ranged from 3.2 km per hour to 3.75 km per hour. Productivity on the original routes averaged 2.0 km per hour. Thus, the optimized routes showed an increased productivity of 60% to almost 90%.

6.2.2 Suffolk County, UK.

Suffolk County, England has a population of 660,000 and experiences icing conditions , as well as a small amount of snow each winter (less than 10 cm of snow on average). The budget for winter maintenance is approximately £1.2 million and prior to 1997 the County had been responsible for maintaining approximately 6900 km of roadways. This total includes 400 km of trunk roads, for which maintenance responsibilities were transferred away from the County (to consultants employed by the central government) in April 1997. Thus, these 400 km were no longer the County's responsibility, and the existing routes needed to be re-designed to treat only the County roads. The following results are from Guttridge (1998) and Thompson (1997).

The planning of routes in Suffolk County was done manually prior to 1996. PSR Group Ltd. was hired to design new routes using GeoRoute to explore whether computerized analysis combined with geographically accurate network databases could provide a valuable alternative to manually designed routes. In addition, from a risk management perspective, computerized route optimization was seen as a proactive strategy to both reduce the likelihood of incidents and to demonstrate retrospectively that adequate service level provision was planned.

Suffolk County began using the GeoRoute software in September 1996 to develop new routes. Two different areas were chosen for a first trial

application of the software: a dense urban area, and another larger area, with similar population, incorporating a variety of urban areas, a sub-urban fringe of the dense urban area, and a multitude of smaller towns and villages. Network and operational details including one-way streets, turn-restrictions, service priorities, road widths, spreading rates, vehicle capacities and level of service requirements were considered. The following results are based on the application of GeoRoute to re-design the Priority 1 salting network. The Priority 1 salting network originally included 37 routes covering 1430 km of county roads. All these routes were limited to a length of two hours. The GeoRoute software produced 35 routes rather than 37, with the two hour treatment time. The software also allowed other service levels (time limits) to be evaluated and produced 33 routes with a 2.5 hour time limit. Thus, the number of routes was reduced by 2 or 4, depending on the time limit. The estimated savings for each route eliminated was £ 11,000 per year, so the total savings for the Priority 1 network is projected to be £ 22,000 - £ 44,000 per year, depending on the level of service.

The cost for route re-design was estimated to be £ 70,000, including staff time and purchase of the GeoRoute software (Guttridge 1998). The cost of re-designing the routes manually was estimated to be at least £ 30,000 (Thompson 1997). In addition, a manual solution (as in the past) would have produced less efficient routes and prevented evaluation of different levels of service (Thompson 1997).

6.3. OTHER RECENT WORK

In addition to the work with CASPER and GeoRoute, several other authors have recently addressed arc routing for snow and ice control in urban and rural areas. However, implementation details are not reported in these papers.

Eglese and Li (1992) and Eglese (1994) describe rural RSIC research based on a 1988 study of winter gritting (spreading abrasives) in Lancashire County, U.K. Trucks are assumed to service each road in a single pass by spreading both sides of the road at the same time. (Note that this is not feasible for plows.) This allows each road to be modeled as an undirected arc. Multiple depots in the network are modeled explicitly. The solution procedure to design routes (based on Male and Liebman 1978) involves using a Clarke-Wright-like greedy heuristic on a cycle node graph derived from the solution of a CPP on the original transportation network. Simulated annealing is then used to improve the truck routes. The objective is to minimize the number of routes and the penalties from exceeding specified distance or time constraints. The approach is tested in three areas which include a total of 694 arcs and 1162 km of roadway.

Detailed results were not reported, but the most significant finding was that the number of depots could be more than halved without increasing the number of trucks.

Li and Eglese (1996) consider the same problem of winter gritting, but incorporate directed and undirected arcs, priority classes, and salt depots that allow vehicles to be refilled without returning to their origin. They describe an interactive PC-based software package that incorporates a "time constraint two phase" heuristic routing algorithm. Each route begins by selecting the unserviced arc farthest from the depot. Phase one extends the route back to the depot, beginning at the end of the initial arc nearest the depot. Phase 2 then extends the route from the far end of the initial arc back to the depot. In each phase, priority is given to selecting arcs with degree one, or arcs adjacent to those with degree one. Time constraints and capacity constraints are checked as each arc is added to ensure route feasibility. The software allows routes to be constructed in an automatic mode, with no user intervention, or with user intervention to specify the next arc to add to the route in each phase. The heuristic was compared to the earlier approach (Eglese 1994) with data from three areas in Lancashire County. Results showed that allowing user intervention in the route design produced up to 15% fewer routes and 17% less total distance compared to the automatic (no user intervention) version of the heuristic. The new heuristic also produced better routes than the earlier approach in Eglese (1994).

Letchford and Eglese (1998) present a different approach for the single vehicle rural postman problem with deadline classes that reflects priorities for snow and ice control. Optimal solutions are found using a dual cutting plane method with valid inequalities as cutting planes. Optimal results are presented for five problems, the largest of which includes 110 undirected arcs, 67 of which require service.

Lotan et al. (1996) describe winter gritting in the province of Antwerp, Belgium. The network to be treated included 747 arcs accounting for over 1000 km of roads, divided into two priority classes. Routes start from one of the main depots, but trucks may be refilled at supplementary salt depots. The problems addressed include defining districts (partitioning the network into sub-networks), locating depots, and routing trucks. The focus of this work is on more strategic aspects, rather than the arc routing details. Interesting findings include the benefits of locating supplementary salt depots near the borders between different districts to allow greater usage, and the tradeoffs in treating two lanes in one pass, rather than making two passes.

7. FUTURE

While several successful software packages for RSIC arc routing have been developed, they have not yet been widely implemented. This may change with the documentation of recent RSIC successes, the continuing increase in sophistication and ease-of-use of software packages, and the increasing familiarity of users with technology. For example, a pilot project for the State of Minnesota is underway in Hennepin County, Minnesota to customize the TransCAD software (Caliper Corporation, Newton Massachusetts) for RSIC (Simcox, 1998). This may lead to development of a package suitable for many jurisdictions in that state.

However, several obstacles remain to the widespread adoption of route optimization for roadway snow and ice control. Because RSIC problems are so complex and site specific, there is unlikely to be a satisfactory "off the shelf" RSIC software package. Considerable cost and effort for customization, data conversion and cleaning, training, maintenance, etc. are the rule rather than the exception. The very different cultures of public works and operations research/management science also hinders usage of arc routing "optimization" for RSIC. Issues such as the loss of freedom to drivers in designing their own routes, and the faith in "black box" route optimization software require greater attention.

The future holds some interesting opportunities as recent technological advances in snowfighting affect RSIC in a variety of ways.

- Road Weather Information Systems (Boselly 1993) are pavement based sensors that collect, monitor and communicate weather and pavement information in real time. Such information could lead to dynamic routing for RSIC to treat only those areas in need at a particular time. Such "just-in-time" routes could be designed dynamically in real-time for each vehicle.

- Truck mounted pavement sensors that use infrared light to detect pavement temperatures (currently used in Indiana) could help better determine when and how to treat a road. Also, electronic spreader controls on trucks can adjust the amount of materials being spread based on vehicle speed, and store data on spreading for later analysis.

- Anti-icing is a preventive approach to RSIC that has shown good promise (Blackburn et al. 1993). Anti-icing seeks to prevent or minimize the formation of an ice-to-pavement bond by spreading chemicals prior to, or quickly after the start of ice or snow. Good routes for anti-icing may or may not coincide with good routes for plowing and spreading during a storm.

- Automatic vehicle location (AVL) using global positioning system (GPS) technology offers some interesting possibilities for RSIC. Vehicle locations can be tracked and monitored to provide real-time data on vehicle location and status (e.g., spreading or not, plowing or not, etc.), as well as an historical record of vehicle activity. This technology has been employed for RSIC by several agencies. The benefits from real-time monitoring include a better ability to reallocate equipment and personnel in real-time, better ability to inform the public of current operations, verification of work completed, and identification of unauthorized travel. The Virginia Department of Transportation saved over $1 million by reducing fraud from the ability to verify RSIC operations by contractors (Miner 1997).

In summary, the history of arc routing for roadway snow and ice control shows a distinct gap between theory and practice. The early promise of applying optimization tools and techniques for arc routing to RSIC gave way to an understanding of the complexities and difficulty of these problems, and the need for custom solutions. Sophisticated and successful interactive decision support system software packages now exist, with documented benefits from several winter's worth of field experience. We may now be on the verge of widespread adoption of route optimization software for RSIC, and finally realizing the promise of arc routing optimization for RSIC.

References

[1] Blackburn, R.R., E.J. McGrane, K.M. Bauer, and E. J. Fleege. (1993). Current Status of U.S. Anti-Icing Technology Development. *Transportation Research Record* 1387, 29-39.

[2] Boselly III, S. Edward (1993). Road Weather Information Systems: What Are They and What Can They Do for You?. *Transportation Research Record* 1387, 191-195.

[3] Bretherton, S. (1997). Personal communication.

[4] Campbell, J.F. and A. Langevin (1995a). Operations Management for Urban Snow Removal and Disposal, *Transportation Research* 29A, 359-370.

[5] Campbell, J.F. and A. Langevin (1995b). The snow disposal assignment problem. *J. of the Operational Res. Soc.* 46, 919-929.

[6] City of Sapporo. (1991). Snow-Sapporo-21 Project. City of Sapporo. June 1991.

[7] Cook, T.M. and B.S. Alprin (1976). Snow and ice removal in an urban environment. *Management Sci.* 23, 227-234.

[8] Eglese, R.W. (1994). Routing winter gritting vehicles. *Discrete Applied Mathematics* 48, 231-244.

[9] Eglese, R.W. and L.Y.O. Li (1992). Efficient routing for winter gritting. *J. of the Operational Res. Soc.* 43, 1031-1034.

[10] Eiselt, H.A., Gendreau, M. and G. Laporte (1995a). Arc routing problems, part I: The Chinese postman problem. *Operations Research* 43, 399-414.

[11] Eiselt, H.A., Gendreau, M. and G. Laporte (1995b). Arc routing problems, part II: The rural postman problem. *Operations Research* 43, 231-242.

[12] Evans, J.R. (1990). Design and Implementation of a Vehicle Routing and Planning System for Snow and Ice Control. Working Paper QA-1990-002, College of Business Administration, University of Cincinnati, Cincinnati, Ohio.

[13] Evans, James R. and M. Weant (1990) Strategic planning for snow and ice control using computer-based routing software. *Public Works* 121 (4), 60-64.

[14] Gélinas, É. (1992). Le problème du postier chinois avec contraintes générales de préséance. M.Sc.A. Dissertation, École Polytechnique de Montréal.

[15] Gendreau M., Hertz, A., G. Laporte (1992). New insertion and post optimization procedures for the traveling salesman problem. *Operations Research* 40 (6), 1086-1094.

[16] Gilbert, J. (1989). Générateur d'itinéraires d'enlèvement de la neige. Publication CRT-648, Centre for Research on Transportation, University of Montreal.

[17] Gini, M. and Y. Zhao (1997). Automated Route Planning and Optimizing Software. Final Report. Published by Minnesota Department of Transportation. February 1997.

[18] Golden, B. and R. Wong (1981). Capacitated arc routing problems. *Networks* 11, 305-315.

[19] Goode, L. (1997). Personal communication.

[20] Goode, L. and T. Nantung (1995). CASPER: The friendly, efficient snow routes planner. TR News 181 (November-December 1995) 20-21.

[21] Gray, D.M. and D.H. Male eds. (1981), Handbook of Snow, Principles, Processes, Management and Use, Pergamon Press, Toronto, Canada.

[22] Guttridge, Andrew. (1998). Rebuilding a Winter Maintenance Service, The Role of Route Optimization. PSR Group Ltd. web site: www.psrgroup.on.ca/WPGuttridge.htm. September 1998.

[23] Haslam, E. and J.R. Wright (1991). Application of routing technologies to rural snow and ice control. *Transpn. Res. Record* 1304, 202-211.

[24] Highway Research Board. (1970). Snow Removal and Ice Control Research. *Highway Research Board Special Report* 115, Washington, D.C.

[25] Kandula, Padma and J.R. Wright (1995). Optimal design of maintenance districts. *Transportation Research Record* 1509, 6-14.

[26] Keyser, J. Hode (1981). Chemicals and Abrasives for Snow and Ice Control, in D.M. Gray and D.H. Male (eds.), Handbook of Snow, Pergamon Press, Toronto, Canada, 1981, pp. 580-612.

[27] Laporte, G. (1997). Modeling and solving several classes of vehicle routing problems as traveling salesman problems. *Computers and Operations Research* 24, 1057-1061.

[28] Laporte, G. (1992). The vehicle routing problem: An overview of exact and approximate algorithms. *European Journal of Operational Research* 59, 345-358.

[29] Lemieux, P.F. and L. Campagna (1984). The snow ploughing problem solved by a graph theory algorithm. *Civil Engineering Systems* 1, 337-341.

[30] Letchford, A.N. and R.W. Eglese (1998). The rural postman problem with deadline classes. *European Journal of Operational Research* 105, 390-400.

[31] Li, L.Y.O. Leon, R.W. Eglese (1996). An interactive algorithm for vehicle routeing for winter - gritting. *Journal of the Operational Research Society* 47, 217-228.

[32] Liebling T.M. (1970). Graphentheorie in Planungs - und Touren problemen am Biespiel des städtischen Strassendienstes. Lecture Notes in Operations Research and Mathematical Systems, Springer-Verlag, Berlin.

[33] Liebling T.M. (1973). Routing problems for street cleaning and snow removal. In R. Deininger, (Ed.), *Models for Environmental Pollution Control,* 363-374.

[34] Lord, B. (1996). Federal Highway Administrator in an address at the Eastern Winter Road Maintenance Symposium & Equipment Expo, Washington, D.C., September 11, 1996.

[35] Lotan, T., D. Cattrysse, D.V. Oudheusden, and K.U. Leuven (1996). Winter gritting in the Province of Antwerp: a combined location and routing problem. *Belgian Journal of Operations Research, Statistics and Computer Science* 36, 141-157.

[36] Male, J.W. and J.C. Liebman (1978). Districting and routing for solid waste collection. *J. Environ. Eng. Div.* 104, 1-14.

[37] Marks, H.D. and R. Stricker (1971). Routing for public service vehicles. *ASCE Journal of the Urban Planning and Development Division* 97, 165-178.

[38] McKelvey, B. (1995). Snow in the Cities: A History of America's Urban Response, University of Rochester Press, Rochester, New York, 1995.

[39] Miner, R. (1997). Personal communication.

[40] Miner, M. (1997). Simulations show ways to boost plow productivity up to 90%. *Better Roads* 67 (4), April 1997, 26-30.

[41] Miner, W.M. and Bretherton, S. (1996) Route Optimization for Winter Maintenance Activities - 1995-1996 *Field Trial of Roadway Snow Plowing.* 1996 TAC Annual Conference. Draft.

[42] Minsk, L.D. (1981). Snow Removal Equipment, in D.M. Gray and D.H. Male (eds.), *Handbook of Snow*, Pergamon Press, Toronto, Canada, 1981, pp. 648-670.

[43] Minsk, L.D. (1970). A Short History of Man's Attempts to Move through Snow, in Snow Removal and Ice Control Research, Special Report 115, Proceedings of an International Symposium held April 8-10, 1970, Highway Research Board, National Academy of Sciences, Washington D.C. 1970, pp. 1-7.

[44] Office of the Legislative Auditor, Snow and Ice Control, A Best Practices Review, Office of the Legislative Auditor, State of Minnesota, St. Paul, Minnesota, May 1995.

[45] Simcox, M. (1998). Personal communication.

[46] Thompson, P. (1997). Personal communication.

[47] Tucker, W.B. and G.M. Clohan (1979). Computer simulation of urban snow removal. *Transportation Research Board Special Research Report Number* 185, 293-302.

[48] Waddell, B. (1994). Snow and ice control excellence with routing software, *Public Works*, 125, Sept. 1994, 72-74, 1994.

[49] Wang, J.-Y. (1992). Computer Aided System for Planning Efficient Routes. Ph.D. dissertation. Purdue University, West Lafayette, Indiana.

[50] Wang, J.-Y., P. Kandula and J.R. Wright. (1995). Evaluation of computer-generated routes for improved snow and ice control. *Transportation Research Record* 1509, 15-21.

[51] Wang, J.-Y., and J.R. Wright (1994). Interactive design of service routes. *Journal of Transportation Engineering* 120, 6, 897-913.

[52] Wang, J.-Y., and J.R. Wright (1992). Integrating GIS and CAD for transportation data base development. *Transportation Research Record* 1304, 212-218.

[53] Winter Urban Environment Research Subcommittee (1994). Harmony Between Road Management and Environment in Winter, Technical Manual of Winter Road Management, International Association of Mayors of Northern Cities, Sapporo, Japan, March 1994.

[54] Wright, J.R. (1997). Personal contact.

[55] Wright, J.R. (1993). The Computer Aided System for Planning Efficient Routes. Joint Highway Research Project Draft Final Report FHWA/IN/JHRP-93-8. Purdue University, West Lafayette, Indiana.

Chapter 11

SCHEDULING OF LOCAL DELIVERY CARRIER ROUTES FOR THE UNITED STATES POSTAL SERVICE

Lawrence Bodin
R. H. Smith School of Business
University of Maryland
College Park, MD 20742
lbodin@rhsmith.umd.edu

Laurence Levy
RouteSmart Technologies
8850 Stanford Boulevard
Columbia, MD 21045
llevy@routesmart.com

1. Introduction 420
2. The Route Adjustment Process 422
   2.1 Inspection 422
   2.2 Route Adjustment 423
   2.3 Auxiliary (or Remnant) Routes 424
3. Types of Delivery Routes 424
   3.1 Park and Loop Routes 425
   3.2 Curbline/ Dismount Routes 426
   3.3 Combined Park and Loop and Curbline/ Dismount Routes 426
   3.4 Relay Box Routes 427
4. Lines of Travel for the Four Types of Postal Delivery Routes 427
   4.1 Walking Line of Travel 428
   4.2 Driving Line of Travel 428
5. Examples of the Lines of Travel for the Four Types of Postal Delivery Routes 429
   5.1 Park and Loop Routes 430
   5.2 Curbline/ Dismount Routes 431

5.3 Combined Park and Loop and Curbline/ Dismount Routes 432
5.4 Relay Box Routes 433
5.5 Discussion 434
6. Algorithm for Route Adjustment 435
6.1 Estimate the Number of Routes to Form 436
6.2 Partition the AOI 437
6.3 Form the Line of Travel for Each Partition 438
6.4 Is the Solution Balanced? 439
7. Manual Intervention 440
8. Conclusions 440

1. INTRODUCTION

The United States Postal Service (USPS) delivers over 166 billion pieces of mail each year to over 100 million delivery locations. The delivery operation of the USPS is so large that it accounts for over 40 percent of the mail volume in the world. The USPS has over 275,000 postal carriers who are involved in USPS local delivery operations on a daily basis (Wright, 1992 and Assad and Golden, 1995).

The local delivery routes for the USPS are broken down into two classes, city routes and rural routes. Historic procedures are in place to determine whether a route is classified as a city route or a rural route. The carriers that service the city delivery routes (the routes considered in this chapter) belong to a union, the National Association of Letter Carriers (NALC). In this chapter, we shall concentrate on the various classes of city delivery routes and describe procedures for performing adjustments to these routes.

The current route adjustment process for the city routes is a well established, time consuming, manual process. This process adheres to a set of rules that both the USPS and the NALC agree ensures fairness and equity. The rules to carry out a route adjustment process are documented in Handbook M-39. The rules for adjusting the rural routes are different than the rules for adjusting the city routes. These rules are not contained in Handbook M-39 and are not described in this chapter.

The agreed-upon *goals of the route adjustment process* for the city routes are as follows:

i. The delivery routing is carried out in an optimum fashion so that each letter carrier has an 8 hour workday.
ii. Each carrier spends the same amount of time each day in the office.
iii. Each carrier spends about the same amount of time each day delivering mail.
iv. A balance in workload among the letter carriers is achieved.
v. Does the route represent an 8 hour workday.

A letter carrier's workday normally consists of two parts. A letter carrier normally spends the first three hours of his workday at the postal facility organizing and sorting the mail on his route. The carrier then travels to his route and spends the remainder of the day delivering the mail.

The USPS anticipates that the city routes (as well as the rural routes) will require realignment with increased frequency as technology improvements in the processing and sorting of the mail, such as automated delivery point sequencing of the mail, are implemented (Cebry, DeSilva and DiLisio, 1992), Assad and Golden, 1995). As these new technologies are implemented within the USPS, it is anticipated that the carrier will spend less time (up to one hour a day) at the postal facility and more time delivering mail. The route adjustment process will be used to implement changes in the routes that must occur when the routes get out of balance and these routes no longer represent 8 hour workdays.

Technology, however, is not the only reason for a route adjustment. The letter carrier's time to carry out his daily activities changes as new residences and offices are built on the route and mail volumes change. Since some routes change faster than other routes, routes become unbalanced. Some routes may require overtime and other routes may have some slack time. The demands in an area may grow to such an extent that an extra letter carrier may have to be added to the area. In all of these cases, the route adjustment process must be used to adjust the routes in order to satisfy the goals of the route adjustment process listed above.

The two types of route adjustments that can be performed are called major route adjustments and minor route adjustments. In a major route adjustment, the existing route structure is either completely or partially discarded and new routes are formed over the region. In a minor route adjustment, street segments are moved from one route to another in order to achieve better balance among the routes. Today, both major and minor route adjustments are accomplished manually by the USPS.

In the terminology of the USPS, the *line of travel* for a route is the sequence of street segments that a carrier must follow in servicing his route. The line of travel for many of the carrier routes is *bimodal* and consists of two parts - the driving line of travel and the walking line of travel. The *driving line of travel* for a route is the sequence of street segments that a carrier must follow while driving his route and the *walking line of travel* for a route is the sequence of street segments that a carrier must follow while walking his route. The driving line of travel and the walking line of travel have to be carefully integrated together to form the line of travel for the route. Since the postal carrier routes can involve two modes of transportation, walking and driving, the lines of travel for the postal carriers can be different than the lines of travel for other arc routing applications.

The major focus of this chapter is to describe the various types of local delivery carrier routes and the resulting lines of travel. As part of this description, algorithms for generating these (possibly bimodal) lines of travel are outlined and examples that illustrate these lines of travel are presented. Also, in this chapter, the USPS route adjustment process is described, an algorithmic approach for doing major route adjustments is outlined and some brief remarks on manual intervention and the current status of automated routing within the route adjustment process is presented.

2. THE ROUTE ADJUSTMENT PROCESS

The current procedure for route inspection and route restructuring consists of the following three steps - preparation, inspection and route adjustment. *Preparation* includes the training of personnel on inspection procedures, a review of the current operations, and the scheduling and planning of the inspection. *Inspection* includes the timing of the in-office activities, the counting of the mail volumes in the office and the recording of actual carrier performance on the route. *Route adjustment* includes the carrying out of the actual route adjustments. The effective execution of the route adjustment step includes the route examiner consulting with the carriers involved in the route adjustment and the implementation of the adjusted routes. The USPS and the NALC have agreed that the route adjustments must be implemented within 52 days of the completion of the inspection of the routes. The inspection and route adjustment steps of this process are now discussed in more detail.

2.1. INSPECTION

In the inspection step, the route examiner attempts to ascertain if there are problems with any of the routes being examined. More specifi-

cally, for any route involved in the route adjustment process, the examiner tries to determine the following:

i. Does the route begin and end close enough to the postal facility to eliminate the need for the carrier driving to the route?
ii. Can some deadheading be eliminated from the route?
iii. Is the *driving line of travel* on a driving route as safe as possible? Can adverse turns be eliminated?
iv. Are there too many park locations on a park and loop route? Too many park locations on a route lead to excessive vehicle movements and delays.
v. Does the route represent an 8 hour workday?

If the route fails any of these considerations just mentioned, then this route is a candidate for a route adjustment.

2.2. ROUTE ADJUSTMENT

If, after completing a route inspection, the inspector determines that a route adjustment is necessary, then the route adjustment procedures are invoked. These procedures decide upon the street segments to remove from the routes (called the tentative amount of *relief on the routes*) and street segments to add to the routes (called the *addition required to the routes*) so that each route represents approximately 8 hours of work on a daily basis. As a result of this realignment process, the following can occur:

i. An existing auxiliary route can be changed in size or eliminated. Auxiliary routes are described in 2.3.
ii. A regular route can be reduced to auxiliary status.
iii. A route can be eliminated entirely.
iv. A new route can be formed.

Other considerations in the route adjustment process are as follows:

i. Some of the routes are to begin and end close enough to the postal facility to eliminate the need for the carrier driving to the route.
ii. Deadheading is reduced.
iii. The routes are compact.
iv. The routes do not cross ZIP Code boundaries.
v. Adverse turns are eliminated on the curbline/ dismount routes so that the routes are as safe as possible.
vi. The number of park locations on a park and loop (walking) route is minimized.

As noted above, if this adjustment process starts with the existing routes and manually moves territory between routes, then this route

adjustment is called a *minor route adjustment.* If the existing routes are not maintained and new routes formed, then this route adjustment is called a *major route adjustment.*

2.3. AUXILIARY (OR REMNANT) ROUTES

In 2.2, the notion of an auxiliary route was presented. An auxiliary (or remnant) route can play a critical role in carrying out a route adjustment. An *auxiliary route* is a partial route that *does not represent an 8 hour workday.* An auxiliary route can be used in the following situation. Suppose that the workload in a region currently undergoing a route adjustment is estimated to be 9.4 carrier routes. This estimate is derived using the estimation procedure presented in 6.1. Then, the route adjuster can form nine 8 hour routes that are balanced and represent a full workday plus an auxiliary route representing .4 of a full route. The auxiliary route can be strategically located in a high growth area to absorb the growth in the area or be combined with the auxiliary route from an adjacent region to form an 8 hour route. These routes can be formed using the procedure given in section 6.

Even though the workload indicates that an auxiliary route is needed, the route adjuster can decide to eliminate the auxiliary route. Thus, in the above example, the route adjuster can decide to use the procedure described in Section 6 either to form 9 complete routes and pay a little overtime to all of the carriers or 10 routes and have each route somewhat under 8 hours in duration. In some cases, overtime can be a cost-effective option in forming routes. If the size of the auxiliary route is small, then the cost of overtime may be less than using one additional carrier. If the area is growing quickly, then forming an additional route can be desirable because these routes can absorb the growth and postpone the need to do another route adjustment.

Auxiliary routes are important considerations in arc routing applications. In several applications that we have encountered, auxiliary routes have led to successful, cost-effective solutions being generated.

3. TYPES OF DELIVERY ROUTES

Park and loop and curbline routes were introduced in the description of the route adjustment process in 2.2. If all of the street segments on the route requiring delivery are serviced by a carrier walking the route, then the route is *called a park and loop route* or *relay box route.* On the other hand, if all of the street segments on the route requiring delivery are serviced by a carrier driving the route, then the route is called a *curbline/ dismount route.* Furthermore, the possibility exists for the

carrier to service some of the street segments while driving the route and service the remaining street segments while walking the route. This route is called a combined *curbline/ dismount and park and loop route.*

A park and loop route and a combined curbline/ dismount and park and loop route involve two modes of transportation - driving and walking. Park and loop routes and combined curbline/ dismount and park and loop routes are examples of *single vehicle bimodal arc routing problems.* Another major example of a bimodal arc routing problem is meter reader scheduling.

In this section, the four major types of city delivery routes serviced by the carriers of the USPS are described. Each route type represents a single vehicle arc routing problem. If $a(m,n)$ and $a(n,m)$ are two arcs or edges on the street segment between intersections m and n, then $a(m,n)$ and $a(n,m)$ are called *counterpart arcs or edges* (Levy, 1987). Assuming that $a(m,n)$ and $a(n,m)$ both require service, then, in most cases, $a(m,n)$ and $a(n,m)$ are serviced by the same carrier. Exceptions to this assumption occur when the counterpart arcs $a(m,n)$ and $a(n,m)$ are streets that are heavily traveled or difficult to cross. In these cases, the USPS may force these two street segments to be on different routes. This situation is an exception and not the ordinary way that the USPS carries out its local delivery operations. In the description of the route types given below, it is assumed that both counterpart arcs are on the same route.

3.1. PARK AND LOOP ROUTES

In a park and loop route, the carrier drives a vehicle from the postal facility to a park location on the route. After parking the vehicle at the **PARK** location, the carrier loads his satchel, walks a set of street segments comprising a **Walking LOOP** out of the **PARK** location and returns to the vehicle. Generally the street segments in a loop are serviced one side of the street at a time. An exception is when the USPS decides that a street segment should be serviced as a *zig-zag* or *meander.* The street segments making up a loop are constrained so that the mail to be delivered in the loop does not exceed the mail capacity of the carrier's satchel. In many cases, a loop takes no more than one hour to complete.

When a carrier returns to the vehicle after having completed a Walking Loop and there is another Walking Loop to be performed by the carrier from the park location, then the carrier refreshes his satchel and begins the next Walking Loop. This process continues until all Walking Loops out of a park location are serviced. Then, the carrier moves the

vehicle to the next park location and repeats the process. Upon finishing all of the Walking Loops in the route, the carrier drives the vehicle back to the postal facility. This return to the postal facility completes the park and loop route.

In the algorithms that we have designed for this problem (Levy, 1987, Levy and Bodin, 1988, Assad and Golden, 1995), each street segment adjacent to a park location is in a different loop. This assumption reflects an actual consideration of the USPS. The USPS assumes that if a carrier walks past a park location on a loop, the carrier will stop at his vehicle and refresh his satchel to service the next set of arcs on the loop. This assumption is perfectly reasonable since there is no need for the carrier to carry mail any longer than necessary.

Thus, in an efficient park and loop route,

i. A different loop emanates out of each street segment adjacent to a park location.
ii. Park locations are generally chosen so that there are at least three street segments incident to it.
iii. The number of park locations in a park and loop route is minimized. Minimizing the number of park locations reduces the amount of deadhead driving on the route.

3.2. CURBLINE/ DISMOUNT ROUTES

In either a curbline or dismount motorized city delivery route, all deliveries are made while the carrier drives the vehicle from one stop to the next stop over all the street segments requiring service. There are no walking deliveries in a curbline/ dismount route. The difference between a curbline route and a dismount route is as follows. In a *curbline route*, more than 50 percent of the possible deliveries are made to customer mailboxes at the curb. The carrier does not leave the vehicle to make the curbline deliveries. In a *dismount route*, more than 50 percent of the deliveries are made by dismount delivery to the door. In this case, the carrier leaves the vehicle and delivers the mail to each customer requiring a dismount delivery. A curbline/ dismount route is the only local postal delivery route guaranteed to involve one mode of transportation from its beginning at the postal facility to its termination at the postal facility.

3.3. COMBINED PARK AND LOOP AND CURBLINE/ DISMOUNT ROUTES

Often, a city delivery route will contain both park and loop and curbline/ dismount service. In a combined park and loop and curbline/ dismount route, the carrier delivers some of the mail while walking (park

and loop) and delivers the remainder of the mail while driving. Both the park and loop route and the curbline/ dismount route are special cases of the combined park and loop and curbline/ dismount route. As noted above, in a park and loop route, there are no deliveries made while driving and, in a curbline/ dismount route, there are no deliveries made while walking a loop. The algorithms for solving the general park and loop and curbline/ dismount problem must consider the bimodal aspect of the problem.

3.4. RELAY BOX ROUTES

A relay box route is a variant of a park and loop route. In a relay box route, the carrier generally does not drive a vehicle but arrives at the initial location by some means of transportation with a full satchel of mail to deliver. The carrier then delivers the mail in the satchel and arrives at a relay box. The carrier then refreshes his satchel with additional mail at the relay box and continues to make deliveries while walking. In essence, the carrier carries out walking loops out of the relax box and the relay boxes in a relay box route are analogous to the park locations in a park and loop route. In a relay box route, the carrier must walk between relay boxes whereas, in a park and loop route, the carrier drives between park locations. Generally, the last loop out of a relay box is designed so that the carrier ends up at the next relay box as the last location of this loop. Upon terminating the route, the carrier generally returns to the postal facility by some means of transportation determined by the USPS.

In a relay box route, the mail is waiting for the carrier when he arrives at the relay box after having delivered the mail that the carrier has brought with him from the postal facility. To accomplish this, another carrier delivers the mail to the relay box in advance of the carrier refreshing his satchel at the relay box. The delivery of the mail to the relay box is a node routing problem with time windows and is not considered in this chapter. Since relay boxes are easy to move, an algorithm for solving this problem can be developed to allow for some flexibility in the location of each of the relay boxes.

4. LINES OF TRAVEL FOR THE FOUR TYPES OF POSTAL DELIVERY ROUTES

In Section 3, the four major types of delivery routes were described. Each delivery route type involves one carrier and is a variant of a single vehicle Chinese Postman problem. Even if the street segments to be serviced on a delivery route (arcs in a network) are identical, the *line of*

travel for the carrier can depend on the type of delivery route.

Let A be the set of street segments requiring service on a single route. $A = B \cup C$ where B is the set of street segments requiring walking service and C is the set of street segments requiring driving service. The networks, $G(A) = [N(A), A]$, $G(B) = [N(B), B]$, and $G(C) = [N(C), C]$ may be disconnected. $G(A)$, $G(B)$ and $G(C)$ are subnetworks of the overall street network that is assumed to be strongly connected.

Disconnectivities force many additional considerations on the formation of the lines of travel. Some of these issues are discussed in Assad and Golden, 1995 and Win, 1989. Further, in the driving line of travel problem, some streets can be zig-zagged and the other streets require two traversals for servicing the counterpart arcs. Thus, the generation of the line of travel can be over a mixed network where some arcs are directed and other arcs are undirected.

4.1. WALKING LINE OF TRAVEL

Assuming arc set B is not empty and the network $G(B)$ is connected, then the procedure for generating the walking line of travel for the arcs in $G(B)$ can be outlined as follows. This approach was first presented in Levy, 1987.

i. The network, $G(B)$, is extracted from the overall street network.
ii. The park locations for the walking line of travel for this route are determined.
iii. The arcs in B are broken down into clusters where each cluster represents the streets to be serviced in a walking loop (see Levy, 1987). The seed points of these clusters are the counterpart arcs that represent a street segment incident to a park location and every street segment incident to a park location is a seed point for a cluster.
iv. The walking line of travel for each cluster is found by solving a variant of a Windy postman problem (Win, 1989).

4.2. DRIVING LINE OF TRAVEL

An algorithm based on a variant of an algorithm for solving either a directed Chinese postman problem or Traveling Salesman problem (Laporte, 1997) can be used to give the driving line of travel. This driving line of travel covers every street segment in $G(C)$ plus the streets that have to be deadheaded to form a continuous directed line of travel. In all cases, the algorithm is based on a directed graph where an undirected street segment is replaced by two directed street segments (in opposite

directions) for deadheading purposes if the street is 2-way and one directed street segment in the direction of traffic if the street is 1-way.

5. EXAMPLES OF THE LINES OF TRAVEL FOR THE FOUR TYPES OF POSTAL DELIVERY ROUTES

In this section, examples of the lines of travel for the four types of postal delivery routes described in Section 3 are presented. In these examples, both counterpart arcs for a street segment are assumed to re quire service by the same carrier. As noted earlier, this assumption is generally a requirement on any USPS city route. In practice, however, it is possible for one side of a street segment to require service and the other side of the street segment to not require service. An example of this situation is when one side of a street segment has residences and small business and the other side of the street segment is a recreation area with no locations requiring service.

We will use the same network (Figure 11.1) to illustrate the different lines of travel. This network is comprised of the street segments that make up a single carrier route. All street segment are counterpart arc pairs and each street segment requires service by either walking the street segment or driving the street segment. The type of service on each street segment depends upon the type of delivery route problem being considered and will be specified in each example.

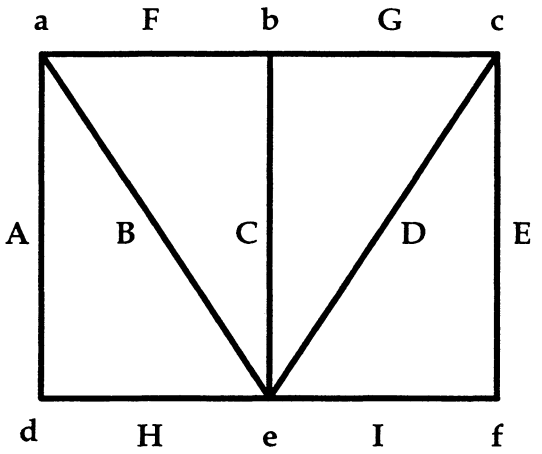


Figure 11.1 Network for Examples of the Different Types of Local Delivery Routes.

5.1. PARK AND LOOP ROUTES

Assume that all edges in Figure 11.1 require service, any service is accomplished by walking the street segment and each side of a counterpart arc pair has to be serviced on a separate traversal of the street segment (i.e. no zig-zagging of a street segment is permitted). Since the carrier walks each street segment while servicing the street segment, the service can be carried out in either direction on the street segment.

5.1.1 Walking Line of Travel

The steps that are followed in determining the walking line of travel are the following.

a. Park Locations
The park locations are d and f.

b. Walking Loops:
Walking Loop 1: A, F, F, A.
Walking Loop 2: H, B, B, H.
Walking Loop 3: I, C, C, I.
Walking Loop 4: E, G, G, D, D, E.

c. Walking Line of Travel for Walking Loop 1
Leave park location d.
Service one side of A.
Service one side of F.
Service the other side of F.
Service the other side of A.
Return to park location d.

Comments: The Walking Line of Travel for the Walking Loops 2-4 can be found in a manner similar to the way the Walking Line of Travel was found for Walking Loop 1. Moreover, in an actual situation, we would probably use e as the only park location for this route since it has four street segments adjacent to it. However, in this example, we used two park locations to better illustrate the Driving Line of Travel. Also, in the description of the walking loops and Walking Line of Travel, street segments are repeated twice because each side of each street segment has to be serviced.

5.1.2 Driving Line of Travel

Leave the Postal Facility.
Drive to Park Location d and Park the Vehicle.
Drive to Park Location f and Park the Vehicle.
Return to the Postal Facility.

5.1.3 Overall Line of Travel

The overall line of travel for this park and loop route joins the walking lines of travel for each walking loop with the driving line of travel to get a complete sequencing of the streets in the postal carrier's line of travel. This overall line of travel is the following.

Deadhead drive from the Postal Facility to Park Location d. Park at d.
Walking Loop 1: A, F, F, A.
(The walking line of travel for this walk loop is specified in 5.1.1.c.)
Walking Loop 2: H, B, B, H.
Deadhead drive from Park Location d to Park Location f.
Walking Loop 3: I, C, C, I.
Walking Loop 4: E, G, G, D, D, E.
Deadhead drive from Park Location f to the Postal Facility.

5.2. CURBLINE/ DISMOUNT ROUTES

Assume that all street segments in Figure 11.1 make up a curbline/ dismount route. The network displayed in Figure 11.1 for this case is $G(C) = [N(C), C]$. Further, each of the counterpart arcs of each street segment has to be serviced by the carrier on a separate traversal of the street segment. In this example, it is assumed that the carrier cannot meander or zig-zagging a street segment, there are no one-way streets, and the service has to be carried out in the direction of travel on the street segment. In this case, this problem turns out to be the generation of an Euler circuit over a directed, connected network where each node is in balance (i.e. each node in $G(C)$ has the same number of arcs into the node as out of the node).

As noted above, all street segments in the curbline/ dismount problem are serviced by driving the street segment and there are no park locations or walking loops. Assuming that the closest node to the depot is node a, all street segments permit two-way traffic and U turns are acceptable, then a line of travel for this example is the following.

Line of Travel:
Deadhead drive from the Postal Facility to a.
Drive the street segments (or arcs in the network) in the following order:
$F, C, C, G, D, D, E, I, B, B, H, A, A, H, I, E, G$ and F.
(There is no deadhead driving in this part of the line of travel.)
Deadhead drive from a to the Postal Facility.

5.3. COMBINED PARK AND LOOP AND CURBLINE/ DISMOUNT ROUTES

In the example of the combined park and loop and curbline/ dismount problem, both sides of every street segment in Figure 11.1 have to be serviced. Assume that all street segments except for C and G have to be served as walking street segments and that C and G have to be serviced as driving street segments. To keep this example close to the Park and Loop example in 4.1, d and f are assumed to be the park locations.

5.3.1 Walking Line of Travel

a. Park Locations:
Park locations are d and f.

b. Walking Loops:
Walking Loop 1: A, F, F, A.
Walking Loop 2: H, B, B, H.
Walking Loop 3: I, I.
Walking Loop 4: E, D, D, E.

c. Walking Line of Travel for Walking Loop 1 (The Walking Line of Travel for the Walking Loops 2-4 can be found in a similar manner.)
Leave park location d.
Service one side of A.
Service one side of F.
Service the other side of F.
Service the other side of A.
Return to park location d.

5.3.2 Driving Line of Travel

Leave the Postal Facility.
Deadhead to Park Location d and park the vehicle at d.
Deadhead to e.
Service one side of C.
Service one side of G.
Service the other side of G.
Service the other side of C.
Deadhead to f and park the vehicle at f.
Deadhead from f to the Postal Facility.

5.3.3 Overall Line of Travel

Deadhead drive from the Postal Facility to Park Location d. Park at d.
Walking Loop 1: A, F, F, A.
Walking Loop 2: H, B, B, H.
Deadhead drive on H from d to e.
Service one side of C while driving.
Service one side of G while driving.

Service the other side of G while driving.
Service the other side of C while driving.
Deadhead drive on E to Park Location f. Park at f.
Walking Loop 3: I, I.
Walking Loop 4: E, D, D, E.
Deadhead drive from e to the Postal Facility.

5.4. RELAY BOX ROUTES

The Relay Box problem is similar to the Park and Loop Problem. As with the Park and Loop Problem, both sides of all of the street segments in Figure 11.1 require walking service. Further, meandering or zig-zagging of a street segment is not permitted. Since each street segment is being walked while being serviced, the service can be carried out in either direction on the street segment. The generation of the line of travel for this relay box route is given below:

a. Relay Box Locations
The Relay Box locations are d and f and these locations are known in advance.

b. Walking Loops:
Walking Loop 1: A, F, F, A.
Walking Loop 2: H, B, B, H.
Walking Loop 3: I, C, C, I.
Walking Loop 4: E, G, G, D, D, E.

c. Walking Line of Travel for Walking Loop 1 (The Walking Line of Travel for the Walking Loops 2-4 can be found in a similar manner.)
Leave the relay box at location d.
Service one side of A.
Service one side of F.
Service the other side of F.
Service the other side of A.
Return to the relay box at location d and reload satchel.

Overall Line of Travel:
Get to d from the Postal Facility by either a USPS vehicle or public transportation.
Carry out the following in the order specified
Walking Loop 1: A, F, F, A.
Walking Loop 2: H, B, B, H.
Deadhead walk from d to f.
Walking Loop 3: I, C, C, I.
Walking Loop 4: E, G, G, D, D, E.
Return to the Postal Facility by USPS vehicle or public transportation.

5.5. DISCUSSION

The examples presented above in this section described different types of single vehicle arc routing problems. These arc routing problems demonstrated the different lines of travel that would be encountered by the USPS depending upon the type of route that was formed. The route can be bimodal; that is to say, the route can have a driving component and a walking component. Other complications that have to be accounted for in an algorithm for finding the line of travel for a route are as follows:

i. The network made up of the street segments to be serviced for both the driving line of travel problem and the walking line of travel problem may be disconnected.

ii. The driving line of travel can involve both one-way streets and two-way streets. There is no meander or zig-zag service permitted on the street segments in the driving line of travel problem. However, the algorithms have to be able to handle this situation if zig-zaggings should occur (Win, 1989).

iii. As demonstrated in the above examples, a bimodal travel path. has the walking loops superimposed on top of driving portion of the line of travel. These walking loops or walking lines of travel account for the park and loop and relay box portions of the routes.

iv. As with most single vehicle arc routing problems, the overall objective in finding this line of travel is to reduce the total nonproductive time on the route. Nonproductive time includes deadhead travel time (both walking and driving) and the time that the USPS carrier spends in refreshing his satchel.

v. Minimizing the number of park locations generally reduces *a*) the nonproductive time on the route by reducing the excessive vehicle movements on park and loop routes and combined park and loop and curbline/dismount routes and *b*) the number of times the satchel of the USPS carrier has to be refreshed or filled up.

vi. The line of travel must be as safe as possible. This safety can be achieved by avoiding adverse or unsafe turns and street crossings. Some adverse or unsafe turns and street crossings can be eliminated by penalizing these situations in the algorithm described in the next section to make these turns undesirable. Further, the route adjuster that inspects the solution to make sure the routes are safe can force certain desirable situations if the route adjuster is not satisfied with the computer-generated routes. However, eliminating these adverse turns can introduce more deadheading.

6. ALGORITHM FOR ROUTE ADJUSTMENT

If minor route adjustments are to be performed, then having an algorithm for generating a line of travel for each route is the only algorithm that is needed. The procedure for generating these lines of travel is outlined in section 6.3 below. If, however, the system is to generate the partitions as well as the line of travel, then a more complicated algorithmic structure evolves. This means that the line of travel algorithms must be imbedded within a graph partitioning procedure that breaks down an *Area of Interest (AOI)* into a set of partitions representing the carrier routes. For the USPS, an AOI can be thought of as a 5 digit zip code area. A 5 digit zip code area generally contains no more than a few thousand street segments and 50 routes - a relatively ideal size.

This overall procedure that is now described assumes the following:

i. The service time on each of the street segments is relatively small (for example, in most cases, the time for a carrier to service a street segment is no more than 15 minutes).
ii. The number of street segments on most of the routes is relatively large (generally at least 15 street segments).
iii. There are no time windows.
iv. An auxiliary route is possible. Each route (except for the auxiliary route) must contain about 8 hours work. This route time includes office time, time to/from the route and delivery time on the route.
vi. Route interlacing or route overlap is as small as possible.

A simple explanation of interlacing routes is as follows. Two routes do not interlace or overlap if it is possible to draw a polygon around each of these two routes and the polygons do not have any area in common. Routes that do not interlace are easy to administer as there is no confusion as to the street segments that belong on each route. In many cases, the route adjuster will manually swap street segments between routes to eliminate interlacing. As with reducing adverse turns, manually swapping street segments to reduce interlacing generally results in additional deadheading. Before computerized routing, the street segments on each route in an area were colored with a different color. In this way, the supervisor in the area could easily determine the streets assigned to each route.

A *priority partition* is a set of streets that have to be serviced early in the day. For example, a priority partition can be the streets that exist in a commercial portion of the AOI. The USPS has decided that it is desirable in many cases to have the commercial enterprises in the AOI receive their mail early in the day. Although priority partitions exist, the algorithm outlined below does not consider priority partitions. We

have modified the algorithm described in this section to consider priority partitions.

In most cases, this algorithm is a double application of a 'cluster first, route second' procedure (Bodin, Golden, Assad and Ball, 1983). The algorithm first partitions the AOI into partitions where each street segment requiring service is assigned to a partition. Then, for each partition, the street segments that require walking service are further broken down into smaller clusters representing park and loops or relay box routes. Then, walking line of travel paths are generated over each of these smaller clusters that represent a park and loop or a relay box route. Finally, a minimum length driving line of travel is formed over all street segments requiring driving service and the park locations of the vehicles. The resulting travel path represents the overall line of travel for the partition. An overall line of travel is generated for each partition, including the auxiliary partition. Details of this overall procedure have been presented in Levy, 1987, Levy and Bodin, 1988, Levy and Bodin, 1989, Bodin and Levy, 1991, Bodin, Fagan and Levy, 1992a and 1992b, and Assad and Golden, 1995.

The statistics for the solution (partitions and lines of travel for each partition) are then examined. If the solution is not balanced, then the next iteration through the procedure is carried out with a revised estimate of the number of vehicles needed to service the stops. If this solution is superior to the *best solution found so far*, then this solution is saved as the best solution found so far. As soon as a balanced solution has been attained or a specified number of iterations have been carried out, then the algorithm terminates. The best solution found so far is manually examined to allow the postal inspector to manually swap street segments between routes in order to derive an improved or safer solution.

6.1. ESTIMATE THE NUMBER OF ROUTES TO FORM

The first step in the algorithm is to estimate the number of routes to service the street segments that require service in the AOI. This estimate need not be an integer. To determine this estimate, the total workload in the AOI is computed. *Total workload* is the total service time on all of the street segments plus allowances for deadhead travel time, time to park the vehicle, time to refresh the satchel, and an estimate of total office time. *Total office time* includes the time necessary to begin and end a day's work plus the time to sort the mail into bins representing the line of travel for the carrier.

To carry out this estimate, a *target workload* is specified. For example, if 8 hour (480 minute) routes are desired, the *target workload* may be 465 minutes. Then, the estimate of the number of routes to form over the AOI is simply the ratio of the total workload in the AOI divided by the target workload.

Since the estimate of the number of routes need not be integer, the need for an auxiliary route can be detected at this early stage in the algorithm. The user must decide to use one of the following in the next step of the algorithm:

i. Partition the AOI with a remnant route,
ii. Increase the estimate of the number of routes to the next largest integer (introducing slack into the routes) and partition the AOI with this increased number of vehicles, or
iii. Decrease the estimate of the number of routes to the next smallest integer and partition the AOI with this decreased number of vehicles, possibly introducing overtime into the routes.

6.2. PARTITION THE AOI

In this step, the AOI is broken down into the number of partitions determined in 6.1 where this partitioning can include the auxiliary route. To carry out this partitioning, an acceptable workload interval is specified where an *acceptable workload interval* is defined as follows [target workload - A, target workload + B] where A and B are generally small (15 minutes, for example). A solution is considered *balanced* if the duration of all of the routes fall in the acceptable workload interval. A balanced solution means that the final set of routes and lines of travel are all approximately the same duration and close to the desired length of a workday.

To accomplish the partitioning, a seed point is selected for each partition and all of the street segments in the AOI are assigned to a partition. This partitioning is carried out in two parts:

i. An initial partitioning of the street segments in the AOI.
ii. An automatic swapping of street segments between partitions.

The goal of the partitioning is that all of the partitions contain about the same workload. Counterpart arcs are assigned to the same partition. If the street segments that require service in the AOI form a disconnected network, then a minimum number of component connections are added to the network of required streets to form a connected network. The partitioning is carried out over this new network.

This partitioning is based strictly on the workload on a street segment and involves all of the walking and driving street segments that require service The partitioning is constructed so that little overlap between the partitions is achieved. As noted above, details of this step can be found in Levy, 1987, Levy and Bodin, 1988, and Assad and Golden, 1995.

6.3. FORM THE LINE OF TRAVEL FOR EACH PARTITION

The AOI has been broken down into partitions where each partition represents the set of street segments to be serviced by a carrier. The problem now is to develop a bimodal line of travel for each partition. This bimodal line of travel is carried out in the following order.

6.3.1 Walking Line of Travel Problem

The *Walking Line of Travel Problem* is the original problem that we studied for the USPS (Levy, 1987 and Levy and Bodin, 1988). In the Walking Line of Travel Problem, all walking street segments in a partition are identified. In GeoMod, the route system we developed for the USPS, identifying the walking street segments is a partition is easy because each street segment has an carrier number associated with it. The walking street segments in the partition are then broken down into clusters where each cluster represents a loop for a park and loop route or a loop for a relay box route. Then, a minimum deadhead time walking line of travel is found for each of the clusters. To accomplish this clustering, the following is carried out.

i. The workload over all of the walking street segments is computed.
ii. A set of park locations is selected.
iii. Each street segment adjacent to a park location is denoted as a seed point in the clustering.
iv. A modified version of a heuristic clustering procedure or set partitioning procedure is solved.

Then, a minimum deadhead time travel path is formed over each of these clusters by solving an Undirected Chinese Postman Problem (UCPP). The UCPP can become complex since the street segments that require driving service can form a highly disconnected network. However, since the underlying street network is strongly connected, a solution to the UCPP can be found. The component connections used in 6.2 are disregarded in this step; these connections were just used to form the initial set of partitions.

6.3.2 Driving Line of Travel Problem

After a Walking Line of Travel Problem has been found for each walking partition, a Driving Line of Travel Problem is solved to find the Driving Line of Travel for that partition. The entities to be serviced on the Driving Line of Travel are the following:

i. The street segments between the route and the postal facility.
ii. All street segments that require driving service in the partition.
iii. All park locations.

This problem turns out to be a small, directed Chinese Postman Problem can be tricky to solve because of this large number of disconnectivities, turn difficulties, street crossing difficulties and the zig-zag service on some street segments.

We have had success in solving this problem as a Traveling Salesman Problem (TSP) where each required street segment is a node in the TSP (Laporte, 1997). The advantage of solving this problem as a TSP is that turn difficulties and street crossing difficulties can be accounted for in the objective function. Also, we have solved this problem as an arc routing problem using a connect-and-balance approach. In a connect-and-balance approach, the disconnected components are joined together by adding in street segments that are to be deadheaded forming a connected network. Once, the network is connected, the problem becomes a more traditional arc routing problem that can be solved by traditional procedures.

6.4. IS THE SOLUTION BALANCED?

After 6.3, the AOI is broken down into partitions, a bimodal line of travel is found over each partition, deadhead time estimates are determined and paths between the postal facility and each of the routes are found. The question then arises - is the solution balanced? That is to say, does the workload for each partition fall in the acceptable workload interval where the acceptable workload interval was defined as follows [target workload - A, target workload + B] (see 6.2). If the bimodal line of travel for all of the routes falls in the acceptable workload interval, then a balanced solution has been found and the algorithm terminates. Otherwise, the workload estimate is revised and the procedure is repeated. However, the workload estimate on iteration Q, $Q > 1$, is more accurate than the workload estimate found on iteration 1 since better estimates of deadhead time on each of the routes are now known.

This procedure continues for a specified number of iterations, P. If a balanced solution is not found on any iteration, the deviation of each

of the routes from the endpoints of the acceptable workload interval is computed and summed. If this sum is smaller than the previous best sum, then this solution is kept as the *best solution found so far.* If a balanced solution is not found on any of the P iterations, then the best solution found so far is specified as the final solution for this problem.

7. MANUAL INTERVENTION

The procedure described in 6.0 is a set of algorithms that gives a solution without manual intervention. However, our experience has shown that integral to a computer system for solving any vehicle routing problem is to allow the user to have the ability to perform manual adjustments to the solution (Bodin and Levy, 1994).

GeoLimited is a computer system that we built for the USPS for performing minor route adjustments manually. GeoLimited allows the route adjuster to transfer territory and set up travel paths while conforming to all of the workrules and regulations agreed upon by the USPS and the NALC. Presently, GeoLimited has been implemented with great success in all 85 USPS districts in the United States. The USPS has selected GeoLimited as the software system of choice for doing manual route adjustments.

8. CONCLUSIONS

In this chapter, both the major route adjustment process and minor route adjustment process for the USPS has been described. Emphasis has been placed on describing the various line of travel problems that the USPS encounters. In particular, bimodal lines of travel that involve walking and driving lines of travel (appropriately integrated together) have been introduced. The algorithms for solving these lines of travel have been imbedded within more traditional graph partitioning procedures in order to outline an approach for solving the major route adjustment problem as well. Furthermore, this approach (described in Section 6) can be modified for use in other applications such as the scheduling of meter reader and residential sanitation vehicles.

In section 7, moreover, the GeoLimited system for minor route adjustments has been briefly described and it was noted that GeoLimited is the USPS system of choice for manual route adjustments. USPS personnel have been trained on the use of GeoLimited and GeoLimited is an integral part of the minor route adjustment process. We can safely say that GeoLimited has been successfully used for route adjustments.

Automatic algorithms such as those described above are not being used by the USPS for major route adjustments. Moreover, the line of travel procedures described above are not used within GeoLimited for generating minimum deadhead travel paths. One of the main reasons why these procedures have not been used is based on the notion of standards. In most arc routing problems, the time to service a street segment or deadhead a street segment is assumed to be independent of the person carrying out the service. This is not the case with the USPS. In their collective bargaining agreements, the USPS and the NALC have agreed that the time to service or traverse a street segment is a function of the carrier carrying out the service. As such, the assignment of work to a partition depends upon the carrier assigned to the partition. Our research assumed that standards exist and this assumption was agreed to by our project director at the USPS. The USPS has groups investigating the use of these algorithms for major route adjustments. However, these groups have to reconcile the differences between the assumptions in the algorithms and a few of the workrules specified in the collective bargaining agreements.

Acknowledgments

The authors wish to thank Professor Moshe Dror and the reviewer of this chapter for their many useful and insightful comments on this chapter. The USPS has sponsored in part some of the research described in this paper with various contracts to Bowne Distinct, LTD and RouteSmart Technologies.

References

[1] Assad, A. and B. Golden (1995), "Arc Routing Methods and Applications", *Handbook in OR & MS*, Volume 8, M. Ball (ed.), Elsevier Science, 375-483.

[2] Bodin, L., B. Golden, A. Assad, and M. Ball (1983). "Routing and Scheduling of Vehicles and Crews. The State of the Art." *Computers & Operations Research*, 10(2), 69-211.

[3] Bodin, L., and L. Levy (1991), "The Arc Partitioning Problem", *European Journal of Operations Research*, 53(3), 393-401.

[4] Bodin, L., and L. Levy (1994), "Visualization in Vehicle Routing and Scheduling Problems", *ORSA Journal of Computing*, 6(3), 261-269.

[5] Bodin, L., G. Fagan and L. Levy (1992a), "The GEOMOD System", *Proceedings of the USPS Advanced Technology Conference*, Vol. 1, United States Postal Service, 413-418.

[6] Bodin, L., G. Fagan and L. Levy (1992b), "Vehicle Routing and Scheduling over Street Networks", *Proceedings of the USPS Advanced Technology Conference*, Vol. 2, United States Postal Service, 425-641.

[7] Cebry, M.E., A.H. DeSilva and F.J. DiLisio (1992), "Management Science Automating Postal Operations: Facility and Equipment Planning in the United States Postal Service", *Interfaces*, 22 (1), 110-130.

[8] Laporte, G. (1997), "Modeling and Solving Several Classes of Arc Routing Problems as Traveling Salesman Problems", *Computers & Operations Research*, 24, 1057-1061.

[9] Levy, L. (1987), "The Walking Line of Travel Problem: An Application of Arc Routing and Partitioning". Ph.D. Dissertation, University of Maryland, College Park, MD.

[10] Levy, L. and L. Bodin (1988), "Scheduling the Postal Carriers for the United States Postal Service: An Application of Arc Partitioning and Routing", *Vehicle Routing: Methods and Studies,* B.L. Golden and A. Assad (eds.), North-Holland, Amsterdam, 359-394.

[11] Levy, L. and L. Bodin (1989), "The Arc Oriented Location Routing Problem", *INFOR*, 27(1), 74-93.

[12] Win, Z. (1989), "On the Windy Postman Problem on Eulerian Graphs", *Mathematical Programming*, 44 (1), 97-112.

[13] Wright, J.W. (1992), *The Universal Almanac*, 1993, Andrews and McMeel, Kansas City, MO.

Chapter 12

LIVESTOCK FEED DISTRIBUTION AND ARC TRAVERSAL PROBLEMS

Moshe Dror
The University of Arizona

Janny M.Y. Leung
The Chinese University of Hong Kong

Paul A. Mullaseril
Mankato State University

1. Introduction 444
   1.1 The Cattle Industry 444
   1.2 The Cattle Yard Operations 444
2. Livestock Feed Distribution as Arc Traversals 447
   2.1 Arc-Routing Models for Pen Inspection and Feed Delivery 448
   2.2 Split Deliveries 450
   2.3 Time Windows 451
3. A Heuristic Approach for Trip Generation 452
   3.1 Generating a Non-split Feasible Solution 452
   3.2 Arc Swapping 455
   3.3 Generating Split Delivery Routes 456
   3.4 Route Addition 456
4. Route-First Cluster-Second Generalized TSP Heuristic 457
   4.1 The Generalized Traveling Salesman Problem 457
   4.2 Transforming the CRPP to an Equivalent GTSP 457
   4.3 Solving the Equivalent GTSP 458
5. Computational Results 458
6. Epilogue 461
7. Appendix 465

1. INTRODUCTION

We describe a setting where the arc routing activity is the principal operational focus of an enterprise. The cattle yard in Arizona studied in this chapter produces cattle for the general meat market. Cattlemen do not set the price for their product, which fluctuates depending on supply, demand and the psychology of the cattle market at the time of sale. Therefore, reducing operational costs is a major concern of a cattle yard, whose main operation is the daily delivery of feed to all the livestock. Since the feeding troughs are located along the sides of the pens, the profitability of the enterprise depends on finding the optimal arc traversals!

1.1. THE CATTLE INDUSTRY

The romantic image of the cattlemen's life on the range has fuelled the imagination of generations of young people all over the world, and is an icon of the United States' historical and cultural heritage. A modern cattle ranch belies this idyllic image. In fact, today's beef and dairy industry is very complex, and forms the largest part of the U.S. food and fibre industry, which, in turn, is the largest segment of the U.S. economy (about 17.5 percent of the gross national product). Cattle ranching, done in all 50 states (see Appendix), contributes 1.6 million jobs to thousands of rural economies and adds, directly and indirectly, $153 billion to the U.S. national economy.

Despite media reports about our changing diet, U.S. beef production increased by nearly 3 billion pounds (14%) and total cattle numbers increased by 7.7 million head (8%) from 1990 to 1996. In fact, production of all meats stayed at near record levels. According to the U.S. Department of Agriculture, sales of cattle and calves totaled more than $36 billion in 1996, the average per capita consumption of red meat and poultry is 209 pounds, and 7 billion beef servings were consumed in restaurants. Steakhouse restaurant traffic has increased 43.6% over the period 1993-1996. Furthermore, growing demand for quality grain-fed U.S. beef in foreign countries has fostered growth in beef exports, which now account for nearly 8 percent of beef output. Export of U.S. cattle, beef and beef products in 1996 totaled $4.8 billion, and the top export markets for U.S. beef are Japan, Korea, Canada, Mexico and Hong Kong.

1.2. THE CATTLE YARD OPERATIONS

This chapter describes the operations of a large cattle yard in Arizona, which employs about 135 cattlemen. This cattle yard houses over 100,000 head of cattle, on average, in about 600 cattle pens dispersed over an area that measures roughly 10 miles by 5 miles. The cattle yard is presently

being expanded to house over 125,000 head of cattle. As shown in Figure 12.1, in addition to pens for keeping livestock, this yard contains a large mixing plant that produces the daily feed rations, storage facilities (both for feed ingredients and finished feed), and loading/unloading facilities including a rail depot for receiving feed ingredients and shipping/receiving of cattle.

The cattle-feed is mixed from ingredients such as hay, straw, grain, alfalfa and trace nutrients at the mixing plant, and is stored in 5 or 6 large holding bins from where it is dispatched to the cattle pens by trucks. The feed has to be freshly cut and mixed; otherwise the cattle will reject it. The feed mixing plant is a large and highly automated facility which can produce the feed requirements of the cattle ranch in a few hours and its main focus is on the consistency and quality of the mix produced for each ration or feed type. A dispatch center close to the mixing plant co-ordinates the loading/unloading and timely dispatch of the feed distribution vehicles and also of the incoming and outgoing cattle. With over 100,000 head of cattle, there is almost a daily movement of new (young) incoming cattle and the shipping-out of mature cattle.

The operational activities in the cattle yard are intuitive and simple and are organized around the feeding needs and plans for the cattle. At the time of initiating this study, the feed distribution management of such large cattle ranches was based primarily on the experience and intuition of the operating crew. No attempt had been made to introduce computer-based planning tools or decision support systems for feed distribution.

This chapter primarily focuses on the daily problem of managing a fleet of trucks for distributing feed at this cattle yard. Almost every activity in the cattle yard is driven by the feeding needs of the cattle. The daily feed distribution operation requires a detailed planning and scheduling of the delivery routes for the trucks. Twice a day about seventy-five trips are made from the feed storage facilities to the pens. Each pen is delivered feed of only one type and the daily time windows for feed delivery are fixed. Feed is specially formulated for the various types and breeds of cattle; as the cattle mature, their diet requirements also change. Therefore, the feed type, volume of feed, and feeding time for each pen may vary from day to day. In this cattle yard, five different types of feed are produced and distributed. Only one type of feed may be carried in a truck in each trip. Since the feed types vary in density, the vehicle capacities also differ by feed type. The delivery trucks are specially equipped to automatically dispense feed into the feeding troughs which are located along a designated side of each cattle pen. The trucks

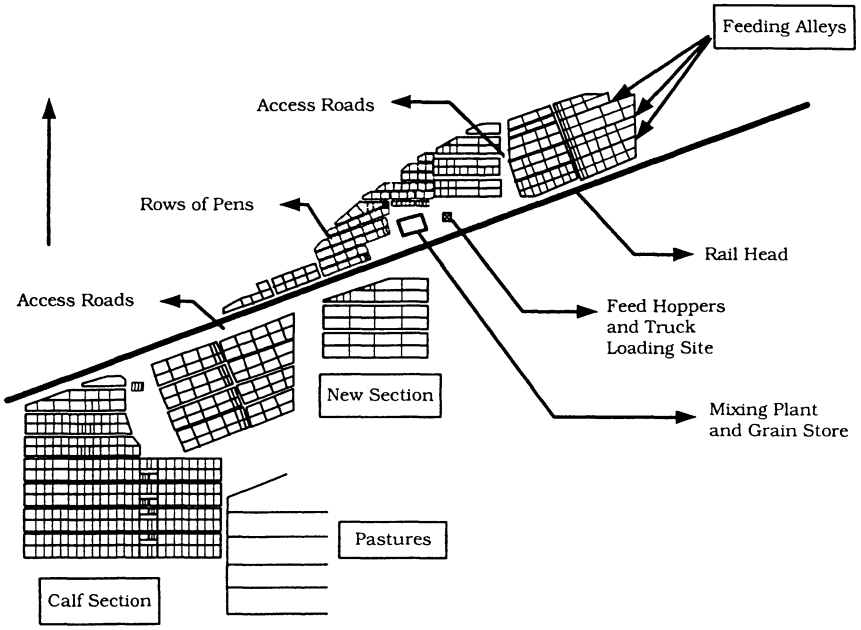


Figure 12.1 A map of the feed-yard.

are equipped to dispense the feed from the driver side only which causes the feed troughs to have a "direction" along each of the cattle pens.

The investigation into feed distribution planning is part of a larger study aimed at making all aspects of the operations at the cattle yard more efficient. The overall management problem is modeled as a hierarchical production mix and distribution problem. The demands placed by the cattle on the system form the input for the feed distribution problem. The solution to the feed distribution problem in turn forms the input for a scheduling problem, which assigns the feed delivery trips generated to the available trucks. The output from this model then drives the production scheduling problem for the mixing plant. From the outset, it was clear that the feed distribution system was the bottleneck in the entire process. The capacity of the feed mixing plant outstripped that of the feed distribution system. The feed mill operators were forced to make frequent stops to allow the distribution people to catch up. Additionally, land and pens are freely available and hence any improvement to the distribution system translates to an equivalent increase in the overall capacity to feed cattle, thereby increasing revenue for the cattle yard in question.

Other important activities of the cattle yard, such as vaccination and hormonal treatments for the cattle, are accomplished by moving an entire cattle lot (the content of a cattle pen) to an appropriate service area in the cattle yard and performing the desired task on each one of the cattle. In addition, calves may outgrow the capacity of the pen where they are housed as they mature, and the cattle lot needs to be split and moved to different pens. Often, the cattle are moved 3 to 4 times during their sojourn in the yard. The planning of cattle movements in co-ordination with the assignment of the appropriate pens is an important and costly activity. The modeling aspects of this activity are described in detail in Dror and Leung (1998).

The models and methods described in this chapter can be a decision-support tool to help the cattlemen effectively manage the feed distribution at the cattle yard. In Section 2, we present arc-routing models of the feed distribution problem. In Section 3, we present heuristic procedures for constructing feasible trips for delivery of cattle feed. We describe extensions made to existing heuristics available in the literature and route improvement procedures adopted, and discuss some issues pertinent to obtaining good solutions. In Section 4, we describe a different heuristic based on a Generalized Traveling Salesman Problem (GTSP) model of the problem. In the last two sections, we describe the computational results from our experiments with these heuristics and summarize the impact of our study in the operation of the cattle yard.

2. LIVESTOCK FEED DISTRIBUTION AS ARC TRAVERSALS

Every morning the operators of the yard must tour all pens with cattle to determine which pens have feeding troughs that are "slick" ("completely empty" in cattlemen parlance). Emergency rations are then delivered to these pens; otherwise, the hungry cattle might 'escape' from these pens and extra manpower is needed to round them up. Also, when a trough is slick, the cattle in the pen tend to gorge themselves when food is available later, which may lead to increased death rates. The main daily activity of the yard is the delivery of a prescribed amount (less any emergency ration already delivered) of a particular feed type to each pen. For any given feed type used on the ranch, there are a number of pens (out of the total of about 600) which require the delivery of a pre-determined feed quantity, each within a specified time window during the day.

2.1. ARC-ROUTING MODELS FOR PEN INSPECTION AND FEED DELIVERY

We represent the cattle yard as a connected graph $G = (N, E \cup A)$, whose edges $e \in E$ correspond to the connecting road segments and the arcs $a \in A$ correspond to the directed road segments along the pens. As noted earlier, these segments are directed because of the delivery mechanism of the trucks and the location of the feeding troughs. The graph for the Arizonan ranch contains 581 nodes and more than 1000 edges and arcs. The problems of slick inspection and feed delivery are cast in a general framework of arc routing.

For slick inspection, since no feed is delivered, it does not matter which direction an arc is traversed. For this problem, we consider the graph $G = (N, E \cup \bar{A})$, where $\bar{A}$ corresponds to the set of arcs A with the arc direction ignored. The slick inspection problem necessitates the finding of a minimum-distance closed traversal of a given subset of edges $S \subseteq \bar{A}$ in the connected graph $G = (N, E \cup \bar{A})$, with an additive non-negative real distance function defined on $E \cup \bar{A}$. In other words, it is an undirected Chinese Postman Problem or a Rural Postman Problem (RPP). The set of required edges S is the collection of edges denoting the sides of the pens with feeding troughs of those pens that contain cattle.

The modeling approach for feed delivery (both for emergency rations and for regular feed delivery) represents a generalization of the Rural Postman Problem. We now consider the graph $G = (N, E \cup A)$. For each feed type, there is a set of required arcs R_f corresponding to the set of pens which require that particular feed type, with a demand quantity associated with each arc in R_f. Since each truck can carry only one feed type, the distribution for each feed type forms a separate problem and is modeled as such. The cardinality of the arcs in R_f varies from 16 for the smallest problem to 348 for the largest.

2.1.1 Mathematical Formulation of the Capacitated Rural Postman Problem (CRPP).

We are given a positive demand q_e for each arc e in the set of required arcs $R \subseteq A$. A circuit in G which includes the depot (node 1) and such that the cumulative demand of the required arcs in this circuit does not exceed some pre-specified 'vehicle' capacity, denoted by W_v, is called a *feasible trip* with respect to vehicle v. The Capacitated Rural Postman Problem (CRPP) is that of determining a set of feasible trips (for a given set of vehicles) of total minimum distance which traverse all the

arcs in R. We formulate the CRPP using the following parameters and variables:

q_{ij} = the demand along arc $(i,j) \in R \subseteq A$,
W_v = the capacity of vehicle v,
c_{ij} = the distance *(length)* of an arc $(i,j) \in A, (c_{ij} \geq 0, \forall (i,j) \in A)$,
V = the upper bound on the number of vehicles,

x_{ijv} = the number of times vehicle v traverses the arc $(i,j) \in A$,
y_{ijv} = a binary variable which takes the value 1 if vehicle v discharges the feed along the arc $(i,j) \in R$, and takes the value 0 otherwise.

$$(CRPP): \qquad \min \sum_{(i,j)\in A} \sum_{v=1}^{V} c_{ij} x_{ijv}$$

subject to

$$\sum_{k\in N} x_{kiv} - \sum_{k\in N} x_{ikv} = 0, i \in N, v = 1, 2, \ldots, V, \tag{12.1}$$

$$\sum_{v=1}^{V} y_{ijv} = 1, \forall (i,j) \in R, \tag{12.2}$$

$$\sum_{(i,j)\in R} q_{ij} y_{ijv} \leq W_v, v = 1, \cdots V, \tag{12.3}$$

$$x_{ijv} \geq y_{ijv}, \forall (i,j) \in R, \tag{12.4}$$

$$M \sum_{i\notin S, j\in S} x_{ijv} \geq \sum_{(j,k)\in A[S]\cap R} x_{jkv}, \begin{cases} \forall S \subseteq N, 1 \notin S, \\ A[S] \cap R \neq \emptyset, \\ v = 1, \ldots, V, \end{cases} \tag{12.5}$$

$$y_{ijv} \in \{0,1\}, \forall (i,j) \in R, v = 1, \ldots, V, \tag{12.6}$$

$$x_{ijv} \in Z^{+}, \forall (i,j) \in A, v = 1, \ldots, V, \tag{12.7}$$

where M is a large constant no smaller than the total distance of any circuit that includes all arcs in R, and $A[S]$ is the set of arcs incident from at least one node in S. (In this formulation, we assume each vehicle makes only one trip. In reality, the index v will denote a trip and trips not overlapping in time could be assigned to the same vehicle.)

The objective function represents the total distance traveled by all the vehicles. Note that arcs can be traversed more than once. The first set of constraints is the common 'flow conservation' constraints for network-flow formulations. The second set of constraints require that at least one traversal is made of each of the arcs in R. The third set of constraints are

the capacity constraints for the vehicles. The next set of constraints require that vehicle v traverse the arc $(i,j) \in R$ if it delivers the demand to this arc. The fifth set of constraints are subtour-elimination constraints which ensure that each trip include the depot. Note that this formulation of CRPP not only has different subtour-elimination constraints than the one given in Golden and Wong (1981), but also the x_{ijv} variables have a different interpretation. In addition, this formulation does not allow 'split delivery' since the demand for any required arc is delivered by exactly one vehicle.

Many practical problems can be modeled as variations of the RPP. The reader interested in applications of arc routing problems is referred to Bodin and Kursh (1978), Manber and Israni (1984), Stern and Dror (1979), Eglese (1994), Eglese and Murdock (1991), and Christofides et al. (1986). Recent surveys are provided by Assad and Golden (1995), and Eiselt et al. (1995a, 1995b). The RPP is a difficult combinatorial optimization problem; it is $\mathcal{NP}$-hard in the strong sense even for completely directed and completely undirected graphs (Garey and Johnson, 1979). Successful exact solution procedures can at present address only problems of much smaller size than the problem that motivated this chapter.

2.2. SPLIT DELIVERIES

In practice, truck loading at the feed storage depot is not very precise. When loading, the driver brings the truck underneath the feed storage bin and opens a chute which releases the feed into the truck. Even though the amount of feed loaded onto the truck is weighed, the scales are not precise enough for exact control, and a truck might have less or more than the required demand of the cattle pens on his feed distribution trip. In principle (and in practice), a truck might dispense a partial amount of feed into a given trough provided it (or another truck) will deliver the remaining amount separately within the required feeding time-window.

When the demand of an arc can be served by more than one vehicle, the feed distribution problem becomes a split-delivery capacitated arc-routing problem. To allow for split delivery, we can replace constraint (6) in the formulation (CRPP) by

$$y_{ijv} \geq 0, \qquad \forall (i,j) \in R, \forall v.$$

The variable y_{ijv} now represents the fraction of the demand of arc (i,j) that is delivered by vehicle v. With split delivery, a feasible trip is a circuit of G that includes the depot node and such that the sum of $q_{ij}y_{ijv}$ over all required arcs (i,j) in the circuit does not exceed the vehicle capacity W_v. The CRPP with split deliveries is the problem of finding a

set of feasible trips that minimizes the total distanced traveled such that the total demand of the required arcs is satisfied.

In most routing problems studied in the literature, the assumption is made that the demand of an arc (or node) is not split between vehicles. The CRPP with split deliveries has not been explicitly addressed in the literature. We are not familiar with any past attempts to solve this problem or any mathematical formulations for this problem even though the formulation follows in quite a "natural" manner (for the routing expert) from the vast literature of general routing problems. A naive view might be that the split-delivery case is "easier" than the non-split problem version. However, Dror et al. (1994) demonstrated that it is not "easier" to solve the Vehicle Routing Problem (VRP), where the nodes carry demand requirements, with split deliveries allowed. Whilst some properties of the optimal VRP solution with split deliveries (see Dror and Trudeau, 1989, 1990, and Dror et al. 1994) can be extended to the split delivery problem for arc routing, allowing for split deliveries represents a generalization of the arc-routing problem and is significantly harder if exact solutions are attempted.

2.3. TIME WINDOWS

Each pen, besides requiring a specified amount of feed of a particular type, requires that the feed be delivered within a specified time-window. The feeding time-window for each pen corresponds to a "mealtime" for the cattle in this pen, which become used to receiving its ration at approximately the same time(s) each day. When driving through the cattle yard, one can easily spot the pens that expect to be fed since the cattle in those pens are alert and standing up watching for the delivery truck.

One aim of our study was to increase the capacity of the feed distribution system by reducing delivery times. The time taken to deliver feed is determined by the distance traveled, given that the feed dispensing rates cannot be further improved. Hence, the objective is to design a set of feed delivery trips that minimizes the total distance traveled by the fleet of trucks, while meeting the time-window constraints for feed delivery. In our observations over a number of visits to the cattle yard and the examination of delivery records, we note that the manual dispatching (feed distribution) solutions constructed daily by the current operating crew did not conform very closely with the delivery time-windows specified for the different pens. In a large cattle yard as the one in this study, there is always movement of cattle in and out of the yard and inside the yard between the different pens. When a certain cattle lot has been moved to a different pen, the feeding time window for this cattle lot must still be observed at its new pen. Thus, the feeding requirements with

respect to the *pen locations* are changing all the time. This makes the feed distribution problem quite complex to grasp and operate manually, and the intuitive solutions of the drivers and dispatchers are often not even "feasible", let alone optimal. Therefore, it made it even more apparent that a computer-generated solution should be able to deliver a more time-responsive and efficient routing and scheduling plan for feed distribution for the cattle yard.

3. A HEURISTIC APPROACH FOR TRIP GENERATION

This section describes our heuristic approach to solve the CRPP with split deliveries and time-windows, which includes several procedures which are extensions of known arc-routing algorithms, modified to handle the complications of time-windows and split deliveries. Our solution strategy is an adaptation of the heuristics proposed for the split delivery for node routing problems, explored in Dror and Trudeau (1989, 1990). We first generate feasible solutions for the corresponding routing problem where split deliveries are not allowed, and then improve the initial non-split delivery routes by generating, in a heuristic fashion, split deliveries. The initial non-split routes are evaluated for split delivery and route consolidation by various improvement procedures. All of our heuristics have been adapted to consider time-windows. Our overall solution approach include four modules:

1 generating a non-split feasible solution (using *Path Scanning* or *Augment-Merge* heuristics),

2 improving the solution by *arc swapping*,

3 generating split-delivery routes by *k-split generation* and

4 modifying the solution by *route addition.*

A road-map of the set of heuristic algorithms applied is given in Figure 12.2.

3.1. GENERATING A NON-SPLIT FEASIBLE SOLUTION

Two heuristics are used to generate a set of feasible (non-split) routes – the extended path-scan heuristic and the modified augment-merge heuristic, which adapts the path-scanning algorithms (introduced by Golden, DeArmon, and Baker, 1983) and the *Augment-Merge* approach (introduced by Golden and Wong (1981) for the capacitated arc-routing problem) to consider time-window constraints.

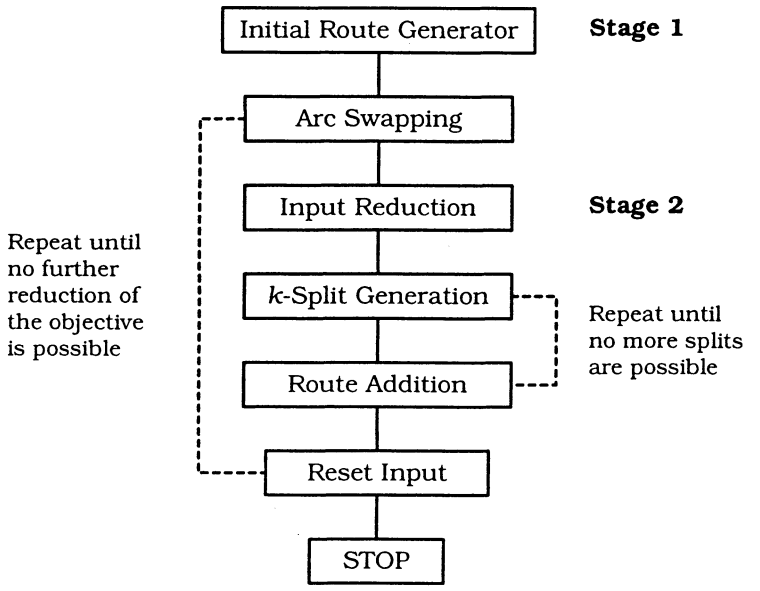


Figure 12.2 A Road-map of the Heuristic Procedures used.

3.1.1 Extended Path Scanning Algorithm for Feeding-Time Feasibility.

In our approach, we construct feasible routes one at a time until the demands of all arcs in the set of required arcs are met. Initially, each route starts at the depot and is extended one arc at a time based on one of the criteria listed below. Only arcs that do not violate the time-window constraints are added. The rules for arc-selection that were tested in the extended path scanning algorithm are:

1 minimizing the distance of insertion and return per unit demand for arc (i', j'),

2 minimizing the distance of insertion per unit demand for arc (i', j'),

3 minimizing the distance of insertion and return,

4 minimizing the distance of return,

5 maximizing the distance from node j' back to the depot if the remaining vehicle capacity is less than half; otherwise minimizing the distance of insertion,

6 minimizing the distance of insertion,

where the distance of insertion is the shortest-path distance from the end of the partial-constructed path to node i' and the return distance is the shortest-path distance from node j' to the depot.

For rules 1 and 3, the initial trips are constructed from arcs close to the depot, whereas subsequent trips include arcs far away from the depot. For rules 2 and 6, the trips tend to fan out radially since the return distance is not considered. Rule 5 essentially selects arcs away from the depot when the truck is relatively empty and selects arcs closer to the depot when it is more than half-full.

3.1.2 Modified Augment-Merge Algorithm.

In the augment-merge approach of Golden and Wong (1981), each arc $(i,j) \in R$ lies initially on a separate route; routes are then merged to effect savings in cost, while remaining feasible with respect to capacity constraints. The merge phase is repeated until no more merging of routes is possible. In our extended augment-merge heuristic, we look for possible merging of routes that are capacity- and time-feasible and also result in net overall savings. This process is repeated until no additional route merging is possible. A further modification for our application is to group together arcs that correspond to pens that are in the same row and may be serviced by the same route into a single composite demand arc, which results in a substantial reduction in the size of the graph and solution time.

We introduce some definitions that will be used both here and in the heuristic described in the next section. Let $G^R = (N(R) \cup \{1\}, R) = \cup_{k=1}^{K} G_k^R$ be the subgraph of $G = (N, A)$ induced by the arc-set R and the depot node with $G_k^R = (N(R_k), R_k), k = 1, \ldots, K$ being the connected components of G^R.

Definition. An *entry* node of R_k is a node $l \in N(R_k)$ such that there is a $j \in N \setminus N(R_k)$ with $(j,l) \in A \setminus R_k$.
An *exit* node of R_k is a node $l \in N(R_k)$ such that there is a $j \in N \setminus N(R_k)$ with $(l,j) \in A \setminus R_k$.

This concept of entry and exit nodes is taken from Dror, Stern and Trudeau (1987) and Dror and Langevin (1997). Our variant of the Augment-Merge algorithm is described as follows.

Phase I: *Augment.*

1 Determine the subset of N that are *entry nodes* for R.

2 Start with $(i,j) \in R$ with node i being an entry node. If the demand for (i,j) is not met yet, start a new trip T_i with an appropriate start time so the demand for arc (i,j) is met at the beginning of its time-window using a shortest path from the depot to node i.

3 Let arc $(i',j') \in R$ be adjacent to (i,j) and determine whether its demand can be met by the same trip (checking vehicle capacity and time-window restrictions). If so, add the arc (i',j') to the trip.

4 Repeat step 3 with (i',j') as (i,j) until (i,j) is not adjacent to any arc in R or the vehicle capacity is reached.

5 Choose another entry node of R and repeat steps 1-3 until there are no more arcs in R with unsatisfied demand.

Note that this version of Augment-Merge is different than the one in Golden and Wong (1981) since its initial routes are generated between pairs of entry and exit nodes and not one route for each arc separately.

Phase-II: *Merge.*

1 Set trip index $v = 1$.

2 For $j \neq v$, evaluate if trip T_v can be merged with trip T_j, without violating vehicle capacity and timing restrictions. If so, we compute the savings in the distance of the trips when merged. Choose T_j with the highest savings and merge with T_v. Re-compute the start and finish times for the merged trip and for each arc on the trip.

3 Set $v = v + 1$, and repeat steps 2-3.

4 Repeat Steps 2 through 4 until no more merging of trips is possible.

3.2. ARC SWAPPING

This step is an improvement heuristic based on the concept of arc-swapping between different routes. This procedure generalizes the classical 2-Opt heuristic for the Traveling Salesman Problem by Lin and Kernighan (1973). The variant for the Vehicle Routing Problem (VRP) is described in Dror and Levy (1986). Our arc-swapping improvement procedure for the CRPP is adapted to include time windows and is run on all feasible solutions obtained, both with and without split deliveries. Arc-swapping heuristics are also discussed in Potvin and Rousseau (1995), and more sophisticated node exchange heuristics, which can be adapted to arc routing, are described in Gendreau, et al. (1992). A survey of heuristic algorithms for arc routing is given in this book by Hertz and Mittaz (2000).

3.3. GENERATING SPLIT DELIVERY ROUTES

In this module, we check to see if the delivery made to an arc can be split across k other candidate routes in such a way that the highest savings can be obtained. Let C_{uv} represent the length of an arc (u, v), and $\overline{C}_{uv}$ represent the shortest-path distance from node u to node v. Let arc (i, j) on route h be the arc that is being considered for splitting between $k \geq 2$ routes. Let a and b be the nodes immediately prior to and after arc (i, j) on route h. Denote by i_t and j_t the consecutive nodes on route t between which we would like to introduce arc (i, j). The savings S_{ij}^k obtained by introducing a k-split delivery at arc (i, j) is:

$$S_{ij}^k = C_{ai} + C_{jb} + C_{ij} - \overline{C}_{ab} + \sum_{t=1}^{k} (\overline{C}_{i_t j_t} - \overline{C}_{i_t i} - C_{ij} - \overline{C}_{j j_t}) \quad (12.8)$$

In any candidate k-split route, the insertion position of arc (i, j) into each candidate route is the one that nets the highest overall savings as shown in the summand in (12.8) above. The k-split generation is performed only if S_{ij}^k is positive.

The candidate routes are chosen from the set of routes whose possible time-span cover the time window on arc (i, j). This list is further pruned to those that can actually deliver to arc (i, j) within its specified time-window. For the Arizonan yard, it was unrealistic to have more than ten trips with trip times that spanned any particular delivery arc, hence k was limited to 10 in our computations. If more than one division of the demand is possible, we choose to assign the demand to routes according to non-decreasing spare capacity of the routes. If the demand delivered to arc (i, j) is already split between a number of routes, we do not just take the partial delivery and split it further. Instead, we remove the split arc (i, j) from all routes and then re-evaluate splitting the delivery.

3.4. ROUTE ADDITION

In route addition, we investigate arcs whose demand is split among several routes, and consider if consolidating them into one new route will effect net savings in distance traversed. Thus, we may be able to realize savings if an arc which is being serviced by two or more routes is excluded from these routes and the demand for the arc is delivered on a separate trip. We note that the "consolidated" arc need not be the only arc in the newly added route; we may add segments from the original routes as well. Various possible permutations of such routes were discussed in Dror and Trudeau (1990) for the node-routing case and in Mullaseril (1996) for the arc-routing case. Calculations for savings is a straightforward extension of the concept first proposed by Clark and

Wright (1964). We do not consider more than 3-route addition in our study.

4. ROUTE-FIRST CLUSTER-SECOND GENERALIZED TSP HEURISTIC

In this section we describe a heuristic approach to CRPP based on the well-established principle in capacitated arc-routing (see, for example, Stern and Dror, 1979) of "route-first and cluster-second", and which uses an exact solution procedure for the Generalized Traveling Salesman Problem (GTSP). (See Noon and Bean, 1991, 1993).

4.1. THE GENERALIZED TRAVELING SALESMAN PROBLEM

In a Generalized Traveling Salesman Problem, the nodes of the given graph $H = (S, L)$ are partitioned into node-sets $S_k, k = 1, \ldots, K$. There are no arcs in L connecting nodes within the same set S_k in the partition. The objective is to find a minimum-cost cycle which includes exactly one node from each node-set in the partition. Clearly, when each node-set in the partition contains exactly one node, the GTSP reduces to the classical Traveling Salesman Problem. A mathematical formulation of the GTSP is given in Noon and Bean (1991).

4.2. TRANSFORMING THE CRPP TO AN EQUIVALENT GTSP

In order to apply the GTSP for routing in the Capacitated Rural Postman Problem, we first calculate the following quantities: For each connected component of required arcs $G_k^R = (N(R_k), R_k)$ of the graph G^R, induced by R and the depot node, and for each pair (s, t) of its entry and exit nodes, let $l_k(s, t)$ denote the distance of the minimum-length traversal of all the arcs in R_k starting at entry node s and ending at exit node t. This traversal is done with respect to $G = (N, A)$; that is, arcs in $A \setminus R_k$ can be used. Clearly, the computation of an optimal traversal of R_k, when required to start at a given entry node and end at a pre-specified exit node of that subgraph, is of $O(|N(R_k)|^3)$ time-complexity. We also compute the shortest-path distance (again relative to $G = (N, A)$) from each exit node in R_k to each entry node in all other components $R_l, l \neq k$.

The graph $H = (S, L)$ for the GTSP is constructed as follows. For $k = 1, \ldots, K$, the set S_k consists of nodes in one-to-one correspondence with the set of *pairs* of entry and exit nodes of R_k. In other words, each node in S_k correspond to an entry-exit node *pair* in R_k. Let node $i \in S_k$ correspond to $(s, t) \in R_k$ and node $i' \in S_k'$ correspond to $(s', t') \in R_k'$,

then the length of arc $(i, i') \in L$ is the sum of $l_k(s,t)$ and the shortest-path distance from t to s' (relative to $G = (N, A)$). Note that by construction, since the depot node is not in $N(R)$, the single depot node is a connected component of G^R.

4.3. SOLVING THE EQUIVALENT GTSP

Every tour of the GTSP corresponds to a trip covering all the arcs of R, where the subsets R_k are traversed in the order indicated by the nodes of the GTSP tour. The GTSP approach may be used to solve for the slick inspection problem which is modeled as an RPP; however, given the size of the problem on hand, this is not implemented as yet.

In case of the feed distribution problem (CRPP) and given the optimal GTSP route, the clustering part of the heuristic is straightforward. Starting from the depot, follow the optimal GTSP route until the capacity of the vehicle is exceeded. Terminate the trip at the last node before exceeding capacity and return to the depot. Start a new trip with the node following the termination point of the previous trip as the first node and continue until vehicle capacity is exceeded again. Repeat this routine until all arcs demands are met.

An optimal solution of the corresponding GTSP represents the "route first" part, and the trip generation the "cluster-second" part, of this heuristic approach to the CRPP. Even though the GTSP is $\mathcal{NP}$-hard in the strong sense, if the number of disconnected graph components G_k^R for this problem is relatively small (not exceeding 30), the solution methodology described in Noon and Bean (1991) solves such problem optimally in reasonable time. However, note that the time-window constraints (but not the capacity constraints) are ignored in this heuristic, so the routes generated may not be feasible with respect to feeding time-windows. On the other hand, since the 'cluster' step of the GTSP heuristic is not guaranteed to be optimal, the GTSP heuristic solution is not necessary optimal for (CRPP), and hence is not necessarily a lower bound to the capacitated arc-routing problem with time windows modeled in this chapter. (See the computational results for problems 400 and 500 in the next section.)

5. COMPUTATIONAL RESULTS

We tested our heuristics using data from the Arizonan cattle yard. Solutions were obtained for seven problems, which are representative of the daily feed requirements for various feed mixes. The problems are for five feed-types, indexed by 100, 400, 500, 700 and 800. The data set for

feed-type 100 is also disaggregated to form two more data sets, indexed by 100A and 100B. These problems ranged in size from 227 nodes and 294 arcs to 581 nodes and 770 arcs, and the number of demand arcs ranged from 16 to 348.

Table 12.1: Problem sizes and results of the GTSP heuristic.

Problem (Feed Type)	100	400	700	800	500	100-A	100-B
Nodes in G	581	581	581	581	581	354	227
Arcs in G	770	770	770	770	770	473	294
Arcs in R	348	16	44	45	55	198	150
GTSP							
Components in H		9	10	20	10	40	32
Nodes in H		36	40	80	40	160	128
Arcs in H		1152	1440	6080	1440	24960	15872
RPP Solution	-	27900	28900	21200	52200	64400	34000
CRPP Solution	-	66635	55538	111836	73448	340064	107093

The computational experiments were performed on an Intel Pentium based machine running at 200Mhz. Table 12.1 reports the size of these problems and the size of the auxiliary graph $H = (S, L)$ generated by the GTSP heuristic. In this table we report the total distance traveled using the GTSP approach, and both the RPP solution (capacity constraints ignored) and the CRPP solution (capacity constraints considered) using the heuristic to break the RPP tour into trips. All heuristics used under 25 seconds CPU time even on a desktop machine. The GTSP code used (Noon and Bean, 1991) did not solve problems with the number of components greater than 70. Thus, for feed-type 100 we did not obtain GTSP-based solutions. The GTSP heuristic generally provides the lowest-cost solution compared to the other heuristics tested, since time-window constraints are ignored.

Table 12.2 shows the comparison between the current routing plan and the solution using the 'best' non-split heuristic as indicated in Tables 12.4 to 12.10. Split delivery solutions obtained by implementing the procedures described in Sections 3.3 and 3.4 are also presented in Table 12.2. A more detailed description of the split delivery arc routing with time windows heuristics are presented in Mullaseril and Dror (1997). The solutions generated by the arc routing procedures presented in this chapter are significantly (up to 42%) better than the current cattle yard distribution solutions. The split delivery option reduces the distribution distance further in three out of five cases. This distance reduction is best illustrated for feed-type 100 where it saves about 5% of the distance traveled over the non-split solution.

Table 12.2: Comparison between heuristic and current routing plans.

Feed Type	Heuristic Used	Heuristic Distance	# of Trips	Split Heuristic Distance	# of Trips	Current Distance	# of Trips	Improvement in Distance
100	*Path Scan-6*	439592	38	418133	39	534809	45	22%
400	*Path Scan-6*	58119	6	58119	6	82932	5	30%
500	*Path Scan-6*	46044	3	46044	3	78836	3	42%
700	*Path Scan-2*	98030	5	95476	5	105560	6	10%
800	*Path Scan-3*	102177	5	102177	5	117298	7	13%

Our computational results are summarized in Tables 12.4 to 12.10. Each data set is solved (heuristically) for both the split and non-split version of the problem, each with and without the arc-swapping module. In generating the initial non-split solution, both the path-scanning heuristic (for each of the six arc selection criteria) and the augment-merge heuristic are used. Our computational results show a large variance among the path-scanning heuristics. The solution values indicate that the sixth path-scanning heuristic is the most effective; it performed the best in 5 out of 7 problems with respect to distance and 4 out of 7 with respect to the total number of trips.

It is interesting to note that the arc-swapping heuristic seems to make very little or no improvement on the Augment-merge solutions. In our opinion, the "best" solution is influenced by the input. In this facility, cattle of similar age, breed and disposition were grouped together and also belonged to the same or closely related time-windows. Hence "greedy" heuristics such as the Path Scan 6 (closest arc), and the augment-merge heuristic tended to perform much better than the other heuristics. The performance of the three improvement procedures (arc-swapping, k-split, route addition) varies considerable across different input (output from the CRPP module) as Tables 12.4 to 12.10 show. At times, the k-split module performs better when the arc-swapping module is not utilized as in the cases of feed-types 100 and 500 using Path Scan-4. However, in general, the use of all three improvement procedures resulted in better solutions.

The cattle operated on a single feed basis prior to this study. To investigate the viability of changing the operation to two feeds per day, we tested our heuristics for the two cases. Case 1 represents the single-feed scenario, while in Case 2, the daily requirements are delivered in two feeding time-windows. Computational runs were done on the same graph (same underlying pen layout) for the two cases. To accommodate for slick feeding, the demand in Case 1 is 75 percent of the total demand in Case

2, assuming that 25 percent of the daily demand is delivered as slick feed. The operators of the feed-yard had three criteria by which to judge the heuristics. The most obvious one is distance traveled and minimizing it will serve to increase the capacity of the feed-yard in terms of number of cattle fed. However, the feed-yard sometimes pays drivers of trucks based on the number of trips delivered and hence would like to minimize that cost. Similarly the wear and tear on the truck and drivers is based on the number of hours of operation and they would like to keep this below certain prescribed limits. Hence we collected information on the distance traveled, total hours of operation and total number of trips for each of the heuristics. Tables 12.4 to 12.10 report in detail the computational results for all seven problems (all heuristic algorithms) for the two cases described above. Figures 12.3 to 12.6 report on the comparison (in terms of rank) of all heuristics with respect to total distance traveled and the total number of trips for the two cases. One can observe that the benefits from two feeds per day outweighs the additional expense.

6. EPILOGUE

Our preliminary computational results point out the inefficiencies in the current operations. Our analysis of the data shows that pens requiring the same feed type with adjacent time windows are dispersed randomly throughout the yard. Hence vehicles have to deadhead between adjacent deliveries within a trip resulting in considerable non-productive time, and which leads to the use of more vehicles than absolutely necessary. Subsequently, the management of the cattle yard revised their distribution policies. The yard adopted what is known in industry parlance as 'zone feeding'. The principle idea in zone feeding is to insure that each row of pens has the same feed delivered to it and with identical time windows. Thus a trip can service a whole row of pens without having to deadhead between two pens. The rows of pens were then assigned time windows in a staggered fashion so that trucks could progressively proceed from one row of pens to the next without skipping a row while making deliveries. This leads to clustered demand patterns that are easier to manage.

In this regard, the GTSP heuristic can be used, in some sense, to provide an approximate target for improvements to the system, since it generates a solution which ignores time-windows and thus, in most cases, should be superior to a solution that is time-constrained. The routes generated by the GTSP heuristic can also help to set the target time-windows when adjusting the demand patterns for zone feeding, by indicating which pens will be delivered on the same route if they have the same or adjacent time windows.

Prior to implementing the zone feeding policy, the information systems in the yard were also vastly improved (see Tracey and Dror, 1996, for the description of a computerized graphical implementation proposed for the new information system). The entire fleet is managed by a dispatch control center where the feeding plans are determined ahead of time and downloaded on to laptop computers provided in each of the trucks. One of the drawbacks of the system in-use is that target times for feeding each of the pens are not available. This information can be provided by the solution to the CRPP with time-windows, which determines the start and finish times for delivery at each pen, enabling more precise monitoring and control of the delivery system.

The zone feeding strategy leads to near-optimal routes provided the demand patterns can be preserved in this state over time. However, as noted, it is often necessary to move cattle to other pens as they mature. When cattle are moved to a new location, they may not be able to immediately adapt to the new feeding time imposed on them by the time windows associated with these pens; it takes some time before they are trained to feed at a new mealtime. (Such time changes cause distress among the cattle, which is not desirable.) Thus, cattle movements and adjustments lead to a gradual breakdown of the clustered demand pattern that zone feeding seeks to maintain. A more zonal demand pattern can be restored by occasional re-arrangement of cattle and a *good* assignment of pens to incoming cattle. Finding the optimal initial pen assignment and the optimization of cattle movement within the yard is an interesting direction for future research.

After examining our computational results, the yard management implemented a new improved feed distribution policy. In the improved plan implemented, two trucks each start at the north-east and south-west corners and each truck starts delivering feed to alternate rows of pens, leapfrogging the other as they proceed through the yard. In certain seasons of high utilization of the yard, more trucks are pressed into the service. This strategy considerably improved the speed of delivery. Hence the yard moved from a single feed a day to two feeds a day, early morning and late afternoon. Two feeds a day is better than a single feed as the cattle have lower tendency to overeat and their diets could be controlled with greater precision leading to other benefits such as lower death rates.

References

[1] Assad, A.A. and B.L. Golden (1995). "Arc Routing Methods and Applications", in M.O. Ball et al., Eds., *Handbooks in OR & MS*, Vol. 8, 375-483, Elsevier Science B.V.

[2] Bodin, L.D. and S.J. Kursh (1978). "A computer-assisted system for the routing and scheduling of street sweepers", *Operations Research* 26, 525-537.

[3] Christofides, N., V. Campos, A. Corberan and E. Mota (1986). "An algorithm for the Rural Postman Problem on a directed graph", *Mathematical Programming Study* 26, 155-166.

[4] Clarke, G. and J.W. Wright (1964) "Scheduling of vehicles from a central depot to a number of delivery points", *Operations Research* 12, 568-581.

[5] Dror, M. and A. Langevin (1997). "A Generalised Traveling Salesman Problem (GTSP) approach to the Directed Clustered Rural Postman Problem", *Transportation Science* 31, 187-192.

[6] Dror, M., G. Laporte and P. Trudeau (1994). "Exact Solutions for Split Delivery Routing", *Discrete Applied Mathematics* 50, 3:239-254.

[7] Dror, M. and J.M.Y. Leung (1998) "Combinatorial optimization in a cattle yard: Feed distribution, vehicle scheduling, lot sizing and dynamic pen assignment" in *Industrial Applications of Combinatorial Optimization*, G. Yu (ed.), Kluwer Academic Publishers, Netherlands, p.142-171.

[8] Dror, M. and L. Levy (1986). "A vehicle routing improvement algorithm - comparison of a 'greedy' and a 'matching' implementation for inventory routing", *Computers and Operations Research*, 13, 33-45.

[9] Dror, M. and P. Trudeau (1989). "Savings by split delivery routing", *Transportation Science* 23, 141-145.

[10] Dror, M. and P. Trudeau (1990). "Split Delivery Routing," *Naval Research Logistics* 37, 383-402.

[11] Eglese, R.W. (1994). "Routing winter gritting vehicles", *Discrete Applied Mathematics* 48, 231-244.

[12] Eglese, R.W. and H. Murdock (1991). "Routing read sweepers in rural area", *J. Oper. Res. Soc.* 42(4), 281-288.

[13] Eiselt, H.A., M. Gendreau and G. Laporte (1995a). "Arc routing problems. Part I: The Chinese Postman Problem", *Operations Research* 43, 231-242.

[14] Eiselt, H.A., M. Gendreau and G. Laporte (1995b). "Arc routing problems. Part II: The Rural Postman Problem", *Operations Research* 43, 399-414.

[15] Garey, M.R., and D.S. Johnson (1979). *Computers and Intractability: A Guide to the Theory of NP-completeness*, Freeman, San Francisco.

[16] Gendreau, M., A. Hertz and G. Laporte (1992). "New insertion and post-optimization procedures for the Traveling Salesman Problem", *Operations Research* 40, 1086-1094.

[17] Golden, B.L. and R.T. Wong (1981). "Capacitated arc routing problems", *Networks*, 11, 305-315.

[18] Golden, B., J. DeArmon and E. Baker (1983). "Computational experiments with algorithms for a class of routing problems", *Comp. Oper. Res.* 10, 47-69.

[19] Hertz, A. and M. Mittaz (2000), "Heuristic Algorithms", in Dror, M. ed. *Arc Routing: Theory, Solutions and Applications* (this book).

[20] Lin, S., B.W. Kernighan (1973). "An effective heuristic algorithm for the traveling salesman problem", *Operations Research*, 21, 498-516.

[21] Manber, U. and S. Israni (1984). "Pierce point minimization and optimal torch path determination in flame cutting", *J. Manufact. Syst.* 3, 81-89.

[22] Mullaseril, P.A., (1996). "Capacitated Rural Postman Problem with Time Windows and Split Delivery", Ph.D. Thesis, MIS Department, University of Arizona, Tucson, Arizona, 85721.

[23] Mullaseril, P.A., M. Dror and J.M.Y. Leung (1997). "Split delivery routing heuristics in livestock feed distribution", *Journal of Operational Research Society* 48, 107-116.

[24] Mullaseril, P.A., and M. Dror (1997). "A set covering approach for node and arc routing problems with split delivery and time windows", Working Paper, MIS Department, University of Arizona (submitted for publication).

[25] Noon, C.E. and J.C. Bean (1991). "A Lagrangean based approach to the asymmetric Generalised Traveling Salesman Problem", *Operations Research* 39, 623-632.

[26] Noon, C.E. and J.C. Bean (1993). "An efficient transformation of the Generalised Traveling Salesman Problem", *INFOR* 31, 39-44.

[27] Potvin, J.-Y. and J.-M. Rousseau (1995) "An exchange heuristic for routeing problems with time-windows", *Journal of the Operational Research Society* 46, 1433-1446.

[28] Stern, H. and M. Dror (1979). "Routing electric meter readers", *Comp. Oper. Res.* 6, 209-223.

[29] Tracey, M. and M. Dror (1996). "Interactive graphical computer application for large-scale cattle feed distribution management", *Decision Support Systems* 19, 61-72.

Appendix

Table 12.3: The cattle and calves business of the 50 U.S. states in 1995.

State	Cash Receipts From Cattle (in $1000)	Number of Businesses (in 1000)	State	Cash Receipts From Cattle (in $1000)	Number of Businesses (in 1000)
1. Texas	6,295,600	149.00	2. Kansas	4,523,400	38.00
3. Nebraska	4,157,800	28.00	4. Colorado	2,081,200	13.00
5. Oklahoma	1,758,600	62.00	6. Iowa	1,705,000	45.00
7. California	1,289,800	25.00	8. South Dakota	1,045,900	21.00
9. Minnesota	835,200	37.00	10. Montana	667,800	12.00
11. Missouri	659,800	75.00	12. Washington	645,000	20.00
13. Illinois	622,300	27.00	14. Idaho	618,100	12.00
15. Wisconsin	611,200	50.00	16. Kentucky	548,200	54.00
17. New Mexico	483,100	9.50	18. Wyoming	461,700	5.70
19. Arizona	434,000	4.30	20. North Dakota	366,300	14.00
21. Pennsylvania	354,700	33.00	22. Arkansas	310,900	32.00
23. Tennessee	310,500	63.00	24. Alabama	308,300	35.00
25. Oregon	294,100	22.00	26. Florida	289,800	21.00
27. Georgia	284,900	29.00	28. Indiana	268,600	28.00
29. Michigan	264,800	19.00	30. Utah	261,400	7.70
31. Ohio	256,700	35.00	32. Virginia	252,400	31.00
33. Mississippi	156,800	30.00	34. North Carolina	152,200	33.00
35. New York	149,200	19.00	36. Louisiana	115,500	18.00
37. Nevada	102,400	1.70	38. South Carolina	91,700	14.00
39. West Virginia	69,300	17.00	40. Maryland	60,000	6.50
41. Vermont	39,500	3.90	42. Maine	16,300	2.80
43. Hawaii	14,600	0.85	44. Connecticut	13,900	1.30
45. New Jersey	9,100	2.00	46. Massachusetts	8,200	1.80
47. New Hampshire	5,600	1.10	48. Delaware	2,600	0.58
49. Rhode Island	600	0.21	50. Alaska	600	0.12
Total United States:	34,275,200	1211.06			

Table 12.4: Performance of Heuristics for Feed Type 100

	CRPP Model						Split Delivery Model							
	With Swap			Without Swap			With Swap					Without Swap			
Feed 100 Case 1	CRPP Distance	Time	Trips	CRPP Distance	Time	Trips	Swap	k-Split Improve	Add Improve	Final Distance	Trips	k-Split Improve	Add Improve	Final Distance	Trips
Path Scan-1	913235	24.5	39	944749	29.2	39	31514	291874	52145	569216	69	271959	29195	643595	69
Path Scan-2	545784	20.2	38	557180	22.1	38	11396	33056	4137	508591	41	40921	7430	508829	41
Path Scan-3	598635	22.4	38	597180	22.4	38	0	49749	0	548886	41	49749	0	548886	41
Path Scan-4	1254323	27.3	39	1399947	36.4	39	145624	50004	35664	1168655	59	499413	35664	864870	59
Path Scan-5	896756	22.5	39	963682	29.3	39	66926	265338	32993	598425	57	244992	20663	698027	49
Path Scan-6	439592	17.5	38	442473	20.0	38	2881	20005	1454	418133	39	10952	0	431521	38
Augment-Merge	559449	28.4	43	559731	28.3	43	282	52017	10727	496705	49	50402	11343	497986	50
Feed 100 Case 2	CRPP Distance	Time	Trips	CRPP Distance	Time	Trips	Swap	k-Split Improve	Add Improve	Final Distance	Trips	k-Split Improve	Add Improve	Final Distance	Trips
Path Scan-1	898444	23.1	44	944756	29.4	44	46312	88964	27244	782236	61	268227	36367	640162	68
Path Scan-2	632224	23.5	45	632224	23.5	45	0	94705	30534	506985	53	88964	27244	516016	61
Path Scan-3	650996	23.0	43	651475	24.1	43	479	118932	19918	512146	55	1189832	20898	511645	55
Path Scan-4	1296578	26.5	42	1394609	36.6	42	98031	59817	36469	1200292	67	631184	36469	726956	67
Path Scan-5	891466	23.5	43	964786	29.5	43	73320	24384	6467	860615	56	233404	8200	723182	56
Path Scan-6	460437	20.3	42	460437	20.3	42	0	12461	0	447976	42	12461	0	447976	42
Augment-Merge	564889	27.5	50	565171	27.4	50	282	53069	6050	505770	54	49963	13084	502124	56

Case 1: Single feed per day with slick feed.

Case 2: Two feeds per day.

Table 12.5: Performance of Heuristics for Feed Type 400

	CRPP Model						Split Delivery Model								
	With Swap			Without Swap			With Swap					Without Swap			
Feed 400 Case 1	CRPP Distance	Time	Trips	CRPP Distance	Time	Trips	Swap	k-Split Improve	Add Improve	Final Distance	Trips	k-Split Improve	Add Improve	Final Distance	Trips
Path Scan-1	72642	2.17	5	72642	2.17	5	0	0	0	72642	5	0	0	72642	5
Path Scan-2	71241	2.15	6	71241	2.15	6	0	0	0	71241	6	0	0	71241	6
Path Scan-3	69935	2.20	6	69935	2.20	6	0	2486	0	67449	6	2486	0	67449	6
Path Scan-4	75812	2.44	5	75812	2.44	5	0	6932	0	68880	5	6932	0	68880	5
Path Scan-5	80350	2.04	6	89253	2.37	6	8903	1238	0	79112	6	1238	0	88015	6
Path Scan-6	58119	2.05	6	58119	2.05	6	0	0	0	58119	6	0	0	58119	6
Augment-Merge	74272	2.46	6	74272	2.46	6	0	0	0	74272	6	0	0	74272	6
Feed 400 Case 2	CRPP Distance	Time	Trips	CRPP Distance	Time	Trips	Swap	k-Split Improve	Add Improve	Final Distance	Trips	k-Split Improve	Add Improve	Final Distance	Trips
Path Scan-1	80829	2.32	6	80829	2.32	6	0	532	706	79591	7	532	706	79591	7
Path Scan-2	65784	2.10	6	65784	2.10	6	0	0	0	65784	6	0	0	65784	6
Path Scan-3	69935	2.20	6	69935	2.20	6	0	137	471	69327	7	137	471	69327	7
Path Scan-4	84823	2.58	6	84823	2.58	6	0	1238	0	83585	6	1238	0	83585	6
Path Scan-5	80342	2.02	6	89253	2.37	6	8911	0	0	80342	6	0	0	89253	6
Path Scan-6	58119	2.05	6	58119	2.05	6	0	0	0	58119	6	0	0	58119	6
Augment-Merge	74272	2.45	6	74272	2.46	6	0	0	0	74272	6	0	0	74272	6

Case 1: Single feed per day with slick feed.

Case 2: Two feeds per day.

Table 12.6: Performance of Heuristics for Feed Type 500

	CRPP Model						Split Delivery Model								
	With Swap			Without Swap			With Swap					Without Swap			
Feed 500 Case 1	CRPP Distance	Time	Trips	CRPP Distance	Time	Trips	Swap	k-Split Improve	Add Improve	Final Distance	Trips	k-Split Improve	Add Improve	Final Distance	Trips
Path Scan-1	159585	1.39	4	168303	4.32	4	8718	47733	0	111852	4	19352	0	148951	4
Path Scan-2	106051	3.11	4	106051	3.11	4	0	808	0	105243	4	808	0	105243	4
Path Scan-3	56866	2.12	3	56866	2.12	3	0	0	0	56866	3	0	0	56866	3
Path Scan-4	156644	3.23	6	197173	8.21	6	40529	45154	0	111490	6	132653	0	64520	4
Path Scan-5	142748	0.42	4	162752	4.10	4	20004	129	0	129805	4	5547	0	157205	4
Path Scan-6	46044	1.59	3	46044	1.59	3	0	0	0	46044	3	0	0	46044	3
Augment-Merge	209432	7.75	13	209432	7.57	13	0	132801	0	76631	13	132801	0	76631	13
Feed 500 Case 2	CRPP Distance	Time	Trips	CRPP Distance	Time	Trips	Swap	k-Split Improve	Add Improve	Final Distance	Trips	k-Split Improve	Add Improve	Final Distance	Trips
Path Scan-1	154316	0.46	4	175895	4.27	4	21579	10196	5049	139071	5	11205	0	164690	4
Path Scan-2	115352	1.48	4	122255	3.29	4	6903	12068	2382	100902	5	14394	2382	105479	5
Path Scan-3	60723	2.12	3	60723	2.12	3	0	0	0	60723	3	0	0	60723	3
Path Scan-4	156644	3.23	6	197173	8.21	6	40529	38090	0	118554	6	94248	2374	100551	7
Path Scan-5	156735	1.30	4	173778	4021	4	17043	8564	0	148171	4	19024	0	154754	4
Path Scan-6	50501	2.02	3	50501	2.02	3	0	0	0	50501	3	0	0	50501	3
Augment-Merge	209432	7.57	13	209432	7.57	13	0	96444	0	112988	13	96444	0	112988	13

Case 1: Single feed per day with slick feed.

Case 2: Two feeds per day.

Table 12.7: Performance of Heuristics for Feed Type 700

	CRPP Model						Split Delivery Model								
	With Swap			Without Swap			With Swap					Without Swap			
Feed 700 Case 1	CRPP Distance	Time	Trips	CRPP Distance	Time	Trips	Swap	k-Split Improve	Add Improve	Final Distance	Trips	k-Split Improve	Add Improve	Final Distance	Trips
Path Scan-1	120620	3.11	5	121998	3.28	5	1378	2554	0	118066	5	2554	0	119444	5
Path Scan-2	98030	3.04	5	98038	3.04	5	8	2554	0	95476	5	2554	0	95484	5
Path Scan-3	104682	3.07	5	104682	3.07	5	0	2554	0	102128	5	2554	0	102128	5
Path Scan-4	141445	4.49	5	159885	7.12	5	18440	21385	3209	116851	6	25787	1079	133019	6
Path Scan-5	124193	2.58	5	143722	3.56	5	19529	0	0	124193	5	7642	2579	133501	7
Path Scan-6	99161	2.57	5	104768	3.07	5	5607	0	0	99161	5	0	0	104768	5
Augment-Merge	159101	5.03	10	159101	5.03	10	0	36715	0	122386	10	36715	0	122386	10
Feed 700 Case 2	CRPP Distance	Time	Trips	CRPP Distance	Time	Trips	Swap	k-Split Improve	Add Improve	Final Distance	Trips	k-Split Improve	Add Improve	Final Distance	Trips
Path Scan-1	148076	3.40	7	152830	4.05	7	4754	9828	0	138248	7	13292	0	139538	7
Path Scan-2	134718	3.50	7	134718	3.50	7	0	8462	2370	123886	8	8462	2370	123886	8
Path Scan-3	137218	3.53	7	137218	3.53	7	0	17296	4151	115771	9	17296	4151	115771	9
Path Scan-4	183072	4.11	7	205808	7.36	7	22736	35625	2029	145418	10	42204	0	163604	7
Path Scan-5	150416	2.43	7	165671	4.28	7	15255	9739	2370	138307	8	8006	0	157665	7
Path Scan-6	137600	3.52	7	137600	3.52	7	0	13998	0	123602	7	13998	0	123602	7
Augment-Merge	159101	5.03	10	159101	5.03	10	0	29796	0	129305	10	29796	0	129305	10

Case 1: Single feed per day with slick feed.

Case 2: Two feeds per day.

Table 12.8: Performance of Heuristics for Feed Type 800

	CRPP Model						Split Delivery Model								
	With Swap			Without Swap			With Swap					Without Swap			
Feed 800 Case 1	CRPP Distance	Time	Trips	CRPP Distance	Time	Trips	Swap	k-Split Improve	Add Improve	Final Distance	Trips	k-Split Improve	Add Improve	Final Distance	Trips
Path Scan-1	168064	4021	5	168064	4.21	5	0	19553	854	147657	6	19553	854	147657	6
Path Scan-2	148705	3.55	5	148705	3.55	5	0	0	0	148705	5	0	0	148705	5
Path Scan-3	102177	2.59	4	102177	2.59	4	0	0	0	102177	4	0	0	102177	4
Path Scan-4	127690	4.02	4	137513	5.07	4	9823	8249	0	119441	4	9922	0	127591	4
Path Scan-5	178801	4.34	6	178801	4.34	6	0	12552	0	166249	6	12552	0	166249	6
Path Scan-6	115890	3.23	5	115890	3.23	5	0	6126	0	109764	5	6126	0	109764	5
Augment-Merge	152255	6.55	11	152255	6.55	11	0	46168	0	106089	11	46166	0	106089	11
Feed 800 Case 2	CRPP Distance	Time	Trips	CRPP Distance	Time	Trips	Swap	k-Split Improve	Add Improve	Final Distance	Trips	k-Split Improve	Add Improve	Final Distance	Trips
Path Scan-1	160410	1.39	6	172504	4.25	6	12094	5672	0	154738	6	5672	0	166832	
Path Scan-2	150298	3.30	6	158763	4.10	6	8465	5672	0	144626	6	5672	0	153091	
Path Scan-3	126089	3.33	6	126089	3.33	6	0	5465	4928	115696	7	5465	4928	115696	
Path Scan-4	154722	5.17	5	154722	5.17	5	0	4994	471	149257	6	4994	471	149257	
Path Scan-5	170115	3.34	6	170795	4.24	6	680	9257	2410	158448	7	141	0	170654	
Path Scan-6	121754	3.31	6	121754	3.31	6	0	640	0	121114	6	640	0	121114	
Augment-Merge	152255	6.55	11	152255	6.55	11	0	46166	0	106089	11	46166	0	106089	

Case 1: Single feed per day with slick feed.

Case 2: Two feeds per day.

Table 12.9: Performance of Heuristics for Feed Type 100-A

	CRPP Model						Split Delivery Model								
	With Swap			Without Swap			With Swap					Without Swap			
Feed 100-A Case 1	CRPP Distance	Time	Trips	CRPP Distance	Time	Trips	Swap	k-Split Improve	Add Improve	Final Distance	Trips	k-Split Improve	Add Improve	Final Distance	Trips
Path Scan-1	589635	16.23	27	616192	19.08	27	26557	119444	6926	463265	32	111235	227	504730	29
Path Scan-2	428270	15.23	25	429394	15.51	25	1124	44798	0	383472	25	44798	0	384596	25
Path Scan-3	424724	15.45	26	424724	15.45	26	0	68331	9190	348203	27	68331	8190	348203	27
Path Scan-4	639086	24.49	26	701302	31.35	26	62216	105169	9638	524279	32	149151	1717	550434	28
Path Scan-5	482044	15.23	26	494730	16.52	26	12686	43963	2196	435885	27	58334	0	436396	26
Path Scan-6	332318	15.55	24	332318	13.55	24	0	14469	13	317836	25	14469	13	317836	25
Augment-Merge	493852	18.43	32	493852	18.43	32	0	77155	865	415832	33	77155	865	415832	33
Feed 100-A Case 2	CRPP Distance	Time	Trips	CRPP Distance	Time	Trips	Swap	k-Split Improve	Add Improve	Final Distance	Trips	k-Split Improve	Add Improve	Final Distance	Trips
Path Scan-1	593971	18.06	29	631066	19.36	29	37095	112853	2456	478662	31	151744	15829	463493	29
Path Scan-2	446779	16.07	27	449805	16.17	27	3026	42527	1540	402712	30	43389	1439	404977	29
Path Scan-3	440110	16.15	27	440110	16.15	27	0	36157	0	403953	27	36157	0	403953	27
Path Scan-4	662379	24.51	29	722241	32.14	29	59862	159062	2464	500853	31	170862	15055	536324	32
Path Scan-5	517066	15.37	29	537458	17.44	29	20392	1530	0	515536	29	63686	1450	472322	31
Path Scan-6	340271	14.08	26	340271	14.08	26	0	3773	0	336498	26	3773	0	336498	26
Augment-Merge	501650	18.31	34	501650	18.31	34	0	73376	700	427574	35	73376	700	42574	35

Case 1: Single feed per day with slick feed.

Case 2: Two feeds per day.

Table 12.10: Performance of Heuristics for Feed Type 100-B

	CRPP Model						Split Delivery Model								
	With Swap			Without Swap			With Swap					Without Swap			
Feed 100-B Case 1	CRPP Distance	Time	Trips	CRPP Distance	Time	Trips	Swap	k-Split Improve	Add Improve	Final Distance	Trips	k-Split Improve	Add Improve	Final Distance	Trips
Path Scan-1	247803	7.17	16	255553	9.44	16	7750	41542	11268	194993	21	65700	11160	178693	20
Path Scan-2	141431	7.42	16	141431	7.42	16	0	10711	390	130330	17	10711	390	130330	17
Path Scan-3	183746	8.17	15	183746	8.17	15	0	18860	4391	160495	22	18860	4391	160495	22
Path Scan-4	325777	4.32	15	390759	12.02	15	64982	115283	41842	168652	22	149195	7939	233625	24
Path Scan-5	212646	8.53	14	212646	8.53	14	0	19601	0	193045	14	19601	0	193045	14
Path Scan-6	123132	7.12	15	123132	7.12	15	0	4475	2492	116165	17	4475	2492	116165	17
Augment-Merge	150973	8.06	18	150973	8.06	18	0	7431	5094	138448	20	7431	5094	138448	20
Feed 100-B Case 2	CRPP Distance	Time	Trips	CRPP Distance	Time	Trips	Swap	k-Split Improve	Add Improve	Final Distance	Trips	k-Split Improve	Add Improve	Final Distance	Trips
Path Scan-1	245842	8.11	19	253102	9.59	19	7260	27670	1981	216191	20	46830	3517	202755	23
Path Scan-2	166480	8.06	17	166480	8.06	17	0	15958	0	150522	17	15958	0	150522	17
Path Scan-3	209674	8.51	16	209674	8.51	16	0	34831	5056	169787	21	34831	5056	169787	21
Path Scan-4	341785	6.07	18	425344	12.29	18	83559	77853	4316	259616	22	145856	6670	272818	25
Path Scan-5	227362	8.49	17	228606	9.20	17	1244	36308	3823	187231	21	38958	2143	187505	20
Path Scan-6	145428	7.46	17	145428	7.46	17	0	622	0	144806	17	622	0	144806	17
Augment-Merge	156056	8.23	17	156056	8.23	17	0	4743	0	151313	17	4743	0	151313	17

Case 1: Single feed per day with slick feed.

Case 2: Two feeds per day.

Figure 12.3: Performance of Heuristics with respect to Distance for Case 1

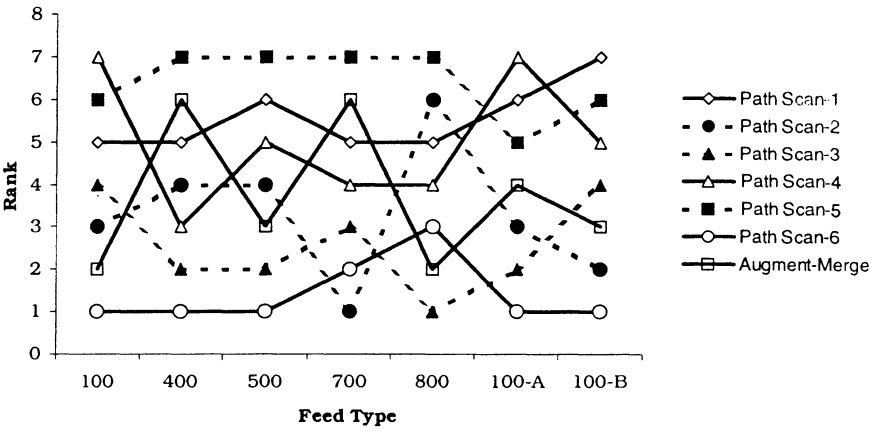


Figure 12.4: Performance of Heuristics with respect to Distance for Case 2

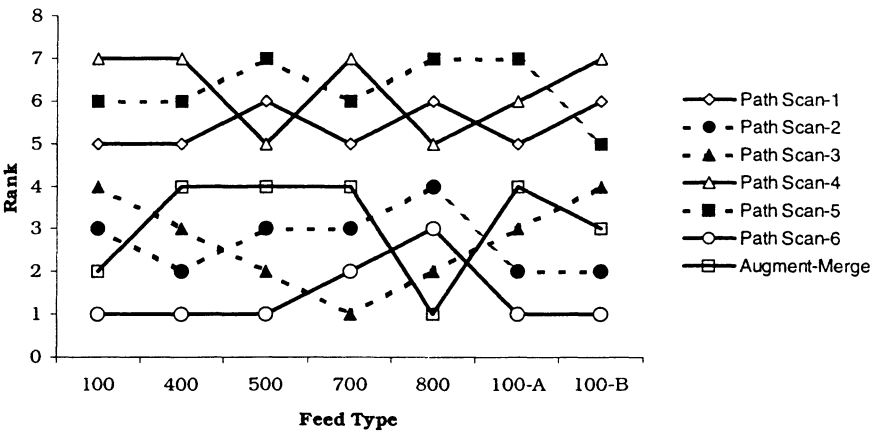


Figure 12.5: Performance of Heuristics with respect to Number of Trips for Case 1

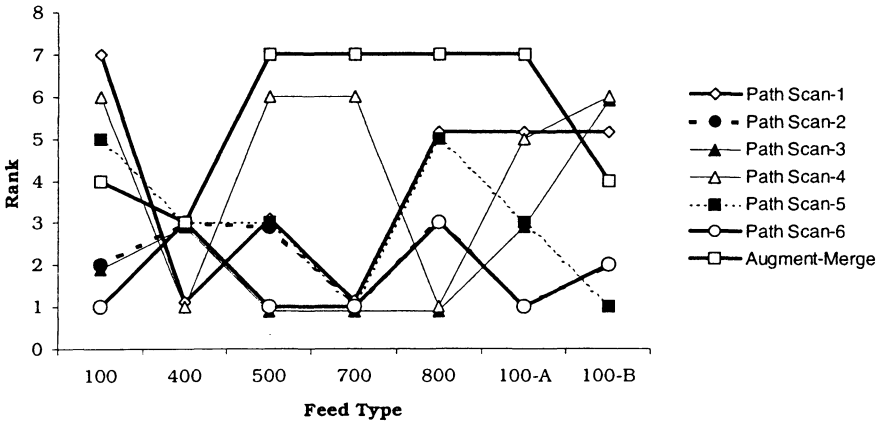
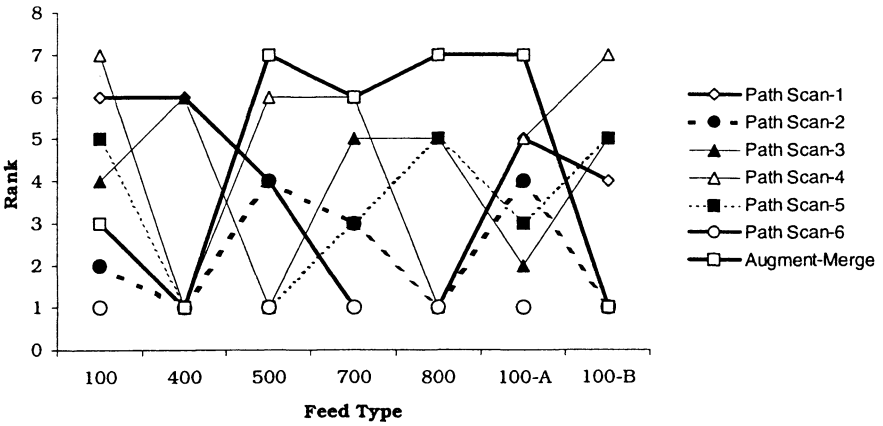


Figure 12.6: Performance of Heuristics with respect to Number of Trips for Case 2



Author Index

Aarts, E. 379, 383.
Agnetis, A. 151, 165.
Alfa, A.S. 9, 13.
Alprin, B.S. 397, 414.
Altinkemer, K. 107, 128, 163, 165.
Andersen, L.D. 47, 85, 150, 165.
Appel, K. 39, 85.
Applegate, D. 124, 125, 128.
Araque, J.R. 225, 227
Arora, S. 157, 158, 159, 165,
Assad, A.A. 1, 12, 13, 148, 156, 163, 165, 200, 227, 235, 271, 286, 287, 323, 326, 420, 421, 425, 436, 438, 441, 450, 462.
Atlas, L. 107.
Augerat, P. 225, 226, 227.
Ausiello, G. 140, 166.
Bachem, A. 107, 128.
Baker, E.K. 269, 273, 365, 367, 385, 452, 464.
Balinski, M.L. 128, 292, 323.
Ball, M.O. 107, 115, 118, 126, 128, 130, 134, 148, 165, 215, 227, 229, 256, 271, 323, 324, 342, 384, 436, 441, 462.
Barnhart, C. 177, 181, 195, 321, 323.
Bauer, K.M. 413, 414.
Bean, J.C. 169, 285, 326, 457, 458, 464.
Beasley, J.E. 297, 300, 323.
Belenguer, J.M. 12, 13, 165, 219, 220, 221, 222, 224, 225, 226, 227, 260, 262, 265, 266, 267, 268, 269, 271, 272.
Beltrami, E. x, xi, 6, 13, 331, 384.
Benavent, E. ix, 8, 12, 13, 146, 161, 165, 166, 206, 219, 220, 221, 222, 224, 225, 226, 227, 228, 239, 260, 262, 265, 266, 267, 268, 269, 271, 272, 324, 377, 384.
Bendish, J. 107, 128.
Bent, W. 149, 150, 166.
Berge, C. 66, 85, 107, 128, 142.
Bermond, J.-C. 58, 85.
Blackburn, R.R. 413, 414.
Bodin, L. x, xi, 6, 9, 13, 101, 107, 331, 384, 425, 436, 438, 440, 441, 442, 450, 463.
Boland, N. 181, 195.
Bondy, J.A. 23, 52, 53, 58, 85.
Boselly, E.S. III. 413, 414.
Bretherton, S. 408, 414, 417.
Brucker, P. 8, 13.
Burkard, R.E. 97, 120.
Campagna, L. 9, 15, 398, 416.
Campbell, J.F. x, 394, 414.
Campos, V. 8, 10, 12, 13, 14, 161, 165, 204, 206, 209, 217, 228, 239, 245, 249, 251, 256, 257, 268, 269, 272, 285, 324, 340, 342, 344, 377, 384, 450, 463.
Carre, B. 78, 85.
Carraresi, P. 93, 128.
Cattrysse, D. 412, 416.
Cebry, M.E. 421, 442.
Chapleau, L. 369, 384.
Christof, T. 202, 228.
Christofides, N. 5, 8, 10, 11, 13, 14, 99, 101, 128, 161, 163, 203, 204, 206, 209, 217, 228, 237, 239, 245, 249, 251, 256, 257, 272, 285, 297, 300, 316, 323, 325, 339, 340, 342, 344, 354, 362, 384, 450, 463.
Chvátal, V. 52, 53, 85, 290, 324.
Clarke, L. 179, 180, 181, 195.
Clarke, G. 367, 384, 457, 463.
Clohan, G.M. 397, 417.

Conforti, M. 195.
Cook, S.A. 136, 139, 282, 324.
Cook, T.M. 397, 414.
Cook, W. 123, 124, 125, 127, 128, 129.
Corberán, A. ix, 8, 10, 12, 13, 14, 146, 148, 161, 165, 203, 204, 205, 206, 209, 210, 211, 212, 213, 217, 224, 225, 226, 227, 228, 239, 245, 246, 247, 249, 251, 254, 256, 257, 258, 259, 268, 269, 270, 272, 285, 324, 340, 342, 344, 345, 349, 360, 361, 377, 384, 450, 463.
Cornuéjols, G. 99, 128, 192, 193, 195, 203, 215, 225, 228, 250, 251, 272.
Crescenzi, P. 140, 165.
Croes, G.A. 384.
Crowder, H.P. 264, 272.
Cunningham, W.H. 129.
Dantzig, G.B. 233, 272.
DeArmon, J.S. 269, 273, 365, 367, 385, 452, 464.
de Bruijn, N.G. 7, 16, 331, 383.
Dejax, P. 290, 293, 294, 298, 301, 302, 304, 320, 323, 325.
Denardo, B.V. 297, 324.
Derigs, U. viii, 91, 97, 100, 101, 106, 107, 111, 115, 117, 118, 120, 121, 122, 123, 124, 125, 126, 128, 129, 130, 134, 142, 143, 165, 166, 324, 330, 384.
DeSilva, A.H. 421, 442.
Desrochers, M. 107, 130, 289, 290, 292, 293, 294, 295, 296, 297, 298, 299, 301, 314, 316, 319, 320, 321, 322, 323, 324, 325.
Desrosiers, J. 282, 288, 289, 292, 293, 294, 295, 298, 312, 314, 319, 321, 322, 323, 324.
Detti, P. 151, 165.
Dial, R. 107, 128.
Dijkstra, E.W. 99, 130.
DiLisio, F.J. 421, 442.
Dirac, G.A. 32, 85.
Drenick, R.F. 98, 131.
Dror, M. vi, vii, viii, ix, xi, xii, 9, 14, 107, 130, 147, 152, 153, 154, 156, 162, 166, 167, 169, 278, 279, 280, 282, 287, 288, 289, 290, 294, 296, 301, 302, 304, 305, 307, 308, 311, 312, 313, 314, 315, 316, 317, 320, 323, 324, 325, 326, 383, 384, 447, 450, 451, 452, 454, 455, 456, 457, 459, 463, 464.
Dumas, Y. 282, 288, 289, 294, 295, 312, 314.
Edmonds, J. viii, 4, 5, 6, 8, 14, 95, 97, 101, 104, 108, 112, 113, 114, 116, 130, 134, 159, 160, 164, 167, 172, 176, 182, 195, 196, 203, 204, 205, 206, 228, 236, 237, 238, 239, 273, 329, 330, 331, 332, 333, 384.
Eglese, R.W. ix, 9, 15, 167, 226, 230, 269, 270, 274, 325, 380, 383, 384, 385, 411, 412, 415, 450, 463.
Eilon, S. 316, 325.
Eiselt, H.A. v, vii, 1, 8, 14, 134, 167, 173, 195, 200, 228, 235, 273, 287, 325, 396, 415, 450, 463.
Euler, L. 3, 4, 14, 96, 130.
Evans, J.R. 398, 399, 415.
Even, S. 4, 14.
Fagan, G. 436, 441.
Ferland, J.A. 369, 384.
Fischetti, M. 249, 273.
Fisher, M. 377, 385.
Fizzell, P.W. 305, 308.
Fleege, E.J. 413, 414.
Fleishmann, B. 203, 228, 236, 250, 273,
Fleischner, H. vii, xii, 2, 4, 5, 8, 14, 20, 45, 46, 47, 50, 54, 55,

56, 58, 75, 85, 86, 141, 150, 151, 165, 167, 279, 325, 329, 385.
Fleury, M. 5, 14.
Fonlupt, J. 203, 215, 228, 250, 251, 272,
Ford, L.R. 6, 7, 14, 83, 85, 86, 143, 167, 172, 195, 206, 228, 240, 273.
Fox, B.L. 297, 324.
Frank, A. 67, 83, 86.
Fredrickson, G.N. 8, 10, 14, 147, 148, 159, 160, 161, 162, 167, 333, 339, 334, 336, 349, 352, 385.
Fuijii, M. 94, 130.
Fulkerson, D.R. 6, 7, 14, 83, 85, 86, 143, 167, 172, 189, 192, 195, 206, 228, 233, 240, 272, 273.
Gabow, H.N. 109, 124, 130, 385.
Gallo, G. 93, 128.
Galil, Z. 167, 330, 385.
Gao, Z.C. 58, 86.
Gambosi, G. 140, 165.
Garey, M.R. 58, 86, 135, 140, 143, 151, 156, 158, 167, 219, 279, 282, 325, 450, 463.
Gastou, G. 189, 190, 193, 196.
Gavish, B. 107, 128, 163, 165.
Gélinas, É. 398, 415.
Gendreau, M. 1, 8, 14, 167, 200, 228, 235, 273, 287, 290, 294, 298, 301, 302, 304, 320, 323, 325, 359, 385, 396, 408, 415, 450, 455, 463, 464.
Gerards, A.M.H. 91, 130.
Ghiani, G. 9, 10, 14, 15, 153, 211, 229, 249, 250, 256, 273, 285, 325.
Ghouila-Houri, A. 78, 85.
Giffin, J.W. 305, 308.
Gilbert, J. 398, 415.
Gini, M. 406, 415.
Golden, B.L. 1, 11, 12, 13, 15, 101, 148, 156, 163, 165, 167, 200, 204, 219, 220, 227, 229, 235, 259, 269, 271, 273, 286, 287, 325, 326, 361, 365, 367, 385, 415, 420, 421, 425, 428, 436, 438, 441, 450, 452, 454, 455, 462, 464.
Gomory, R.E. 172, 189, 190, 195, 235, 246, 273.
Goode, L. 401, 406, 415.
Gottlieb, E.S. 222, 229.
Graham, R.L. 68, 86, 169.
Gray, D.M. 390, 415.
Greistorfer, P. 376, 385.
Grötschel, M. ix, xii, 8, 85, 87, 99, 123, 131, 169, 171, 192, 193, 195, 200, 206, 208, 215, 226, 229, 237, 238, 240, 243, 244, 253, 262, 273, 274.
Guan, M.-G. (Mei-Ko Kwan) 5, 8, 15, 75, 86, 95, 131, 134, 148, 149, 167, 168, 172, 174, 181, 195, 203, 204, 229, 243, 274, 336, 385.
Guéguen, C. 290, 294, 298, 301, 302, 304, 320, 323, 325.
Guenin, B. 193, 195.
Gun, H. 217, 229, 257, 274.
Guttridge, A. 410, 415.
Haimovich, M. 163, 167.
Haken, W. 39, 85.
Hall, L.A. 225, 227.
Hane, C. 177, 195.
Haouari, M. 162, 166, 293, 325.
Harary, F. 87, 151, 168.
Harche, F. 225, 228, 267, 274.
Harris, B. 189, 192, 195.
Haslam, E. 9, 15, 395, 400, 403, 416.
Hasselström, D. 94, 131.
Hecht, M.S. 148, 167, 349, 352, 385.
Held, M. 100, 131.

Hell, P. 68, 86.
Hertz, A. x, 10, 12, 15, 249, 250, 274, 285, 326, 355, 359, 375, 377, 380, 382, 383, 385, 408, 415, 455, 463.
Hierholzer, C. 4, 15.
Hochbaum, D.S. 157, 168.
Hoffman, A.J. vi, xii,
Holland, O. 130, 131, 237, 238, 262, 273.
Hu, T.C. 246, 273.
Improta, G. 9, 14, 153, 285, 325.
Iri, M. 98, 131.
Israni, S. 9, 15, 152, 168, 450, 464.
Jackson, B. 58, 86.
Jaeger, F. 46, 58, 87.
Jaikumar, R. 377, 385.
Jansen, K. 161, 164, 168.
Johnson, D.S. 58, 86, 131, 134, 135, 137, 139, 140, 141, 143, 151, 156, 158, 167, 168, 219, 228, 279, 282, 283, 325, 326, 450, 463.
Johnson, E.L. viii, xii, 4, 5, 6, 8, 14, 95, 97, 101, 104, 130, 134, 159, 160, 164, 167, 172, 176, 177, 179, 180, 182, 181, 189, 190, 191, 193, 195, 196, 203, 204, 205, 206, 228, 236, 237, 238, 239, 264, 272, 273, 321, 323, 326, 329, 330, 331, 332, 333, 384.
Johnson, S.M. 233, 272.
Jünger, M. 211, 226, 229, 232, 235, 271, 274.
Kandula, P. 401, 406, 417.
Kann, V. 140, 165.
Kapoor, A. 195.
Kappauf, C.H. 168, 206, 229.
Karp, R.M. 100, 131, 141, 168, 195, 200, 229.
Kasami, T. 94, 130.
Kernighan, B.W. 101, 131, 455, 464.
Keyser, H.J. 393, 416.
Khachian, L.G. 192, 196.
Kim, C.E. 148, 167, 349, 352, 385.
Koch, J. 39, 85.
Koehler, G.J. 168, 206, 229.
König, D. 2, 6, 15, 87.
Kozin, F. 98.
Krabs, W. 130.
Kruskal Jr., J.B. 99, 131.
Kuhn, H.W. 131.
Kursh, S. x, xi, 9, 13, 450, 463.
Labbe, M. 164, 168.
Ladner, R. 107.
Langevin, A. ix, x, 147, 156, 166, 383, 384, 394, 414, 454, 463.
Lapalme, G. 369, 384.
Laporte, G. v, vii, 1, 8, 10, 12, 14, 15, 134, 141, 164, 167, 168, 173, 195, 200, 211, 228, 229, 235, 249, 250, 256, 273, 274, 284, 285, 287, 290, 305, 307, 324, 325, 326, 355, 359, 375, 377, 380, 382, 383, 385, 396, 408, 416, 428, 439, 442, 450, 451, 455, 463, 464.
Lawler, J.K. ix, xii, 91, 109, 131, 168, 326, 330, 385.
Lehman, A. 196.
Lemieux, P.F. 9, 15, 398, 416.
Lenstra, A.H.G. ix, xii, 9, 15, 147, 168, 203, 229, 245, 250, 274, 279, 326, 338, 379, 383, 385.
Letchford, A.N. ix, 9, 10, 15, 146, 167, 213, 216, 219, 223, 224, 225, 226, 230, 245, 247, 249, 251, 253, 254, 255, 269, 270, 272, 274, 325, 412, 416.
Leung, J.M.Y. vii, xi, xii, 156, 166, 167, 278, 279, 280, 289, 305, 316, 325, 447, 463, 464.
Leuven, K.U. 412, 416.

Levy, L. x, xi, xii, 107, 425, 426, 436, 438, 440, 441, 442, 455, 463.
Lewis, P.M. III. 355, 386.
Li, L.Y.O. 380, 383, 385, 386, 411, 412, 415.
Lichtenstein, D. 149, 168.
Liebling, T.M. 238, 274, 397, 416.
Liebman, J.C. 411, 415.
Lin, S. 101, 131, 455, 464.
Lin, Y. 332, 385.
Liu, D.Q. 9, 13.
Lord, B. 391, 416.
Lotan, T. 412, 416.
Lovász, L. 38, 59, 64, 67, 68, 83, 85, 87, 91, 131, 169.
Lucas, M.E. 5, 15.
Lukka, A. 355, 386.
Lund, C. 157, 158, 159, 165.
Machol, R.E. 91, 131.
Maffioli, F. 229.
Magazine, M.J. 148, 165, 215, 227, 256, 271, 342, 384.
Magnanti, T.L. 225, 227, 229, 271, 323, 324.
Male, D.H. 390, 411, 415.
Malich, M. 107, 128.
Manber, U. 9, 15, 149, 150, 152, 166, 168, 450, 464.
Marchetti-Spaccamela, A. 140, 166.
Marks, H.D. 397, 417.
Marsh, A.B. 129.
Marsten, R. 177, 195.
Martello, S. 229, 260, 275.
Marti, R. 11, 14, 345, 349, 360, 361, 384.
Matsui, S. 98.
Matthews, M. 57, 87.
McGeoch, C.C. 131.
McGrane, E.J. 413, 414.
McKelvey, B. 390, 417.
Megiddo, N. 167.
Meloni, C. 151, 165.
Mercure, H. 164, 168.
Metz, A. 106, 107, 121, 122, 124, 125, 128, 129, 130, 131, 330, 384.
Micali, S. 330, 385.
Miller, D.L. 107, 131.
Miner, R. 414, 417.
Miner, M.W. 408, 415.
Minieka, E. 8, 15, 148, 168, 333, 336, 386.
Minsk, L.D. 390, 393, 417.
Mittaz, M. x, 12, 15, 285, 326, 375, 377, 380, 382, 383, 385, 455, 464.
Monma, C.L. 130, 229, 271, 323, 324.
Mosterts, S. 191, 196.
Mota, E. 8, 10, 12, 13, 14, 161, 165, 204, 206, 209, 217, 228, 239, 245, 249, 251, 256, 257, 268, 269, 272, 285, 324, 340, 342, 344, 377, 384, 450, 463.
Mullaseril, P.A. vii, xi, xii, 141, 167, 168, 278, 279, 287, 289, 305, 307, 311, 314, 315, 316, 317, 323, 325, 326, 456, 459, 464.
Muraldharan, B. 103, 104, 131, 132.
Murdock, H. 450, 463.
Murota, K. 98.
Naddef, D. 203, 215, 225, 226, 227, 228, 250, 251, 272.
Nanchen-Hugo, P. 10, 12, 15, 249, 250, 274, 385.
Nantung, T. 401, 415.
Nash-Williams, C.St.J.A. 151, 168.
Nemhauser, G.L. 99, 104, 129, 130, 131, 177, 179, 180, 181, 195, 195, 200, 229, 230, 271, 321, 323, 324.
Newman, J.R. 3, 4, 14
Ninomiya, N. 94.

Nobert, Y. 8, 15, 169, 176, 196, 206, 207, 208, 230, 240, 241, 242, 243, 275, 285, 326.
Noon, C.E. 169, 285, 326, 457, 458, 464.
Olafason, S. 107.
Orloff, C.S. 6, 9, 16, 147, 169, 203, 230, 250, 275.
Oudhensden, D.V. 412, 416,
Pacciarelli, D. 151, 165.
Padberg, M.W. ix, xii, 182, 196, 200, 216, 226, 229, 230, 235, 236, 237, 253, 262, 264, 266, 272, 275.
Pandit, R. 103, 104, 132.
Papadimitriou, C.H. 8, 16, 77, 117, 132, 135, 137, 139, 143, 144, 145, 146, 159, 164, 168, 169, 200, 204, 229, 230, 239, 275, 279, 283, 326, 334, 386.
Pearn, W.L. 11, 12, 13, 16, 269, 275, 286, 287, 326, 363, 370, 386.
Picard, J.-C. 8, 15, 169, 176, 196, 206, 207, 208, 230, 240, 241, 242, 243, 275, 285, 326.
Plummer, M.D. 38, 59, 64, 67, 68, 83, 85, 87, 91, 131.
Protasi, M. 140, 166.
Protvin, J.-K. 455, 464.
Pulleyblank, W. 114, 130, 149, 168, 192, 193, 195.
Quandt, R.E. 292, 323.
Queyranne, M. 201, 230, 242, 275.
Raghavachari, B. 159, 160, 161, 169.
Ralphs, T.K. 146, 206, 209, 230.
Rao, M. 182, 196, 222, 229, 237, 253, 262, 266, 275.
Ratliff, H.D. 242, 275.
Reeves, C.R. 379, 386.
Reinelt, G. 202, 211, 226, 228, 229, 232, 235, 271, 274.
Reingold, E.M. 98.
Richter, R.B. 58, 86.
Riha, S. 55, 87.
Rinaldi, G. 200, 211, 225, 226, 227, 229, 230, 232, 235, 236, 267, 274, 275.
Rinnooy Kan, A.H.G. ix, xii, 9, 15, 147, 163, 168, 203, 229, 245, 250, 274, 279, 326, 338, 385.
Riskin, E. 107.
Roberts, F.S. 27, 67, 68, 85, 87, 196.
Robertson, N. 39, 87.
Rohe, A. 123, 124, 125, 127, 129.
Romero, A. 11, 14, 203, 208, 217, 228, 230, 257, 258, 259, 272, 275, 345, 349, 360, 361, 384.
Rosenkrantz, D.J. 355, 386.
Ross, G. 379.
Rousseau, J.-M. 369, 383, 384, 386, 455, 464.
Roy, S. 383, 386.
Ryjacek, Z. 58, 87.
Salazar, J.J. 249, 273.
Salevlsbergh, M.W.P. 321, 323.
Salminen, L. 355, 386.
Sanchis, J.M. ix, 14, 16, 146, 148, 165, 203, 205, 208, 209, 210, 211, 212, 213, 217, 224, 228, 245, 246, 247, 249, 251, 254, 255, 257, 258, 259, 270, 272, 324.
Sanders, D. 39, 87.
Savall, J.V. 217, 230, 257, 275.
Schrijver, A. 85, 87, 113, 132.
Sekanina, M. 56, 87.
Seymour, P.D. 39, 87, 196.
Shenoi, R. 181, 195.
Shmoys, D.B. ix, xii, 135, 168, 169, 326.
Sierksma, G. 305, 326.
Sigismondi, G. 177, 195.
Simcox, M. 399, 413, 417.
Soland, R. 379.

Solomon, M.M. 282, 288, 289, 292, 293, 294, 295, 298, 304, 312, 314, 319, 321, 322, 323, 324, 326.
Soumis, F. 282, 288, 289, 290, 294, 295, 296, 312, 314, 324.
Spencer, J. 196.
Steiglitz, K. 77, 117, 132.
Stern, H.I. vi, xii, 9, 14, 152, 153, 154, 167, 169, 383, 384, 450, 454, 457, 464.
Sterns, R.E. 355, 386.
Stone, A.H. 196.
Stougie, L. 163, 168.
Stricker, R. 9, 16, 397, 417.
Su, S.I. 386.
Sumner, D. 57, 87.
Tardos, E. 135, 169.
Tarjan, R.E. 98, 109, 119, 130, 132.
Thienel S. 271, 274.
Thoft-Christensen, P. 206, 228, 272.
Thomas, R. 39, 87.
Thomassen, C. 57, 87.
Thompson, P. 410, 411, 417.
Tijssen, G.A. 305, 326.
Toth, P. vi, xii, 249, 260, 273, 275.
Tracey, M. 464.
Trudeau, P. 9, 14, 152, 153, 154, 167, 288, 289, 290, 305, 307, 312, 324, 383, 384, 451, 452, 454, 455, 463.
Tucker, W.B. 397, 417.
Tutte, W.T. 53, 87, 189, 196.
Ulusoy, G. 373, 386.
Vaidya, P.M. 119.
Veerasamy, J. 159, 160, 161, 169.
van Emde Boas, P. 135, 136, 138, 169,
van Aardenne-Ehrenfest, T. 7, 16, 331, 383.
Vance, P.H. 321.
Verhoog. T.W. 107, 130.
Vigo, D. vi, xii.
Volkmann, L. 67, 87.
Vuskovic, K. 195.
Waddell, B. 399, 417.
Wand, R. 107, 132.
Wang, J.-Y. 400, 401, 402, 404, 417.
Wang, Y. 201, 230.
Watson-Gandy, C.D.T. 316, 325.
Weant, M. 398, 399, 415.
Weber, G. 104, 131, 132.
Welz, S.A. 219, 220, 260, 264, 268, 275.
Wenger, E. 75, 86.
Weyl, H. 200, 230.
Win, Z. 8, 16, 103, 132, 148, 161, 169, 171, 195, 204, 206, 208, 209, 219, 229, 230, 240, 243, 244, 275, 337, 338, 377, 386, 428, 434, 442.
Wolfe, P. vi, xii.
Wolsey, L.A. 200, 230.
Wong, R.T. 11, 12, 15, 156, 163, 167, 204, 219, 220, 229, 259, 273, 325, 361, 367, 385, 415, 450, 452, 454, 455, 464.
Wright, J.R. 9, 15, 367, 384, 396, 400, 401, 402, 403, 406, 416, 420,
Wright, J.W. 442, 457, 463.
Yannakakis, M. 135, 169.
Zao, Y. 332, 385.
Zhang, C-Q. 45, 46, 75, 87.
Zhao, Y. 406, 415.
Zhu, J. 179, 180, 195.
Zowe, J. 130.