Multicriteria Discrete Optimization – and Related Topics

Christian Roed Pedersen

PhD thesis, July 2006

Multicriteria Discrete Optimization – and Related Topics

Christian Roed Pedersen, Department of Operations Research,
University of Aarhus, Denmark

PhD thesis, July 2006

Thesis committee:

- Prof. Dr. Kathrin Klamroth, University of Erlangen-Nuremberg.
- Prof. Jørgen Tind, University of Copenhagen.
- Prof. Jørgen Aase Nielsen, University of Aarhus.

Thesis advisor:

- Prof. Kim Allan Andersen, Aarhus School of Business.

Christian Roed Pedersen
Department of Operations Research
University of Aarhus
Ny Munkegade, Building 1530
DK-8000 Aarhus C
Denmark
roed@imf.au.dk

Subject classification:

- *MSC2000* – Primary: 90B50; secondary: 90C10, 90C27, 90C35, 90B80, 90C59, 90B35.
- *OR/MS* – Networks/graphs: flow algorithms, matchings; programming: integer, multiple criteria; transportation models: assignment, network; scheduling: applications.

Preface

This thesis represents the outcome of my research during my four years as a PhD student at the Department of Operations Research, University of Aarhus, Denmark. My main field of interest has been multicriteria discrete optimization, which constitutes the backbone of this thesis. Other areas such as logical inference, constraint programming, telecommunications network design and job scheduling have also gained some interest during the last four years, but only the latter is included in this thesis. By inclusion of the practical scheduling project in the last chapter, this thesis covers all main areas within operations research – theory, algorithmic developments, implementational work and practical applications.

In the late spring of 2003, I established a contact to Morten Bech Kristensen (Sonofon), Lars Jørgensen (Sonofon) and Lars Grynderup (DM-Data) together with my supervisor Prof. Kim Allan Andersen (Department of Business Studies, Aarhus School of Business, Denmark) and a fellow PhD student at my department, Rasmus Vinther Rasmussen. This initiated a cooperation on a practical job scheduling problem. Rasmus Vinther Rasmussen, Kim Allan Andersen and I developed a tabu search heuristic capable of solving the practical problem and providing a significantly improved solution. The results are to appear in *Computers and Operations Research* [128], and the work is reproduced in Chapter 10.

In August 2003, during the ISMP conference in Lyngby, my supervisor and I had an informal discussion with Prof. Kaj Holmberg (Division of Optimization, Linköping Institute of Technology, Sweden) regarding applications of the well-known Chinese postman problem. Later, the idea of working with a bicriterion version of the Chinese postman problem was born. Since then, the goal of developing an exact solution procedure for different bicriterion Chinese postman instances has been pursued by Kim Allan Andersen, myself and also by PhD Lars Relund Nielsen (Department of Genetics and Biotechnology, Research Centre Foulum, Denmark), who joined the project in the early summer of 2005. The project resulted in the subproblem on the bicriterion multi modal assignment problem presented in the working paper [126] submitted to an international journal on Operations Research. Also, the other subproblem of ranking assignments resulted in the working paper [127] to be submitted in revised form to an international journal on Operations Research. These two topics constitute the basis for Chapters 4

and 6 of this thesis. Also, the ongoing research presented in Chapter 5 on ranking solutions for the minimum cost integer flow problem and for the transportation problem, and Section 9.1 on extensions of the bicriterion multi modal assignment problem are biproducts of this original problem.

In the winter 2004/2005, I visited Prof. Dr. Horst W. Hamacher at the AG Optimization, University of Kaiserslautern, Germany, for three months. During this period, I worked together with Horst W. Hamacher and Stefan Ruzika (AG Optimization, University of Kaiserslautern, Germany), on the multicriteria minimum cost flow problem. Since new ideas are often facilitated by a complete knowledge on the existing literature, we decided to write a review paper on this problem class which resulted in the paper [69] to appear in *European Journal of Operational Research*. This is presented in Chapter 7 of this thesis. At the same time, we were engaged in developing a new solution method for the multicriteria minimum cost integer flow problem, which resulted in the yet unpublished ideas presented in Section 9.2.

During the review process for the multicriteria minimum cost integer flow problem, we learned that an instance can, in general, have exponentially many nondominated points in the size of an input. As a logical consequence of this fact, Horst W. Hamacher, Stefan Ruzika and I started, during the spring of 2005, to develop a structured way of generating a representative system for the nondominated set of general bicriterion discrete optimization problems. The results are to appear in *Operations Research Letters* [70] and are reproduced in Chapter 8.

Acknowledgements

I am indebted to a number of people who have been crucial to the completion of this PhD project.

First of all, I wish to thank my supervisor Kim Allan Andersen for his encouragement, contributions and suggestions of great value. Many ideas, presented in this thesis, are the fruits of our long discussions, and for this I am very grateful.

I would like to thank my colleagues at the Department of Operations Research, University of Aarhus, Denmark, for always providing a pleasant working atmosphere and for many interesting conversations during my four years as a PhD student. In particular, credits are due to Rasmus Vinther Rasmussen for excellent cooperation and to Randi Mosegaard for essential proof reading. Also, I am thankful for the many enlightening discussions and indispensable coffee breaks at the office with Rasmus Vinther Rasmussen and Trine Krogh Kristoffersen.

I wish also to thank my colleague Lars Relund Nielsen, who played an important role in the work on the various assignment problems addressed in this thesis. In particular, his extensive and very competent implementational help was of great value.

Next, let me express my gratitude to all the members of the AG Optimization, University of Kaiserslautern, Germany, for their comprehensive hospitality and enjoyable company during my three months visit in the winter 2004/2005. In particular, I am deeply indebted to Horst W. Hamacher and to Stefan Ruzika from whome I benefitted richly during a very fruitful cooperation reaching beyond my stay in Kaiserslautern. I also owe my gratitude to Nico Behrent for his far-reaching service and competent support with technical and practical problems of any kind.

I thank Morten Bech Kristensen, Lars Jørgensen and Lars Grynderup for data supply and for rewarding collaboration on the practical scheduling project. Also, thanks are due to Marta Margarida Braz Pascoal and Anthony Przybylski for providing test instances and for helpful comments on the various assignment projects presented in this thesis.

On the Faculty of Science, University of Aarhus, Denmark, I am particularly grateful for the tireless support by librarian Sanne Fjord Jensen, who helped me find papers from all over the world. Also, I extend my gratitude to the computer staff – to Michael Kjærgaard Sørensen and Frank Brøndum Dabelstein for extensive support on the Linux system, and to Lars Madsen for excellent type setting support.

Last but definitely not least, my heartfelt appreciation goes to my family and to my friends. Without their love and never-ending support the completion of this PhD project would have been an unbearable burden. Especially, I owe a debt of gratitude to my parents for always believing in me and encouraging me to pursue my dreams.

Aarhus, July 2006

Christian Roed Pedersen

Contents

Chapter 1

Introduction

In this thesis the main focus is on multicriteria discrete optimization problems. Of particular interest are multicriteria combinatorial optimization problems, and even more specifically I consider problem classes which can be described using a network formulation. However, work on general multicriteria discrete problems is also presented. In coherence with other multicriteria literature, I shall mainly address bicriterion optimization problems, but instances with more than two objectives will also be discussed. Related to bicriterion optimization problems, I also present new results on ranking solutions to single criterion combinatorial optimization problems. I conclude the thesis with another single criterion optimization problem in Chapter 10, where I discuss a practical scheduling problem provided by the Danish telecommunications net operator Sonofon.

The remaining part of this thesis is presented in four major parts including a total of nine chapters. To facilitate an easy reference of the figures and tables used, I have included a list of each of these in the back of the thesis together with the references, an author index and the subject index. Also, in an attempt to unify the notation used in the thesis, I have included a list of notation starting at page 173.

To make this thesis self-contained, I familiarize in the first part – comprised of Chapters 2 and 3 – my reader with basic knowledge on combinatorial optimization problems and on multicriteria optimization. Real-world decision making is generally imposed with more than one objective to be simultaneously optimized. This holds true for combinatorial optimization problems arising from many different fields of operations such as transportation planning, vehicle routing, mail distribution, etc. Observing the multicriteria nature of such problems, interest in multicriteria combinatorial optimization has prospered mainly during the last twenty years and has provided many specialized solution methods. Such solution methods for multicriteria problems, and in particular for bicriterion combinatorial optimization problems, rely heavily on efficient solution methods for their single

criterion counterparts.

In Chapter 2, I present various well-known concepts on combinatorial optimization. I introduce terminology and notation to be used throughout this thesis and present a number of relevant problem classes on which the thesis elaborates. The special problem class of ranking the K best solutions to a specific optimization problem is introduced in Section 2.2 along with a brief description of two general ranking algorithms.

In Chapter 3, I introduce terminology and notation for later developments on multicriteria optimization problems. General theoretical multicriteria results are presented. This leads to a discussion of multicriteria solution methods, including a thorough introduction to the well-known two-phase method for bicriterion optimization problems in Section 3.2.1.

In the second part of this thesis – composed of Chapters 4 and 5 – I present new results on ranking the K best solutions to a given combinatorial optimization problem. Obtaining near-optimal solutions to a single criterion problem by ranking is interesting on its own, however, we shall also see that such developments are useful for bicriterion problems – to be considered in the third part of the thesis.

In Chapter 4, which is an extension of Pedersen, Nielsen, and Andersen [127], I consider the problem of ranking solutions for the classical linear assignment problem. A new algorithm partitioning the set of feasible assignments is presented where, for each partition, the optimal assignment is calculated utilizing an efficient reoptimization technique. Computational results are included in Section 4.3 to validate the efficiency of the new ranking algorithm.

In Chapter 5, I extend the results from Chapter 4 by discussing how to rank solutions for the minimum cost integer flow problem and for the transportation problem. This chapter presents the preliminary results of an ongoing cooperation with Kim Allan Andersen and Lars Relund Nielsen and cannot, in its present form, be considered complete.

The third part – being Chapters 6 to 9 – constitutes the backbone of this thesis. Here, I present both theoretical results as well as several new algorithmic developments for multicriteria discrete optimization problems. The third part concludes with Chapter 9 proposing various ideas for further lines of research on this topic.

In Chapter 6, which is based on Pedersen, Nielsen, and Andersen [126], I consider the bicriterion multi modal assignment problem, which is a new generalization of the classical linear assignment problem. I propose a two-phase solution procedure exploiting the ranking assignments scheme presented in Chapter 4. Reports on extensive tests are given, including results on the special class of the bicriterion assignment problem.

Based on Hamacher, Pedersen, and Ruzika [69], Chapter 7 presents a review of theory and algorithms to solve the multicriteria minimum cost flow problem. For both the continuous and the integer version, exact and approximation algorithms are presented. In addition, a section on compromise solutions presents correspond-

ing results. Facilitating a direct comparison of all the solution procedures for the multicriteria minimum cost flow problem, Table 7.3 on page 109 summarizes the classification of all the reviewed papers.

For bicriterion discrete optimization problems, the number of nondominated points can, in general, be exponential in the size of an input. Therefore, in Chapter 8, based on Hamacher, Pedersen, and Ruzika [70], I propose two algorithms to compute a finite representative system for the nondominated set of a bicriterion discrete optimization problem. The theoretical performance of both algorithms are investigated, and the algorithms are evaluated with respect to a number of quality measures. A few comments on a recent computational study of the box algorithms are provided in Section 8.4.1.

Building on the results of Chapters 6 and 7, I present, in Chapter 9, ideas for further lines of research on a number of multicriteria network problems. Section 9.1 is the temporary results of my joint work with Kim Allan Andersen and Lars Relund Nielsen. It shows the bicriterion multi modal assignment problem to be a subproblem of the bicriterion directed Chinese postman problem which constitutes a larger and more complex problem class. Related extensions and algorithmic ideas are presented. The ideas presented in Section 9.2, were obtained together with Horst W. Hamacher and Stefan Ruzika. Here, I outline a rough idea for a new exact solution procedure for the multicriteria minimum cost integer flow problem exploiting the knowledge gained by the review process behind Chapter 7. Also, trying to extend the results for single criterion flow problems into a multicriteria set-up, one of the core directions for further research on this topic is highlighted.

The fourth and final part of this thesis consists of the job scheduling problem presented in Chapter 10. Job scheduling problems constitute a fundamental modelling tool within the operations research community, and have been studied intensively for more than 50 years. The concern is to allocate scarce resources to a set of tasks in the pursuit of optimizing one or more objects. Tasks may be executions of computer programs, transmissions of data packets on the internet and assembling of parts for watches, and the corresponding resources may then be servers (or CPUs), broadband connections and assembly-machinery. Even though scheduling problems can often be described using 0-1 variables they, generally, share no common particular structure. For an excellent survey of scheduling applications and theoretical developments, I refer the reader to the comprehensive book edited by Leung [100]. Since many scheduling problems are known to be $\mathcal{NP}$-hard, heuristic methods are often applied. Within these, *tabu search* due to Glover [60] and Glover and Laguna [61], in particular, have shown promising results for large-scale scheduling problems.

In Chapter 10, based on Pedersen, Rasmussen, and Andersen [128], I present a practical large-scale scheduling problem with elastic jobs faced by the Danish telecommunications net operator Sonofon. The jobs are processed on three servers and restricted by precedence constraints, time windows and capacity limitations. A tabu search procedure, including a specialized heuristic method to identify an

initial feasible solution, is presented, and extensive computational results are given. For the specific practical problem, the new solution method leads to a significant decrease in the makespan which can prevent Sonofon from purchasing additional unnecessary hardware.

The contributions of this thesis are given in the last three parts. In particular, Chapters 4 and 5 contribute to the field of ranking solutions for combinatorial optimization problems, Chapters 6, 7, 8, and 9 contribute to the field of multicriteria discrete optimization, and Chapter 10 contributes to the field of job scheduling. Chapters 5 and 9 present ongoing work and future lines of research, and cannot be considered complete.

I

Preliminaries

Chapter 2

Combinatorial optimization problems

Even though little consensus exists, in literature, for the term *combinatorial optimization*, it is generally accepted that combinatorial optimization problems deal with discrete decision making in a system with a finite or countable infinite number of alternatives, [1, 96, 110, 144]. Due to its rich field of real-life applications, combinatorial optimization problems have been in the interest of operations researchers consecutively for more than 40 years. Results are numerous and run from model formulations over increased knowledge of polyhedral structure to an excessive algorithmic development. Over the past four decades a variety of text books has contributed greatly to the field of combinatorial optimization. Among these, the works by Ahuja, Magnanti, and Orlin [2], Dell'Amico, Maffioli, and Martello [31], Grötschel, Lovász, and Schrijver [63], Lawler [96], Nemhauser and Wolsey [110], Papadimitriou and Steiglitz [121], and Schrijver [144] should be mentioned.

To a combinatorial optimization problem we often associate a finite set of decision variables $x_1, \ldots, x_m$, where, normally, non-negativity of x_j is assumed. Hence $x_j \in \{l_j, \ldots, u_j\}$ for some integral lower and upper bounds $l_j \geq 0$ and $u_j \leq \infty$. Let $\mathcal{X}$ denote the finite or countable infinite *feasible set* of alternatives. A vector $x \in \mathcal{X}$ is referred to as the *decision vector*. Then a generic description of a combinatorial minimization problem is

$$\min \left\{ y(x) \in \mathbb{R}^1 \ : \ x \in \mathcal{X} \right\} \ , \tag{2.1}$$

where one searches an *optimal solution* to the decision problem. Here, optimality relates to some cost criterion with which a quantitative measure of the quality for each of the feasible solutions can be given. In a context where the cost corresponding to a decision variable is linearly proportional to the value of the variable, we introduce the m-dimensional *cost vector* $c := (c_1, \ldots, c_m)$. In such a set-up, the

most common cost criterion is the *sum objective*

$$y(x) := \sum_{j=1}^{m} c_j x_j = cx \ , \tag{2.2}$$

to which I shall direct my main attention in this thesis. Another widely used cost criterion is the *bottleneck objective*

$$y(x) := \max_{1 \leq j \leq m} c_j x_j \ , \tag{2.3}$$

which bears a resemblance to the makespan criterion used in Chapter 10.

It should be mentioned that, in some applications on combinatorial optimization, the decision variables are restricted to be *binary variables* (also referred to as 0-1 *variables*). Hence $x_j \in \{0,1\}, \forall j$, and therefore $\mathcal{X} \subseteq \mathbb{B}^m$, where $\mathbb{B}^m$ is the set of m-dimensional binary vectors.

Throughout this thesis I shall focus mainly on combinatorial problems possessing a certain structure of the feasible set, and in the next section I formally introduce these problem classes.

2.1 Graph problems

Some of the most prominent combinatorial problems are the *graph problems*. As the name suggest, graph problems can be graphically displayed using a *graph* (or a *network*), and hence they share the property of being easy understandable even for non-operations research practitioners. Graph problems constitute a strong modelling equipment for a large variety of practical problems and also serve as important subproblems in more complex models. In the following, I introduce the notation associated with the graph problems discussed in this thesis. In doing so, I shall adopt the terminology from the main textbooks on network flows and graph theory, Ahuja et al. [2] and Bondy and Murty [12].

A *directed graph* $G = (N, A)$, is generally defined by a set N of nodes and a set of directed *arcs* A, with $n := |N|$ and $m := |A|$. For the arc $(i,j) \in A$, the nodes i and j are referred to as the *from-node* (or the tail) and the *to-node* (or the head), respectively. An arc (i,j) is *incident to* nodes i and j, and is said to be an *outgoing arc* of i and an *incoming arc* to j. For a node $i \in N$, the *indegree* (*outdegree*) corresponds to the number of incoming (outgoing) arcs. In accordance with other literature on graphs, I shall denote by $d^-(i)$ and $d^+(i)$ the indegree and the outdegree of node i, respectively. The *degree* of a node, is the sum of the indegree and the outdegree of that node.

In a directed graph, a *walk* is a collection of nodes and arcs, $i_1 - a_1 - i_2 - a_2 - \cdots - i_{r-1} - a_{r-1} - i_r$, satisfying that, for all $1 \leq p \leq r-1$, either $a_p = (i_p, i_{p+1}) \in A$ or $a_p = (i_{p+1}, i_p) \in A$. Sometimes, I suppress the explicit statement of the nodes (or the arcs) in a walk. A *directed walk* is a walk in which for any two consecutive

nodes i_p and i_{p+1} on the walk, $a_p = (i_p, i_{p+1}) \in A$. Adding the arc (i_r, i_1) or (i_1, i_r) to the walk between i_1 and i_r yields a *closed walk*.

A *(directed) trail* is a (directed) walk with no repetition of arcs. A trail from i_1 to i_r together with the arc (i_r, i_1) or (i_1, i_r) yields a *closed trail*, whereas a directed trail from i_1 to i_r together with the arc (i_r, i_1) is a *closed directed trail*.

A *(directed) path* is a (directed) trail if all nodes are distinct. A directed path from i_1 to i_r together with the arc (i_r, i_1) is a *directed cycle*. Both for trails, paths and cycles, we say that (i, j) is a *forward arc* if i is visited prior to j in the trail/path/cycle. Otherwise, (i, j) is a *backward arc*.

The nodes i and j are said to be *connected*, if the graph contains at least one path from i to j. If all pairs of nodes in a directed graph are connected, the graph is *connected*. Furthermore, a graph is said to be *strongly connected* if it has at least one directed path between every ordered pair of nodes. In a strongly connected directed graph $G = (N, A)$, a *directed Euler tour* is a closed directed trail that contains all arcs of G. Hence, every arc in G is in an Euler tour exactly once. If a graph contains an Euler tour it is called an *Eulerian graph*.

Each arc $(i, j) \in A$ is associated with a non-negative lower and a positive upper bound capacity referred to as l_{ij} and u_{ij}, respectively. I also associate to each arc $(i, j) \in A$ the non-negative cost $c_{ij} \geq 0$ that denotes the cost per unit flow on that arc. In this thesis, I focus entirely on situations where the flow cost varies linearly with the amount of flow. However, many problems of practical interest address the so-called *fixed charge problems*, in which a fixed cost is paid to open an arc as well as a variable cost component linearly dependent on the amount of flow (see Hirsch and Dantzig [74]).

Let $b \in \mathbb{Z}^n$ be a vector of demand (if $b_i < 0$, $i \in N$) and supply (if $b_i > 0$, $i \in N$) satisfying $\sum_{i \in N} b_i = 0$. If $b_i = 0$ for some $i \in N$, node i is a transshipment node. I assume integrality of all parameters (l, u, c and b). The decision variables x_{ij} in a graph problem represent the flow on each arc $(i, j) \in A$.

An *undirected graph* is defined in the same manner as a directed graph, except that edges are unordered pairs of distinct nodes. Hence, the undirected graph $G = (N, E)$ is constituted by a set of nodes N and a set of undirected *edges* E. Also, a *mixed graph* $G = (N, E, A)$, includes both a set of directed arcs A, and a set of undirected edges E. Every concept for directed graphs transfers more or less without exception to undirected and mixed graphs.

A special case of directed or undirected graphs are the *bipartite graphs* (also called *bipartite networks*), in which the node set N can be partitioned into the two sets W and V such that for every $(i, j) \in A$ either $i \in W$ and $j \in V$ or vice versa.

In the subsequent sections, I introduce the notions of a number of classical graph problems that will be discussed in this thesis.

2.1.1 The minimum cost flow problem

The *minimum cost flow problem (MCF)* is a fundamental network flow problem with a rich history of theoretical as well as algorithmic developments. The work by Hitchcock [75], Kantorovich [87], and Koopmans [90] on the transportation problem (see Section 2.1.4), which is a special instance of MCF, even predates the development of *linear programming* and the *simplex method*, usually credited to Dantzig [27, 28]. The minimum cost flow problem has a large library of applications, such as distributing a product from manufacturing plants to warehouses, distributing a product from warehouses to retailers, routing telephone calls through communication systems, transporting passengers and/or vehicles through a transportation system, etc, Ahuja et al. [2].

A function $x : A \to \mathbb{R}$ is called a *(network) flow* if it satisfies the *flow conservation constraints*

$$\sum_{j\,:\,(i,j)\in A} x_{ij} - \sum_{j\,:\,(j,i)\in A} x_{ji} = b_i \quad \forall\, i \in N \tag{2.4}$$

and the *capacity constraints*

$$l_{ij} \le x_{ij} \le u_{ij} \quad \forall\, (i,j) \in A\,. \tag{2.5}$$

The set of all flows satisfying the flow and the capacity constraints is the *flow polyhedron*, denoted $\mathcal{P}_{flow}$.

$$\mathcal{P}_{flow} = \{x : x \text{ satisfies (2.4) and (2.5)}\}$$

The minimum cost flow problem can be concisely stated as the following mathematical program:

$$\min\{cx \;:\; x \in \mathcal{P}_{flow}\}\,, \tag{2.6}$$

where $c \in \mathbb{N}_0^m$ is the non-negative integer cost vector. MCF is in general a continuous problem, i.e. the flows x_{ij} may take on fractional values. If we want to enforce integrality of the obtained solution, we solve the *minimum cost integer flow problem (MCIF)*

$$\min\{cx \;:\; x \in \mathcal{X}_{flow}\}\,, \tag{2.7}$$

where $\mathcal{X}_{flow} := \mathcal{P}_{flow} \cap \mathbb{Z}^m$ is the *flow integer lattice*.

To a minimum cost flow solution x corresponds the *residual network (or incremental graph)*, $G(x) = (N, A_f \cup A_b)$ defined as follows.

$$A_f = \{(i,j) \;:\; (i,j) \in A \;\wedge\; l_{ij} \le x_{ij} < u_{ij}\}$$
$$A_b = \{(j,i) \;:\; (i,j) \in A \;\wedge\; l_{ij} < x_{ij} \le u_{ij}\}$$

I assume that for no pair of nodes, i and j, the network G contains both the arcs (i,j) and (j,i), since this could yield parallel arcs in $G(x)$ and thereby constitute an unnecessary complication. This assumption does not impose any loss of generality, since by proper node additions and divisions of costs, one can always transform any general network to an equivalent network with this property. With this assumption, for any $(i,j) \in A$, either $(i,j) \in A_f$, or $(j,i) \in A_b$, or $l_{ij} = x_{ij} = u_{ij}$ and hence the variable x_{ij} is fixed.

Forward arc $(i,j) \in A_f$ has cost c_{ij} and *residual capacity* $r_{ij} = u_{ij} - x_{ij}$. *Backward arc* $(j,i) \in A_b$ has cost $c_{ji} = -c_{ij}$ and residual capacity $r_{ji} = x_{ij} - l_{ij}$. All lower bound capacities in $G(x)$ are 0. A flow ξ in $G(x)$ is referred to as a *residual flow* (or *incremental flow*).

By defining the set of *node potentials* $\pi = (\pi(1), \ldots, \pi(n))$ as the dual node variables, one can consider the *reduced cost residual network* $G^\pi(x)$ with *reduced costs* on arcs $(i,j) \in A$, $c^\pi_{ij} = c_{ij} - \pi(i) + \pi(j)$ and $-c^\pi_{ij}$ instead of c_{ij} and c_{ji}, respectively. The following well-known result connects the costs of directed paths and cycles in $G(x)$ and $G^\pi(x)$, (see e.g. [2]).

Lemma 2.1

(a) *For any directed cycle O and for any set of node potentials π,*

$$c^\pi(O) := \textstyle\sum_{(i,j)\in O} c^\pi_{ij} = \sum_{(i,j)\in O} c_{ij} := c(O) \ .$$

(b) *For any directed path P from node s to node t and for any set of node potentials π,*

$$c^\pi(P) := \textstyle\sum_{(i,j)\in P} c^\pi_{ij} = \sum_{(i,j)\in P} c_{ij} - \pi(s) + \pi(t) \ := c(P) - \pi(s) + \pi(t) \ .$$

To ease notation in later developments, I introduce the two operators $\oplus$ and $\ominus$ for addition and subtraction of two flows, respectively.

Definition 2.2

$\oplus$: Addition of a feasible flow x and a feasible incremental flow ξ in $G(x)$ yields the feasible flow $\hat{x} := x \oplus \xi$ with the following arc flows.

$$\forall\,(i,j) \in A \ : \ \hat{x}_{ij} := \begin{cases} x_{ij} + \xi_{ij} & \text{if } \ (i,j) \in A_f \ \wedge \ \xi_{ij} > 0 \\ x_{ij} - \xi_{ji} & \text{if } \ (j,i) \in A_b \ \wedge \ \xi_{ji} > 0 \\ x_{ij} & \text{otherwise} \end{cases}$$

$\ominus$: Subtracting the feasible flow x from the feasible flow $\hat{x}$ yields the feasible incremental flow $\xi := \hat{x} \ominus x$ in $G(x)$ with the following arc flows.

$$\forall\,(i,j) \in A \ : \ \begin{cases} \xi_{ij} := \hat{x}_{ij} - x_{ij} \ \wedge \ \xi_{ji} := 0 & \text{if } \hat{x}_{ij} > x_{ij} \\ \xi_{ji} := x_{ij} - \hat{x}_{ij} \ \wedge \ \xi_{ij} := 0 & \text{if } \hat{x}_{ij} < x_{ij} \\ \xi_{ij} = \xi_{ji} := 0 & \text{otherwise} \end{cases}$$

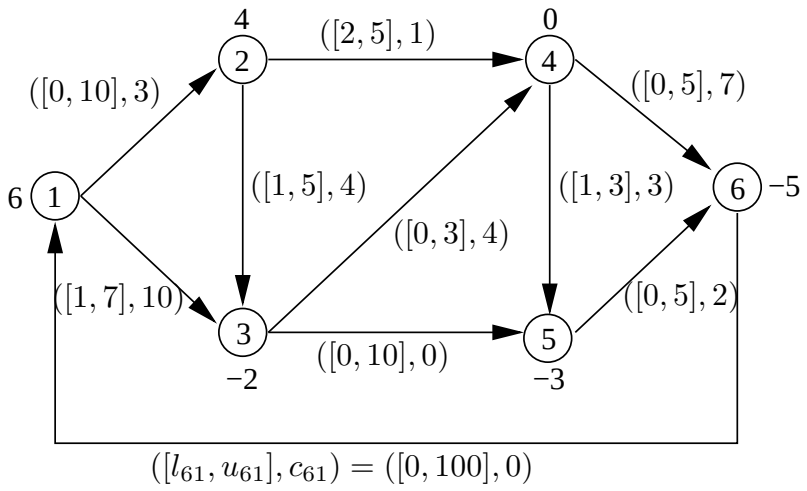


Figure 2.1: The graph of a minimum cost flow problem.

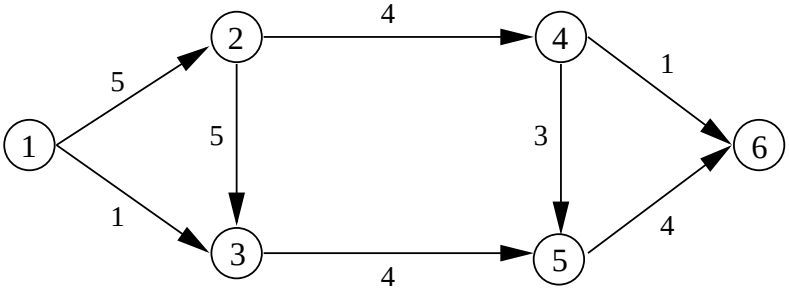


Figure 2.2: Optimal solution to the MCF in Figure 2.1.

In Definition 2.2 it is implicitly used that for the incremental flow ξ in $G(x)$, $\xi_{ij}\xi_{ji} = 0$ can, without loss of generality, be assumed for all $(i, j) \in A$.

Example 2.3 An instance of the minimum cost flow problem having $n = 6$ nodes and $m = 10$ arcs is drawn in Figure 2.1. Supplies and demands are depicted next to the node numbers, and for each arc (i, j) is displayed the lower and upper capacity bounds as well as costs, $([l_{ij}, u_{ij}], c_{ij})$. The optimal solution x, for the MCF of Figure 2.1 is displayed in Figure 2.2 showing only non-zero arc flows. The optimal value is $y(x) = 73$.

The minimum cost flow problem was proven to be polynomial time solvable by Edmonds and Karp [37] and Dinic [34], independently. In 1985, Tardos [156] developed the first strongly polynomial time algorithm for MCF. Also, the well-known *out-of-kilter method* discovered independently by Minty [104] and Fulkerson [51], remains popular due to its easiness of understanding.

2.1.2 The shortest path problem

The *shortest path problem (SP)* is among the most important combinatorial optimization problems. Not only can many practical applications be formulated in terms of an SP. Shortest path algorithms also occur as subroutines in more complex problems (e.g. the Chinese postman problem that will be discussed in Section 2.1.5). Applications stem from many different areas of operations and include the task of finding a least cost traversal in a graph, an optimal replacement strategy in a production environment and a solution for the classical *binary knapsack problem*, etc., Ahuja et al. [2], Bellman [10], Dijkstra [33], Fulkerson [52], and Lawler [96].

The definition of the shortest path problem with non-negative costs can be formalized looking at a directed uncapacitated network, $G = (N, A)$. Here, SP is to find a least cost directed path between two distinguished nodes, s and t, where the nodes s and t are referred to as the *source node* and the *sink node*, respectively. Letting $b_s = 1$, $b_t = -1$, and $b_i = 0$ for all other nodes i in the graph, SP is seen to be a special instance of the minimum cost flow problem, where $l_{ij} = 0$, and $u_{ij} = \infty$. If we let $\mathcal{P}_{path}$ denote the *path polyhedron*,

$$\mathcal{P}_{path} = \left\{ x : \sum_{j\,:\,(i,j)\in A} x_{ij} - \sum_{j\,:\,(j,i)\in A} x_{ji} = \begin{cases} 1 & \text{if } i = s \\ 0 & \text{if } i \neq s,t \\ -1 & \text{if } i = t \end{cases}, \right. \\ \left. x_{ij} \geq 0,\ \forall\,(i,j) \in A \right\} \quad (2.8)$$

the following mathematical formulation of SP can be given.

$$\min\{cx \ :\ x \in \mathcal{P}_{path}\} \quad (2.9)$$

Due to total unimodularity of the constraint matrix, $\mathcal{P}_{path}$ is an integer polyhedron. Therefore, a simplex based solution method will yield an integral optimum for SP.

The most well-known shortest path algorithm for directed graphs is *Dijkstra's method* [33], showing with time complexity $\mathcal{O}(n^2)$ that SP is strongly polynomially solvable. Many later algorithms are improvements to Dijkstra's method based on various heap structures, (see e.g. [144]).

2.1.3 The assignment problem

The *linear assignment problem (AP)* is another well-known graph problem and may be considered as the problem of assigning n workers to n jobs. Each worker must be assigned to exactly one job. Assigning worker i to job j is associated with the cost $c_{ij} \geq 0$, and the objective is to minimize total cost. Practical applications

of the assignment problem are numerous and include assigning teachers to school classes, tenants to apartments, jobs to machines, etc, and AP also arises as a subproblem in more complex decision systems, Ahuja et al. [2], Egerváry [38], and Kuhn [92, 93].

Formally, the graph corresponding to an AP is bipartite having two equally sized node sets W and V (i.e. $n := |W| = |V|$ and $n^2 = m := |A|$). The graph may be assumed to be undirected $G = (W \cup V, E)$, with $i \in W$ and $j \in V$ for all $(i, j) \in E$, *or* directed $G = (W \cup V, A)$, with $i \in W$ and $j \in V$ for all $(i, j) \in A$. Adapting the latter notation yields the *assignment polyhedron*,

$$\mathcal{P}_{AP} = \Bigg\{ x \ : \sum_{j \,:\, (i,j) \in A} x_{ij} = 1, \ \forall\, i \in W, \quad \sum_{i \,:\, (i,j) \in A} x_{ij} = 1, \ \forall\, j \in V, \\ x_{ij} \geq 0, \ \forall\, (i,j) \in A \Bigg\} . \quad (2.10)$$

Letting c_{ij} be the cost of the arc $(i, j) \in A$, AP is stated below.

$$\min\{cx \ : \ x \in \mathcal{P}_{AP}\} \quad (2.11)$$

Again, $\mathcal{P}_{AP}$ is an integer polyhedron. However, enforcing integrality, we get

$$\mathcal{X}_{AP} := \mathcal{P}_{AP} \cap \mathbb{Z}^m \ . \quad (2.12)$$

Obviously, the assignment problem is a special instance of the minimum cost flow problem in a network $G = (W \cup V, A)$, with $b_i = 1, \forall i \in W$, $b_j = -1, \forall j \in V$, $l_{ij} = 0$ and $u_{ij} = 1, \forall\, (i, j) \in A$.

The Hungarian mathematician Egerváry [38] was the first to (implicitly) state an algorithm for the assignment problem and this inspired Kuhn [92, 93] to develop the *Hungarian method* of time complexity $\mathcal{O}(n^4)$. Later, with the introduction of *successive shortest path procedures* the complexity was reduced to $\mathcal{O}(n^3)$ (see e.g. Tomizawa [158]).

2.1.4 The transportation problem

The *transportation problem (TP)* is a generalization of AP. First considered by Hitchcock [75], Kantorovich [87], and Koopmans [90], independently, TP addresses the problem of transporting goods from a set of supply nodes to a number of demand nodes. Multiple practical applications of TP arise in various fields of operations, like distribution of goods from warehouses to retailers, distribution of goods from manufacturing plants to warehouses, shipping of empty cargo ships, etc, Ahuja et al. [2] and Koopmans [90].

As for AP, a transportation problem can be described using a directed bipartite graph $G = (W \cup V, A)$, with arc costs $c_{ij} \geq 0$, $\forall\, (i, j) \in A$. Nodes in W are the supply nodes each with supply s_i, and nodes in V are the demand nodes each with

demand d_j. We let $n_1 := |W|$, $n_2 := |V|$ and as before $n_1 \cdot n_2 = m := |A|$. Hence, the transportation problem can be stated as

$$\min\{cx \ : \ x \in \mathcal{P}_{TP}\} \ , \tag{2.13}$$

using the *transportation polyhedron*,

$$\mathcal{P}_{TP} = \Bigg\{ x : \sum_{j \, : \, (i,j) \in A} x_{ij} = s_i, \ \forall \ i \in W, \sum_{i \, : \, (i,j) \in A} x_{ij} = d_j, \ \forall \ j \in V, \\ x_{ij} \geq 0, \ \forall \, (i,j) \in A \Bigg\} \ . \tag{2.14}$$

Without loss of generality it can be assumed, that the transportation problem is *balanced*, hence $\sum_{i \in W} s_i = \sum_{j \in V} d_j$. Enforcing integrality in $\mathcal{P}_{TP}$, we get the transportation integer lattice

$$\mathcal{X}_{TP} := \mathcal{P}_{TP} \cap \mathbb{Z}^m \ . \tag{2.15}$$

The transportation problem is a special instance of the minimum cost flow problem in a network $G = (W \cup V, A)$, with $b_i = s_i, \forall i \in W$, $b_j = -d_j, \forall j \in V$, and $l_{ij} = 0$ and $u_{ij} = \min\{s_i, d_j\}, \forall \, (i,j) \in A$. Notice that by setting $|W| = |V| = n$, $s_i = 1, \forall i \in W$ and $d_j = 1, \forall j \in V$, TP reduces to an instance of the assignment problem.

Tardos [156] and Galil and Tardos [53] gave the first strongly polynomial-time algorithm for the transportation problem. Later improvements are due to Orlin [119] among others.

2.1.5 The Chinese postman problem

The *Chinese postman problem (CPP)* is a graph problem that dates back to the Chinese mathematician Meigu Guan (or Kwan Mei-Ko) [64]. A postman is allocated to a segment of roads which he has to traverse while delivering mail. Apart from the obvious application of CPP within mail delivery, several other real-life applications can be given, such as school bus routing, snow plowing, gritting roads in winter, street cleaning, garbage collection, etc, Dror [35], Edmonds and Johnson [36], and Eiselt, Gendreau, and Laporte [45].

The three most classical instances of CPP are the *directed Chinese postman problem (DCPP)* considering a directed graph $G = (N, A)$, the *undirected Chinese postman problem (UCPP)* on an undirected graph $G = (N, E)$, and the *mixed Chinese postman problem (MCPP)* on a mixed graph $G = (N, E, A)$. DCPP is of primal interest to me. To each arc (i, j) a non-negative length c_{ij} is associated, and the objective is to identify a closed directed walk of minimum length that visits each arc at least once.

Letting x_{ij} denote the number of times arc (i, j) is traversed in a directed walk, DCPP can be formulated as the mathematical programme

$$\min\{cx \ : \ x \in \mathcal{P}_{DCPP}\} \tag{2.16}$$

using the *directed Chinese postman polyhedron*,

$$\mathcal{P}_{DCPP} = \left\{ x : \sum_{j \,:\, (i,j)\in A} x_{ij} - \sum_{j \,:\, (j,i)\in A} x_{ji} = 0, \ \forall i \in N, \right.$$
$$\left. x_{ij} \geq 1, \ \forall\, (i,j) \in A \right\}. \tag{2.17}$$

Obviously, DCPP is an instance of the minimum cost flow problem with $b_i = 0$ for all nodes and $l_{ij} = 1$ and $u_{ij} = \infty$ for all arcs.

The Chinese postman problem can be solved by a two-phase method. First, a question of how to minimally expand a particular graph for it to contain an Euler tour must be addressed. This problem introduced by Guan [64] is referred to as the *augmentation problem* and has become a core problem of arc routing. The augmentation problem for DCPP further divides into an all-pairs shortest path computation in a directed graph and into a classical transportation problem. For UCPP, the augmentation problem splits into an all-pairs shortest path computation (in an undirected graph) and into a perfect (non-bipartite) matching problem. All these problems are strongly polynomial-time solvable, [35, 36, 45]. Second, the actual Euler tour must be identified. This can be performed in linear time both in a directed and in an undirected graph, [73, 162]. Therefore, it can be concluded, that both DCPP and UCPP are strongly polynomial-time solvable, opposed to MCPP which is known to be $\mathcal{NP}$-hard by transformation from 3SAT, [120].

2.1.6 Overview of graph problems

In Figure 2.3, I give a schematic representation of the connection between the various problem classes discussed in this section. A particular problem class can be interpreted as a subproblem of its ancestors in the tree of Figure 2.3. The "B" in the box for TP and AP points out that these problems are defined on bipartite graphs.

2.2 Identifying the K best solutions

In the task of reflecting real-life practical applications with mathematical models, as the ones described in Section 2.1, the analyst is often forced to make a number of simplifying assumptions. Such simplifications occur in instances where certain qualitative features are not displayable in a model. Therefore, it seems

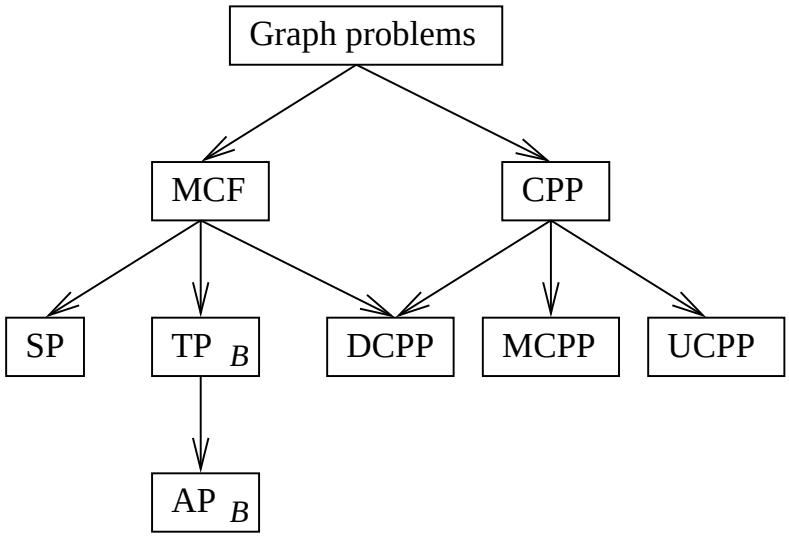


Figure 2.3: Overview of graph problems.

natural to widen the focus and provide the decision maker with a larger variety of alternatives, instead of only one optimal solution. Finding the K best solutions to a mathematical problem may provide essential knowledge in this instance. If for instance the objective values of the k best solutions are relatively close, but the $(k+1)$st solution has a decisively worse objective value, it is of the utmost importance for the decision maker to ensure implementation of one of the first k solutions.

Simplifications may also arise from the need to keep the mathematical model suitably simple, in order to be able to compute an optimal solution for it. Therefore, leaving out displayable but complicating constraints may occur. Here, an optimal solution to the original problem, including the complicating set of constraints, can be found by enumerating suboptimal solutions for the simplified mathematical model until a solution satisfying the complicating constraints is found.

The idea of ranking alternatives for mathematical models dates back to Hoffman and Pavley [76] and Bock, Kantner, and Haynes [11] identifying the K shortest paths and K shortest loopless paths in a network, respectively. The latter work was improved by Yen [168] finding the K shortest loopless paths in directed networks using a decomposition procedure. Also the work by Murty [106], identifying the K best assignments, stands as a milestone in ranking literature. In 1972, Lawler [95] generalized the work of Murty [106] and Yen [168] deriving a procedure for finding the K best solutions to general combinatorial problems with binary variables.

Despite their differences, any ranking algorithm can be classified using two identifiers: A specific *branching technique* is used to partition the set of feasi-

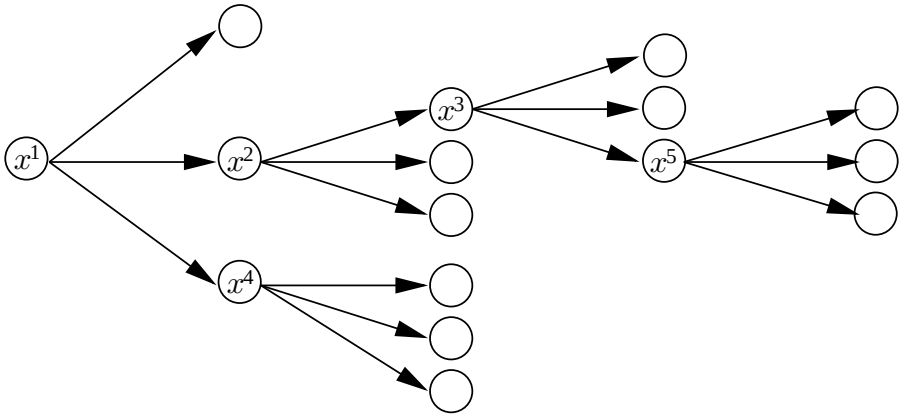


Figure 2.4: Branching technique due to Lawler [95].

ble solutions into smaller subsets, and a *solution method* needed for ranking the solutions is used to find a solution in each subset.

In the general algorithm by Lawler [95] for combinatorial problems with m binary variables, the feasible set of solutions is partitioned into at most $m-1$ non-empty disjoint subsets for each additional ranking made. The solutions already identified are excluded from all these subsets. When the $k-1$ best solutions have been identified, the kth best solution is obtained using a solution method to find the optimal solution in each subset of the current partition. The branching procedure of Lawler can be visualized in terms of the rooted tree in Figure 2.4. Each node in the tree corresponds to a subset of feasible solutions, and with x^k it is denoted that the kth best solution is found in the given subset. Whenever a subset contains the next current best solution, the corresponding subset is further partitioned into subsets, as displayed by the directed arcs of Figure 2.4. In Chapters 4 and 5, I present two implementations of the general ranking algorithm by Lawler for the assignment problem and the minimum cost integer flow problem, respectively.

Opposed to the ranking algorithm by Lawler stands another general method for combinatorial problems with m binary variables by Hamacher and Queyranne [68] referred to as the *binary search tree algorithm*. Here, each set of feasible solutions is at any point of the algorithm partitioned into exactly two disjoint subsets by considering the best solution x^*, and the second best solution $\tilde{x}$ for the current set. In one subset, $\tilde{x}$ is excluded and x^* remains optimal. In the other subset, x^* is excluded and $\tilde{x}$ becomes optimal. When the $k-1$ best solutions have been identified, the kth best solution is obtained by using a solution method to identify the second best solution in each subset in the current partition. The branching procedure of Hamacher and Queyranne is visualized in Figure 2.5 with the same notation as in Figure 2.4. For readers requiring more information on ranking using binary search, I refer the reader to the original paper by Hamacher and Queyranne [68] and on its utilizations in Chegireddy and Hamacher [19] and Hamacher [66].

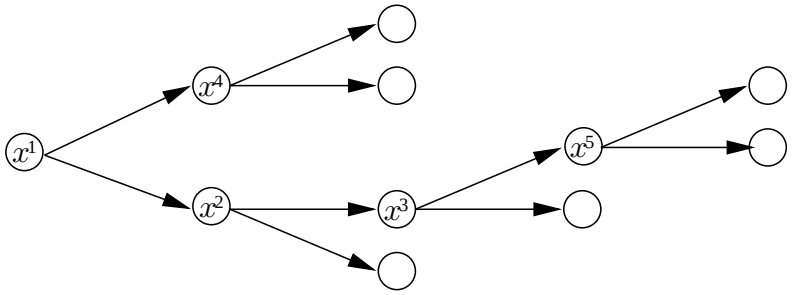


Figure 2.5: Branching technique due to Hamacher and Queyranne [68].

The K best solutions have been determined for various combinatorial optimization problems in literature. I have no intention of providing the reader with a complete list of references on ranking algorithms in this thesis. Therefore, I shall mention only a few references related to the problem classes described in Section 2.1.

K best network flows: To the best of my knowledge, this problem has only been considered by Hamacher [66] and Hamacher and Hüsselman [67] who utilize a version of the binary search tree algorithm. In Chapter 5, I present a new algorithm for ranking integer flows by modifying the branching technique by Lawler.

K shortest paths: First addressed by Hoffman and Pavley [76].

K shortest loopless paths: First addressed by Bock et al. [11] and improved by Yen [168].

K best assignments: Early work by Murty [106]. Improved by Chegireddy and Hamacher [19] and Pascoal, Captivo, and Clímaco [123]. The latest developments including further improvements by Pedersen et al. [127] can be found in Chapter 4.

K best extreme points of $\mathcal{P}_{TP}$: First applied by Murty [107] as a subroutine for the fixed charge transportation problem.

K best solutions of $\mathcal{X}_{TP}$: I am not aware of any other developments for ranking transportation solutions (including non-extreme solutions) than what is presented in Section 5.3 of this thesis.

K best undirected Chinese postman solutions: Sole work by Saruwatari and Matsui [141].

Chapter 3

Multicriteria optimization

A description of real-world applications as single criterion optimization problems is seldom realistic, since real-life decision making is often, by nature, imposed with more objectives to be simultaneously optimized. Modelling with multiple objectives (or criteria) serves its right in many different fields of operations such as production planning, transportation, network planning, scheduling, etc. For a larger range of applications, refer to the extensive survey by White [166] citing more than 500 papers on multicriteria optimization. Due to this large applicability, interest in multicriteria optimization has prospered, resulting in a large literature on the subject. The textbooks by Steuer [152] and Ehrgott [39] offer a solid introduction to multicriteria theory, algorithms and applications.

Since both combinatorial and multicriteria optimization serve as important modelling tools, it is somewhat surprising, that focus on combining these subjects arose only twenty years ago. Since then, growing interest in multicriteria combinatorial optimization problems has provided us with many more or less specialized solution algorithms, enabling us to address increasingly complex problems. Most of the combinatorial problems introduced in Chapter 2 exist in a multicriteria (or bicriterion) version in literature. For general multicriteria combinatorial problems an early review is provided by Ulungu and Teghem [160] and recent developments are surveyed in Ehrgott and Gandibleux [42]. For multicriteria network problems early reviews are offered by Current and Marsh [24] and Current and Min [25], and recent knowledge on theory and algorithms for the special case of the multicriteria minimum cost flow problem is presented in Hamacher et al. [69] and reproduced in Chapter 7. Also, a forthcoming special issue in *Annals of Operations Research* shows an ongoing profound interest in the topic, [44].

Remember that by $\mathcal{X}$ and $\mathcal{P}$ I denote a discrete feasible set and a polyhedral feasible set, respectively. Let me by $\mathcal{S}$ refer to a general feasible set of solutions in $\mathbb{R}^m$. A multicriteria optimization problem can then be stated

$$\min \{y(x) := (y_1(x), \ldots, y_Q(x)) \ : \ x \in \mathcal{S}\} \ , \tag{3.1}$$

where Q objectives must be simultaneously minimized. $\mathcal{S}$ is the *decision space*, and $\mathcal{Y} := \{y := y(x) \in \mathbb{R}^Q \ : \ x \in \mathcal{S}\}$ denotes the corresponding *criterion space*. Elements of $\mathcal{S}$ are referred to as *solutions*, whereas elements of $\mathcal{Y}$ are called *(criterion) points* or *objective vectors*. When the feasible set $\mathcal{S}$ is that of a combinatorial problem, (3.1) is referred to as a *multicriteria (or multiobjective) combinatorial optimization problem (MOCO)*.

My main interest is on problems with sum objectives (see (2.2)). Therefore, $y_j(x) := c^j x, \forall\, j = 1, \ldots, Q$. If we consider a polyhedral set of feasible solutions

$$\mathcal{P} := \{x \in \mathbb{R}^m : Ax = b,\ x \geq 0\}\ ,$$

with constraint matrix A and right-hand side vector $b \in \mathbb{R}^n$, the general *multicriteria (or multiobjective) linear program (MOLP)*

$$\min\{Cx \ : \ x \in \mathcal{P}\} \tag{3.2}$$

is obtained, where $C = (c^1, \ldots, c^Q)^T$ with rows $c^1, \ldots, c^Q$ denotes a $Q \times m$ linear objective matrix.

Enforcing integrality of all decision variables in (3.2) yields the feasible set $\mathcal{X} := \mathcal{P} \cap \mathbb{Z}^m$, and the *multicriteria (or multiobjective) linear integer program (MOLIP)*

$$\min\{Cx \ : \ x \in \mathcal{X}\}\ . \tag{3.3}$$

In general, the various objectives, for a particular problem, are conflicting in the sense that no solution simultaneously optimizes all objectives. Therefore, one is interested in finding solutions which have the property that none of the objectives can be improved without worsening one of the other objectives. Finding all or a suitable subset of these *Pareto* or *efficient solutions* is the goal of multicriteria optimization. I formalize these concepts below.

3.1 Multicriteria terminology

For single criterion optimization, the concept of optimality is well-defined. However, minimizing a vector-valued objective function requires some more explanation since there is no complete order defined in $\mathbb{R}^Q$ for $Q \geq 2$. Respecting common practice in the field of multicriteria optimization, I shall deploy the Pareto concept of optimality based on the following binary relations defined for any pair $y^1, y^2 \in \mathbb{R}^Q$.

$$\begin{array}{lcll} y^1 \leqq y^2 & \Leftrightarrow & y^1_j \leq y^2_j & j = 1, \ldots, Q \\ y^1 \leq y^2 & \Leftrightarrow & y^1_j \leq y^2_j & j = 1, \ldots, Q \quad \wedge \quad y^1 \neq y^2 \\ y^1 < y^2 & \Leftrightarrow & y^1_j < y^2_j & j = 1, \ldots, Q \end{array}$$

A (criterion) point $y^2 \in \mathcal{Y}$ is *dominated* by $y^1 \in \mathcal{Y}$ if $y^1 \leq y^2$. If no point $y^1 \in \mathcal{Y}$ dominates $y^2 \in \mathcal{Y}$ it is *nondominated*. To each nondominated point $y \in \mathcal{Y}$ exists at least one x with $y = y(x)$, which is then an *efficient solution*. Therefore, according to the Pareto concept of optimality, the *efficient* or *Pareto set*, $\mathcal{S}_E$, and the *weakly efficient* or *weakly Pareto set*, $\mathcal{S}_{wE}$, are defined as

$$\mathcal{S}_E := \{x \in \mathcal{S} \ : \ \nexists \bar{x} \in \mathcal{S} \text{ with } y(\bar{x}) \leq y(x)\}$$
$$\mathcal{S}_{wE} := \{x \in \mathcal{S} \ : \ \nexists \bar{x} \in \mathcal{S} \text{ with } y(\bar{x}) < y(x)\} \ .$$

Obviously, for a discrete feasible set $\mathcal{X}$, the corresponding notation is $\mathcal{X}_E$ and $\mathcal{X}_{wE}$, respectively. The images

$$\mathcal{Y}_N := y(\mathcal{S}_E) \quad \text{and} \quad \mathcal{Y}_{wN} := y(\mathcal{S}_{wE})$$

of these sets under the vector-valued mapping y are called the *nondominated set* and the *weakly nondominated set*, respectively. In Figure 3.1, examples of nondominated and dominated criterion points are indicated by dots and crosses, respectively.

Let me define the *Pareto cone* by $\mathbb{R}^Q_{\geqq} := \{y \in \mathbb{R}^Q : y \geqq 0\}$. Following the terminology of Steuer [152], I use the denotation

$$\mathcal{Y}^{\geq} := \text{conv}(\mathcal{Y}_N) \oplus \mathbb{R}^Q_{\geqq}$$

(the shaded grey area of Figure 3.1) where $\oplus$ denotes the usual direct sum and "conv" is the convex hull operator. For MOLP and MOLIP, the *efficient frontier* is defined as the set $\{y \in \text{conv}(\mathcal{Y}_N) : \text{conv}(\mathcal{Y}_N) \cap (y \oplus (-\mathbb{R}^Q_{\geqq})) = y\}$, which in the two-dimensional case of Figure 3.1 is equivalent to the boundary of $\mathcal{Y}^{\geq}$. For MOLP, the efficient frontier is identical with the set $\mathcal{Y}_N$. In the case of $Q = 2$ objectives, the efficient frontier of MOLP is known to be piecewise linear and convex. Its breakpoints are the *extreme nondominated points* which are images of *extreme efficient solutions* in the decision space. If the coefficient matrix is totally unimodular for MOLIP, the efficient frontier is the nondominated set of its continuous relaxation.

If a nondominated criterion point is on the efficient frontier, it is called a *supported* nondominated criterion point. Otherwise it is an *unsupported* nondominated criterion point. The corresponding solutions in decision space are denoted *supported efficient solutions* and *unsupported efficient solutions*, respectively. It is important to notice that, for MOLP, only supported efficient solutions exist, whereas, for MOLIP, unsupported efficient solutions may exist even if the constraint matrix for the considered MOLIP instance is totally unimodular.

Two different notions of connectivity based on topology and graph theory, respectively, are used in the context of multicriteria programming.

A set S is called *topologically connected* if there does not exist non-empty open sets S_1 and S_2 such that $S \subseteq S_1 \cup S_2$ and $S_1 \cap S_2 = \emptyset$. For MOLP the

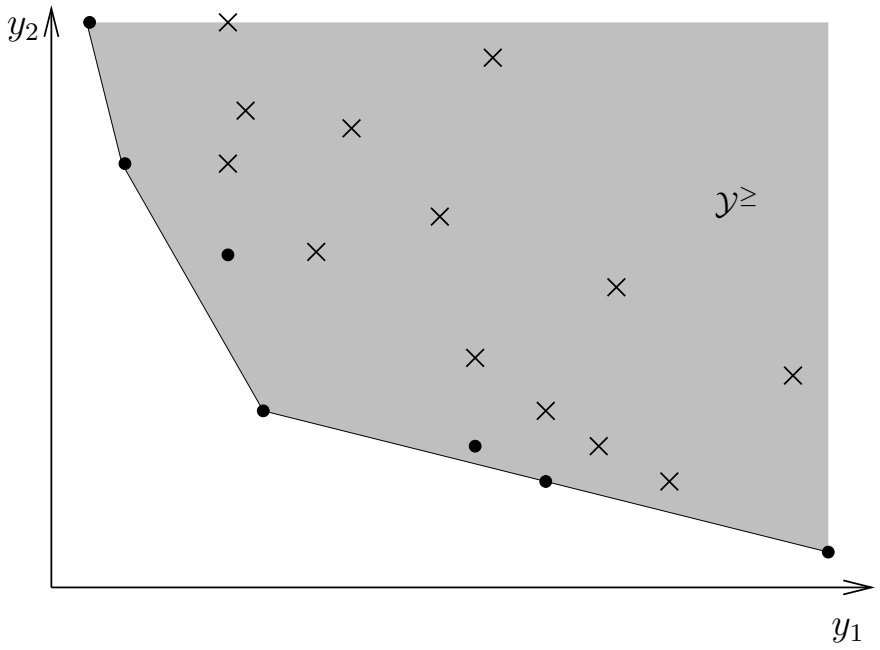


Figure 3.1: The set $\mathcal{Y}^{\geq}$ for a bicriterion linear integer program with nondominated (dots) and dominated (crosses) criterion points. Unsupported nondominated criterion points are located in the interior of $\mathcal{Y}^{\geq}$.

efficient set $\mathcal{X}_E$ and the efficient frontier $\mathcal{Y}_N$ are topologically connected as shown by Naccache [109] and Warburton [164]. In contrast, neither $\mathcal{X}_E$ nor $\mathcal{Y}_N$ are topologically connected for MOLIP.

Let $\mathcal{G} = (\mathcal{N}, \mathcal{E})$ denote the *adjacency graph of MOLP* where $\mathcal{N}$ is the set of efficient basic feasible solutions. An edge between two nodes of $\mathcal{N}$ is included in $\mathcal{E}$ if and only if the corresponding efficient basic feasible solutions can be obtained from each other by a single pivot operation. Isermann [83] showed that the adjacency graph $\mathcal{G}$ of MOLP is connected. This *graph theoretical connectedness* of the set of all basic solutions in $\mathcal{X}_E$ makes it possible to find the entire efficient and nondominated sets by simple pivot exchange arguments for MOLPs (see Figure 3.2).

The same figure illustrates that the adjacency graph is, in general, not connected for the integer version, MOLIP. Here, $\mathcal{N}$ is the set of efficient basic feasible solutions to the continuous relaxation of (3.3). Consequently, it is not possible for MOLIPs to generate the entire efficient set by "travelling" across the adjacency graph in a simplex based manner only. This result was shown for the bicriterion shortest path problem and for the bicriterion spanning tree problem by Ehrgott and Klamroth [43]. Obviously, it depends on the definition of the adjacency graph in the integer case. Other definitions of the node and edge set may salvage connectivity.

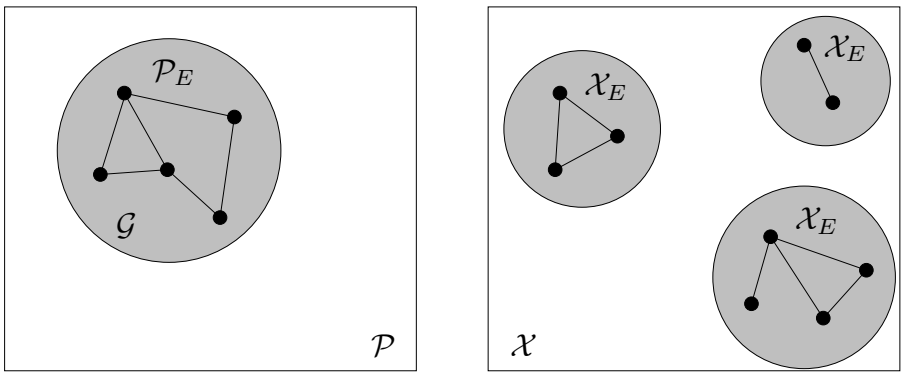


Figure 3.2: Illustration of adjacency graph: connectedness for MOLP (left) and disconnectedness for MOLIP (right).

3.2 Multicriteria solution methods

The manner in which the preferences of the decision maker are revealed plays a crucial role in the determination of the correct solution method for a multicriteria optimization problem. If such preferences are revealed prior to the decision making process, explicit or implicit usage of these are facilitated in the search for an overall desired (optimal) solution. If the preferences become known during the decision making process, interactive methods involving a ranging degree of dialogue with the decision maker are preferred, [42, 152]. In this thesis, no assumptions regarding the preferences of a decision maker are made prior to or during the optimization process. Therefore, the focus is entirely on non-interactive methods.

If the preferences of the decision maker are not known until after the optimization process, the solution methods must compute either all efficient solutions, all nondominated points or a suitable approximation of either of these. Obviously, the number of efficient solutions is no less than the number of nondominated points and efficient solutions corresponding to the same nondominated point may be interpreted as alternative solutions. Therefore, the predominant thought within multicriteria optimization is to identify all nondominated points with one efficient solution corresponding to each nondominated point. This is equivalent to identifying a *minimal complete set of efficient solutions* [55, 72].

Imposing even simple combinatorial optimization problems with more than one sum objective, normally means raising the complexity of the problem to $\mathcal{NP}$-completeness. Also for many MOCOs, the nondominated set may have an exponential number of elements, making such instances *intractable*.

Definition 3.1 A multicriteria combinatorial optimization problem is called *intractable*, if the size of $\mathcal{Y}_N$ can be exponential in the size of an instance.

Respecting both these features, a growing interest to find approximations of the nondominated sets has developed within the last twenty years. Several distinct

approaches to determine approximations of the nondominated set are utilized. One main idea is to apply a multicriteria version of a (meta)heuristic. Research in multicriteria metaheuristics covers, among others, work in tabu search, ant colony systems, simulated annealing, genetic algorithms and evolutionary algorithms, (see e.g. Ehrgott and Gandibleux [42]). Forthcoming issues on multicriteria metaheuristics proves the ongoing relevance of this topic, [3, 32].

Despite the importance of multicriteria heuristical methods, I have chosen not to consider them in this thesis. Instead, focus is on exact solution methods (identifying $\mathcal{Y}_N$), and on methods computing approximations in a non-heuristical manner. One well-known approximation concept for the nondominated set is the *ε-approximation* due to Warburton [165]. Any nondominated point is kept within a prespecified range from the nearest point in the approximation. In Definitions 6.1 and 6.2 on page 67 the concepts of ε-dominance and ε-approximation are formally introduced.

A desire to control the quality of an approximation has led to the developments of *representative systems* possessing provable quality features. Recent work is presented in Hamacher et al. [70] and reproduced in Chapter 8 to which I also postpone the formal discussion of the different quality measures. In Chapter 8, two new methods for deriving representative systems of the nondominated points are presented.

The main focus within multicriteria combinatorial optimization is on instances with $Q = 2$ objectives – and this will also be of main interest in this thesis. The bicriterion instance distinguishes itself from general $Q > 2$ objective problems because of its possibility for generating solution algorithms that benefit richly from geometrical features of the criterion space (in $\mathbb{R}^2$). Among these algorithms, is the well-known *two-phase method* originally proposed for the bicriterion assignment problem by Ulungu and Teghem [161].

In the two-phase method, no cuts are added to the original feasible set during execution. This stands opposed to other popular bicriterion optimization procedures, like for instance the ε-constraint method (see Chankong and Haimes [18]). It is evident, that the preservation of the original constraint structure, allows the two-phase method to apply iteratively the specialized solution algorithms for the single criterion instances. During the last decade, the two-phase method has proved to be successful for solving even very difficult MOCOs, (see e.g. [105, 114, 161, 163]).

Also, a choice between a pure simplex-based bicriterion optimization procedure and a two-phase method can in some way be guided by the graph theoretical adjacency results stated in Section 3.1. Since the adjacency graph of MOCOs are, in general, unconnected, simplex-based solution procedures would fail in generating all nondominated points. Deploying the two-phase method correctly does not suffer from this flaw. Due to its high applicability, I have chosen to present the two-phase method in general terms in the next section. It is exploited in Chapter 6, addressing the bicriterion multi modal assignment problem.

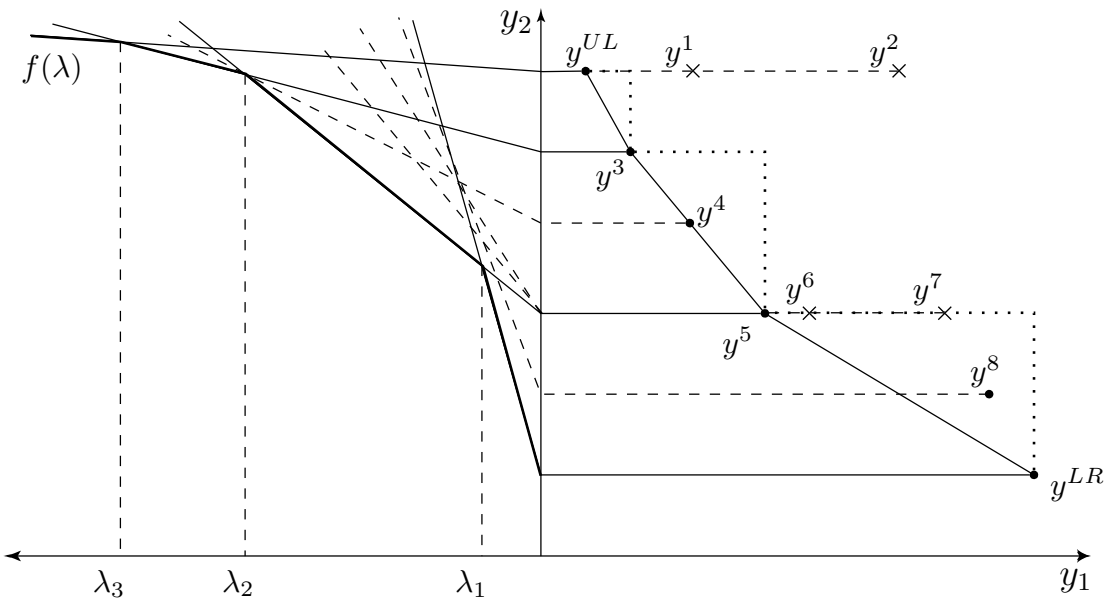


Figure 3.3: The criterion space and its corresponding parametric space.

3.2.1 The two-phase method

The two-phase method is a general approach for solving bicriterion combinatorial optimization problems. As the name suggests, the two-phase method divides the search for nondominated points into two phases.

In phase one, the supported extreme nondominated points are found. These extreme points define a number of triangles in which unsupported nondominated points may be found. Phase two proceeds to search the triangles one at a time. Both phases make use of a parametric minimization problem defined as follows.

$$\begin{array}{ll} \min & f_\lambda(x) = (\lambda c^1 + c^2)x \\ \text{s.t.} & x \in \mathcal{X} \end{array} \tag{3.4}$$

with $\lambda \geq 0$.

The method is best illustrated using an example. Suppose that the points in the right hand side of Figure 3.3 represent the criterion space of a bicriterion discrete minimization problem. Points y^{UL}, y^3, y^4, y^5 and y^{LR} are supported nondominated points of which y^4 is the only nonextreme. The point y^8 is the only unsupported nondominated point. The remaining points are dominated.

Consider the parametric minimization problem (3.4). For a fixed criterion point $y = (c^1x, c^2x)$, $f_\lambda(x)$ define a line with slope $y_1 = c^1x$ and intersection $y_2 = c^2x$ in the *parametric space* as illustrated on the left hand side of Figure 3.3. The lower envelope of the lines in the parametric space defines a non-decreasing piecewise linear function $f(\lambda)$ with break points λ_i. Note that each line on $f(\lambda)$ corresponds

```
1  procedure PhaseOne()
2     y^UL := (c^1 x^UL, c^2 x^UL), where x^UL is optimal for lex min(c^1 x, c^2 x);
3     y^LR := (c^1 x^LR, c^2 x^LR), where x^LR is optimal for lex min(c^2 x, c^1 x);
4     if (y^UL = y^LR) then STOP (only one nondominated point);
5     Y_N := {y^UL, y^LR};
6     y^+ := y^UL; y^- := y^LR;
7     while (y^+ ≠ y^LR) do
8        λ := (y_2^+ − y_2^−)/(y_1^− − y_1^+);
9        solve (3.4) with optimal solution x* and cost y* = (c^1 x*, c^2 x*);
10       if (f_λ(x*) < y_1^+ λ + y_2^+) then add y* between y^+ and y^− in Y_N;
11       else y^+ := y^−;
12       y^− := Next(Y_N, y^+);
13    end while
14 end procedure
```

Figure 3.4: Phase one – Finding supported extreme nondominated points.

to an extreme nondominated point. As a result, each extreme nondominated point can be found by identifying the point with minimal parametric weight for fixed λ values, i.e. solving (3.4). This is done in phase one, which uses a NISE[1] like algorithm (see [23]) as shown in Figure 3.4. This idea was first applied to the bicriterion transportation problem by Aneja and Nair [4].

The procedure first finds the *upper left* and the *lower right* point (y^{UL} and y^{LR} in Figure 3.3). Given two extreme nondominated points y^+ and y^-, we calculate the *search direction* λ defined by the slope of the line between the points and solve (3.4). That is, we find the value of λ where the two lines corresponding to y^+ and y^- meet in the parametric space. If the optimal solution $x^\star$ of (3.4) corresponds to a new extreme nondominated point, then the parametric weight $f_\lambda(x^\star)$ must be below the parametric weight of y^+ and y^- (line 10 of Figure 3.4). The points $y^+, y^\star$ and y^- then define two new search directions and the **while** step is repeated on the points y^+ and $y^\star$. Otherwise no new extreme nondominated point has been found and we proceed with the two next points in $\mathcal{Y}_N$, i.e. we call the `Next` function that returns the point following y in $\mathcal{Y}_N$. The procedure stops when no additional extreme nondominated points can be found.

Phase one in its present description, is actually an implementaion of *the weighted sum method* known to be capable of producing each supported efficient solution both for MOLPs and MOLIPs with any number Q, of objectives, Geoffrion [59] and Isermann [82].

Since there may exist unsupported nondominated criterion points, it is not, in general, possible to find all nondominated points during the first phase. This can be seen in Figure 3.3 where unsupported nondominated points inside the

[1] Non-inferior set estimation.

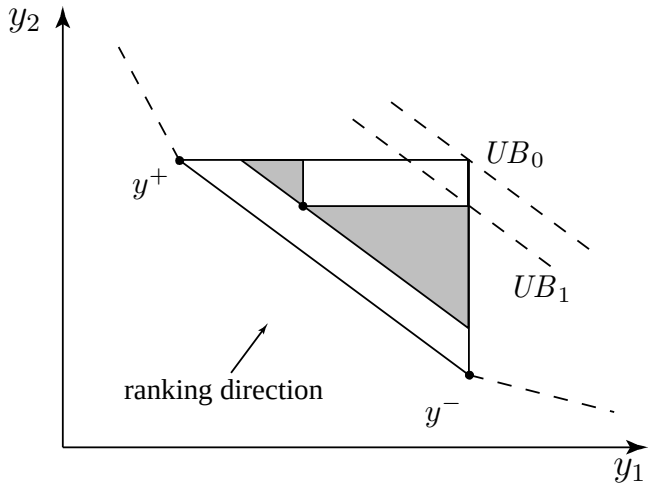


Figure 3.5: A triangle, $\triangle(y^+, y^-)$, defined by the supported extreme nondominated points y^+ and y^-.

triangles, such as y^8, correspond to a dashed line lying above $f(\lambda)$. These points are found in phase two which searches each triangle defined by the set of extreme nondominated points found in phase one. Distinct techniques for searching the triangles are presented in the literature. However, I shall only describe a generic method applying a ranking procedure, (see Section 2.2).

Consider the triangle $\triangle(y^+, y^-)$ defined by the extreme nondominated points y^+ and y^- (see Figure 3.5). The second phase searches each triangle using a K best procedure to rank the parametric weight $f_\lambda(x)$ in the *ranking direction* defined by the slope between the two points defining the triangle. The search stops when the parametric value $f_\lambda(x)$ reaches an upper bound. Initially, the upper bound is $UB_0 = y_1^-\lambda + y_2^+$. When a new unsupported nondominated point is found inside the triangle, the upper bound is updated to UB_1 as can be seen in Figure 3.5.

A pseudo code for phase two is given in Figure 3.6 where initialization is done on lines 2-4. In the main loop the parametric weight $f_\lambda(x)$ is ranked until the upper bound is reached. Procedure `KBest` returns the cost vector y^k of the k'th best solution. If it is nondominated, we add y^k to the nondominated set and update the upper bound using `UpdateUB`. Finally, we update the lower bound LB, to the current parametric weight and repeat the loop.

A detailed description of procedure `KBest` relies on the specific problem class under consideration. An implementation for the multi modal assignment problem is discussed in Chapter 6. Furthermore, note that in general `UpdateUB` finds the upper bound by using the points in the nondominated set. However, special properties such as integrality of the nondominated points may be used to improve the bound, as we will also see in Chapter 6.

```
1  procedure PhaseTwo(△(y⁺, y⁻))
2      λ := (y₂⁺ − y₂⁻)/(y₁⁻ − y₁⁺);
3      𝒴_N := {y⁺, y⁻};
4      k := 1;  LB := λy₁⁺ + y₂⁺;  UB := UpdateUB(𝒴_N);
5      while (LB ≤ UB) do
6          y^k := KBest(k, λ);
7          if (NonDom(y^k)) then
8              𝒴_N := 𝒴_N ∪ {y^k};
9              UB := UpdateUB(𝒴_N);
10         end if
11         LB := λy₁^k + y₂^k;  k := k + 1;
12     end while
13 end procedure
```

Figure 3.6: Phase two – Finding unsupported nondominated points.

II

Ranking solutions

Chapter 4

Ranking assignments using reoptimization

The *linear assignment problem* (AP) is a well-known problem and may be considered as the problem of assigning n workers to n jobs. Each worker must be assigned to exactly one job. The objective is to minimize total cost.

In an annotated bibliography authored by Dell'Amico and Martello [29], more than 100 papers on the problem are mentioned. Kuhn [92, 93] suggested the first polynomial method for the solution of AP, called the *Hungarian method* with $\mathcal{O}(n^4)$ complexity. Since 1955 several other algorithms for AP have been developed. Some of the most efficient algorithms are the class of *successive shortest path procedures*[2] with an $\mathcal{O}(n^3)$ complexity, (see e.g. Tomizawa [158] and Jonker and Volgenant [86]). An excellent survey is given by Dell'Amico and Toth [30] including comparative tests of several implementations of different AP algorithms.

The assignment problem can be generalized to ranking the first K assignments in nondecreasing order of cost. Applications of ranking assignments are numerous and include the ones presented in the general overview of ranking methods in Section 2.2. Also, ranking assignments appear as a subproblem within algorithms for solving the bicriterion assignment problem and related extensions, as we will see in Chapter 6.

Several algorithms for ranking assignments have been suggested. Recall that any ranking algorithm is classified by two identifiers: The branching technique and the solution method.

Murty [106] suggested a branching technique where the set of possible assignments is partitioned into at most $n-1$ disjoint subsets for each additional ranking made. The Hungarian algorithm was used to find the best assignment for each subset resulting in an $\mathcal{O}(Kn^5)$ complexity. However, applying e.g. a successive shortest path procedure may improve the overall complexity to $\mathcal{O}(Kn^4)$.

[2] Also known as shortest augmenting paths algorithms.

The general framework by Hamacher and Queyranne [68] for ranking solutions of combinatorial problems, (presented in Section 2.2), was specialized for bipartite matchings by Chegireddy and Hamacher [19]. The branching technique partitions the set of feasible assignments into at most two subsets for each additional ranking and, for each subset, the second best assignment has to be calculated. Different solution methods are suggested. One consists of identifying the second best assignment by a shortest cycle determination in an auxiliary network. The shortest cycle can be found by solving at most n shortest paths problems resulting in an overall $\mathcal{O}(Kn^3)$ time complexity.

Recently, Pascoal et al. [123] presented a ranking algorithm with the same branching technique as in [106]. However, by considering the subsets in reverse order when applying their solution method, they are able to reoptimize the solution from the previous subset considered and find the best assignment by solving a single shortest path problem yielding the same time complexity as in [19].

The solution methods in all the above ranking algorithms use shortest path methods to find the best assignment for each subset. Methods based on shortest paths are *dual algorithms*. Dual feasibility exists and primal feasibility has to be reached. Tomizawa [158] noted that the original costs in the assignment may be replaced with the reduced costs when using successive shortest path procedures. Since the reduced costs are non-negative, the shortest path may be found using the algorithm of Dijkstra [33].

In spite of the connection between the dual variables and successive shortest path procedures, no one has considered updating the dual variables of the previous solution before the shortest path procedure is applied to a subset. We shall see that such an update offers an improvement to the overall algorithm for ranking assignments. The new algorithm presented in this chapter uses the branching technique of Murty [106]. For each subset, a solution method is used where only one single shortest path problem has to be solved. Hence, the overall time complexity of the proposed method is $\mathcal{O}(Kn^3)$.

After a short overview over the dual properties in AP and over the successive shortest path procedures in Section 4.1, the new ranking algorithm is presented in Section 4.2. In Section 4.3, computational results are given. They include comparative tests against any other known and available implementation of algorithms, with time complexity $\mathcal{O}(Kn^3)$, for ranking assignments.

4.1 Preliminaries

As in Section 2.1.3, the assignment problem (AP) is defined on the bipartite directed graph $G = (W \cup V, A)$, with n nodes in each of the subsets $W = \{1, \ldots, n\}$ and $V = \{n+1, \ldots, 2n\}$ and $m = n^2$ arcs in A. Note that non-existing arcs can be represented as arcs having infinite cost. For sake of easy reference, the assignment

polyhedron

$$\mathcal{P}_{AP} = \Bigg\{ x : \sum_{j\,:\,(i,j)\in A} x_{ij} = 1,\ \forall\, i \in W, \sum_{i\,:\,(i,j)\in A} x_{ij} = 1,\ \forall\, j \in V, \\ x_{ij} \geq 0,\ \forall\, (i,j) \in A \Bigg\}, \quad (4.1)$$

and the related mathematical formulation of AP

$$\min\{cx \ :\ x \in \mathcal{P}_{AP}\} \quad (4.2)$$

are restated here.

In an optimal solution to (4.2), $x_{ij} = 1$ if node i is assigned node j, and zero otherwise. A feasible solution x to (4.2) is called an *assignment*. Using the network formulation, an assignment may alternatively be written as $a = \{(1, j_1), \ldots, (n, j_n)\}$ where $(i, j) \in a$ if and only if $x_{ij} = 1$. A *partial primal solution* is a solution in which less than n variables is assigned value one and the constraints in $\mathcal{P}_{AP}$ are satisfied with a $\leq$ sign instead of equality. Note that a partial primal solution corresponds to a *partial assignment* $a = \{(i_1, j_{i_1}), \ldots, (i_q, j_{i_q})\}$.

By associating *dual variables* $\pi(i)$ and $\pi(j)$ with the constraints above (see Section 2.1.1), the corresponding dual problem is

$$\begin{aligned} \max \quad & \sum_{i\in W} \pi(i) - \sum_{j\in V} \pi(j) \\ \text{s.t.} \quad & \pi(i) - \pi(j) \leq c_{ij},\ \forall\, (i,j) \in A\ . \end{aligned} \quad (4.3)$$

Given the reduced costs $c^{\pi}_{ij} = c_{ij} - \pi(i) + \pi(j),\ \forall\, (i,j) \in A$, *the complementary slackness optimality conditions* become

$$x_{ij} c^{\pi}_{ij} = 0,\ \forall\, (i,j) \in A\ . \quad (4.4)$$

4.1.1 The successive shortest path procedure

Successive shortest path procedures for AP are dual methods. Dual feasibility exists and the optimal solution is built step-by-step by iteratively adding assignments to a current partial primal solution.

A successive shortest path procedure consists of two phases. In phase one, the cost matrix $[c_{ij}]$ is preprocessed and a partial primal solution (or partial assignment) and a dual feasible solution satisfying the complementary slackness optimality conditions (4.4) are determined. In phase two, the partial primal solution is augmented by adding one row/column assignment at a time, until the solution

```
1 procedure SuccSP()
2     (a, π) := Preprocess([c_ij]);
3     while (|a| < n) do
4         G^π(a) := BuildResNetwork(a, π);
5         P := FindAugmentPath(G^π(a));
6         a := AugmentSolution(P);
7         π := AdjustDualSolution(P);
8      end while
9   end procedure
```

Figure 4.1: The successive shortest path procedure.

becomes feasible. At each step in phase two, the dual solution is updated so that complementary slackness still holds. Hence, at the end of the second phase, the current primal and dual solutions are optimal.

A pseudo-code for the successive shortest path procedure is given in Figure 4.1. Phase one is executed by function `Preprocess` which returns a partial assignment a and a dual feasible solution π satisfying (4.4). Phase two is executed on lines 3–8.

If $|a| < n$ then all nodes in W have not been assigned to a node in V and function `BuildResNetwork` builds the residual network $G^\pi(a) = (W \cup V, A_f \cup A_b)$ constructed from G and the current partial solution a. In accordance with the introduction of the residual network in Section 2.1.1, let

$$A_f = \{(i,j) \ : \ (i,j) \in A \ \wedge \ (i,j) \notin a\} \text{ and}$$
$$A_b = \{(j,i) \ : \ (i,j) \in A \ \wedge \ (i,j) \in a\} \ .$$

Each forward arc (i,j) in A_f is assigned reduced cost c^π_{ij} and each backward arc (j,i) in A_b is assigned cost $-c^\pi_{ij} = 0$ due to (4.4).

It is easy to see that any directed path in $G^\pi(a)$ contains an arc in A_f and an arc in A_b, alternatingly. Such paths are called *alternating paths*. If the directed path P starts in an unassigned node in W and terminates with an unassigned node in V, it is called an *augmenting path*.

It is well-known that, by removing assignments in a corresponding to the backward arcs in P and adding the forward arcs in P to a, the number of assignments in the (partial) assignment $\bar{a}$ increases by one. Furthermore, since AP is a special instance of MCF, the following result can be derived from the general optimality results presented in Section 5.1.

Corollary 4.1 *Let a be a partial assignment and π the corresponding dual variables fulfilling the complementary slackness optimality conditions. Let P in $G^\pi(a)$ be a shortest augmenting path and set*

$$\bar{a} = a \oplus P \ . \tag{4.5}$$

Then $\bar{a}$ is a minimum cost (partial) assignment with $|a| + 1$ assignments.

Hence, finding the shortest augmenting path in $G^{\pi}(a)$ is equivalent to finding a minimum cost (partial) assignment $\bar{a}$ with $|a|+1$ assignments. As a result, AP can be solved by identifying at most n successive shortest augmenting paths. Since the reduced costs are non-negative, each path can be determined through Dijkstra's method running in $\mathcal{O}(n^2)$ time. Therefore the overall computational complexity of a successive shortest path procedure is $\mathcal{O}(n^3)$.

In procedure `SuccSP` the shortest augmenting path P is found using function `FindAugmentPath` and next the (partial) assignment a is updated as in (4.5) using function `AugmentSolution`. Finally, the dual variables are updated using function `AdjustDualSolution` such that (4.4) holds.

For an efficient implementation of procedure `SuccSP` see for instance Jonker and Volgenant [86]. Here extensive preprocessing is used in `Preprocess` and network $G^{\pi}(a)$ is maintained implicitly in the data structures. Function `FindAugmentPath` calculates the augmented path using a specialized version of Dijkstra's method and the new solution is found in `AugmentSolution` by traversing the path. Finally, the dual solution is adjusted by traversing the path. For further details see [86].

4.2 Ranking assignments

Consider the problem of ranking the first K assignments in nondecreasing order of cost, i.e. finding the K best assignments $a^1, \ldots, a^K$ satisfying

1. $y\left(a^i\right) \leq y\left(a^{i+1}\right),\ i = 1, \ldots, K-1$
2. $y\left(a^K\right) \leq y\left(a\right),\ \forall a \in {}_a\mathcal{X}_{AP} \setminus \left\{a^1, \ldots, a^K\right\}$

where $y(a)$ denotes the cost (or the objective function value) of assignment a. Above ${}_a\mathcal{X}_{AP}$ denotes the set of all feasible integral solutions to (4.2) using the network formulation for an assignment, i.e. $a \in {}_a\mathcal{X}_{AP}$ if and only if the corresponding $x \in \mathcal{X}_{AP} = \mathcal{P}_{AP} \cap \mathbb{Z}^m$.

The branching technique described in [106] is used where the set ${}_a\mathcal{X}_{AP}$ is partitioned into smaller subsets as follows. Given the optimal assignment $a^1 = \{(1, j_1), \ldots, (n, j_n)\}$, the set ${}_a\mathcal{X}_{AP} \setminus \{a^1\}$ is partitioned into $n-1$ disjoint subsets $\mathcal{X}^i$, $i = 1, \ldots, n-1$, where

$$\mathcal{X}^i = \{a \in {}_a\mathcal{X}_{AP} \ :\ \{(1, j_1), \ldots, (i-1, j_{i-1})\} \in a, (i, j_i) \notin a\},\ i = 1, \ldots, n-1\ .$$

We say that $\{(1, j_1), \ldots, (i-1, j_{i-1})\}$ is *forced* to be in all assignments belonging to $\mathcal{X}^i$. Clearly, the second best assignment a^2 can be found by using a solution method to find the optimal assignment in the sets $\mathcal{X}^i$, $i = 1, \ldots, n-1$. Moreover, the branching technique can be applied recursively to subsets $\mathcal{X}^i \subset {}_a\mathcal{X}_{AP}$.

The pseudo code for the ranking algorithm, named `K-AP`, is shown in Figure 4.2. The algorithm implicitly maintains a candidate set Φ of pairs $(\bar{a}, \bar{\mathcal{X}})$, where $\bar{a}$ is

```
1  procedure K-AP()
2      a := SuccSP();
3      Φ := {(a, _aX_AP)};
4      for (k := 1 to K) do
5          (â, X̂) := arg min{y(ā) : (ā, X̄) ∈ Φ};
6          if ((â, X̂) = null) then STOP; else OUTPUT a^k := â;
7          Φ := Φ \ {(â, X̂)};
8          for (i := 1 to n − 1) do
9              â* := FindOptimal(X̂^i);
10             if (y(â*) < ∞) then Φ := Φ ∪ {(â*, X̂^i)};
11         end for
12     end for
13 end procedure
```

Figure 4.2: The ranking assignments algorithm.

the optimal assignment in (sub)set $\bar{\mathcal{X}}$. Assuming that the first $k-1$ assignments $a^1, \ldots, a^{k-1}$ have been found, the current candidate set represents the partition of ${}_a\mathcal{X}_{AP} \setminus \{a^1, \ldots, a^{k-1}\}$. Assignment a^k is then found by selecting and removing the pair $(\hat{a}, \widehat{\mathcal{X}})$ containing the assignment with minimum cost in the candidate set (lines 5–7). Next, the branching technique is used to partition $\widehat{\mathcal{X}}$, possibly obtaining new pairs that are added to the candidate set (lines 8–11). Note that, in practice, it is not necessary to consider all subsets in the partitioning of $\widehat{\mathcal{X}}$. Consider the case where (i, j_i) was forced to be contained in any assignment belonging to $\widehat{\mathcal{X}}$ in some previous partition. Therefore $\widehat{\mathcal{X}}^i = \emptyset$, since (i, j_i) is not allowed in any assignment of $\widehat{\mathcal{X}}^i$. In this case, it may be assumed that $\widehat{\mathcal{X}}^i$ is not generated by the algorithm.

Function `FindOptimal` represents the solution method applied to find the optimal assignment in a given subset. Consider that partition $\widehat{\mathcal{X}}$ has optimal assignment $\hat{a} = \{(1, j_1), \ldots, (n, j_n)\}$ and assume that all assignments in $\widehat{\mathcal{X}}$ cannot contain $(l_1, t_1), \ldots, (l_q, t_q)$ due to previous partitions. Recall that $(1, j_1), \ldots, (i-1, j_{i-1})$ are forced to be in all assignments belonging to subset $\widehat{\mathcal{X}}^i$ $(\subseteq \widehat{\mathcal{X}})$. Therefore, assuming $\widehat{\mathcal{X}}^i$ is non-empty, the optimal assignment can be found solving an AP of size $n - (i-1)$ where

1. Rows $\{1, \ldots, i-1\}$ and columns $\{j_1, \ldots, j_{i-1}\}$ have been removed from the reduced cost matrix $\left[c^{\pi}_{ij}\right]$, i.e. these indices are not considered in (4.2) and (4.3).

2. The reduced cost in cells (i, j_i) and $(l_1, t_1), \ldots, (l_q, t_q)$ is set to infinity.

Given a non-empty subset $\widehat{\mathcal{X}}^i$, let $AP(\widehat{\mathcal{X}}^i)$ denote the AP defined as above by subset $\widehat{\mathcal{X}}^i$. If the successive shortest path procedure, `SuccSP`, is used as the solution

method to find the optimal assignment to $AP(\widehat{\mathcal{X}}^i)$, $i = 1, \ldots, n-1$, the overall complexity of `K-AP` is $\mathcal{O}(Kn^4)$. However, the optimal assignment to $AP(\widehat{\mathcal{X}}^i)$ can be found using reoptimization – thereby reducing the complexity of the algorithm.

Let $\hat{a}$ denote the optimal assignment in subset $\widehat{\mathcal{X}}$ found by solving $AP(\widehat{\mathcal{X}})$, let $\hat{\pi}$ denote the corresponding dual values and let the partial assignment $a(i)$ be defined by removing from $\hat{a}$ the single assignment $\{(i, j_i)\}$, hence

$$a(i) := \hat{a} \setminus \{(i, j_i)\} \; . \tag{4.6}$$

The following lemma is well-known.

Lemma 4.2 *$a(i)$ is a partial assignment of size $n-1$, and $\hat{\pi}$ remains dual feasible to $AP(\widehat{\mathcal{X}}^i)$ and satisfies the complementary slackness optimality conditions* (4.4)*.*

However, by updating the dual variables according to the following scheme

$$\begin{aligned} \pi(i) &= \hat{\pi}(i) + \min_{j \in V \setminus \{j_1, \ldots, j_i\}} \{c_{ij} - \hat{\pi}(i) + \hat{\pi}(j)\} \\ \pi(r) &= \hat{\pi}(r), \; r \in \{i+1, \ldots, n\} \\ \pi(j_i) &= \hat{\pi}(j_i) - \min_{r \in \{i+1, \ldots, n\}} \{c_{rj_i} - \hat{\pi}(r) + \hat{\pi}(j_i)\} \\ \pi(j) &= \hat{\pi}(j), \; j \in V \setminus \{j_1, \ldots, j_i\} \end{aligned} \tag{4.7}$$

the following revised version of Lemma 4.2 can be derived. As we will see in Section 4.3.3, this provides a speed up of the overall algorithm due to the present implementation of the method to find a shortest augmenting path.

Lemma 4.3 *$a(i)$ is a partial assignment of size $n-1$ and π defined in* (4.7) *is a dual feasible solution to $AP(\widehat{\mathcal{X}}^i)$, satisfying the complementary slackness optimality conditions* (4.4)*.*

Proof. The reduced cost matrix to $AP(\widehat{\mathcal{X}}^i)$ using π from (4.7) can be seen in Figure 4.3, where it is utilized that $\pi(r) = \hat{\pi}(r)$, $r \in \{i+1, \ldots, n\}$, and $\pi(j) = \hat{\pi}(j)$, $j \in V \setminus \{j_1, \ldots, j_i\}$.

Only the reduced costs in column j_i and row i have changed. Due to (4.7), the reduced costs in row i satisfy

$$c_{ij} - \pi(i) + \pi(j) \geq c_{ij} - (\hat{\pi}(i) + (c_{ij} - \hat{\pi}(i) + \hat{\pi}(j))) + \hat{\pi}(j) = 0, \;\; \forall j \in V \setminus \{j_1, \ldots, j_i\} \; .$$

Similar results hold for the reduced costs in column j_i. Hence π is a dual feasible solution to $AP(\widehat{\mathcal{X}}^i)$. Moreover, since the reduced costs corresponding to the elements in assignment $a(i)$ have not changed, the complementary slackness optimality conditions (4.4) still hold. □

A pseudo-code for the reoptimization algorithm is given in Figure 4.4. Here, first the partial assignment $a(i)$ and the dual values π are calculated due to (4.6)

$$
\begin{array}{c}
 \\ i \\ i+1 \\ \vdots \\ n
\end{array}
\begin{array}{c}
\begin{array}{cccc} j_i & j_{i+1} & \cdots & j_n \end{array} \\
\left[\begin{array}{cccc}
\infty & c_{ij_{i+1}} - \pi(i) + \hat{\pi}(j_{i+1}) & \cdots & c_{ij_n} - \pi(i) + \hat{\pi}(j_n) \\
c_{i+1j_i} - \hat{\pi}(i+1) + \pi(j_i) & c^{\hat{\pi}}_{i+1j_{i+1}} & \cdots & c^{\hat{\pi}}_{i+1j_n} \\
\vdots & \vdots & \ddots & \vdots \\
c_{nj_i} - \hat{\pi}(n) + \pi(j_i) & c^{\hat{\pi}}_{nj_{i+1}} & \cdots & c^{\hat{\pi}}_{nj_n}
\end{array}\right]
\end{array}
$$

Figure 4.3: The reduced cost matrix to $AP(\hat{\mathcal{X}}^i)$.

```
1 procedure FindOptimal(X̂^i)
2     a(i) := CreatePartial(â);
3     π := ModifyDual(π̂);
4     G^π(a(i)) := BuildResNetwork(a(i), π);
5     P := FindAugmentPath(G^π(a(i)));
6     a := AugmentSolution(P);
7   end procedure
```

Figure 4.4: Finding the optimal solution for a subset.

and (4.7), respectively. As an alternative to updating the dual variables, one could substitute in line 3 of Figure 4.4 the dual variables by $\hat{\pi}$. Next, the residual network corresponding to $a(i)$ and the dual solution is built. Finally, the shortest augmenting path and the corresponding solution are found. Due to Corollary 4.1 and the fact that the length of the partial assignment $a(i)$ is $n-1$, the following result holds true.

Theorem 4.4 *Using partial assignment $a(i)$ and dual values π (or $\hat{\pi}$), the optimal assignment in subset $\widehat{\mathcal{X}}^i$ can be found by solving a single shortest path problem.*

Using Dijkstra's method to find the shortest path, function `FindOptimal` runs in $\mathcal{O}(n^2)$. Therefore, the following time complexity of `K-AP` is obtained, which is equal to the best known time complexity for ranking the K best assignments.

Theorem 4.5 *The K best assignments using procedure `K-AP` can be found in $\mathcal{O}(Kn^3)$ time.*

4.3 Computational results

In this section, computational results for the algorithms presented in this chapter are given. In order to validate the effectiveness of the new algoritms, comparative tests against other ranking algorithms from the literature are performed. The algorithms are executed in their original implementation, i.e. the test results must be interpreted with caution. To the best of my knowledge all algorithms with time complexity $\mathcal{O}(Kn^3)$ for ranking assignments – available to me – are included. All tests were performed on an Intel Xeon 2.67 GHz computer with 6 GB RAM using a Red Hat Enterprise Linux version 4.0 operating system.

4.3.1 Implementational details

Three new versions of the ranking assignment algorithm `K-AP` were implemented.

The first algorithm, `K-AP_SuccSP`, is a straightforward implementation where the successive shortest path procedure, `SuccSP`, is used as function `FindOptimal` (line 9 of Figure 4.2) to find the optimal assignment to each subproblem $AP(\widehat{\mathcal{X}}^i)$. The successive shortest path procedure implementation of Jonker and Volgenant [86] is applied in a slightly modified version allowing problems of varying size to be solved. `K-AP_SuccSP` has time complexity $\mathcal{O}(Kn^4)$.

The second algorithm, `K-AP_Reopt`, and the third algorithm, `K-AP_NoDualUp`, both have time complexity $\mathcal{O}(Kn^3)$. They use the reoptimization solution method given in Figure 4.4, where `FindAugmentPath` finds the shortest augmenting path using the modified implementation of Jonker and Volgenant [86]. In `K-AP_Reopt` the dual variables are updated according to (4.7), whereas, in `K-AP_NoDualUp`, the set of previously optimal dual variables $\hat{\pi}$ is reused.

In all three algorithms, the candidate set Φ of pairs $(\bar{a}, \bar{\mathcal{X}})$ is maintained implicitly using a binary tree as described in Nielsen [112, p137], and a 4-heap is used to sort the costs in nondecreasing order, see Tarjan [157]. In each node of the branching tree, no information on the solution is stored apart from the solution value. Therefore, a given solution must be recalculated before branching on this solution can be performed. On the downside, this results in an increased running time of the algorithm. However, on the positive side, the memory requirements are smaller. The algorithms were implemented in C++ and compiled with the GNU C++ compiler version 3.4.5 using optimize option `-O3`.

In literature, few other algorithms exist for ranking assignments. The three methods included below is, to the best of my knowledge, the only available implementations with time complexity $\mathcal{O}(Kn^3)$.

An executable version of the algorithm `K-AP_VMA_1` from Pascoal et al. [123] was provided to me by the authors (in [123] the algorithm is referred to as `VMA`). In `K-AP_VMA_1`, for solving the occuring shortest path problems, a label correcting algorithm is utilized. An internal upper bound on the allowable number of assignment nodes to be stored imposes an implicit limit on the instance size that can be solved by the current implementation of this algorithm. `K-AP_VMA_1` has been

implemented in C and compiled using the GNU C++ compiler version 3.3.5 with optimize option `-O4`.

Ranking assignments occur as a subproblem for the bicriterion assignment problem (see also Chapter 6) in the paper by Przybylski, Gandibleux, and Ehrgott [131]. The algorithm `K-AP_CH` originally suggested by Chegireddy and Hamacher [19] is employed for this purpose. Using a binary search tree branching technique ([68]), the solution method solves shortest path problems on bipartite graphs as a subprocedure. `K-AP_CH`, provided to me by the authors Przybylski et al. [131], is implemented in C and has been compiled with the GNU C++ compiler version 3.4.5 using optimize option `-O3`.

The last implementation of an algorithm ranking the K best assignments in $\mathcal{O}(Kn^3)$ was recently drawn to my attention. It consist of a new implementation, due to Przybylski [130], of the algorithm `K-AP_VMA_1` from [123]. This algorithm is referred to as `K-AP_VMA_2`. It is implemented in C and has been compiled with the GNU C++ compiler version 3.4.5 using optimize option `-O3`.

4.3.2 Test instances

To yield consistency in literature, the algorithms were tested on the instances used in Pascoal et al. [123] plus a few larger instances not reported upon in that paper. Two separate groups of instances are considered, where, only for the first group, repetitions are made for a given problem size. Therefore the main part of the results presented in this section are on the first group of test instances.

The first group of test instances consists of assignment problems on complete bipartite networks of size $n \in \{50, 100, \ldots, 300\}$. Costs are drawn randomly in $\{0, \ldots, 99\}$, which is a much smaller cost range than previously stated in [123].[3] For each problem size, ten instances generated with different seeds are available. For a given instance, the $K = 100$ best assignments are ranked.

Denote by $\delta_{1,k}$ the relative percentage increase in cost from the optimal solution x^1 to the kth best solution x^k, hence

$$\delta_{1,k} = \frac{y(x^k) - y(x^1)}{y(x^1)} \cdot 100 \, .$$

Also, let $x^{\max} \in \mathcal{X}_{AP}$ be the worst assignment in terms of costs, hence $x^{\max} := \arg\max\{cx \, : \, x \in \mathcal{X}_{AP}\}$. Notice that, $x^{\max}$ can be found running algorithm `K-AP` with $K = 1$ on a modified AP in which all cost entries, c_{ij}, are substituted by $\tilde{c}_{ij} = maxCost - c_{ij}$, where $maxCost$ is the maximal cost entry for the original AP.

In Table 4.1 is provided some statistics for the first group of test instances. For each problem size, I display the average of the optimal solution values, the average of the relative percentage increase in cost for $k = 50$ and $k = 100$, and the average worst solution value. The relative percentage increase in cost tends to decrease

[3] The correction is due to Pascoal [122].

size	ave. $y(x^1)$	ave. $\delta_{1,50}$	ave. $\delta_{1,100}$	ave. $y(x^{\max})$
50	125.2	2.40	3.05	4811
100	112.7	0.81	1.00	9787.5
150	87.5	0.47	0.47	14759.2
200	72	0	0	19726.3
250	52.8	0.19	0.19	24697.8
300	37.4	0	0	29661.7

Table 4.1: Statistics for the test instances from Pascoal et al. [123].

with the dimension, so an increasing number of the $n!$ feasible solutions becomes alternative optima. Since the cost of the worst assignment is much higher than the optimal solution value, the importance of choosing an optimal or near-optimal solution, by ranking, is justified.

For the second group of test instances, the focus in on ranking the $K = 100$ best assignments for somewhat larger complete bipartite networks of size $n \in \{100, 200, \ldots, 800\}$ and cost randomly drawn in $\{1, \ldots, 100\}$. These instances were taken from the OR-library[4] and were first used in Beasley [8]. Only one instance of each problem size is solved. Analysing the second group of test instances, again show the number of alternative optima to be high. In fact, only for the instance of size 100, $y(x^1) \neq y(x^{100})$.

4.3.3 Test results

Table 4.2 displays, for each possible problem size of the first group of test instances, the average CPU times (in seconds) for ranking the 100 best assignments with the algorithms `K-AP_SuccSP` and `K-AP_Reopt`. Also displayed is the ratio between these CPU times. Because of the higher time complexity for `K-AP_SuccSP`, it is not surprising that this ratio is increasing with dimension.

The effect of updating the dual variables due to (4.7) is displayed numerically in Table 4.3 giving results for the algorithms `K-AP_NoDualUp` and `K-AP_Reopt`. It is evident that updating the dual variables improves the algorithm. This is a result of the present implementation of the function `FindAugmentPath` using the specialized Dijkstra's method in [86]. Updating the dual variables means decreasing the reduced costs corresponding to the unassigned column j_i. Therefore this column enters the list of indices to scan next faster than if the reduced costs corresponding to column j_i had not been decreased. Since all reduced costs are non-negative, this leads to a faster termination of the function `FindAugmentPath`.

In Figure 4.5 on page 45, for all five algorithms of time complexity $\mathcal{O}(Kn^3)$, I display the CPU time against K for the largest problem size ($n = 250$) from the first group of test instances on which all algorithms was capable of ranking

[4] `http://people.brunel.ac.uk/~mastjjb/jeb/info.html` (see also [9]).

size	ave. CPU K-AP_SuccSP	K-AP_Reopt	ratio
50	0.24	0.06	4.09
100	5.34	0.67	7.96
150	22.62	2.30	9.85
200	61.97	5.61	11.04
250	126.01	10.46	12.04
300	175.09	13.92	12.58

Table 4.2: Average CPU times against size for Pascoal et al. [123] instances with $K = 100$ using algorithms `K-AP_SuccSP` and `K-AP_Reopt`.

size	ave. CPU K-AP_NoDualUp	K-AP_Reopt	ratio
50	0.06	0.06	0.97
100	0.71	0.67	1.06
150	2.52	2.30	1.10
200	6.21	5.61	1.11
250	11.71	10.46	1.12
300	15.45	13.92	1.11

Table 4.3: Average CPU times against size for Pascoal et al. [123] instances with $K = 100$ using algorithms `K-AP_NoDualUp` and `K-AP_Reopt`.

the 100 best assignments. The algorithm `K-AP_VMA_1` was unable to rank the 100 best assignments for any of the size $n = 300$ instances, because the upper bound limit of number of assignment nodes was reached. The problem sizes not displayed indicates similar results as shown in Figure 4.5.

The two algorithms `K-AP_CH` and `K-AP_VMA_1` are significantly slower than the remaining three algorithms. Obviously, comparing running times of algorithms implemented by different researchers must be performed with caution. The algorithms `K-AP_CH` and `K-AP_VMA_2` were both implemented by Przybylski [130] using for `K-AP_VMA_2` a better data structure than for `K-AP_CH`. However, the results still indicate that using a revised version of the branching technique due to Murty [106] outperforms the binary search tree algorithms due to Hamacher and Queyranne [68], Przybylski [130]. This result is substantiated by numerical tests in Pascoal et al. [123].

Since the two algorithms `K-AP_VMA_2` and `K-AP_Reopt` were indicated jointly by Table 4.3 and Figure 4.5 to be the two best algorithms, the remaining test results

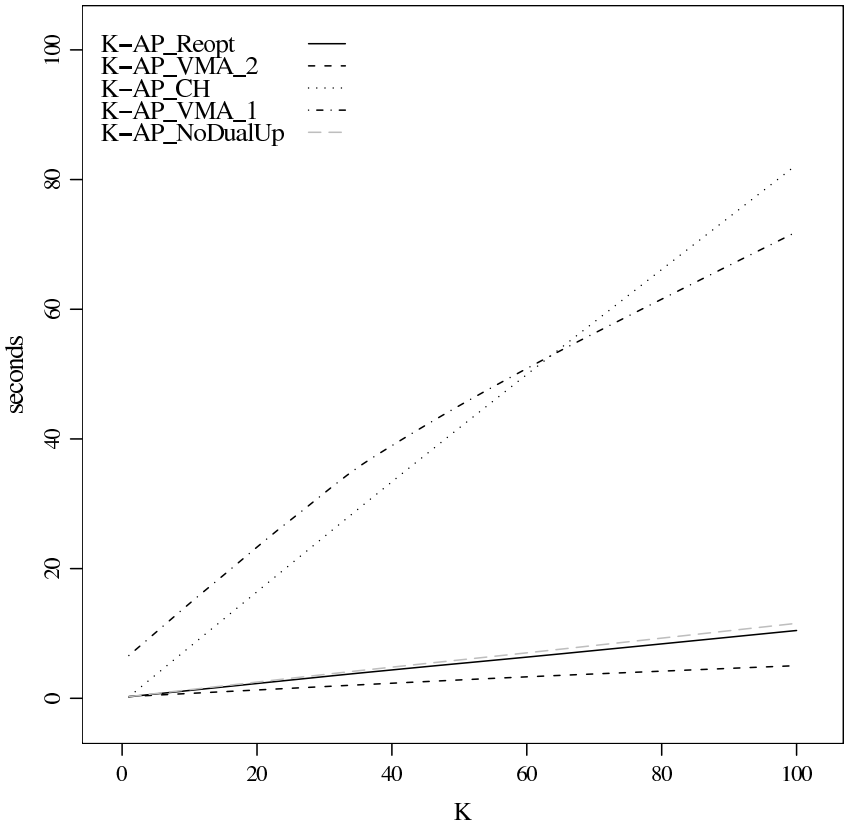


Figure 4.5: CPU time against K for Pascoal et al. [123] instances of size $n = 250$.

cover these two algorithms solely.

In Figures 4.6 and 4.7 is shown the CPU time against K for the individual problem sizes for the first and the second group of instances, respectively. The CPU time of both algorithms seems to increase linearly with the number of solutions to rank. According to these two figures, none of these two algorithms are, in their present implementaion, capable of outperforming the other. Remember though, in `K-AP_Reopt` the times to recalculate the optimal solution for a given subset taken out of the candidate set Φ is included. This is not the case for `K-AP_VMA_2`. The total running time for finding the 100 best solutions for all test instances are 709.50 seconds for `K-AP_VMA_2` and 589.22 seconds for `K-AP_Reopt`, respectively. Therefore, on average, the algorithm `K-AP_Reopt` performs approximately 20 per cent faster than `K-AP_VMA_2`.

For the first group of test instances, the CPU times for ranking $K = 50$ and $K = 100$ assignments with `K_AP_Reopt` are displayed against problem size in Figure 4.8(a) on page 48. The algorithm shows more than a linear growth in CPU time with increasing K. However, it is less than exponential growth, as can be seen in Figure 4.8(b) displaying logarithm of CPU times.

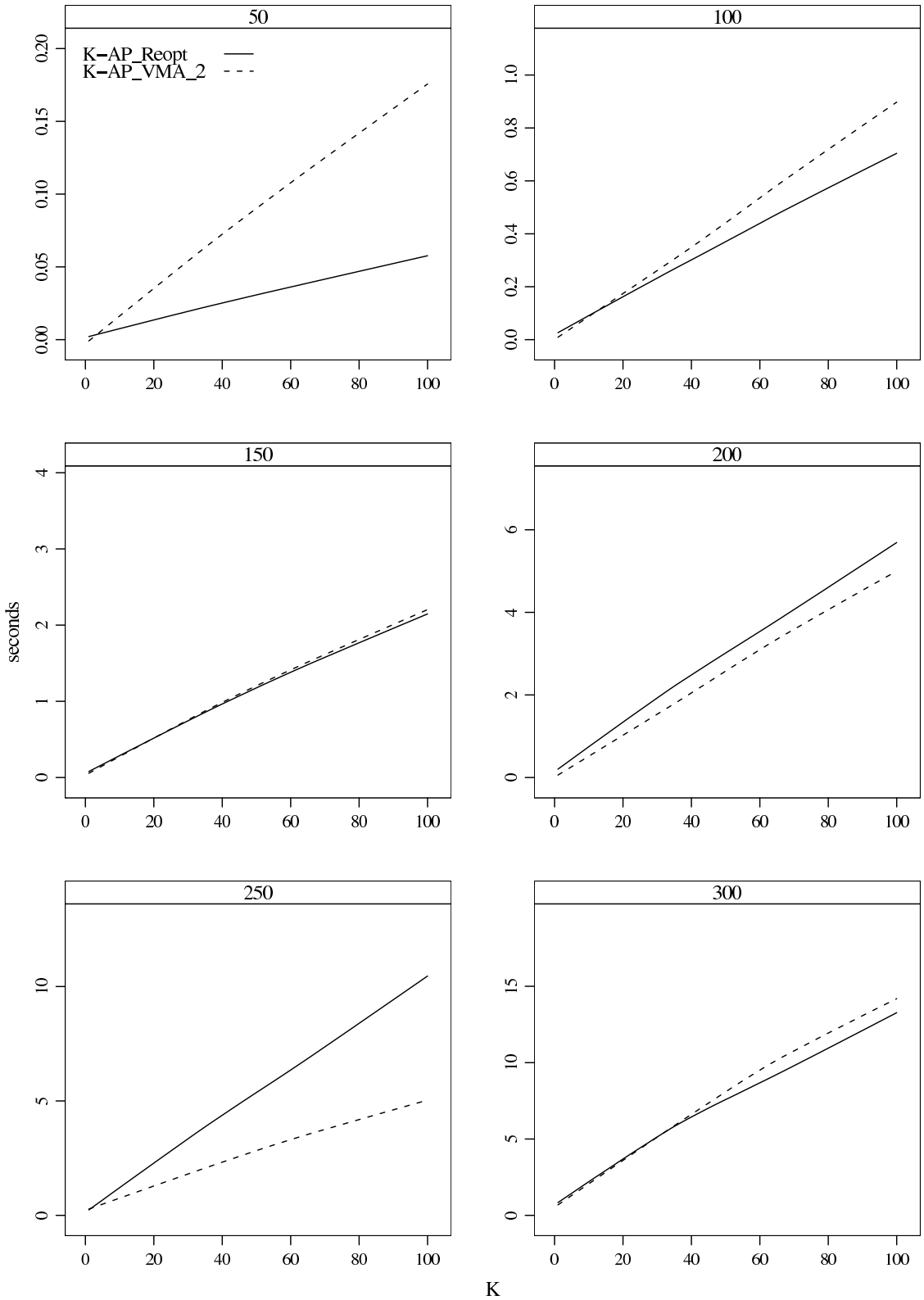


Figure 4.6: CPU time against K for Pascoal et al. [123] instances.

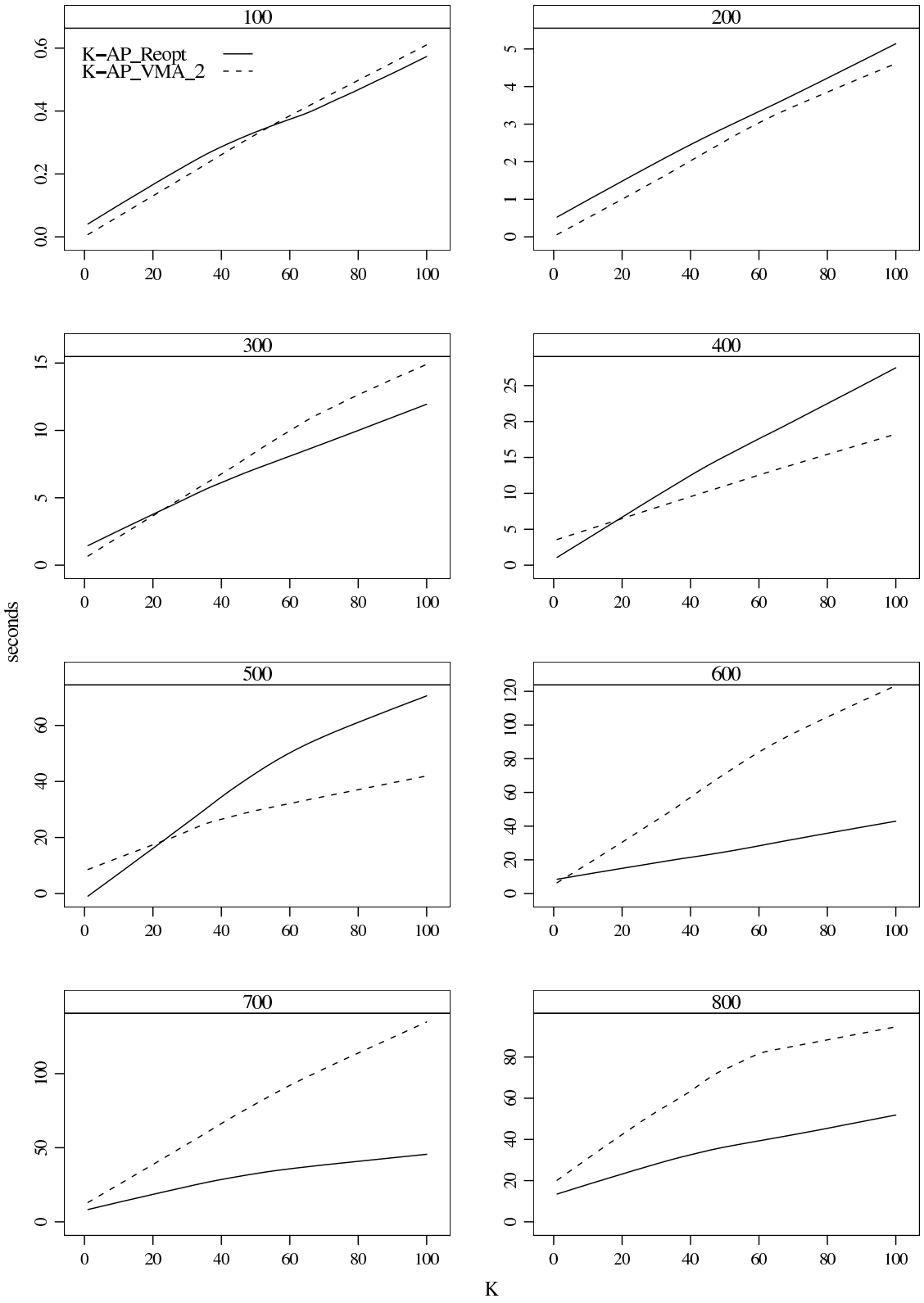
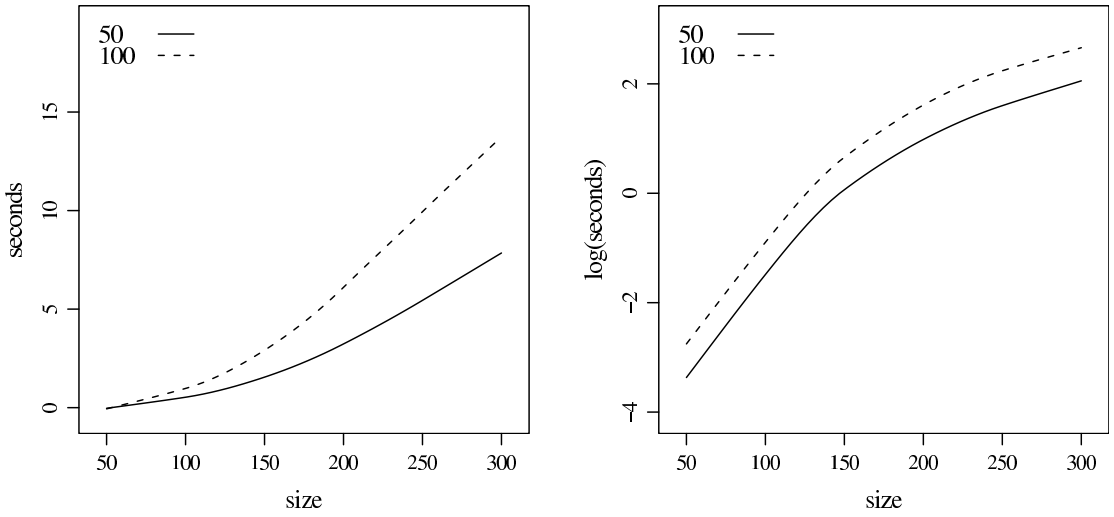


Figure 4.7: CPU time against K for Beasley [8] instances.



(a) CPU time

(b) Logarithm of CPU time

Figure 4.8: Average running time against size for Pascoal et al. [123] instances with $K = 50$ and $K = 100$.

Chapter 5

Ranking integer flows

The minimum cost integer flow problem (MCIF) is a relevant and well-studied generalization of the assignment problem. Let me for brevity merely refer to Section 2.1.1 and to the textbook by Ahuja et al. [2] for applications of network flow models. Finding the K best solutions of MCIF may provide a decision maker with vital information on a given problem and can even be interpreted as an important subroutine for other more complex problem classes.

Despite its importance, the problem on ranking integer (network) flows has, to the best of my knowledge, only been studied by Hamacher [66] and Hamacher and Hüsselman [67]. They developed an algorithm using the binary search tree branching technique (presented in Section 2.2) originally intended for combinatorial optimization problems on binary variables, Hamacher and Queyranne [68]. Utilizing an improved version of Dijkstra's shortest path method with time complexity $\mathcal{O}(m + n\log(n))$, the total running time of the suggested method is $\mathcal{O}(K(mn\log(n) + m^2))$.

In Chapter 4, it was indicated, through numerical tests, that a variant of the branching technique due to Murty [106] outperforms the binary search tree algorithm when ranking assignments. It seems natural to investigate if this observation holds true when these branching techniques are applied to find the K best solutions for the minimum cost integer flow problem. Therefore, I propose, in this chapter, a new algorithm for ranking integer flows in nondecreasing order of cost, relying on a modified version of the branching technique due to Lawler [95] which is a generalization of the branching technique by Murty [106]. Because of time limitations, I have not been able to implement the presented ideas, and hence the work in this chapter should not be considered complete. Obviously, implementation of the algorithm, generation of test instances and a thorough testing of the algorithm constitute the next phase of this project. Comparative runs against the algorithm in [67] should be performed.

The transportation problem (TP) is an important special instance of MCIF.

Again, ranking solutions of a TP may provide useful information to a decision maker and may also be helpful as subroutine for other problem classes, including multicriteria extensions of TP, as we shall see in Section 9.1.

Ranking algorithms for the transportation problem are, as far as I know, exclusively concentrated on generating the K best vertex solutions in the continuous relaxation of $\mathcal{X}_{TP}$ (denoted $\mathcal{P}_{TP}$). Such vertex solutions provide the necessary information when searching for an optimal solution to the fixed charge transportation problem, McKeown [103], Murty [107], and Sadagopan and Ravindran [140]. However, when searching for the K best transportation solutions, non-extreme integral solutions from $\mathcal{P}_{TP}$ has the same importance as vertex solutions. Therefore, it is required to extend the vertex ranking algorithms to include non-extreme integral solutions. I serve this goal by strengthening the algorithm for ranking integer flows presented in this chapter.

The remaining parts of this chapter are organized as follows. In Section 5.1, I give some preliminaries facilitating the description of the ranking algorithm and the theoretical results stated in Section 5.2. In Section 5.3, additional features to be included in the algorithm ranking solutions for the transportation problem are discussed and the suggested algorithm is illustrated by an example in Section 5.3.1.

5.1 Preliminaries

Recall from Section 2.1.1 that MCIF on a directed graph $G = (N, A)$ with $N := \{1, \ldots, n\}$ and $m := |A|$, is to minimize a single criterion cx over the feasible set $\mathcal{X}_{flow}$, where

$$\mathcal{X}_{flow} = \Bigg\{ x \; : \sum_{\{j:(i,j)\in A\}} x_{ij} - \sum_{\{j:(j,i)\in A\}} x_{ji} = b_i \quad \forall\, i \in N, \\ [l_{ij} \le x_{ij} \le u_{ij} \;\wedge\; x_{ij} \in \mathbb{Z}],\ \forall\,(i,j) \in A \Bigg\}. \quad (5.1)$$

Denote by a *pseudoflow* a function $x : A \to \mathbb{R}$ satisfying only the capacity constraints, and not neccesarily the flow conservation constraints from (5.1). The definitions of node potentials π and reduced arc costs c^{π}_{ij}, and the following equivalent optimality results for MCIF prove useful in the development of the algorithm for ranking integer flows, see e.g. Ahuja et al. [2].

Theorem 5.1 *For a minimum cost flow problem, a feasible primal solution x and a set of node potentials π are optimal if and only if the following* reduced cost optimality conditions *are fulfilled.*

$$c^{\pi}_{ij} \ge 0 \text{ for every arc } (i,j) \text{ in } G^{\pi}(x) \quad (5.2)$$

Theorem 5.2 *For a minimum cost flow problem, a feasible primal solution x and a set of node potentials π are optimal if and only if the reduced costs c^π and the flow satisfy the following* complementary slackness optimality conditions *for every arc $(i,j) \in A$.*

$$\begin{cases} c_{ij}^\pi > 0 \;\; \Rightarrow \;\; x_{ij} = l_{ij} \\ l_{ij} < x_{ij} < u_{ij} \;\; \Rightarrow \;\; c_{ij}^\pi = 0 \\ c_{ij}^\pi < 0 \;\; \Rightarrow \;\; x_{ij} = u_{ij} \end{cases} \tag{5.3}$$

The following two lemmas regarding pseudoflows are also well-known, [2].

Lemma 5.3 *Suppose that a pseudoflow x and a set of node potentials π satisfy the reduced cost optimality conditions of equation* (5.2)*. Consider the residual network $G^\pi(x)$. Let d denote the vector of shortest path distances from some node s to all other nodes in $G^\pi(x)$. Then the pseudoflow x also satisfies the reduced cost optimality conditions with respect to node potentials $\pi' = \pi - d$.*

Lemma 5.4 *Suppose that a pseudoflow x satisfies the reduced cost optimality conditions and suppose that x' is obtained from x by sending flow along a shortest path P between two specified nodes in $G^\pi(x)$, hence $x' = x \oplus P$. Then x' also satisfies the reduced cost optimality conditions.*

5.2 Ranking integer flows

Consider the problem of ranking the first K integer flows in nondecreasing order of cost, i.e. finding the K best solutions $x^1, \ldots, x^K$ satisfying

1. $y\left(x^i\right) \leq y\left(x^{i+1}\right),\ i = 1, \ldots, K-1$
2. $y\left(x^K\right) \leq y\left(x\right),\ \forall x \in \mathcal{X}_{flow} \setminus \left\{x^1, \ldots, x^K\right\}$

where $y(x)$ is the cost (or the objective function value) of a flow x.

As in the ranking algorithm for AP a specific branching technique and a specific solution method are needed. These concepts are discussed separately in the subsequent sections.

5.2.1 Branching technique

Assume that the capacity constraints of $\mathcal{X}_{flow}$ are maintained updated during the algorithm. A branching technique inspired by the one in the previous chapter is utilized. Given the optimal solution x^1, the set $\mathcal{X}_{flow} \setminus \{x^1\}$ is partitioned into smaller subsets in lexicographical increasing index-order.

Definition 5.5 The arc (i,j) is lexicographically smaller than the arc (p,q) if the following holds.

$$(i,j) <^{\text{lex}} (p,q) \;\Leftrightarrow\; (i<p) \;\vee\; (i=p \;\wedge\; j<q)\;.$$

For a given arc $(p,q) \in A$, let

$$I^{(p,q)} := \left\{(i,j) \;:\; (i,j) <^{\text{lex}} (p,q)\right\}$$

be all variables lexicographically smaller than (p,q). The set $\mathcal{X}_{flow} \setminus \{x^1\}$ is partitioned into the disjoint subsets $\mathcal{X}^{pq}$, for all arcs $(p,q) \in A$, where

$$\mathcal{X}^{pq} := \left\{x \in \mathcal{X}_{flow} \;:\; x_{ij} := x^1_{ij},\; \forall\,(i,j) \in I^{(p,q)} \;\wedge\; x_{pq} \neq x^1_{pq}\right\}\;. \tag{5.4}$$

In $\mathcal{X}^{pq}$, the variable x_{pq} is prevented from having the same value as in x^1 and any lexicographically smaller variable from x^1 is *fixed* to the same value in all flows in $\mathcal{X}^{pq}$. Variable x_{ij} is fixed by setting $l_{ij} = u_{ij} = x_{ij}$. The set $\mathcal{X}^{pq}$ is subdivided into the following two disjoint subsets, representing a *downward* and an *upward* branch.

$$\begin{aligned}\mathcal{X}^{pq} &= {}_d\mathcal{X}^{pq} \cup {}_u\mathcal{X}^{pq} \\ &:= \left\{x \in \mathcal{X}^{pq} \;:\; x_{pq} \leq x^1_{pq} - 1\right\} \cup \left\{x \in \mathcal{X}^{pq} \;:\; x_{pq} \geq x^1_{pq} + 1\right\}\end{aligned} \tag{5.5}$$

In the downward branch ${}_d\mathcal{X}^{pq}$, the upper bound u_{pq} is set to $x^1_{pq} - 1$, and in the upward branch ${}_u\mathcal{X}^{pq}$, the lower bound l_{pq} is set to $x^1_{pq} + 1$. Notice that the information gained by the additional constraint on variable x_{pq} may be propagated to other variables by adjusting their bounds, which may again initiate further bound propagation. It must be investigated, through numerical tests, to which extent such propagation could be utilized.

Let $MCIF(\mathcal{X})$ denote the reduced MCIF defined by subset $\mathcal{X}$. Clearly, the second best solution x^2 can be found by using a solution method to find the optimal solution to all the reduced problems. Moreover, the branching technique can be applied recursively to subsets ${}_d\mathcal{X}^{pq} \subset \mathcal{X}_{flow}$ and ${}_u\mathcal{X}^{pq} \subset \mathcal{X}_{flow}$.

The pseudo code for the ranking algorithm, named `K-MCIF`, is shown in Figure 5.1. The algorithm implicitly maintains a candidate set Φ of pairs $(\tilde{x}, \widetilde{\mathcal{X}})$, where $\tilde{x}$ is the optimal flow in (sub)set $\widetilde{\mathcal{X}}$. Assuming that the first $k-1$ solutions $x^1, \ldots, x^{k-1}$ have been found, solution x^k is found by selecting and removing the pair $(\hat{x}, \widehat{\mathcal{X}})$ containing the flow with minimum cost in the candidate set (lines 5–7). Next, the branching technique is used to partition $\widehat{\mathcal{X}}$, possibly obtaining new pairs that are added to the candidate set (lines 8–13). The function `FindOptimal`, returning the optimal solution in a given subset, is discussed in Section 5.2.2. The validity of the ranking algorithm is established with the following theorem.

```
1  procedure K-MCIF()
2      x^1 := MCIF();
3      Φ := {(x^1, X_flow)};
4      for (k := 1 to K) do
5          (x̂, X̂) := arg min{y(x̃) : (x̃, X̃) ∈ Φ};
6          if ((x̂, X̂) = null) then STOP; else OUTPUT x^k := x̂;
7          Φ := Φ \ {(x̂, X̂)};
8          for ((p, q) ∈ A) do
9              _dx̂* := FindOptimal(_dX̂^pq);
10             if (y(_dx̂*) < ∞) then Φ := Φ ∪ {(_dx̂*, _dX̂^pq)};
11             _ux̂* := FindOptimal(_uX̂^pq);
12             if (y(_ux̂*) < ∞) then Φ := Φ ∪ {(_ux̂*, _uX̂^pq)};
13         end for
14     end for
15 end procedure
```

Figure 5.1: The algorithm for ranking integer flows.

Theorem 5.6 *The algorithm K-MCIF in Figure 5.1, finds the K best solutions to the minimum cost integer flow problem.*

Proof. At any iteration of the algorithm, the integer flow with minimum cost in the candidate set Φ is chosen. Therefore, all I need to show is that, when the $k-1$ best integer flows $x^1, \ldots, x^{k-1}$ have been found, the union of the subsets of feasible solutions in Φ is exactly the feasible solutions in $\mathcal{X}_{flow} \setminus \{x^1, \ldots, x^{k-1}\}$.

Consider $\widehat{\mathcal{X}}$ with optimal integer flow $\hat{x}$. By definition, the sets $\widehat{\mathcal{X}}^{pq}$ are disjoint and $\widehat{\mathcal{X}}^{pq} \subseteq \widehat{\mathcal{X}}$. Furthermore, $\hat{x}$ is excluded from all these subsets, so

$$\bigcup_{(p,q)\in A} \widehat{\mathcal{X}}^{pq} \subseteq \widehat{\mathcal{X}} \setminus \{\hat{x}\} \ . \tag{5.6}$$

Choose an arbitrary $\tilde{x} \in \widehat{\mathcal{X}} \setminus \{\hat{x}\}$. In particular $l_{ij} \leq \tilde{x}_{ij} \leq u_{ij}$. Let (p,q) be the lexicographically smallest index having $\tilde{x}_{pq} \neq \hat{x}_{pq}$. Two cases are possible:

(i): $l_{pq} \leq \tilde{x}_{pq} < \hat{x}_{pq} \leq u_{pq}$. Then $\tilde{x} \in {}_d\widehat{\mathcal{X}}^{pq}$.

(ii): $l_{pq} \leq \hat{x}_{pq} < \tilde{x}_{pq} \leq u_{pq}$. Then $\tilde{x} \in {}_u\widehat{\mathcal{X}}^{pq}$.

In either case, $\tilde{x} \in \bigcup_{(p,q)\in A} \widehat{\mathcal{X}}^{pq}$, so

$$\bigcup_{(p,q)\in A} \widehat{\mathcal{X}}^{pq} \supseteq \widehat{\mathcal{X}} \setminus \{\hat{x}\} \ . \tag{5.7}$$

Combining (5.6) and (5.7), the result follows, since, in iteration k, the branching technique is shown to exclude x^k and only x^k from further consideration. □

Notice that, in practice, it is not always necessary to consider all subsets in the partitioning of $\widehat{\mathcal{X}}$. The few rules presented in the following proposition are easily established.

Proposition 5.7 *In* $\widehat{\mathcal{X}}$,

(a) *if* $\hat{x}_{pq} = l_{pq}$, *the downward branch* ${}_d\widehat{\mathcal{X}}^{pq} = \emptyset$.

(b) *if* $\hat{x}_{pq} = u_{pq}$, *the upward branch* ${}_u\widehat{\mathcal{X}}^{pq} = \emptyset$.

(c) *branching on a variable, that would update the lower and upper arc bounds* $\{l_{ij}\}_{ij}$ *and* $\{u_{ij}\}_{ij}$ *to fulfil either one of the following relations, is infeasible.*

$$\exists i \in N \; : \sum_{\{j:(i,j)\in A\}} u_{ij} - \sum_{\{j:(j,i)\in A\}} l_{ji} < b_i$$

$$\exists i \in N \; : \sum_{\{j:(i,j)\in A\}} l_{ij} - \sum_{\{j:(j,i)\in A\}} u_{ji} > b_i$$

5.2.2 Solution method

This section discusses the solution method for a given subset of feasible integer flows. I distinguish between the downward branch and the upward branch and start by discussing the downward branch.

Assume that partition $\widehat{\mathcal{X}}$ has optimal primal solution $\hat{x}$ and corresponding optimal node potentials $\hat{\pi}$. Furthermore, suppose that all variables in $I^{\text{fix}(\widehat{\mathcal{X}})} := \{x_{i_1,j_1}, \ldots, x_{i_e,j_e}\}$ have been fixed to a specific value in previous partitions, i.e. all feasible flows of $\widehat{\mathcal{X}}$ must contain these values. Therefore, assuming ${}_d\widehat{\mathcal{X}}^{pq}$ is non-empty, $MCIF({}_d\widehat{\mathcal{X}}^{pq})$ corresponds to a reduced MCIF with n nodes and no more than $m - e$ arcs with the following attributes.

1. The fixed units are subtracted from the original amount of supply and demand at all corresponding nodes, hence

$$\hat{b}_i = b_i - \sum_{\{j\,:\,(j,i)\in A\cap(I^{(p,q)}\cup I^{\text{fix}(\widehat{\mathcal{X}})})\}} \hat{x}_{ji} + \sum_{\{j\,:\,(i,j)\in A\cap(I^{(p,q)}\cup I^{\text{fix}(\widehat{\mathcal{X}})})\}} \hat{x}_{ij}$$

2. Arcs $(i,j) \in A \cap (I^{(p,q)} \cup I^{\text{fix}(\widehat{\mathcal{X}})})$ are deleted.

3. In the lower and upper bounds of $\widehat{\mathcal{X}}$, l_{ij} and u_{ij}, are included any other arc constraints obtained by previous branchings on x_{ij}.

4. A constraint of $x_{pq} \leq \hat{x}_{pq} - 1$ is present. This constraint says that at least one unit allocated from node p to node q in $\hat{x}$ must be taken away.

Define the temporary partial primal solution ${}_dx(p,q)$ from $\hat{x}$ by subtracting one unit from the (p,q) allocation, hence

$$ {}_dx(p,q)_{ij} := \hat{x}_{ij},\ \forall\,(i,j) \neq (p,q)\,, \quad {}_dx(p,q)_{pq} := \hat{x}_{pq} - 1\,. \tag{5.8} $$

Consider the residual network to ${}_dx(p,q)$, $G^{\pi}({}_dx(p,q)) = (N, A_f \cup A_b)$.

$$ \begin{aligned} A_f &= \{(i,j)\ :\ (i,j) \in A\ \wedge\ l_{ij} \leq {}_dx(p,q)_{ij} < u_{ij}\}, \\ A_b &= \{(j,i)\ :\ (i,j) \in A\ \wedge\ l_{ij} < {}_dx(p,q)_{ij} \leq u_{ij}\} \end{aligned} $$

Notice that no arc $(i,j) \in I^{(p,q)} \cup I^{\mathrm{fix}(\widehat{\mathcal{X}})}$ is present in either A_f or A_b since, for such an arc, $l_{ij} = {}_dx(p,q)_{ij} = u_{ij}$. Each forward arc (i,j) in A_f is assigned reduced cost c^{π}_{ij} and each backward arc (j,i) in A_b is assigned cost $c^{\pi}_{ji} = -c^{\pi}_{ij}$. Since $\hat{x}$ is optimal to $MCIF(\widehat{\mathcal{X}})$, all costs in this residual network are positive due to the complementary slackness optimality conditions of (5.3).

The optimal solution $\hat{x}$ to $MCIF(\widehat{\mathcal{X}})$ remains feasible for $MCIF({}_d\widehat{\mathcal{X}}^{pq})$ except that the variable x_{pq} breaks its new upper bound limit by exactly one unit. Hence, all variables are *in-kilter* except x_{pq}, which is *out-of-kilter* with a kilter status of one unit. Exploiting the same techniques as the *out-of-kilter method*, the optimal solution for $MCIF({}_d\widehat{\mathcal{X}}^{pq})$ can be found by sending along a single shortest path from p to q in $G^{\pi}({}_dx(p,q))$ one unit of flow. Below is a more detailed description of the updating procedure.

Because of the constraint $x_{pq} \leq \hat{x}_{pq} - 1$, there cannot be send any more flow from p to q in ${}_d\widehat{\mathcal{X}}^{pq}$. Furthermore, it can never be optimal to take back more than one unit along x_{pq}, since the reduced cost of the shortest path from p to q in $G^{\pi}({}_dx(p,q))$ plus c^{π}_{pq} is non-negative due to optimality of $\hat{x}$ in $\widehat{\mathcal{X}}$. Recall that, in Section 2.1.1, it was assumed that, between each pair of nodes i and j, the arcs (i,j) and (j,i) are not allowed to be in A simultaneously, ensuring that no parallel arcs exist in the residual graph. Therefore, in $G^{\pi}({}_dx(p,q))$, arc $(p,q) \notin A_f$, and arc (q,p) can be disregarded from A_b when considering ${}_d\widehat{\mathcal{X}}^{pq}$.

Let d denote the shortest path distances from node p to all other nodes in $G^{\pi}({}_dx(p,q))$. If a shortest path to some node t does not exist, the corresponding shortest path distance is set to a large positive number M. Notice, that this corresponds to adding artificial arcs with cost M to the original network. Obviously, such artificial arcs will never be part of any optimal solution. Also, let P denote a shortest directed path from node p to node q sending one unit of flow. If no such path P with cost less than M exists, the subproblem of the current branch is infeasible.

The vector of dual node potentials π is updated according to the scheme in Lemma 5.3,

$$ {}_d\hat{\pi}^{*} := \hat{\pi} - d\,, \tag{5.9} $$

```
1 procedure FindOptimal(${}_d\widehat{\mathcal{X}}^{pq}$)
2     ${}_dx(p,q)$ := ObtainTemporary($\hat{x}$);
3     $G^\pi({}_dx(p,q))$ := BuildResNetwork(${}_dx(p,q), \pi$);
4     $d$ := FindShortestPathDistances($G^\pi({}_dx(p,q))$);
5     $P$ := FindAugmentPath($G^\pi({}_dx(p,q))$);
6     ${}_d\hat{x}^*$ := AugmentSolution(${}_dx(p,q), P$);
7     ${}_d\hat{\pi}^*$ := ModifyDual($\hat{\pi}, d$);
8   end procedure
```

Figure 5.2: Finding the optimal solution for a downward branch.

and the primal (partial) solution is updated according to Lemma 5.4.

$$
\begin{aligned}
{}_d\hat{x}^* &:= {}_dx(p,q) \oplus P \\
&\Updownarrow \\
{}_d\hat{x}^*_{ij} &:= \begin{cases} {}_dx(p,q)_{ij} + 1 & \text{if } (i,j) \in P \cap A_f \\ {}_dx(p,q)_{ij} - 1 & \text{if } (j,i) \in P \cap A_b \\ {}_dx(p,q)_{ij} & \text{otherwise} \end{cases}
\end{aligned}
\tag{5.10}
$$

Exploiting the general theory presented in Section 2.1.1 and in Section 5.1, the following result is obtained.

Theorem 5.8 *${}_d\hat{x}^*$ is the optimal primal solution to $MCIF({}_d\widehat{\mathcal{X}})$ and ${}_d\hat{\pi}^*$ is the optimal dual variables. The optimal cost value is $y({}_d\hat{x}^*) := y(\hat{x}) + c(P) - c_{pq} = y(\hat{x}) + c^\pi(P) - c^\pi_{pq}$.*

A pseudo-code for the reoptimization algorithm to the downward branch is given in Figure 5.2. First, the temporary partial flow ${}_dx(p,q)$ is obtained from the previous solution $\hat{x}$. Second, the residual network is built in which a shortest path tree and a shortest augmenting path from p to q are found. Finally, the corresponding primal and dual optimal solutions are updated.

Next, the upward branch is briefly discussed. Most of the results and developments from the downward branch applies without changes. In the upward branch, ${}_u\widehat{\mathcal{X}}^{pq}$, the optimal solution $\hat{x}$ to $MCIF(\widehat{\mathcal{X}})$ remains feasible for $MCIF({}_u\widehat{\mathcal{X}}^{pq})$, except that the variable x_{pq} breaks its new lower bound limit by exactly one unit, since a new constraint $x_{pq} \geq \hat{x}_{pq} + 1$ is present. This constraint says that one more unit allocation between node p and node q must be added to $\hat{x}$. Therefore, all variables except x_{pq} are in-kilter, and x_{pq} is out-of-kilter with a kilter status of one unit. Again this is exploited by addition of one unit along a shortest path from node q to node p in the residual graph $G^\pi({}_ux(p,q))$ to the temporary flow ${}_ux(p,q)$ defined from $\hat{x}$ by the following relations.

$$
{}_ux(p,q)_{ij} := \hat{x}_{ij},\ \forall\,(i,j) \neq (p,q)\,, \quad {}_ux(p,q)_{pq} := \hat{x}_{pq} + 1\,. \tag{5.11}
$$

In the residual network $G^{\pi}({}_{u}x(p,q))$, let d denote the shortest path distances from node q to all other nodes. Also, let P denote a shortest directed path from node q to node p sending one unit of flow. The dual and primal solutions are updated as in (5.9) and (5.10), yielding ${}_{u}\hat{\pi}^{*} := \hat{\pi} - d$ and ${}_{u}\hat{x}^{*} := {}_{u}x(p,q) \oplus P$, respectively. Similar to Theorem 5.8, the next result follows.

Theorem 5.9 *${}_{u}\hat{x}^{*}$ is the optimal primal solution to $MCIF({}_{u}\widehat{\mathcal{X}})$ and ${}_{u}\hat{\pi}^{*}$ is the optimal dual variables. The optimal cost value is $y({}_{u}\hat{x}^{*}) := y(\hat{x}) + c(P) + c_{pq} = y(\hat{x}) + c^{\pi}(P) + c^{\pi}_{pq}$.*

A procedure similar to `FindOptimal` in Figure 5.2 applies for the upward branch. Using an improved Dijkstra's method to build the shortest path tree, function `FindOptimal` runs in $\mathcal{O}(m+n\log(n))$, see e.g. [144]. Since the function `FindOptimal` is employed to no more than $2m$ branches in each iteration, the following time complexity of `K-MCIF` is obtained, which equals the current best complexity obtained in [66].

Theorem 5.10 *The algorithm for ranking integer flows, `K-MCIF` in Figure 5.1, has time complexity $\mathcal{O}(K(mn\log(n) + m^2))$.*

5.3 Ranking transportation solutions

Recall that the transportation problem (TP) is a special instance of the minimum cost integer flow problem, and that TP can be displayed by the bipartite directed graph $G = (W \cup V, A)$. Refer to a feasible solution x to a TP as a *transportation solution*. Obviously, the algorithm for ranking integer flows, `K-MCIF`, and the results presented in the previous section also apply for ranking transportation solutions. However, utilizing specialized properties for TP, additional rules can be used to improve the performance of the algorithm for ranking transportation solutions.

Let $TP(\mathcal{X})$ denote the reduced TP defined by subset $\mathcal{X}$ and let $\hat{x}$ be the optimal solution to $TP(\widehat{\mathcal{X}})$. Due to the bipartite structure of TP, the branching technique from Section 5.2.1 can be strengthened by branching only on arcs (p,q) with $\hat{x}_{pq} > l_{pq}$. Therefore, in this section, $I^{(p,q)}$ is redefined accordingly to be all variables strictly larger than their lower bound and lexicographically smaller than (p,q).

$$\widetilde{I}^{(p,q)} := \left\{(i,j) \ : \ \hat{x}_{ij} > l_{ij} \ \wedge \ (i,j) <^{\text{lex}} (p,q)\right\}$$

The set $\widehat{\mathcal{X}} \setminus \{\hat{x}\}$ is split into the disjoint subsets $\widehat{\mathcal{X}}^{pq}$ for (p,q) with $\hat{x}_{pq} > l_{pq}$ using $\widetilde{I}^{(p,q)}$ instead of $I^{(p,q)}$ in (5.4). Also, $\widehat{\mathcal{X}}^{pq}$ is again split into the sets ${}_{d}\widehat{\mathcal{X}}^{pq}$ and ${}_{u}\widehat{\mathcal{X}}^{pq}$, like in (5.5). Let `K-TP` denote the branching algorithm for ranking transportation solutions which is similar to `K-MCIF`, except that the condition in the **for** statement in line 8 of Figure 5.1 is substituted with $(p,q) \in A \ : \ \hat{x}_{pq} > l_{pq}$. The validity

of the strengthened branching technique used in K-TP is obtained by the following proposition.

Proposition 5.11 *Consider $\widehat{\mathcal{X}}$ with optimal transportation solution $\hat{x}$. Then the following holds true.*

$$\bigcup_{(p,q)\,:\,\hat{x}_{pq}>l_{pq}} \widehat{\mathcal{X}}^{pq} = \widehat{\mathcal{X}} \setminus \{\hat{x}\} \ .$$

Proof. By definition, the sets $\widehat{\mathcal{X}}^{pq}$ are disjoint and $\widehat{\mathcal{X}}^{pq} \subseteq \widehat{\mathcal{X}}$. Furthermore, $\hat{x}$ is excluded from all these subsets, so

$$\bigcup_{(p,q)\,:\,\hat{x}_{pq}>l_{pq}} \widehat{\mathcal{X}}^{pq} \subseteq \widehat{\mathcal{X}} \setminus \{\hat{x}\} \ .$$

Choose an arbitrary $\tilde{x} \in \widehat{\mathcal{X}} \setminus \{\hat{x}\}$. In particular $l_{ij} \leq \tilde{x}_{ij} \leq u_{ij}$. Let (p,q) be the lexicographically smallest index having $\tilde{x}_{pq} \neq \hat{x}_{pq}$. Three cases are now possible:

(i): $l_{pq} \leq \tilde{x}_{pq} < \hat{x}_{pq} \leq u_{pq}$. Then $\tilde{x} \in {}_d\widehat{\mathcal{X}}^{pq}$.

(ii): $l_{pq} < \hat{x}_{pq} < \tilde{x}_{pq} \leq u_{pq}$. Then $\tilde{x} \in {}_u\widehat{\mathcal{X}}^{pq}$.

(iii): $l_{pq} = \hat{x}_{pq} < \tilde{x}_{pq} \leq u_{pq}$. No branching is performed on x_{pq}.

In case (iii), let (i,j) be the lexicographically second smallest index having $\tilde{x}_{ij} \neq \hat{x}_{ij}$. The same three cases as above apply, and in cases (i) and (ii) $\tilde{x}$ belongs to ${}_d\widehat{\mathcal{X}}^{ij}$ and ${}_u\widehat{\mathcal{X}}^{ij}$, respectively. In case (iii), the search for an index fulfilling either (i) or (ii) continues. Ultimately, instance (i) must be found, proving that $\tilde{x} \in \bigcup_{(p,q)\,:\,\hat{x}_{pq}>l_{pq}} \widehat{\mathcal{X}}^{pq}$, so

$$\bigcup_{(p,q)\,:\,\hat{x}_{pq}>l_{pq}} \widehat{\mathcal{X}}^{pq} \supseteq \widehat{\mathcal{X}} \setminus \{\hat{x}\} \ ,$$

which yields the desired result. □

Notice that the above result holds, in particular, because of the bipartite nature of TP. If $\hat{x}$ is optimal in $\widehat{\mathcal{X}}$, and $\hat{x}_{ij} = l_{ij}$, $\forall\,(i,j) \in A$, then $\widehat{\mathcal{X}} = \{\hat{x}\}$. This is not necessarily true for the minimum cost integer flow problem as can be seen in the following example. Therefore, branching is performed on a larger set of arcs in Section 5.2.1.

Example 5.12 Consider the MCIF of Figure 5.3 with optimal solution $x^1 := \{x_{ij} = 1,\ \forall\,(i,j) \in A\}$. Clearly, $x_{ij} = l_{ij}$, $\forall\,(i,j) \in A$, but $\mathcal{X}_{flow} = \{x^1, x^2\}$ with $x^2 := \{x_{ij} = 2,\ \forall\,(i,j) \in A\}$.

A few more rules speed up the performance of the algorithm K-TP.

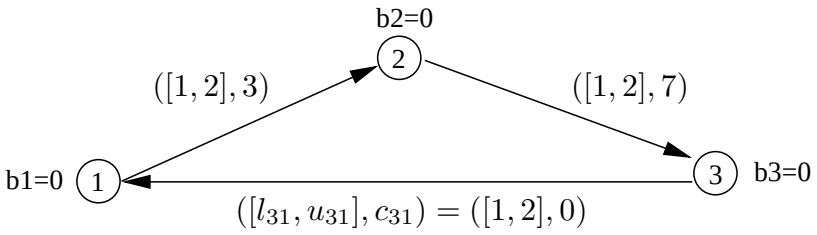


Figure 5.3: The graph of a minimum cost flow problem.

Proposition 5.13 *Let $\hat{x}$ be the optimal solution to $TP(\widehat{\mathcal{X}})$. Then the following holds.*

(a): $\forall p \in W$, *for* $q = \max\{j \in V \ : \ \hat{x}_{pj} > l_{pj}\}$ *the upward branch* ${}_u\widehat{\mathcal{X}}^{pq} = \emptyset$.

(b): $\forall q \in V$, *for* $p = \max\{i \in W \ : \ \hat{x}_{iq} > l_{iq}\}$ *the upward branch* ${}_u\widehat{\mathcal{X}}^{pq} = \emptyset$.

Proof. (a): Assume that upward branching on x_{pq} is feasible, so $x_{pq} > \hat{x}_{pq}$. For this to be true, there must exist a $j \neq q \in V$, so that

$$(p, j) \notin \widetilde{I}^{(p,q)} \cup I^{\mathrm{fix}(\widehat{\mathcal{X}})} \ \wedge \ x_{pj} < \hat{x}_{pj} \ .$$

However, this is impossible, since for $j < q$, $\hat{x}_{pj} = l_{pj}$ because $(p, j) \notin \widetilde{I}^{(p,q)}$, and for $j > q$, $\hat{x}_{pj} = l_{pj}$, because $q = \max\{j \in V \ : \ \hat{x}_{pj} > l_{pj}\}$. (b): Can be proved similarly. □

Proposition 5.14 *Let $\hat{x}$ be the optimal solution to $TP(\widehat{\mathcal{X}})$. Then, for $x_{pq} = \mathrm{lex\,max}\{(i, j) \ : \ \hat{x}_{ij} > l_{ij}\}$ the branch $\widehat{\mathcal{X}}^{pq} = {}_u\widehat{\mathcal{X}}^{pq} \cup {}_d\widehat{\mathcal{X}}^{pq} = \emptyset$.*

Proof. Upward branching on (p, q) is infeasible due to Proposition 5.13, so I need only to prove that downward branching is also infeasible. Assume that downward branching on x_{pq} is feasible, so $x_{pq} < \hat{x}_{pq}$. Therefore,

$$\begin{array}{c} \exists j \neq q \in V \ : \ (p, j) \notin \widetilde{I}^{(p,q)} \cup I^{\mathrm{fix}(\widehat{\mathcal{X}})} \ \wedge \ x_{pj} > \hat{x}_{pj} \\ \Downarrow \\ \exists k \neq p \in W \ : \ (k, j) \notin \widetilde{I}^{(p,q)} \cup I^{\mathrm{fix}(\widehat{\mathcal{X}})} \ \wedge \ x_{kj} < \hat{x}_{kj} \end{array}$$

This is again impossible, since for $(k, j) <^{\mathrm{lex}} (p, q)$, $\hat{x}_{kj} = l_{kj}$, because $(k, j) \notin \widetilde{I}^{(p,q)}$, and for $(p, q) <^{\mathrm{lex}} (k, j)$, $\hat{x}_{kj} = l_{kj}$, because $(p, q) = \mathrm{lex\,max}\{(i, j) \ : \ \hat{x}_{ij} > l_{ij}\}$. □

5.3.1 An example of ranking transportation solutions

Example 5.15 Consider the problem of ranking all solutions in the TP having the cost matrix of Table 5.1. The strengthened branching technique for ranking transportation solutions is applied.

$i\backslash j$	4	5	
1	1	1	$b_1 = 2$
2	1	5	$b_2 = 2$
3	3	1	$b_3 = 2$
	$b_4 = -3$	$b_5 = -3$	

Table 5.1: The cost matrix of TP with supplies and demands stated.

The optimal primal solution to the initial TP is $x^1 = \{x_{14} = x_{15} = 1, x_{24} = x_{35} = 2\}$, and a dual optimum is $\pi = (\pi(1), \pi(2), \pi(3), \pi(4), \pi(5)) = (0, 0, 0, -1, -1)$ with $y(x^1) = 6$.

Following propositions 5.7, 5.13, and 5.14, the first solution yields three branches all of which are feasible. Notice that the upward branches ${}_u\mathcal{X}^{15}$ and ${}_u\mathcal{X}^{24}$ are empty due to Proposition 5.13 (a) and (b), respectively. Downward branching on x_{15} would yield $u_{14} + u_{15} = 1 + 0 < 2 = b_1$ and is therefore infeasible (Proposition 5.7 (c)). Finally, any branching on x_{35} is prohibited by Proposition 5.14.

$$
\begin{aligned}
{}_d\mathcal{X}^{14} &= \{x \in \mathcal{X}_{TP} \ : \ x_{14} \leq 0\} \\
{}_u\mathcal{X}^{14} &= \{x \in \mathcal{X}_{TP} \ : \ x_{14} \geq 2\} \\
{}_d\mathcal{X}^{24} &= \{x \in \mathcal{X}_{TP} \ : \ x_{14} = 1, x_{15} = 1, x_{24} \leq 1\}
\end{aligned}
$$

Consider the residual graph $G^\pi(x^1)$ in Figure 5.4. Disregarding for ${}_d\mathcal{X}^{14}$ the arcs $(1, 4)$ and $(4, 1)$, the shortest 1-4-path is $P : 1 - 5 - 3 - 4$ with $c^\pi(P) = 2$ and $d = (0, 2, 0, 2, 0)$. Following the rules of Section 5.2.2, the primal solution ${}_d x^* = \{x_{15} = x_{24} = 2, x_{34} = x_{35} = 1\}$ is obtained, with dual solution updated to ${}_d\pi^* = (0, -2, 0, -3, -1)$ and with total cost $y({}_d x^*) = y(x^1) + c^\pi(P) - c^\pi_{14} = 6 + 2 - 0 = 8$.

For the upper branch ${}_u\mathcal{X}^{14}$ the arcs $(1, 4)$ and $(4, 1)$ are again disregarded. The shortest 4-1-path is $P : 4 - 2 - 5 - 1$ with $c^\pi(P) = 4$ and $d = (4, 0, 4, 0, 4)$. The primal solution is updated to ${}_u x^* = \{x_{14} = 2, x_{24} = x_{25} = 1, x_{35} = 2\}$, with dual solution ${}_u\pi^* = (-4, 0, -4, -1, -5)$ and with total cost $y({}_u x^*) = y(x^1) + c^\pi(P) + c^\pi_{14} = 6 + 4 + 0 = 10$.

Similarly, the shortest path computation for ${}_d\mathcal{X}^{24}$ yields $P : 2 - 5 - 3 - 4$ with $c^\pi(P) = 6$ and $d = (M, 0, 4, 6, 4)$. So the updated solutions are ${}_d x^* = \{x_{14} =$

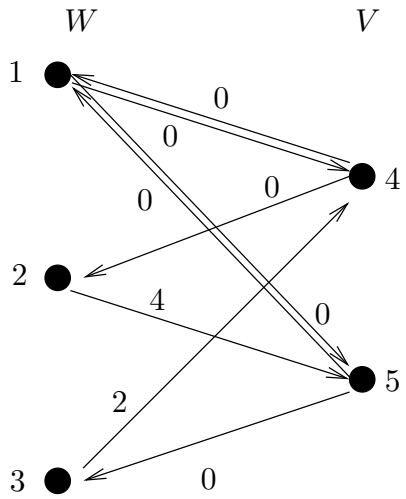


Figure 5.4: Residual graph $G^{\pi}(x^1)$.

$x_{15} = x_{24} = x_{25} = x_{34} = x_{35} = 1\}$ and ${}_d\pi^* = (-M, 0, -4, -7, -5)$ with total cost $y({}_dx^*) = y(x^1) + c^{\pi}(P) - c^{\pi}_{24} = 6 + 6 - 0 = 12$.

Therefore the best transportation solution in $\mathcal{X}_{TP} \setminus \{x^1\}$ is found in ${}_d\mathcal{X}^{14}$ and hence $x^2 = \{x_{15} = x_{24} = 2, x_{34} = x_{35} = 1\}$ with $y(x^2) = 8$. Branching continues on this solution.

With $K = 7$, further branching would in fact rank all solutions for this small TP in nondecreasing order of cost. The corresponding branching tree can be seen in Figure 5.5 showing only non-empty subsets. Lower and upper bound constraints on arcs are assumed to be inhereted implicitly downwards in the branching tree. Notice that x^4 corresponds to a non-extreme solution of the original TP.

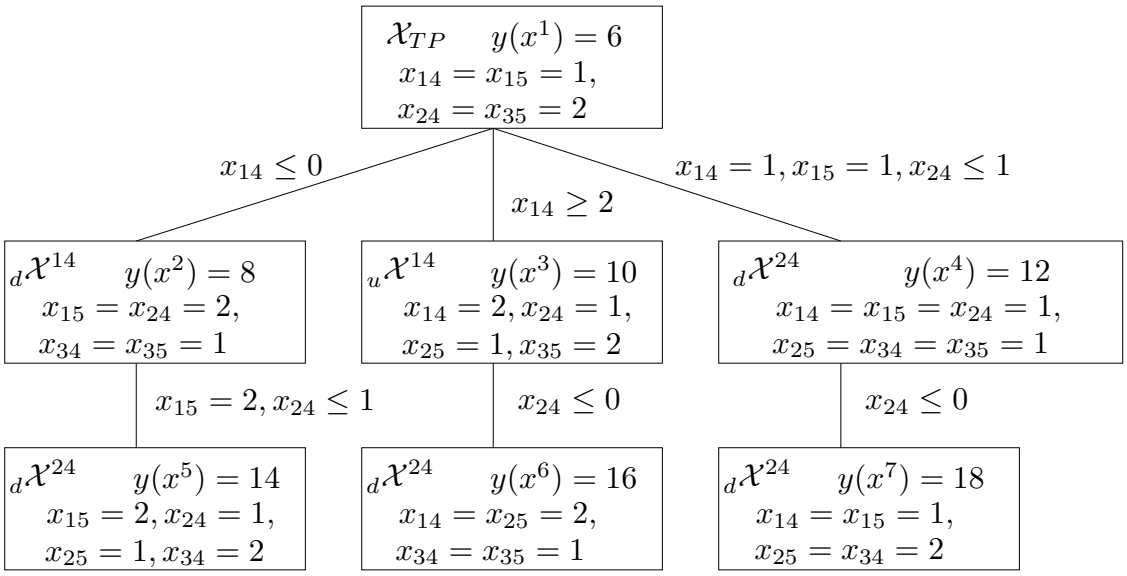


Figure 5.5: Branching tree for Example 5.15.

III

Multicriteria discrete optimization

Chapter 6

The bicriterion multi modal assignment problem

In general, a description of real world applications as single criterion optimization problems is seldom realistic, since they are often, by nature, imposed with more objectives to be simultaneously optimized. Assigning workers to jobs with minimal cost and time yields the *bicriterion assignment problem (BiAP)*, which is an important generalization of the classical linear assignment problem (discussed in Section 2.1.3 and Chapter 4). Within the last ten years focus on BiAP has risen. Ulungu and Teghem [161] presented the first exact solution method for BiAP, proposing a two-phase method identifying in phase one all supported efficient solutions and in phase two all unsupported efficient solutions, (see Section 3.2.1). In that paper, a scheme resembling total enumeration in all nonbasic variables is employed. The method in [161] was implemented by Tuyttens, Teghem, Fortemps, and Van Nieuwenhuyze [159], showing – with large CPU times – the limitations of this algorithm. Recently, an improvement of this algorithm was given in Przybylski et al. [131], proposing also a two-phase method applying ranking for BiAP.

The main focus on BiAP in literature, however, seems to be on heuristical methods. Tuyttens et al. [159] use a version of the MOSA method which is an extension of simulated annealing to deal with multiple objectives. Gandibleux, Morita, and Katoh [54] use genetic information for BiAP, and population based heuristics using path relinking are described in Gandibleux et al. [55] and Przybylski et al. [131].

In this chapter, I deal with another highly relevant extension of the assignment problem, which has, to the best of my knowledge, not yet been discussed in literature. Imagine a large global company with n specialists spread across the world and suppose that exactly n jobs have to be performed by these n specialists, such that each worker is assigned to exactly one job. Moreover, a specialist i has L_{ij} different mode choices of transportation to reach the destination of job j, among

$i\backslash j$	$n+1$	$\cdots$	$2n$
1	$\binom{c^1_{1,n+1,1}}{c^2_{1,n+1,1}}, \ldots, \binom{c^1_{1,n+1,L_{1,n+1}}}{c^2_{1,n+1L_{1,n+1}}}$	$\cdots$	$\binom{c^1_{1,2n,1}}{c^2_{1,2n,1}}, \ldots, \binom{c^1_{1,2n,L_{1,2n}}}{c^2_{1,2n,L_{1,2n}}}$
$\vdots$	$\vdots$	$\ddots$	$\vdots$
n	$\binom{c^1_{n,n+1,1}}{c^2_{n,n+1,1}}, \ldots, \binom{c^1_{n,n+1,L_{n,n+1}}}{c^2_{n,n+1,L_{n,n+1}}}$	$\cdots$	$\binom{c^1_{n,2n,1}}{c^2_{n,2n,1}}, \ldots, \binom{c^1_{n,2n,L_{n,2n}}}{c^2_{n,2n,L_{n,2n}}}$

Figure 6.1: The cost matrix of BiMMAP.

which exactly one must be chosen. For this problem, it seems relevant to consider two weight criteria to be minimized simultaneously, namely *travel time* and *travel or assignment cost*. For worker i, job j and mode choice l, let c^1_{ijl} and c^2_{ijl} denote travel or assignment cost and travel time, respectively.

Since a specialist i has multiple modes of transportation and routes to choose from in order to reach the destination of job j, several two-dimensional cost vectors exist for each i and j. Therefore, the *bicriterion multi modal assignment problem (BiMMAP)* is an extension of BiAP containing, in each assignment cell (i, j) in the *assignment cost matrix*, several two-dimensional cost vectors/points as illustrated in Figure 6.1. The objective is to identify all nondominated criterion points for the problem.

Obviously, BiMMAP can be displayed by the directed bipartite graph $G = (W \cup V, A)$, with node sets $W := \{1, \ldots, n\}$ and $V := \{n+1, \ldots, 2n\}$ and $A := \{(i, j, l)\}_{ijl}$ the set of three-indexed arcs. Letting x_{ijl} be a binary variable with value 1, if i is assigned to j using mode choice l, and 0 otherwise the following mathematical formulation of this bicriterion problem is obtained.

$$\begin{aligned} \min \quad & \sum_{(i,j,l)\in A} c^1_{ijl} x_{ijl} \\ \min \quad & \sum_{(i,j,l)\in A} c^2_{ijl} x_{ijl} \\ \text{s.t.} \quad & x \in \mathcal{X}_{MMAP} \end{aligned} \tag{6.1}$$

in which

$$\mathcal{X}_{MMAP} = \Bigg\{ x \ : \ \sum_{j\in V} \sum_{l=1}^{L_{ij}} x_{ijl} = 1, \ \forall i \in W, \ \sum_{i\in W} \sum_{l=1}^{L_{ij}} x_{ijl} = 1, \ \forall j \in V, \\ x_{ijl} \in \{0,1\}, \ \forall (i,j,l) \in A \Bigg\} \tag{6.2}$$

is the *multi modal assignment integer lattice*. Assume that for each cell (i, j) the costs satisfy

$$0 \leq c^1_{ij1} < c^1_{ij2} < \cdots < c^1_{ijL_{ij}} \quad \text{and} \quad c^2_{ij1} > c^2_{ij2} > \cdots > c^2_{ijL_{ij}} \geq 0 \tag{6.3}$$

Moreover, c^1_{ijl} and c^2_{ijl} are integer for all $(i, j, l) \in A$.

A feasible solution x to (6.1) is called a *multi modal assignment* or short an *assignment*. For $x \in \mathcal{X}_{MMAP}$, $y = (y_1, y_2) = (c^1 x, c^2 x)$ is the corresponding criterion point. In coherence with previous notation, I let the *multi modal assignment polyhedron*, $\mathcal{P}_{MMAP}$ be the continuous relaxation of $\mathcal{X}_{MMAP}$. Note that if $L_{ij} = 1$ for all i, j, $\mathcal{P}_{MMAP}$ reduces to the assignment polyhedron, $\mathcal{P}_{AP}$.

It is well-known that unsupported nondominated points may exist for BiAP [39], and hence also for BiMMAP. Also, since BiMMAP is a generalization of BiAP, it holds true that BiMMAP is intractable and $\mathcal{NP}$-complete ([39, 148]). Moreover, because of (6.3), we have that all cost vectors for a cell are nondominated. This is no restriction, since a dominated cost vector in a given cell will never be used in an efficient assignment.

Recall from Section 3.1 the definition of the adjacency graph. Along the same lines as in Przybylski, Gandibleux, and Ehrgott [132], it can be shown that the adjacency graph for (6.1) may not be connected. In particular, this means that it may not be possible to find the full set of nondominated points by simple pivot operations. Therefore, to find such a full set of nondominated points, I utilize the two-phase method presented in Section 3.2.1 which is not based upon simplex operations. Special features for the current problem class are included in the procedure.

Acknowledging that ranking procedures have been applied with great success for other bicriterion optimization problems (see e.g. Nielsen et al. [114]), ranking of multi modal assignments is employed as a subroutine. The subroutine is an efficient extension of the algorithm to find the K best assignments in a classical AP presented in Chapter 4.

For practical reasons it may be enough to find an approximation of the nondominated set and it may for some large problem sizes be too time-consuming to find all the nondominated criterion points. In such a case the concepts of ε-domination and ε-approximation introduced by Warburton [165] can be used to control the quality of the set of criterion points reported.

Definition 6.1 A point $y = (y_1, y_2)$ *ε-dominates* point $\hat{y} = (\hat{y}_1, \hat{y}_2)$, if $(1 - \varepsilon)y$ dominates $\hat{y}$, i.e. if $(1 - \varepsilon)y \leq \hat{y}$.

Definition 6.2 A set $\widetilde{\mathcal{Y}}$ is an *ε-approximation* of a nondominated set $\mathcal{Y}_N$, if, for each point $\hat{y} \in \mathcal{Y}_N$, there exists $y \in \widetilde{\mathcal{Y}}$ that ε-dominates it.

Using a large library of test instances for BiMMAP, numerical results indicating the effectiveness of the implemented method are given. The concept of

approximating the nondominated points is shown to have a large effect on the computational performance and it is shown that the approximative algorithm is a serious rival to heuristical methods. Since BiMMAP is a generalization of BiAP, I also report computational results for some BiAP instances previously solved in literature showing that the new algorithm outperforms all known exact solution methods.

The remaining part of this chapter is organized as follows. In Section 6.1 and Section 6.2, I describe the specialized version of the two-phase method for the exact solution method and for the approximation method, respectively. Section 6.3 provides a description on how to rank multi modal assignments, and computational results for BiMMAP and BiAP are given in Section 6.4.

6.1 Finding $\mathcal{Y}_N$ with the two-phase method

To solve BiMMAP using the two-phase method presented in general terms in Section 3.2.1, the following parametric minimization problem must be solved iteratively.

$$\begin{aligned} \min\ & f_\lambda(x) = \left(\lambda c^1 + c^2\right) x = \sum_{(i,j,l)\in A} (\lambda c^1_{ijl} + c^2_{ijl}) x_{ijl} \\ \text{s.t.}\quad & x \in \mathcal{X}_{MMAP} \end{aligned} \tag{6.4}$$

For a given value of $\lambda \geq 0$ and a given cell (i,j), define

$$l^\star_{ij}(\lambda) = \arg\min_{1\leq l\leq L_{ij}} \{\lambda c^1_{ijl} + c^2_{ijl}\}\ . \tag{6.5}$$

That is, for a specific value of λ, the minimal parametric cost entry $l^\star_{ij}(\lambda)$, in the assignment cell (i,j), is chosen. It is straightforward to see that (6.4) reduces to the following problem:

$$\begin{aligned} \min\ & f_\lambda(x) = \left(\lambda c^1 + c^2\right) x = \sum_{i\in W} \sum_{j\in V} c_{ijl^\star_{ij}(\lambda)} x_{ijl^\star_{ij}(\lambda)} \\ \text{s.t.}\quad & x \in \mathcal{X}_{AP} \end{aligned} \tag{6.6}$$

which is a single criterion assignment problem. This is summarized in Proposition 6.3.

Proposition 6.3 *The parametric problem* $\min\{f_\lambda(x)\ :\ x \in \mathcal{X}_{MMAP}\}$ *reduces to the classical AP* $\min\{\sum_{ij} c_{ijl^\star_{ij}(\lambda)} x_{ijl^\star_{ij}(\lambda)}\ :\ x \in \mathcal{X}_{AP}\}$.

Therefore, the minimal cost assignment for BiMMAP, given a fixed value of λ, can be found in procedure `PhaseOne` of Figure 3.4 on page 28 by solving a single criterion AP. Moreover, since each criterion point in BiMMAP is integer, the following obvious proposition may be used to reduce the computational effort in phase one.

Proposition 6.4 *Consider two supported extreme nondominated points y^+ and y^-. Then no further supported extreme nondominated points can be found below the line connecting y^+ and y^- in phase one, if any of the following conditions are fulfilled*

$$y_1^- - y_1^+ = 1 \quad \text{or} \quad y_2^+ - y_2^- = 1 \,. \tag{6.7}$$

That is, we simply skip the search between y^+ and y^- if (6.7) holds in phase one. Condition (6.7) may also be used in phase two to skip searching some triangles. If any of the conditions in (6.7) are fulfilled, we do not have to apply procedure `PhaseTwo` of Figure 3.6 on page 30 to triangle $\triangle(y^+, y^-)$ since no unsupported nondominated point can exist in this triangle. Observe that this also holds true even if the points y^+ and y^- are supported nonextreme nondominated points. Hence, storing such nonextreme points in phase one (by using a $\leq$ sign instead of a $<$ sign on line 10 in Figure 3.4) may reduce the computational effort in phase two.

Furthermore, since all nondominated points have integer coordinates, the upper bound used in phase two may be improved.

Proposition 6.5 *Given the triangle $\triangle(y^+, y^-)$ with previously found nondominated points $\{y^+ = y^1, \ldots, y^q = y^-\}$ ordered in increasing order of the first objective and a search direction given by λ, define*

$$UB^{IP} := \max_{r=1,\ldots,q-1} \{\lambda\big(y_1^{r+1} - 1\big) + \big(y_2^r - 1\big)\} \,. \tag{6.8}$$

Then all unsupported nondominated criterion points in $\triangle(y^+, y^-)$ have parametric weight below or equal to UB^{IP}.

Proof. Consider a non-found nondominated point (y_1, y_2) located in $\triangle(y^+, y^-)$. Due to integrality of y_1 and y_2 we have

$$\begin{array}{l} \exists r \in \{1, \ldots, q-1\} : \; y_1 \leq y_1^{r+1} - 1 \;\wedge\; y_2 \leq y_2^r - 1 \\ \Downarrow \\ \lambda y_1 + y_2 \leq \lambda\big(y_1^{r+1} - 1\big) + \big(y_2^r - 1\big) \end{array}$$

Since nondominated points can be located between any two consecutive points y^r and y^{r+1}, we obtain expression (6.8) for the upper bound. □

As a result, the upper bound of (6.8) can be used in function `UpdateUB` of procedure `PhaseTwo`. Furthermore, note that upper bound (6.8) is valid for all bicriterion problems with integer criterion points and is an improvement to the upper bound previously reported in literature [159, 161].

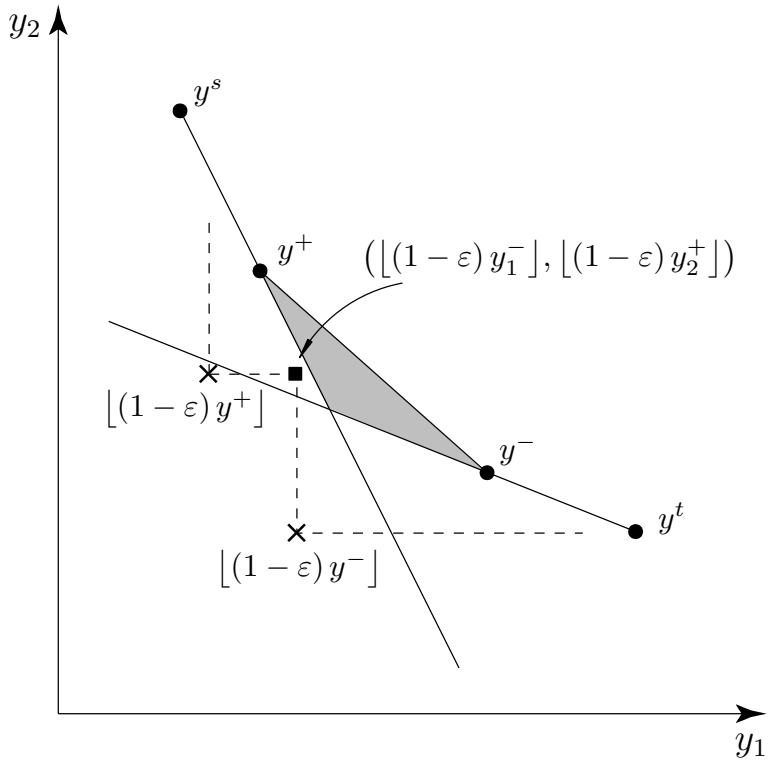


Figure 6.2: Using ε-dominance in the first phase.

6.2 Finding an ε-approximation with the two-phase method

In some cases it may be sufficient to find an approximation of the nondominated set. In this section, I consider the problem of finding an ε-approximation ($\varepsilon > 0$) of the nondominated set using the two-phase method. Only slightly modified versions of phase one and two are needed.

First, consider phase one and a set of extreme nondominated points found during phase one as illustrated in Figure 6.2. Note that any new supported extreme nondominated point between y^+ and y^- must belong to the shaded area in Figure 6.2. As a result we can skip searching for such points between y^+ and y^- if the following proposition is satisfied.

Proposition 6.6 *Given supported extreme nondominated points* $\{y^{UL}, \ldots, y^s, y^+, y^-, y^t, \ldots, y^{LR}\}$ *found during phase one, each supported extreme nondominated point between* y^+ *and* y^-*, i.e. inside the shaded area of Figure 6.2, is* ε*-dominated by either* y^+ *or* y^- *if*

$$\begin{aligned} & \lambda_1 \lfloor(1-\varepsilon)\,y_1^-\rfloor + \lfloor(1-\varepsilon)\,y_2^+\rfloor \leq \lambda_1 y_1^+ + y_2^+ \\ \textit{or} \quad & \lambda_2 \lfloor(1-\varepsilon)\,y_1^-\rfloor + \lfloor(1-\varepsilon)\,y_2^+\rfloor \leq \lambda_2 y_1^- + y_2^- \ , \end{aligned} \tag{6.9}$$

where λ_1 is defined by the slope of the line between y^s and y^+ and λ_2 is defined by the slope of the line between y^- and y^t.

Proof. I only show the result if the first condition in (6.9) is satisfied. The second case is shown with similar arguments. Therefore, assume that $\lambda_1 \lfloor (1-\varepsilon)\, y_1^- \rfloor + \lfloor (1-\varepsilon)\, y_2^+ \rfloor \leq \lambda_1 y_1^+ + y_2^+$. Furthermore, suppose that there exists a supported extreme nondominated point (y_1, y_2) in the shaded area between y^+ and y^- (see Figure 6.2), which is not ε-dominated by neither y^+ nor y^-. It follows that $(1-\varepsilon)\, y_1^- > y_1$ and $(1-\varepsilon)\, y_2^+ > y_2$. Due to integrality of y_1 and y_2, this means:

$$\lambda_1 y_1 + y_2 \leq \lambda_1 \lfloor (1-\varepsilon)\, y_1^- \rfloor + \lfloor (1-\varepsilon)\, y_2^+ \rfloor \tag{6.10}$$

The nondominated point (y_1, y_2) is necessarily strictly above the line through y^s and y^+ (otherwise y^+ is not a supported extreme nondominated point found before (y_1, y_2)). Therefore, $\lambda_1 y_1 + y_2 > \lambda_1 y_1^+ + y_2^+$. This implies that

$$\lambda_1 \lfloor (1-\varepsilon)\, y_1^- \rfloor + \lfloor (1-\varepsilon)\, y_2^+ \rfloor \leq \lambda_1 y_1^+ + y_2^+ < \lambda_1 y_1 + y_2 \tag{6.11}$$

Combining equations (6.10) and (6.11) leads to a contradiction. □

Observe that Proposition 6.6 also holds true for the case where no points are identified to the left (or to the right) of the points y^+ and y^- by an appropriate choice of λ_i, $i = 1, 2$.

Corollary 6.7 *If $y^+ = y^{UL}$ ($y^- = y^{LR}$) Proposition 6.6 holds true by choosing $\lambda_1 = \infty$ ($\lambda_2 = 0$).*

Proposition 6.6 and Corollary 6.7 can be used in procedure `PhaseOne` to skip the search between two points y^+ and y^-.

In phase two, the upper bound (6.8) can be further strengthened if an ε-approximation is wanted.

Proposition 6.8 *Given the triangle $\triangle(y^+, y^-)$ with previously found nondominated points $\{y^+ = y^1, \ldots, y^q = y^-\}$ ordered in increasing order of the first objective and a search direction given by λ, define*

$$UB^{IP}(\varepsilon) := \max_{r=1,\ldots,q-1} \{\lambda \lfloor (1-\varepsilon) y_1^{r+1} \rfloor + \lfloor (1-\varepsilon) y_2^r \rfloor\} \,. \tag{6.12}$$

Then all criterion points in $\triangle(y^+, y^-)$ with parametric weight above $UB^{IP}(\varepsilon)$ are ε-dominated by the current ε-approximation of the triangle.

Proof. Let $y = (y_1, y_2)$ be a nondominated point in $\triangle(y^+, y^-)$. Therefore, there exists an $r \in \{1, \ldots, q-1\}$ such that y is located between y^r and y^{r+1}. If y is not

ε-dominated then

$$
\begin{array}{ll}
 & y_1 < (1-\varepsilon)\, y_1^{r+1} \ \wedge \ y_2 < (1-\varepsilon)\, y_2^{r} \\
\Downarrow & \text{(since } y \text{ is integral)} \\
 & y_1 \leq \left\lfloor (1-\varepsilon)\, y_1^{r+1} \right\rfloor \ \wedge \ y_2 \leq \left\lfloor (1-\varepsilon)\, y_2^{r} \right\rfloor \\
\Downarrow & \\
 & \lambda y_1 + y_2 \leq \lambda \lfloor (1-\varepsilon)\, y_1^{r+1} \rfloor + \lfloor (1-\varepsilon)\, y_2^{r} \rfloor
\end{array}
$$

Since nondominated points can be located between any two consecutive points y^r and y^{r+1}, expression (6.12) is obtained for the upper bound. □

It follows that for $\varepsilon > 0$, the upper bound in function `UpdateUB` of procedure `PhaseTwo` can be updated using (6.12). Also, note that if we consider the shaded area between y^+ and y^- (see Figure 6.2) not searched in phase one (i.e. satisfying (6.9)), then $UB^{IP}(\varepsilon)$ for $\triangle(y^+, y^-)$ will be less than the parametric weight of y^+. Therefore, $\triangle(y^+, y^-)$ is not searched in procedure `PhaseTwo` either. Furthermore, there can be some possible gains in storing supported nonextreme points in phase one as well. The more supported nondominated points that are identified in the first phase, the more likely are $UB^{IP}(\varepsilon)$ to be below the parametric weight of y^+ and hence $\triangle(y^+, y^-)$ is not searched.

Note that, the approximation found in phase one is a subset of the supported nondominated points, since the optimal solution of (6.4) corresponds to a supported nondominated point. Moreover, because a ranking procedure is applied in the second phase, a dominated point cannot be found before a point dominating it. These comments yield the following result.

Proposition 6.9 *The approximation of the nondominated set for an $\varepsilon > 0$ is a subset of $\mathcal{Y}_N$.*

6.3 The K best multi modal assignments

In this section, I describe the method to rank multi modal assignments in nondecreasing order of cost, used in phase two when searching a triangle. That is, ranking assignments using the single criterion parametric costs defined for a given parameter λ.

As in Chapter 4, an assignment x may alternatively be denoted by its network formulation as $a = \{(1, j_1, l_1), \ldots, (n, j_n, l_n)\}$ where $(i, j, l) \in a$ if and only if $x_{ijl} = 1$. Denote by ${}_a\mathcal{X}_{MMAP}$ the corresponding set of all feasible solutions to (6.1).

Without loss of generality, assume that each cell (i, j) contains entries $c_{ij1} \leq \cdots \leq c_{ijL_{ij}}$. The objective is to determine the K best assignments $a^1, \ldots, a^K$, in a single criterion multi modal assignment problem, such that

- $c(a^i) \leq c(a^{i+1}),\ i = 1, \ldots, K-1$

- $c(a^K) \leq c(a),\ \forall a \in {}_a\mathcal{X}_{MMAP} \setminus \{a^1, \ldots, a^K\}$

where $c(a)$ denotes the cost of assignment a.

Given the optimal assignment $a^1 = \{(1, j_1, l_1),\ (2, j_2, l_2), \ldots, (n, j_n, l_n)\}$ of ${}_a\mathcal{X}_{MMAP}$, the set ${}_a\mathcal{X}_{MMAP} \setminus \{a^1\}$ is partitioned into $n-1$ disjoint subsets $\mathcal{X}^i$, $i = 1, \ldots, n-1$, where

$$\mathcal{X}^i = \{a \in {}_a\mathcal{X}_{MMAP} \ :\ (1, j_1, l_1), \ldots, (i-1, j_{i-1}, l_{i-1}) \in a,\ (i, j_i, l_i) \notin a\}$$

Similarly to the original method by Murty [106] it follows that

$$\bigcup_{i=1}^{n-1} \mathcal{X}^i = {}_a\mathcal{X}_{MMAP} \setminus \{a^1\}\ .$$

Clearly, the second best assignment a^2 can be identified using a solution method to find the minimal cost assignment in the sets $\mathcal{X}^i$, $i = 1, \ldots, n-1$. Moreover, the branching technique above can be applied recursively to subsets $\mathcal{X}^i \subset {}_a\mathcal{X}_{MMAP}$.

In general, the algorithm maintains a candidate set Φ of pairs $(\bar{a}, \bar{\mathcal{X}})$, where $\bar{a}$ is the minimum cost assignment in (sub)set $\bar{\mathcal{X}}$. Suppose we have found the $k-1$ best assignments $a^1, \ldots, a^{k-1}$, then the current candidate set Φ represents a disjoint partition of ${}_a\mathcal{X}_{MMAP} \setminus \{a^1, \ldots, a^{k-1}\}$. The kth best assignment is then found as the pair $(\hat{a}, \widehat{\mathcal{X}}) \in \Phi$ which contains the assignment $\hat{a}$ with minimum cost $c(\hat{a})$ among all assignments in the candidate set Φ.

Consider the solution method, i.e. how to determine the minimum cost assignments in $\widehat{\mathcal{X}}^i$ when applying the branching technique to some subset $\widehat{\mathcal{X}} \subset {}_a\mathcal{X}_{MMAP}$. Without loss of generality, assume that the minimum cost assignment in subset $\widehat{\mathcal{X}}$ is given by

$$\hat{a} = \{(1, j_1, l_1), (2, j_2, l_2), \ldots, (n, j_n, l_n)\}\ . \tag{6.13}$$

Furthermore, assume that, according to previous partitions, no assignments in $\widehat{\mathcal{X}}$ can contain $(m_1, p_1, h_1), \ldots, (m_q, p_q, h_q)$. Recall that any assignment belonging to $\widehat{\mathcal{X}}^i$ must contain $(1, j_1, l_1), \ldots, (i-1, j_{i-1}, l_{i-1})$. Assuming that $\widehat{\mathcal{X}}^i$ contains an assignment, it can be found as follows:

1. Delete rows $\{1, 2, \ldots, (i-1)\}$ and columns $\{j_1, j_2, \ldots, j_{i-1}\}$ from the cost matrix.

2. The cost of entries (i, j_i, l_i) and $(m_1, p_1, h_1), \ldots, (m_q, p_q, h_q)$ in the cost matrix is set to infinity.

Given a non-empty subset $\widehat{\mathcal{X}}^i$, let MMAP($\widehat{\mathcal{X}}^i$) denote the multi modal assignment problem defined by the two steps above. The following result follows from Proposition 6.3.

Corollary 6.10 *The minimal cost multi modal assignment in MMAP($\widehat{\mathcal{X}}^i$) can be found by solving a classical assignment problem using the minimal cost of each cell in MMAP($\widehat{\mathcal{X}}^i$).*

Due to Corollary 6.10, the algorithm for ranking classic linear assignments from Chapter 4 can be used with the slightly more general branching technique described above. Since the general branching technique described above does not create more subsets than the classical branching technique, the overall complexity for ranking the K best multi modal assignments is the same.

Theorem 6.11 *The complexity for finding the K best multi modal assignments is $\mathcal{O}(Kn^3)$.*

Actually, in some cases the minimal cost assignment for subset $\widehat{\mathcal{X}}^i$ can be found without solving an AP. Given subset $\widehat{\mathcal{X}}$, assume without loss of generality that each cell (i,j) in MMAP($\widehat{\mathcal{X}}$) contains L_{ij} entries $c_{ij1} \leq \cdots \leq c_{ijL_{ij}}$ (not set to infinity). Moreover, let $\hat{\pi}$ denote the dual variables of the optimal assignment (6.13) found by solving AP($\widehat{\mathcal{X}}$). Hence the corresponding reduced cost for each cell (i,j) becomes $c_{ij}^{\hat{\pi}} = c_{ij1} - \hat{\pi}(i) + \hat{\pi}(j)$. By disregarding cell (i,j), the minimum reduced costs in row i and column j are

$$c_{i\cdot}^{\hat{\pi}} = \min_t \{c_{it}^{\hat{\pi}} \mid t \neq j\} \text{ and } c_{\cdot j}^{\hat{\pi}} = \min_s \{c_{sj}^{\hat{\pi}} \mid s \neq i\} ,$$

respectively. Note that $c_{i\cdot}^{\hat{\pi}}, c_{\cdot j}^{\hat{\pi}} \geq 0$, $\forall i,j$, due to optimality of $\hat{\pi}$. Now, consider subset $\widehat{\mathcal{X}}^i$. In MMAP($\widehat{\mathcal{X}}^i$) c_{ij_i1} is set to infinity. If $L_{ij_i} > 1$, c_{ij_i1} is replaced by c_{ij_i2} in AP($\widehat{\mathcal{X}}^i$). That is, AP($\widehat{\mathcal{X}}^i$) uses the same costs as AP($\widehat{\mathcal{X}}$) except in cell (i,j_i) where c_{ij_i2} is used. Hence we have the following proposition to enhance the performance of the procedure.

Proposition 6.12 *Assume $L_{ij_i} > 1$ and $c_{i\cdot}^{\hat{\pi}} + c_{\cdot j_i}^{\hat{\pi}} \geq c_{ij_i2} - c_{ij_i1}$. Then a minimal cost assignment for subset $\widehat{\mathcal{X}}^i$ is*

$$a(i) = (\hat{a} \setminus \{(i,j_i,1)\}) \cup \{(i,j_i,2)\} .$$

Proof. Define

$$\Delta_{ij_i} = c_{ij_i2} - c_{ij_i1} \geq 0 .$$

The optimal assignment $\hat{a}$ of AP($\widehat{\mathcal{X}}$) is primal feasible and satisfies the complementary slackness conditions $\hat{x}_{ij1} c_{ij}^{\hat{\pi}} = 0$, $\forall (i,j)$. Consider AP($\widehat{\mathcal{X}}^i$) with dual variables $\bar{\pi}$. If $\Delta_{ij_i} \leq c_{i\cdot}^{\hat{\pi}}$, set $\bar{\pi}(i) = \hat{\pi}(i) + \Delta_{ij_i}$ and keep the remaining dual values unchanged. Then $c_{it}^{\bar{\pi}} \geq 0$, $\forall t \neq j_i$ and

$$c_{ij_i}^{\bar{\pi}} = c_{ij_i2} - (\hat{\pi}(i) + \Delta_{ij_i}) + \hat{\pi}(j_i) = c_{ij_i}^{\hat{\pi}} = 0 .$$

Hence the assignment $a(i)$ of AP($\widehat{\mathcal{X}}^i$) is primal feasible and satisfies, together with $\hat{\pi}$, the complementary slackness conditions, i.e. $a(i)$ is an optimal solution.

If $\Delta_{ij_i} > c^{\hat{\pi}}_{i\cdot}$, we set $\bar{\pi}(i) = \hat{\pi}(i) + c^{\hat{\pi}}_{i\cdot}$ and $\bar{\pi}(j_i) = \hat{\pi}(j_i) - \Delta_{ij_i} + c^{\hat{\pi}}_{i\cdot}$ and keep the remaining dual values unchanged. Hence $c^{\bar{\pi}}_{it} \geq 0$, $\forall t \neq j_i$, $c^{\bar{\pi}}_{sj_i} \geq 0$, $\forall s \neq i$, and

$$c^{\bar{\pi}}_{ij_i} = c_{ij_i2} - \left(\hat{\pi}(i) + c^{\hat{\pi}}_{i\cdot}\right) + \left(\hat{\pi}(j_i) - \Delta_{ij_i} + c^{\hat{\pi}}_{i\cdot}\right) = c^{\hat{\pi}}_{ij_i} = 0$$

Again, assignment $a(i)$ is optimal. □

Using Proposition 6.12, we do not have to solve AP($\widehat{\mathcal{X}}^i$) if $c^{\hat{\pi}}_{i\cdot} + c^{\hat{\pi}}_{\cdot j_i} \geq c_{ij_i2} - c_{ij_i1}$. The minimal cost assignment $a(i)$ is simply obtained by assigning the rows to the same columns as in assignment $\hat{a}$ and, in cell (i, j_i), by using entry 2 instead of entry 1.

6.4 Computational results

In this section, I report the computational experience on BiMMAP test instances. Moreover, since BiMMAP is an extension of BiAP, I also report some results on test instances for BiAP. All tests were performed on an Intel Xeon 2.67 GHz computer with 6 GB RAM using a Red Hat Enterprise Linux version 4.0 operating system.

6.4.1 Implementational details

The algorithms have been implemented in C++ and compiled with the GNU C++ compiler version 3.4.5 using optimize option `-O`.

The cost matrix of BiMMAP (see Figure 6.1 on page 66) is stored using a two-dimensional array of *Cell* objects. Each Cell object contains an array holding the cost entries and an ordered array holding the parametric costs of the entries for a specific λ.

In phase one, for a given search direction specified by λ, the parametric costs are updated and ordered in nondecreasing order. Due to Proposition 6.3, I consider the smallest entry in each cell and solve the resulting AP using the implementation given by Jonker and Volgenant [86]. Furthermore, the algorithm takes advantage of Proposition 6.4 or Proposition 6.6 (if $\varepsilon > 0$) whenever possible.

In phase two, the parametric costs are again updated and ordered in nondecreasing order for a given ranking direction specified by λ. Next, the K best multi modal assignment procedure described in Section 6.3 is utilized for searching a triangle, using the upper bounds given in Proposition 6.5 or Proposition 6.8 (if $\varepsilon > 0$).

The K best multi modal assignment procedure was implemented using the reoptimization algorithm from Chapter 4 for ranking assignments for the classical AP, with the slightly more general branching technique given in Section 6.3. In particular, note that, when considering a subset where an entry in the ordered array of parametric costs is removed, the new entry with minimal cost is the next

cost in the array. That is, the new minimal cost is found by increasing a local pointer by one.

All nondominated points found by the ranking procedure are stored in a single linked list available in both phase one and two.

6.4.2 BiMMAP test instances

The bicriterion multi modal assignment problem has, to the best of my knowledge, not previously been studied in literature, and hence no available test instances exist for this problem. To facilitate a comprehensive computational study of the BiMMAP algorithm, a problem generator, *APGen*[5], was build for this problem class. As a side-effect this generator can be used to generate a variety of BiAP instances. In the following I give a brief description of the generator, and refer readers requiring more information on this topic, to the full documentation paper [113]. A BiMMAP instance is generated specifying a number of parameters:

n – the size of the problem.

$maxEnt$ – the maximal number of entries in each assignment cell.

$minEnt$ – the minimal number of entries in each assignment cell (default 1).

$maxCost$ – the maximal cost value (minimum cost value is 0).

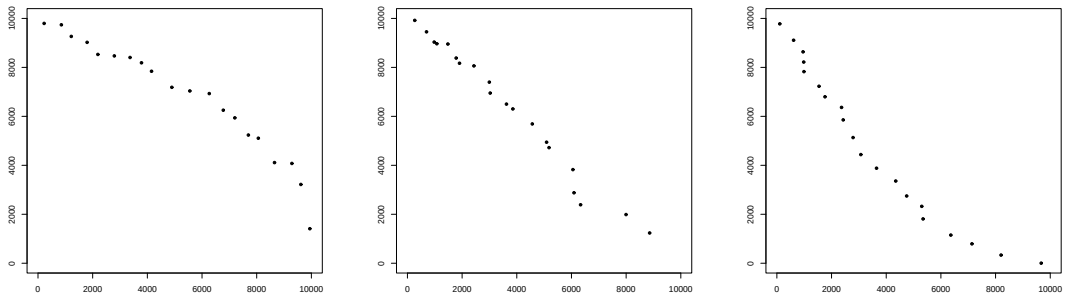
method – a choice between three different ways of generating cell entries.

shape – for a given method, the shape parameter describes the shape of the entries in a given cell.

Obviously, for a given cell, no entries are allowed to be dominated by other entries in that cell, since this would correspond to a dominated solution. The number of entries in a cell is chosen randomly in the *entry range* $\{minEnt, \ldots, maxEnt\}$. To describe best the six different versions of *method* and *shape* used for generating BiMMAP instances, all two-dimensional cost vectors for a given cell having 20 entries are displayed in Figures 6.3 to 6.5.

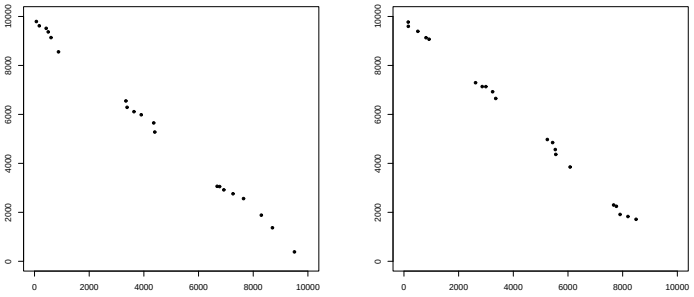
As can be seen in Figure 6.3, with method 1 the shape parameter describes the curve of the function along which the entries are generated. A negative shape corresponds to generating the entries along a concave-like function, using shape 0 generates entries fluctuating along a straight line, and finally, a positive shape means generating entries along a convex-like function. Therefore, using a negative (positive) shape parameter tends to generate many unsupported (supported) entries in the given cell. We shall see, that this has a strong influence on the difficulty of the considered problem and hence the computational performance of the algorithm.

[5] The problem generator and the test instances used in this chapter are downloadable from the following webpage `http://www.research.relund.dk/`.



(a) shape −60 (b) shape 0 (c) shape 60

Figure 6.3: Cell entries for method 1.

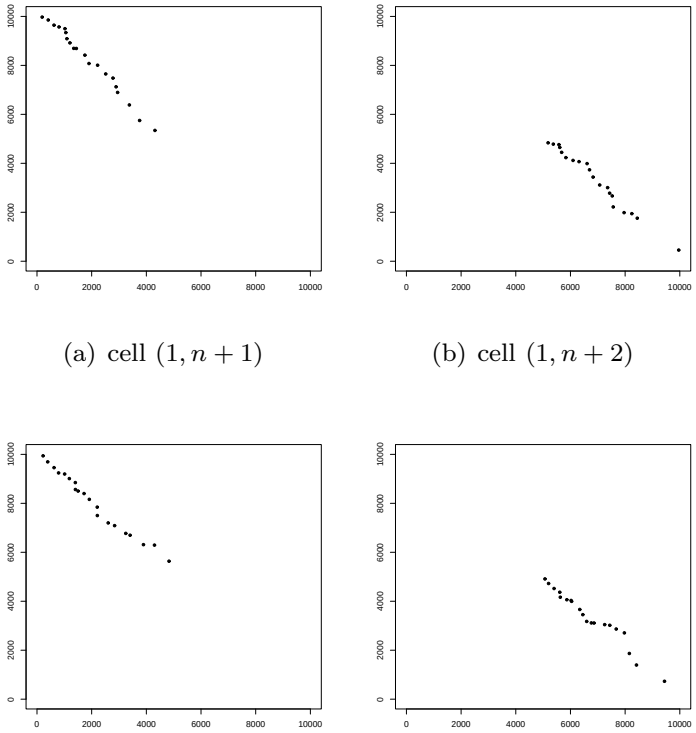


(a) shape 3 (b) shape 4

Figure 6.4: Cell entries for method 2.

For method 2, the entries in a given cell are generated in a number of groups corresponding to the shape parameter (see Figure 6.4). The groups are equally distributed in the cost-space and the entries are all generated fluctuating along the straight line between $(0, maxCost)$ and $(maxCost, 0)$. Note, to use method 2, the parameter $minEnt$ must be chosen sufficiently large, since at least 2 points are required in each group.

Finally, for method 3, the shape parameter has the same meaning as for method 1. However, here the cost space is divided into four regions by halving both criterion axes, and the entries are generated either in the upper left cost region or in the lower right cost region in consecutive cells. This can be seen in Figure 6.5, where the entries in the four consecutive assignment cells $(1, n+1), \ldots, (1, n+4)$ are shown.



(a) cell $(1, n+1)$ (b) cell $(1, n+2)$

(c) cell $(1, n+3)$ (d) cell $(1, n+4)$

Figure 6.5: Cell entries for cells $(1, n+1)$ to $(1, n+4)$ for method 3 and shape 0.

To provide a broad class of test instances and facilitate statistical analysis, 100 instances of each of the following 80 possible configurations were generated.

- $n \in \{4, 6, 8, 10\}$.
- Cost ranges : $\{0, \ldots, 500\}$ and $\{0, \ldots, 10000\}$.
- Entry ranges : $\{2, \ldots, 8\}$ (not for method 2) and $\{10, \ldots, 30\}$.
- $(method, shape) \in \{(1, -60), (1, 0), (1, 60), (2, 3), (2, 4), (3, 0)\}$.

The two different ranges of number of entries are chosen to reflect a situation close to BiAP (few entries) and a situation very far away from BiAP (many entries), respectively. Note that the number of possible assignments increases exponentially with the number of entries in each cell. For a problem with $n = 10$ and entry range $\{10, \ldots, 30\}$, the total number of assignments ranges between $10! \cdot 10^{10} \approx 3.6 \cdot 10^{16}$ in the best case and $10! \cdot 10^{30} \approx 3.6 \cdot 10^{36}$ in the worst case. In comparison, the BiAP with size 10 has $10! \approx 3.6 \cdot 10^6$ feasible assignments.

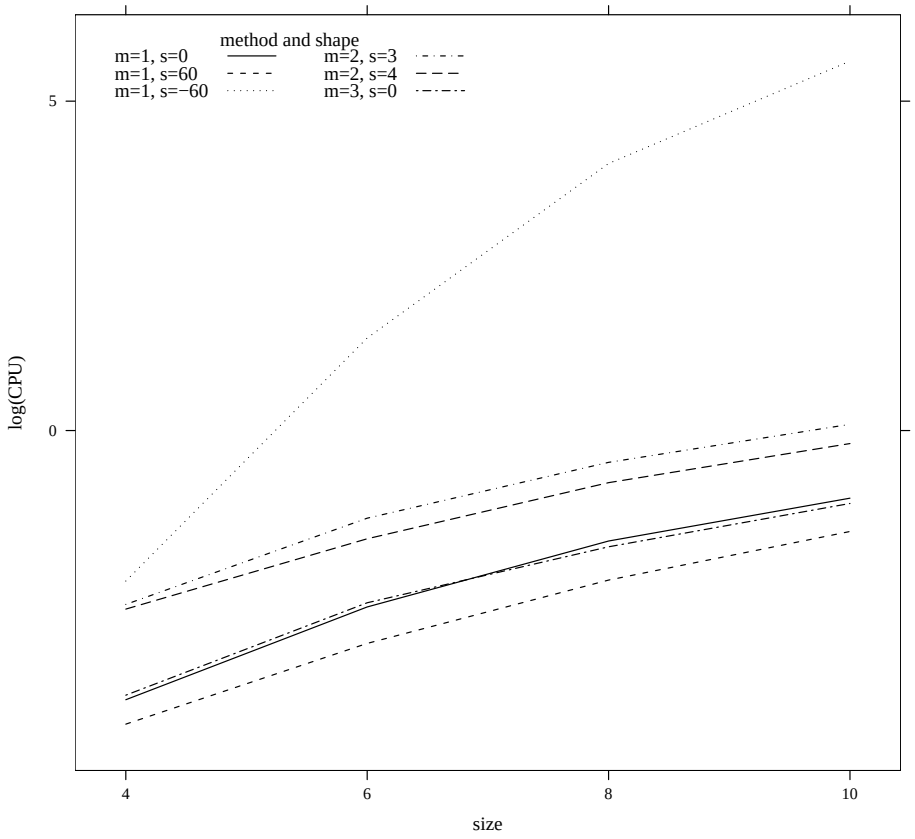
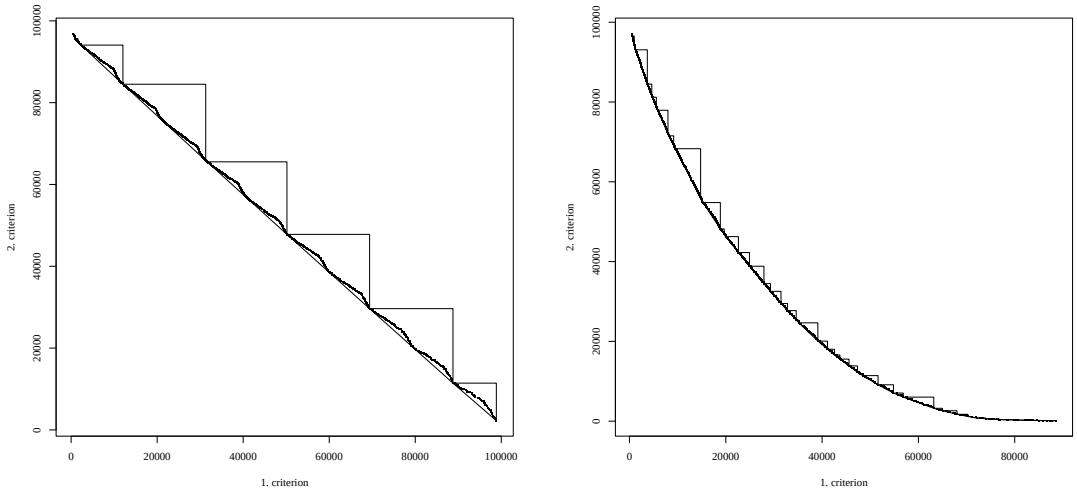


Figure 6.6: Logarithm of average CPU time against n (cost range $\{0, \ldots, 10000\}$ and entry range $\{10, \ldots, 30\}$).

6.4.3 BiMMAP test results

Giving the results of the extensive amount of tests, I first display the logarithm of the CPU time (in seconds) averaged over the 100 instances against problem size n for the highest possible cost range and the highest possible entry range (Figure 6.6). It can be seen, that, for none of the six different classes, the running time is increasing exponentially with problem size. Also notice that the most difficult class is by far using method 1 with shape -60, whereas the easiest class is method 1 and shape 60.

To yield a possible explanation of the difference in difficulty of these two problem classes, it is important to note that phase two is the major time consumer in this algorithm. More specifically, comparing the running times for all the 8000 exactly solved instances, phase two uses an average of 98 per cent of the total CPU time. Consider Figure 6.7 where the nondominated points in the criterion space have been plotted for two test instances using method 1, shape -60 and



(a) shape −60 (b) shape 60

Figure 6.7: Nondominated points (method 1, $n = 10$, entry range $\{10, \ldots, 30\}$ and cost range $\{0, \ldots, 10000\}$).

method 1, shape 60, respectively. Triangles are drawn between consecutive supported extreme nondominated points.

For the test instance with shape −60, only a limited number of supported extreme nondominated criterion points exist. Notice that these extreme points are far from each other resulting in large triangles to search in the second phase. More important, all the supported extreme nondominated points are located almost on a straight line. Therefore, the ranking directions for the triangles are more or less the same. As a result, the ranking procedure of the time demanding phase two initiated in the first large triangle has to generate many points before reaching the upper bound of the triangle making this single triangle search extremely time consuming. Remember though, that nondominated points generated which are outside the triangle currently searched, are stored. This may enable the algorithm to finish searching other triangles faster and hence enhance computational performance.

In contrast, the test instance with shape 60 has a higher number of extreme nondominated points in the criterion space, resulting in small triangles to search. Moreover, the ranking directions are more diverse and hence fewer points have to be ranked when searching a triangle.

Considering other instances the above relationships proved to have general validity. The larger a triangle is, the longer the search in this triangle continues in phase two. Therefore, with many extreme points at termination of phase one, only small triangles need to be searched, resulting in a lower overall running time

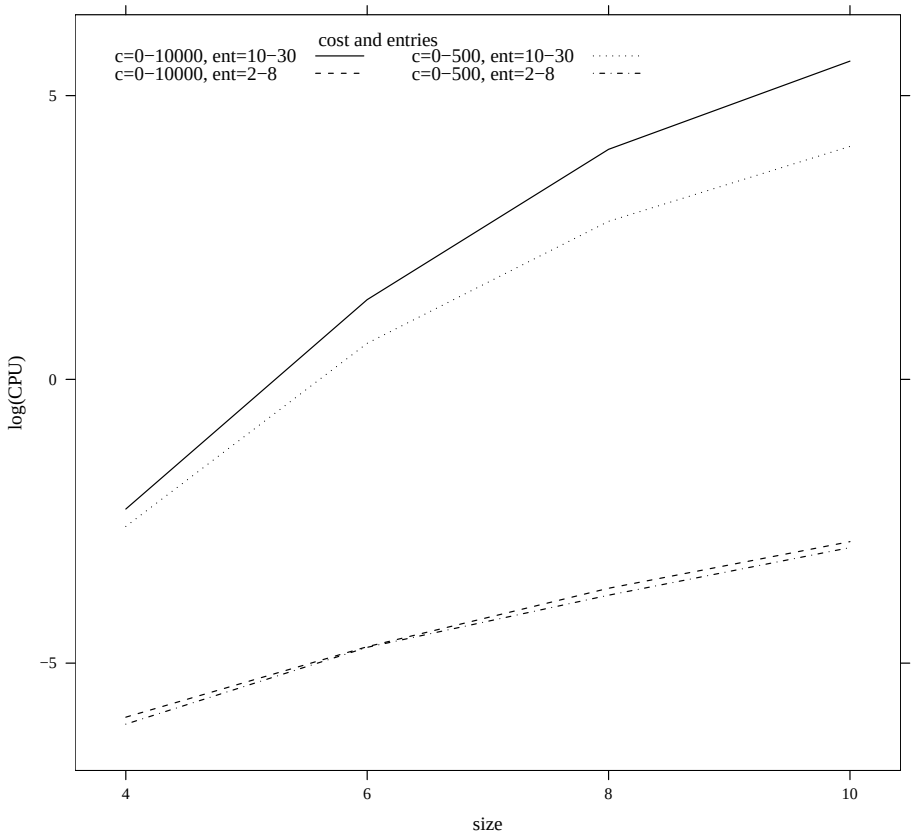


Figure 6.8: Logarithm of average CPU time against n (method 1 and shape -60).

compared to an instance with a few extreme points and larger triangles. Moreover, test instances where the ranking directions for the triangles are more or less the same are harder to solve.

Since method 1 shape -60 has established itself as the most difficult problem class, I focus entirely on such instances from here on.

In Figure 6.8, the logarithm of average CPU time against the problem size is shown for the four different configurations of entry range and cost range. It can be seen, that the most difficult case is the one with the most entries and highest cost range. The most significant factor is the entry range, obviously resulting from the increased number of feasible solutions. Also, for a given number of entries, the most difficult case arises with the highest possible cost range. In this respect, the BiMMAP follows the classical single criterion assignment problem since this problem is known to be easiest solvable with relatively small costs [86].

From here on, focus is entirely on test instances using method 1, shape -60 cost range $\{0, \ldots, 10000\}$ and entry range $\{10, \ldots, 30\}$. In Table 6.1, I give the nu-

size	ave CPU	max CPU	ave $\lvert\mathcal{Y}_{sN}\rvert$	ave $\lvert\mathcal{Y}_{usN}\rvert$	ave $\lvert\mathcal{Y}_{N}\rvert$
4	0.10	0.27	7	206	213
6	4.07	37.36	10	445	455
8	57.61	427.93	13	744	757
10	272.45	3693.79	16	1135	1151

Table 6.1: Exact results (method 1, shape -60, entry range $\{10, ..., 30\}$ and cost range $\{0, ..., 10000\}$).

merical results for the exact solution of these instances. In the first three columns are depicted the size of the problem, average CPU time (seconds) and maximal CPU time, respectively. In the remaining three columns I report average number of supported nondominated points ($\mathcal{Y}_{sN}$), average number of unsupported nondominated points ($\mathcal{Y}_{usN}$), and average of the total number of nondominated points ($\mathcal{Y}_N$), respectively.[6] Obviously, all columns are increasing in size. However, it is interesting to note the relatively high number of unsupported nondominated criterion points making these instances very difficult.

Now I describe the results for finding an approximation of the nondominated set. Two small values 0.01 and 0.05 of ε are chosen to ensure that a sufficiently accurate approximation is found. I also include the results for the exactly solved instances ($\varepsilon = 0$). In Figure 6.9 is displayed the logarithm of average CPU time against size for all three ε values for instances with method 1, shape -60 cost range $\{0, \ldots, 10000\}$ and entry range $\{10, \ldots, 30\}$.

Finding an approximation can be seen to have a strong influence on the running time of the algorithm. Even for these small ε values (and hence good approximations) there are significant savings in computation time. I note (not displayed) that the number of identified nondominated points decreases with increasing ε, as some supported extreme nondominated points may not be considered in the first phase, and in the second phase, fewer alternatives for each triangle are ranked.

In Figure 6.10 on page 84, I display the empirical cumulative distribution functions of CPU time for the 100 test instances of the above configuration for problem size 10. This clearly shows that the majority of problems are solved fast, whereas only a few difficult instances are solved relatively slowly. The numerical results are summarized in Table 6.2, giving for each ε the average CPU time (seconds), the 90 per cent fractile of CPU time and the maximum CPU time.

[6] The average number of nondominated points is rounded to the nearest integer.

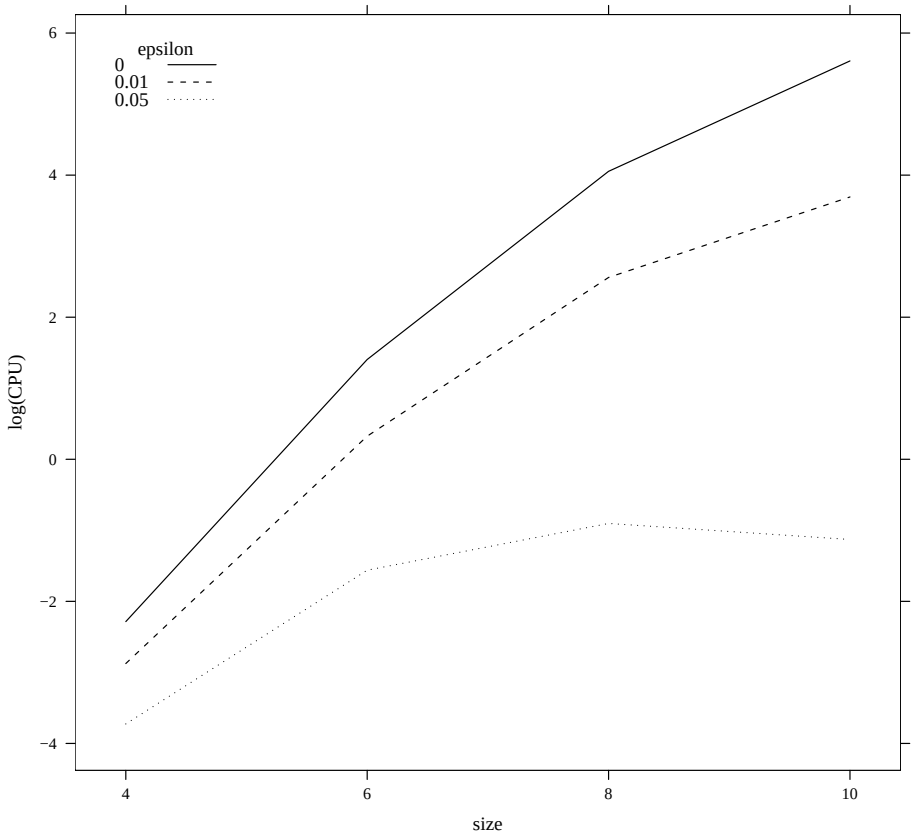


Figure 6.9: Logarithm of average CPU time against n (method 1, shape -60, entry range $\{10, ..., 30\}$ and cost range $\{0, ..., 10000\}$).

	$\varepsilon = 0$			$\varepsilon = 0.01$			$\varepsilon = 0.05$		
size	ave.	90%	max	ave.	90%	max	ave.	90%	max
4	0.10	0.20	0.27	0.06	0.11	0.18	0.02	0.05	0.09
6	4.07	8.46	37.36	1.38	3.05	16.37	0.21	0.40	2.28
8	57.61	154.05	427.93	12.93	41.59	95.36	0.41	1.13	2.43
10	272.45	643.31	3693.79	40.15	91.74	513.58	0.32	0.76	4.43

Table 6.2: CPU times for $\varepsilon \in \{0, 0.01, 0.05\}$ (method 1, shape -60, entry range $\{10, ..., 30\}$ and cost range $\{0, ..., 10000\}$).

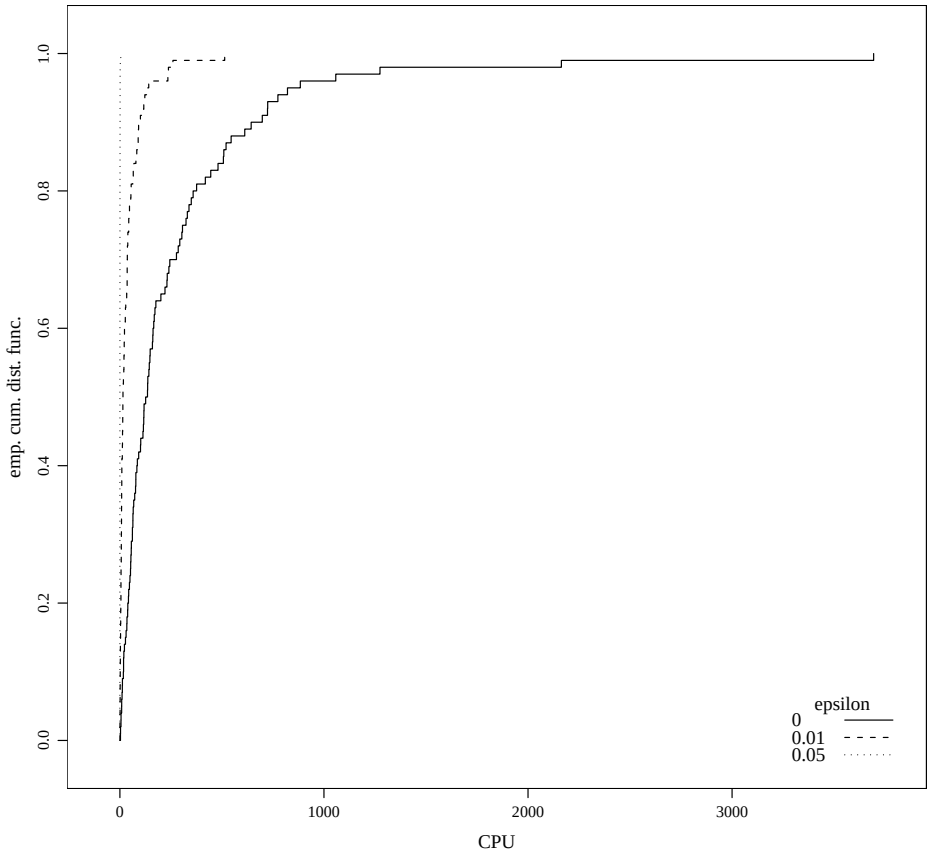


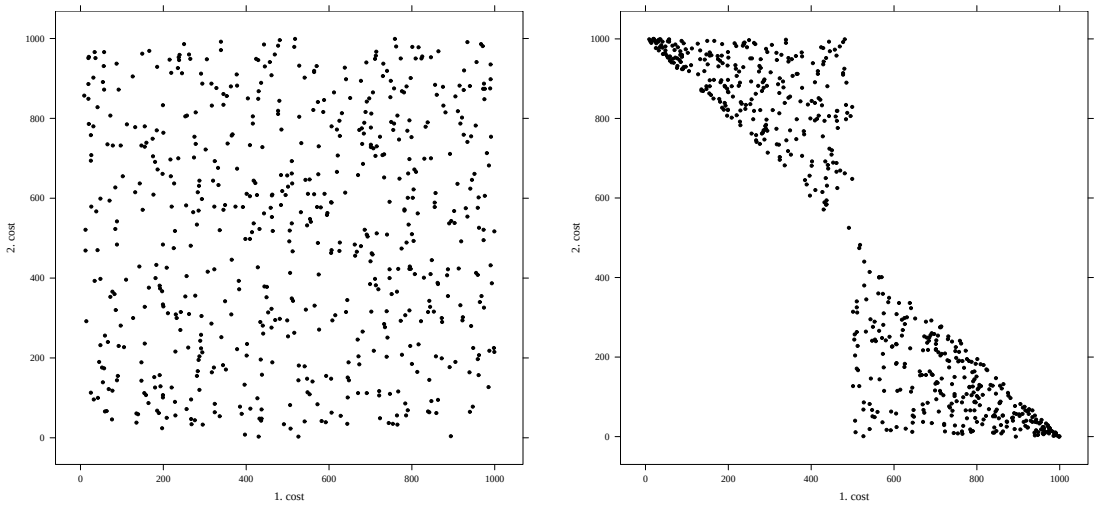
Figure 6.10: Empirical cumulative distribution function of CPU time for different ε values.

6.4.4 Results for BiAP

Since BiMMAP is an extension of BiAP, it is natural to test the performance of the current implementation on this problem class. To yield consistency in literature, I obtained the test instances used in Tuyttens et al. [159] which are BiAP instances of size $\{5, 10, \ldots, 50\}$. Also previously used in literature are BiAP instances of size $\{60, 70, \ldots, 100\}$ found in Gandibleux et al. [54].[7] These instances have recently been solved by an exact method in [131] and by a heuristic in [55] acknowledging the current interest in this field. For all the problem classes only one instance is solved, and hence limited statistics can be performed for those data sets. For all instances, costs are chosen randomly in the rather narrow interval $\{0, \ldots, 19\}$.

To provide statistics based on a broader class of instances, 100 instances were generated, using APGen, for each of the following sizes $\{5, 10, \ldots, 100\}$ with costs

[7] `http://www.univ-valenciennes.fr/ROAD/MCDM/ListMOAP.html`



(a) Random. (b) Negatively correlated.

Figure 6.11: Cost generation for BiAP test instances.

randomly chosen in $\{0, \ldots, 1000\}$. This wide interval leaves room for identifying a large number of large triangles to search in phase two, and hence increase the difficulty of the problem. Also, to investigate the effect of negatively correlated costs, I generated 100 instances of each of the sizes $\{5, 10, \ldots, 100\}$, again with costs in the interval $\{0, \ldots, 1000\}$. The difference in costs generated randomly and negatively correlated is shown in Figure 6.11. We shall see that negatively correlated cost vectors have a strong influence on the difficulty of the considered problem, and hence the running time of the algorithm. In general bicriterion problems with negatively correlated costs are harder to solve, see e.g. [26].

In Figure 6.12, the time against size for the instances previously found in literature is shown. I note that the number of nondominated points found here corresponds exactly to the number of efficient solutions found by CPLEX in [54], making this set a minimal complete set of efficient solutions. In [54], the results were already concluded to be questioning the validity of the results from [159], and this is substantiated by my results. A comparison of the new CPU time with the CPU time of the exact algorithms reported upon in [54, 131, 159] must necessarily include a discussion on the efficiency of the different computers used. Applying *Linpack Benchmark-Peaks* from Netlib [111] to reflect the relative performance of the computers, we can see that the new algorithm developed in this chapter outperforms the exact methods previously proposed in literature. Furthermore, the running time of the exact method is even competitive to the CPU time for the heuristics proposed in [54, 55].

Figure 6.13 on page 87 shows average CPU time against size for the negatively correlated and random BiAP test instances. For the negatively correlated

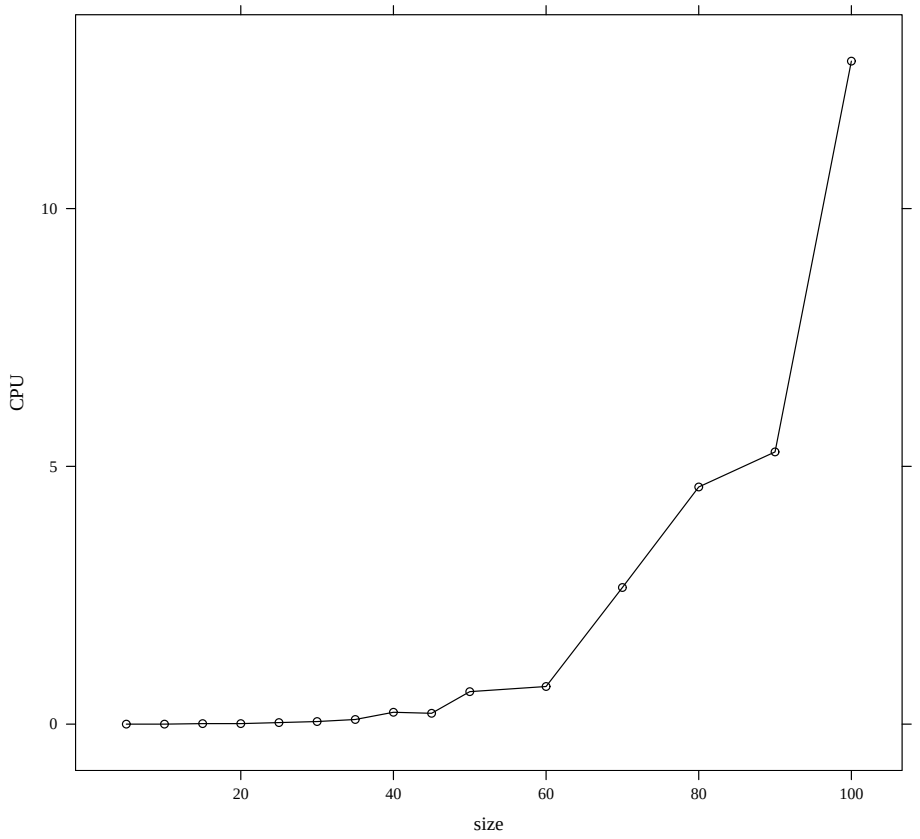


Figure 6.12: CPU time against n for the BiAP instances from [54] and [159].

instances, the algorithm was only capable of solving instances of problem size up to 40 within a reasonable amount of time. This shows the complex nature of such instances, as is also previously seen for other bicriterion problems. The increased difficulty follows mainly because we have more (see Table 6.3) and larger triangles (not shown). Moreover, note that the number of nondominated points is much higher. For the size 40 problem reported upon in [159], a total of 127 nondominated points was found, 73 of which were unsupported. In Table 6.3, we see that the average number of nondominated points and the average number of unsupported nondominated points for the size 40 negatively correlated instances (random instances) are 2402 (333) and 2346 (297), respectively.

Having CPU times no larger than 182 seconds for the random instances and 2635 seconds for the negatively correlated instances, the new algorithm proves capable of solving BiAP problems rather efficiently.

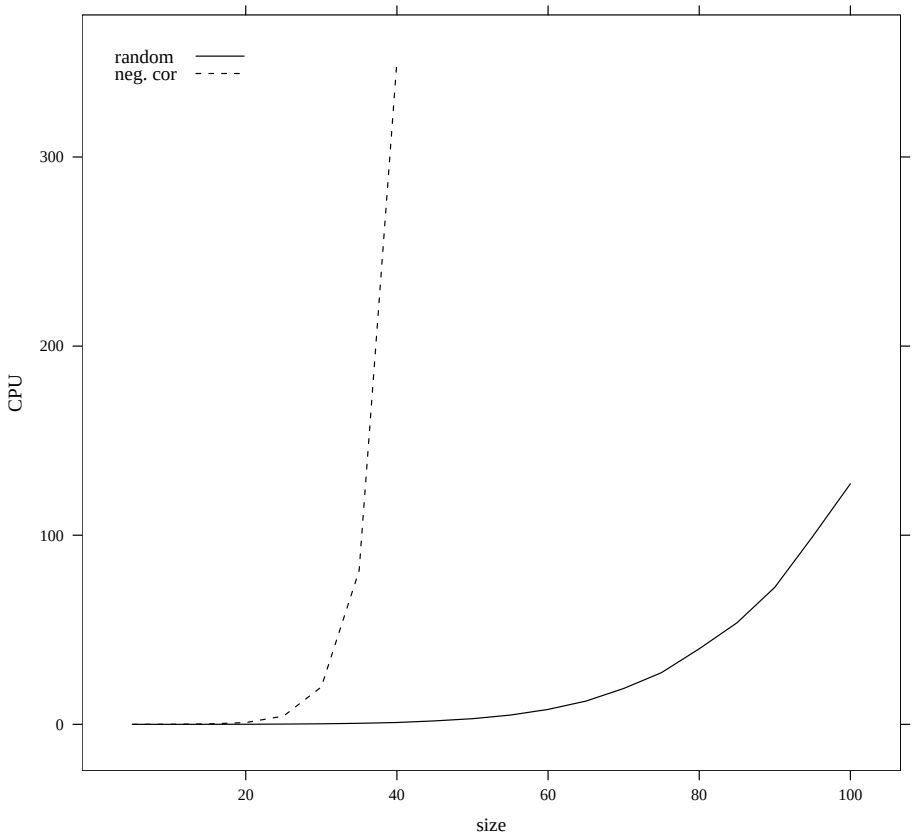


Figure 6.13: Average CPU time against n for the new BiAP instances.

	negatively corr. data			random data		
size	ave $\lvert\mathcal{Y}_{sN}\rvert$	ave $\lvert\mathcal{Y}_{usN}\rvert$	ave $\lvert\mathcal{Y}_{N}\rvert$	ave $\lvert\mathcal{Y}_{sN}\rvert$	ave $\lvert\mathcal{Y}_{usN}\rvert$	ave $\lvert\mathcal{Y}_{N}\rvert$
5	6	15	20	4	2	6
10	13	115	129	8	17	24
15	20	295	315	12	41	53
20	28	539	567	18	72	90
25	35	878	913	23	119	141
30	42	1299	1341	27	167	195
35	49	1773	1822	33	231	263
40	56	2346	2402	37	297	333

Table 6.3: Exact results for the new BiAP instances.

6.4.5 Comments on the usefulness of the IP upper bound

I conclude this section by commenting on the usefulness of the IP strengthening of the upper bound from (6.8) of Proposition 6.5, when searching a triangle. To collect outputs for making such statistics, I intentionally neglected to focus on CPU time by actually disabling this IP strengthening.

Calculating the number of times the IP upper bound would succeed in terminating the search of a triangle when the bound without IP strengthening would fail and dividing this by the total number of iterations, the average ratio for the negatively correlated BiAP instances is 5.80 per cent. This corresponds to saying that, in more than one out of 20 iterations the search in a given triangle would be terminated when using the integer based upper bound, whereas the upper bound without integrality would still be higher than the lower bound. For the random instances this ratio is 7.67 per cent and finally, for the random BiAP instances found in literature, the ratio is 79.55 per cent. Obviously, since only a limited amount of instances have previously been reported upon in literature, this last number must be interpreted with caution. However, it seems that the IP strengthening is more valuable for instances having a narrow cost range. This yields another explanation for the new instances being significantly harder than the instances previously seen in literature. The same ratio for all the 8000 exactly solved BiMMAP instances is 11.48 per cent.

Chapter 7

The multicriteria minimum cost flow problem

Network flow problems are inherently multicriteria problems in nature and have as such received an increasing amount of interest within the multicriteria literature during the past 30 years. Applications within transportation planning faces conflicting criteria like minimization of cost for selected routes, minimization of arrival times at the destination points, minimization of deterioration of goods, maximization of safety, etc. Since these applications typically require flow values to be integer, one also has prominent examples showing the need to deal with multicriteria discrete optimization problems. Network flow problems are obviously a very good starting point for this research, since the network flow feasibility polyhedron is known to have the integrality property. I refer to Section 2.1.1 for a short introduction to the single criterion minimum cost flow problem and to Chapter 3 for general multicriteria optimization. For a more thorough introduction on these topics, appropriate references are Ahuja et al. [2], Ehrgott [39], and Steuer [152].

In this chapter a review of theoretical results and solution algorithms for the class of multicriteria minimum cost flow problems is provided. I have chosen not to consider any of the various special cases of the minimum cost network flow problem (shortest path, assignment, transportation problem, etc.). All of these subclasses have received considerable attention in the operations research or – more specifically – in the network design and routing literature and deserve their own reviews, since there exist various algorithms exploiting the special structure of those problems. For excellent existing reviews I refer the interested reader to for instance Current and Marsh [24], Current and Min [25], Skriver [149], and Ulungu and Teghem [160]. Although some of these survey papers include sections on the general multicriteria minimum cost flow problem, I consider it necessary to deal with this subject by itself to be able to elaborate more extensively on some details which could be of interest to the reader. I tried my best to include all

relevant literature which is available as publication in a journal or as a report on the internet. To any author whose paper I left out unintendedly, I apologize in advance.

This chapter is organized as follows. The mathematical formulation of the multicriteria minimum cost flow problem is introduced in Section 7.1. In Section 7.2 and Section 7.3, I review the exact and approximate methods for the multicriteria minimum cost flow problem with continuous flows and integral flows, respectively. In Section 7.4 the results on compromise solutions are discussed and Section 7.5 summarizes all results in tabular form.

7.1 Problem formulation

The *multicriteria minimum cost flow problem (MMCF)* can be concisely stated as the following mathematical programme:

$$\min\{(c^1, \ldots, c^Q)^T x \ : \ x \in \mathcal{P}_{flow}\} \ , \tag{7.1}$$

where $c^1, \ldots, c^Q \in \mathbb{N}_0^m$ are non-negative integer cost vectors. Note that the objective function $Cx = (c^1, \ldots, c^Q)^T x$ is composed of Q linear functions, whereas the decision space remains the same as for the classical minimum cost flow problem, previously denoted MCF. Obviously, MCF is a special case of MMCF (with $Q = 1$) and MMCF is a special case of MOLP. Another special case which is addressed in most papers on MMCF is the *bicriterion minimum cost flow problem (BiMCF)*. MMCF is in general a continuous problem, i.e. the flows x_{ij} may take on fractional values. If we want to enforce integrality, the *multicriteria minimum cost integer flow problem (MMCIF)*

$$\min\{(c^1, \ldots, c^Q)^T x \ : \ x \in \mathcal{X}_{flow}\}$$

is solved.

Both MMCF and MMCIF become easy if the componentwise ordering is replaced by the *lexicographical ordering*. In the latter case, two vectors are compared by looking at the first component in which they differ. The order of this component then defines the lexicographical order of the vectors. Virtually all algorithms for the classic minimum cost flow problem can be carried over to the lexicographical minimum cost flow problem (see for instance, in a more general context, Hamacher [65] or Calvete and Mateo [16]).

For the bicriterion minimum cost flow problem Ruhe [134] uses a pathological graph, introduced by Zadeh [169], to show that there can be an exponential number of supported extreme nondominated criterion points making even the continuous case *intractable*, in general. Furthermore, BiMCF is known to be a generalization of the bicriterion shortest path problem (BiSP) which is $\mathcal{NP}$-hard and has a $\#\mathcal{P}$-complete decision problem. These results obviously carry over to MMCF. More

on general complexity results for multicriteria optimization problems can be found in the book by Ehrgott [39].

7.2 The continuous multicriteria minimum cost flow problem

Similar to the situation in linear programming, there are many more papers on the continuous multicriteria minimum cost flow problem MMCF than on its integer counterpart. An obvious reason is that some of the techniques of MOLP can just be specialized to MMCF. First, I consider papers which aim at computing all efficient solutions. Then methods to find a representative system for the efficient set are discussed.

7.2.1 Exact methods

The papers reviewed in this section present algorithms for finding the entire set of efficient solutions of BiMCF – either in decision space or in objective space ([98, 101, 133, 145, 147]). I found no papers on an exact solution method for MMCF with more than two objectives. All but one paper ([147]) exploit jointly the topological and the graph theoretical connectivity of the set of efficient solutions, (see Section 3.1).

Basic feasible solutions of MMCF (subsequently called *basic flows*) correspond to spanning trees ([2]) and a pivot operation corresponds to the insertion of a non-tree edge into the tree and the deletion of an edge in the tree. Hence, two basic feasible flows are adjacent if their tree representations have $n-2$ arcs in common. Due to the graph theoretical connectivity property for MMCF, one can start with an arbitrary efficient basic feasible flow and obtain the entire efficient frontier by iteratively performing network pivot operations, i.e. by exchanging one arc in trees at a time.

Entering a nonbasic arc (s,t) in the tree representation of an efficient basic flow x yields a unique cycle O along which δ_{st} units of flow can be sent before a new basic feasible flow $\hat{x}$ is obtained. Note $\delta_{st} = \min\{r_{ij} : (i,j) \in O\}$ equals the minimal capacity of the arcs in O in the incremental graph $G(x)$. The only flow change between x and $\hat{x}$ is on the arcs of this unique cycle. Only the introduction of nonbasic arcs that have out-of-kilter status with respect to at least one of the criteria can lead to other efficient flows, [41]. If two adjacent basic feasible flows x and $\hat{x}$ are both efficient, then so are all of their convex combinations. Any such convex combination can be found for $\delta_{st} > 0$ by sending a suitable amount of flow $0 \leq \delta < \delta_{st}$ through the cycle. In terms of polyhedral structure of MMCF this corresponds to moving along the *efficient edge* connecting x and $\hat{x}$ in $\mathcal{P}_{flow}$, [98].

Malhotra and Puri [101] provide a generalization of the out-of-kilter method to solve BiMCF with a uniform capacity for all arcs, i.e. $u_{ij} = \eta$ for all arcs

$(i,j) \in A$. As the authors state, this idea can be modified to address general BiMCF. The efficient frontier is built in a left-to-right fashion, starting with the lexicographical minimum for the first objective. Due to the definition of a lexicographical minimal flow, all arcs are in-kilter with respect to the first objective. If BiMCF does not have an ideal point minimizing both objectives simultaneously, some arcs are out-of-kilter for the second objective. The algorithm strives for an efficient flow with all arcs being in-kilter for the second objective, i.e. a lexicographical minimal flow for the second objective. Therefore, for a given flow, the arcs being out-of-kilter for the second objective (and in-kilter for the first objective) are said to be *eligible arcs*. For each eligible arc, Pareto cycles including this arc are found in the incremental graph of the current flow and added to the current flow. The obtained flows are argued to be efficient points on the frontier of BiMCF. This argumentation can easily be seen to be wrong (see Section 9.2), and in fact not only the generation of dominated objective vectors but also possible oversights of extreme points on the efficient frontier might occur. Even for the small network given by Malhotra and Puri themselves, it can be seen that an extreme nondominated criterion point exists, which is not found by the proposed algorithm. The incorrect output of the algorithm is partly due to the fact that only one set of dual node variables (node potentials) is introduced.

This mistake is corrected by – among others – Lee and Pulat [98], Pulat et al. [133], and Sedeño-Noda and González-Martín [145] who use one set of dual variables for each objective. These papers are based on the idea of Gass and Saaty [57] and its generalization by Geoffrion [58] that was also utilized in Chapter 6. Corresponding to (3.4), BiMCF is formulated as the parametric minimization problem

$$\begin{aligned} \min \;\; & f_\lambda(x) = \left(\lambda c^1 + c^2\right) x \\ \text{s.t.} \;\; & x \in \mathcal{P}_{flow} \,, \end{aligned} \tag{7.2}$$

where $0 \leq \lambda \leq \infty$ (or $\delta \leq \lambda \leq \phi$ where δ and ϕ are suitably chosen lower and upper bounds). It is well known that all optimal solutions for (7.2) are efficient solutions of BiMCF, and that conversely all efficient solutions of BiMCF can be found as optimal solutions of some (7.2) problem, [58].

Similar to [101], the algorithm by **Lee and Pulat** [98] implements a revised version of the out-of-kilter method. Initially uniform weights are placed on both objectives and the resulting single criterion minimum cost flow problem, $\min\{(c^1 + c^2)x : x \in \mathcal{P}_{flow}\}$ is solved by the out-of-kilter method after which the flow is adjusted to be basic by a rerouting procedure or node price adjustment procedure. Then, from this compromise solution, the frontier is searched to the left by considering arcs that are out-of-kilter with respect to the first objective and to the right by considering arcs that are out-of-kilter for the second objective. To perform the move from one basic flow to another, the labeling procedure from the single criterion out-of-kilter method is modified and a procedure for ensuring the attainment of a basic feasible flow is applied. The arc entering the basis

is chosen upon a determination of the smallest ratio between reduced costs for the two criteria. This corresponds to choosing an arc that results in the steepest slope of the line between two consecutive points on the efficient frontier. Since it is mistakenly believed that efficient basic feasible flows necessarily yield extreme nondominated criterion points, the authors neglect to focus on the case of multiple nonbasic arcs having the same minimal value of the ratio of the two reduced costs. Consequently, possible nonextreme nondominated points are generated.

The algorithm by **Pulat, Huarng, and Lee** [133] is conceptually the same as the one by Malhotra and Puri, since it again utilizes adjacency results for searching the frontier in a left-to-right fashion. Apart from the addition of a second set of node prices, another distinction is that Pulat et al. use the network simplex method to solve BiMCF in its parametric programming formulation (7.2). Pulat et al. realize that some efficient basic feasible flows may not correspond to extreme nondominated criterion points. However, since the authors aim at a complete description of all efficient basic feasible flows, in case of several arcs yielding a minimal ratio of the reduced costs, all of them are introduced in the current basic feasible flow. As noted above, all nonbasic efficient flows can be found by sending flow in the unique cycle between any two adjacent efficient basic feasible flows. Hence, a complete description of all efficient basic feasible flows is required to find an explicit expression for the entire efficient set.

Sedeño-Noda and González-Martín [145] build on the ideas of the three preceding papers. The parametric problem in (7.2) is solved in a left-to-right fashion using the network simplex method for the single criterion optimizations. The desired output of the algorithm is the set of extreme nondominated criterion points. In each iteration from a list Γ of arcs yielding the minimal ratio of the reduced costs, one arc is chosen to enter the basic tree of the current efficient basic feasible flow x. This will result in a new efficient basic feasible flow $\hat{x}$ which corresponds either to an extreme nondominated point or a nonextreme nondominated point. The former case is identified if all other arcs in Γ fulfill the optimality condition of the second objective, and in this case the obtained point is stored. In the latter case, arcs violating the optimality condition for the second objective still exist in Γ, and one of these is introduced into the basic tree of $\hat{x}$ yielding a new efficient basic feasible flow. If the list Γ is empty, an extreme nondominated point has been identified and a new list of arcs is built. The algorithm stops if no more arcs are out-of-kilter with respect to the second objective.

The papers reviewed above all exploit the graph theoretical connectivity of the efficient set. In contrast, **Sedeño-Noda and González-Martín** [147] modify a method proposed by Aneja and Nair [4] which iteratively applies the weighted sum problem. The method was originally designed for the bicriterion transportation problem. The method by Sedeño-Noda and González-Martín starts by finding the two lexicographical minima. For each two identified consecutive points on the frontier, a search for additional extreme nondominated criterion points in the triangle between these points and the corresponding local ideal point (y_1^*, y_2^*), is

performed using Lagrangian theory and the network simplex method. The algorithm finds the extreme point closest to the line between the ideal point and some appropriately chosen aspiration level vector e with $e_j > y_j^*$, $j = 1, 2$. Whenever a new point is identified, two new areas are introduced for future search. The output of the procedure is the set of all extreme nondominated criterion points.

Four of the five described algorithms have been implemented, all using NETGEN [89] as problem generator. In fact, three of the algorithms have been implemented by Sedeño-Noda and González-Martín on the same sample sets making a direct comparison possible [98, 145, 147]. In total, 180 networks (five replications of 36 set-ups) with the number of nodes ranging between 25 and 40, the number of arcs ranging from 100 to 400 and the cardinality of maximum capacity on the arcs ranging from 10 to 100000 were considered. For these examples, the method by Sedeño-Noda and González-Martín [145] proved superior in all instances. The method by Sedeño-Noda and González-Martín [147] performs better than the method by Lee and Pulat [98] for sparse networks. Pulat et al. [133] test their algorithm on 90 networks grouped in three different set-ups with the number of nodes ranging from 10 to 100 and the number of arcs ranging from 30 to 500.

7.2.2 Approximation methods

As mentioned in Section 7.1, Ruhe [134] shows that the exact computation of the efficient frontier is, in general, intractable, since there may exist an exponential number of extreme nondominated criterion points. Therefore several approaches are proposed to find representative subsets of the efficient frontier. All these approaches aim at the development of computationally appealing algorithms. The literature following this thought can be divided into two groups: The first group argues for the computation of a limited number of points leading to a final preferred solution in reasonable computation time. I have chosen to present this group in Section 7.4 along with the algorithms finding compromise solutions for MMCIF. The second group approximates the frontier as a whole with provable good quality. All approaches in the second group are applicable to BiMCF only and utilize sets $\mathcal{L}$ and $\mathcal{U}$ which "sandwich" the efficient frontier (see Figure 7.1), i.e. $\mathcal{Y}_N \subseteq \big((\mathcal{L} \oplus \mathbb{R}^2_{\leqq}) \cap (\mathcal{U} \oplus (-\mathbb{R}^2_{\leqq}))\big)$.

All articles using the sandwich idea for BiMCF follow the algorithm of **Burkard, Hamacher, and Rote** [14] for approximating univariate convex functions $g : \mathbb{R} \to [a, b]$. Therefore this idea is presented in its general framework.

It is assumed that, for any $z \in [a, b]$, the left and right derivatives $g^-(z)$ and $g^+(z)$ are obtainable. Let $a = z_1 < z_2 \cdots < z_n = b$ be a finite partition of the interval $[a, b]$. The points z_i, $i = 1, \ldots, n$, are referred to as *breakpoints*. Two piecewise linear functions $u(z)$ and $l(z)$ approximate $g(z)$ from above and from

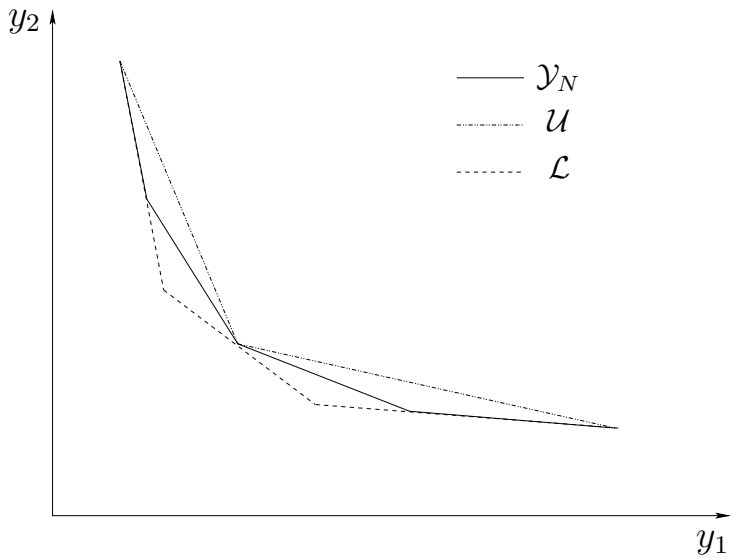


Figure 7.1: Illustration of sandwich idea.

below, where

$$u(z) := g(z_i) + \frac{g(z_{i+1}) - g(z_i)}{z_{i+1} - z_i}(z - z_i) \tag{7.3}$$

$$l(z) := \max\{g(z_i) + g^+(z_i)(z - z_i),\ g(z_{i+1}) + g^-(z_{i+1})(z - z_{i+1})\} \tag{7.4}$$

for $z_i \leq z \leq z_{i+1},\ i = 1, 2, \ldots, n-1$. At any stage of the algorithm, $l(z)$ and $u(z)$ satisfy $l(z) \leq g(z) \leq u(z)$ for all $z \in [a, b]$.

The error of the current approximation is measured in [14] by $\max\{u(z) - l(z) : z \in [a, b]\}$. Let z_l and z_r be two consecutive breakpoints in the current partition satisfying $z^* \in [z_l, z_r]$ where $z^* = \arg\max\{u(z) - l(z) : z \in [a, b]\}$. Thus $[z_l, z_r]$ is an interval with the largest error. A new breakpoint $z_{new} := (z_r - z_l)/2$ is added to the current finite partition of the interval $[a, b]$, and the value $g(z_{new})$ is found. This choice of the new breakpoint is referred to as the *interval bisection rule*. The approximating functions $u(z)$ and $l(z)$ are updated due to equations (7.3) and (7.4). This scheme is iteratively repeated until the error of the approximation falls below a prescribed value. It is shown that the approximating functions $l(z)$ and $u(z)$ converge uniformly to $g(z)$. Furthermore, the error of the approximation decreases quadratically with the number of breakpoints and, given an accuracy level, an upper bound on the number of breakpoints needed for satisfying this accuracy level can be obtained.

When transferring the interval bisection rule to BiMCF, one objective function is treated as the independent variable z. The other objective function plays the role of $g(z)$. In other words, one objective function is considered a convex function of the other objective function. Note that evaluating $g(z_{new})$ for some given z_{new}

corresponds to minimizing one objective function subject to a fixed value of the other objective function (which is known as the ε-constraint method). In terms of BiMCF this means solving MCFs with one additional side constraint.

Ruhe [135] (see also the monography of Ruhe [136]) introduces the Hausdorff distance between the lower and upper approximation to measure the error of the approximation. For $\mathcal{L} := \{(z, l(z)) \ : \ z \in [a, b]\}$ and $\mathcal{U} := \{(z, u(z)) \ : \ z \in [a, b]\}$ the Hausdorff distance between $\mathcal{L}$ and $\mathcal{U}$ is

$$d(\mathcal{L},\mathcal{U}) := \max\Big\{ \sup_{y\in\mathcal{L}} \inf_{\hat{y}\in\mathcal{U}} ||\hat{y} - y||_2, \sup_{\hat{y}\in\mathcal{U}} \inf_{y\in\mathcal{L}} ||\hat{y} - y||_2 \Big\} \ . \tag{7.5}$$

In contrast to the error measure in [14], the Hausdorff distance is invariant under rotation and does not favour one objective function over the other.

In Ruhe [135], new breakpoints are generated by applying the so-called *chord rule*, which was originally introduced by Aneja and Nair [4] in the context of computing the efficient extreme points of a bicriterion transportation problem. Given an interval $[z_l, z_r]$ with the maximal error, the new breakpoint is computed by

$$z_{new} := \arg\min\{g(z) - \alpha \cdot z \ : \ z \in (z_l, z_r)\} \tag{7.6}$$

where α is the slope of the upper approximating function $u(z)$, $z \in [z_l, z_r]$.

Using the chord rule for the approximation of $\mathcal{Y}_N$ for BiMCF, a weighted sum problem (which corresponds to solving MCF) has to be solved in each iteration which is in general easier than solving an ε-constraint problem.

Fruhwirth, Burkard, and Rote [50] introduce two new rules, the *angle bisection* and the *slope bisection rule* for generating breakpoints. As in the chord rule, the new breakpoint is computed by (7.6), but the rules vary in the choice of the parameter α. For the angle bisection rule, α equals the slope of the bisector of the outer angle of the triangle formed by the graphs $(z, u(z))$ and $(z, l(z))$, $z \in [z_l, z_r]$ in the cutting point $\tilde{z}$ between the two linear functions that determine $l(z)$ in $[z_l, z_r]$. In case of the slope bisection rule, α is the mean of the slopes of the two linear functions that determine $l(z)$ in $[z_l, z_r]$.

For chord, angle bisection and slope bisection rule, the error decreases quadratically with the number of breakpoints. Consequently, an upper bound on the number of MCF evaluations can be derived to obtain a given accuracy level when applying any of the three rules (see [50, 135]).

Burkard, Rote, Ruhe, and Sieber [13], Fruhwirth et al. [50], and Ruhe [136] report on numerical studies comparing different partition rules. The chord rule and the angle bisection rule are compared with a special implementation of the angle bisection rule which exploits the bound on the number of MCFs that have to be solved to construct an approximation with a desired accuracy level. A primal network simplex algorithm is employed to solve the occuring MCFs. Nine network instances were generated with $n = 100, 400, 800$ and $m = 10n$. Capacities and costs for the arcs were taken independently at random from the intervals

$\{10, \ldots, 500\}$ and $\{1, \ldots, 500\}$, respectively. The special implementation of the angle bisection rule outperforms the angle bisection as well as the chord rule in terms of memory and time consumption.

A derivative-free modification of the sandwich approximation approach was proposed by **Yang and Goh** [167]. For each interval $[z_i, z_{i+1}]$, the upper approximating function is computed as in (7.3). The lower approximating function consists of a piecewise linear function that is parallel to the upper approximation. New breakpoints are computed with the chord rule. The algorithm is applied to bicriterion quadratic minimum cost flow problems.

In **Ruhe and Fruhwirth** [137], the sandwich algorithm is used in order to compute an ε-optimal approximation for BiMCF. Here, a subset $\mathcal{T} \subset \mathcal{P}_{flow}$ is called ε-optimal if for all $x \in \mathcal{P}_{flow}$ there is a solution $\hat{x} \in \mathcal{T}$ such that $c^i\hat{x} \leq (1+\varepsilon)c^i x$ for $i = 1, 2$. Notice the equivalence to the concept of an ε-approximation from Definition 6.2 on page 67. In their pseudo-polynomial time algorithm, Ruhe and Fruhwirth modify the rule for determining additional breakpoints. Instead of solving only one MCF as in all previous sandwich algorithms, two MCFs are solved in each iteration. Two general questions are addressed: Given a value ε, an ε-optimal set of small cardinality is determined and, given the cardinality of $\mathcal{T}$, an ε-optimal set having a high level of accuracy is computed. Ruhe and Fruhwirth compare two realizations of their algorithm numerically. Networks are generated with $n = 600$ and $m = 6000,\ 9000,\ 12000$ and $n = 900$ and $m = 9000,\ 18000$, respectively. The results are averaged over twenty instances for each of the five network sizes. Capacities and costs are chosen randomly distributed in $\{500, \ldots, 5000\}$ and $\{1, \ldots, 1500\}$, respectively. The numerical study shows that already for relatively small sized approximating sets, the accuracy level is quite high.

7.3 The multicriteria minimum cost integer flow problem

All approaches for finding all integer efficient solutions of MMCIF address problems with two objective functions only – *the bicriterion minimum cost integer flow problem (BiMCIF)*. The algorithms are comprised of two phases, resembling the two-phase method for bicriterion discrete optimization problems described in Section 3.2.1.

Phase 1: Find all supported integer efficient flows.

Phase 2: Find all unsupported integer efficient flows.

Some of the papers concentrate on Phase 1 only. These approaches can be considered as approximations of the integer nondominated set. Analogously to the structure in Section 7.2, these approximation approaches are discussed in Section 7.3.2, while the exact approaches are discusssed in Section 7.3.1. When presenting the latter group of papers, I therefore focus on Phase 2 only assuming that Phase 1

has already been solved. Several general ideas are employed to implement Phase 2. Firstly, structural results are used such as the fact that each efficient flow is an extreme efficient solution of BiMCIF with modified capacities. Secondly, it is exploited that consecutive supported nondominated points on the efficient frontier define triangles with remaining nondominated points restricting the search area for undiscovered nondominated points. Thirdly, well-known concepts from different research areas like the ε-constraint method and the branch-and-bound method are combined and applied in the bicriterion flow context.

Three approaches for finding all efficient/nondominated points for BiMCIF are presented subsequently. Although literature shows that two of them lack in argumentation, they contain a thorough analysis of some structural properties of BiMCIF and I have therefore chosen to present their main ideas. Below, I discuss some general results for BiMCIF which will be needed in the descriptions of the algorithms in Sections 7.3.1 and 7.3.2.

Lee and Pulat [99] elaborate on the cycle relationship among basic feasible flows and investigate the structure of solutions of BiMCIF. Let x^r and x^{r+1} be two efficient basic solutions of BiMCF with criterion points y^r and y^{r+1} which are graph theoretically adjacent in $\mathcal{Y}_N$. Due to the total unimodularity of the constraint matrix, x^r and x^{r+1} are integral. Consider the basic tree T of x^r and let (s,t) be the nonbasic arc whose inclusion in T yields x^{r+1}. It is well-known that the union of T and (s,t) contains a unique cycle. Let $\delta_{st} \in \mathbb{N}_0$ denote the maximum amount of flow that can be sent along this cycle. The number of supported integer nonextreme nondominated points on the line connecting y^r and y^{r+1} is $\delta_{st} - 1$. Furthermore, such integer points are equidistant from each other, [99, 133]. Figure 7.2 illustrates this result. Here, $\delta_{st} = 3$. Augmenting the cycle induced by (s,t) by 1 and 2 units results in the points y' and y'', respectively.

Let $(i,j) \neq (s,t)$ be a nonbasic arc in the bases of x^r and x^{r+1} whose arc flow x_{ij} is equal to its lower bound l_{ij}. Assume it is feasible to introduce arc (i,j) into the bases of x^r and x^{r+1}. Let $x^{r'}$ and $x^{r+1'}$ be the resulting flows when sending one unit of flow along the cycles originating from adding arc (i,j) to the basic trees of x^r and x^{r+1}, respectively. Then, $x^{r'}$ and $x^{r+1'}$ are efficient flows for a modified BiMCF with respect to an increased lower bound $l'_{ij} := l_{ij} + 1$. Note that the bases of x^r and $x^{r'}$ (x^{r+1} and $x^{r+1'}$) are the same. Hence, it is possible to introduce the nonbasic arc (s,t) in the basis of $x^{r'}$ to obtain $x^{r+1'}$, (see Figure 7.2).

Let $y^p := Cx^p$ for $p = r, r', r+1$, and $r+1'$. It is shown in Lee and Pulat [99] that the slopes of the two lines joining y^r and y^{r+1} and $y^{r'}$ and $y^{r+1'}$, respectively, are identical. Furthermore, the number of integer flows lying on each of these lines differ by at most one. The distances between these integer points lying on the parallel lines are the same. In Figure 7.2, the number of points on the line connecting $y^{r'}$ and $y^{r+1'}$ is one less than the number of points on the line connecting y^r and y^{r+1}. The distances between points on these lines are identical, e.g. $||y^{r'} - y'''||_2 = ||y^r - y'||_2$.

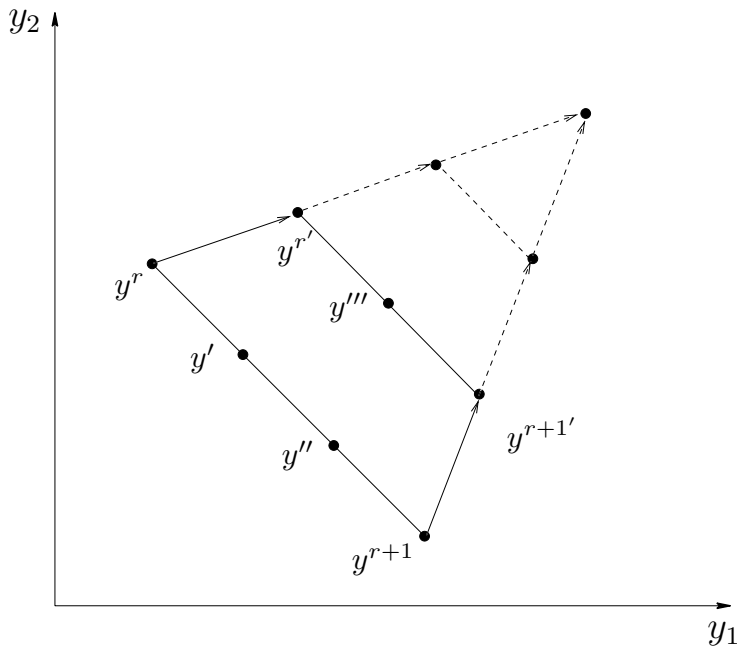


Figure 7.2: Illustration of the structure of $\mathcal{Y}$.

Note that these results can be generalized when increasing the lower bound l_{ij} of nonbasic arc (i,j) by more than one unit assuming feasibility is maintained. These generalizations are indicated by the dashed lines in Figure 7.2. Obviously, analogous arguments can be used if $x_{ij} = u_{ij}$ instead of $x_{ij} = l_{ij}$. Based on this idea, Lee and Pulat [99] prove that any efficient integer flow $x \in \mathcal{X}_E$ can be obtained as a supported efficient solution with respect to modified lower and upper capacities on some arcs. Ehrgott [41] generalizes this result to MMCIFs with more than two objectives.

7.3.1 Exact methods

Lee and Pulat [99] assume that all supported extreme nondominated points and all other supported nondominated points are explicitly known. The general results above are used to compute candidate points located inside the triangles defined by consecutive supported nondominated points. Instead of explicitly testing all combinations of modified lower and upper capacities, an implicit search for efficient flows is performed.

For each supported integer efficient extreme point x^r, it is checked whether a nonbasic arc $(i,j) \neq (s,t)$ can enter the basic tree of x^r possibly leading to an unsupported efficient solution $x^{r'}$. Integer points on the line that connects $y^{r'}$ and $y^{r+1'}$ are generated as well. Rules are established that exclude nonbasic arcs from consideration which yield candidates that are dominated by previously found candidates. For example, it is observed that if $y^{r'} \in \{y^r\} \oplus \mathbb{R}^2_{\underline{\geqq}}$ and the

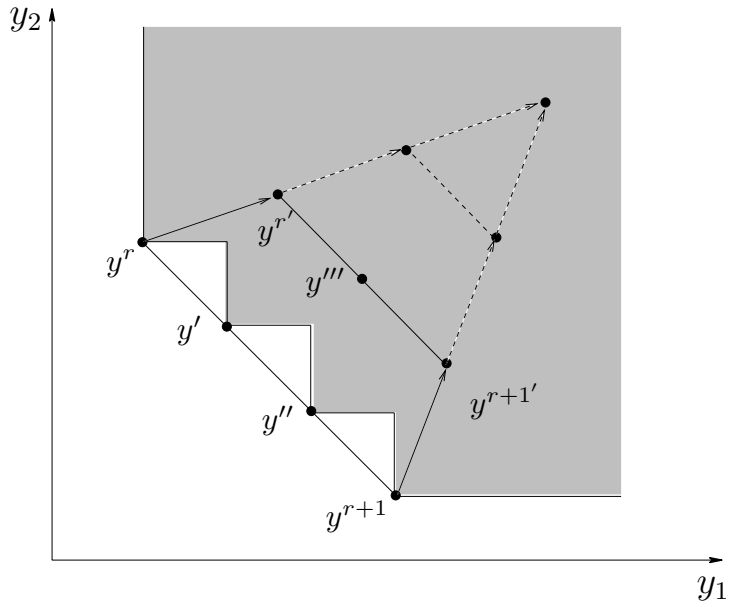


Figure 7.3: Illustration of a rule eliminating dominated points.

number of integer points on the line connecting $y^{r'}$ and $y^{r+1'}$ is less than or equal to the number of integer points on the line connecting y^r and y^{r+1}, then all points resulting from introducing (i, j) to the basic tree of x^r are dominated (see Figure 7.3).

Thus, not all feasible flows are generated – i.e. the search is, indeed, implicit. After all candidates have been generated by checking all supported efficient extreme points and all possible nonbasic arcs, a filtering process deletes dominated points from the candidate set. It is claimed that the remaining points are nondominated and all nondominated points have been found. The algorithm was implemented and tested on six different problem set-ups. The number of nodes and arcs vary between 10 to 20 and 20 to 50, respectively. Five different instances have been randomly generated and solved for each set-up. For more details, see the PhD thesis by Lee [97].

Huarng, Pulat, and Ravindran [79] extend the algorithm by Lee and Pulat focusing on degeneracy phenomena.

Sedeño-Noda and González-Martín [146] claim that the algorithm stated in [99] – and thus also in [79] – is incorrect. They argue that Lee and Pulat might miss some efficient flows since their algorithm introduces only two nonbasic arcs (i, j) and (s, t) at a time whereas more than two arcs are needed in general. Sedeño-Noda and González-Martín use a left-to-right approach starting with the efficient solution that is optimal with respect to objective function c^1 to find unsupported efficient integer flows. Each efficient flow found by the algorithm is associated with a set of nonbasic candidate arcs, the introduction of which might possibly yield an unsupported efficient flow. Filtering rules similar to those in [99]

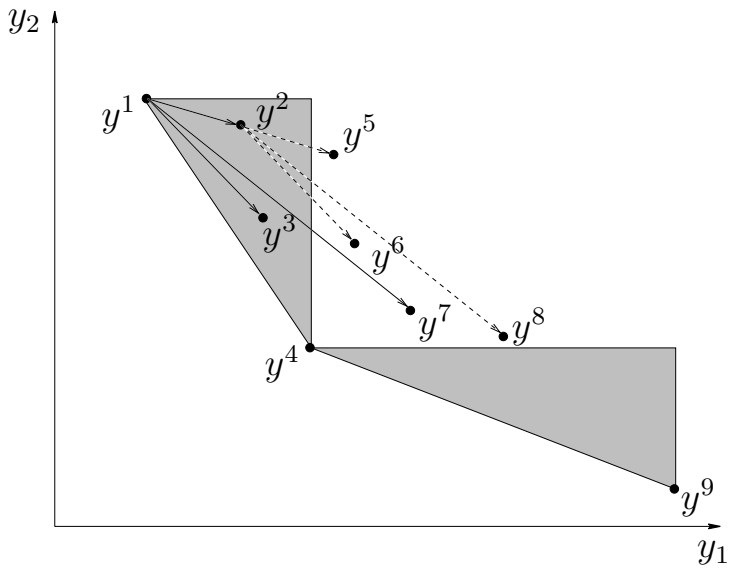


Figure 7.4: Illustration of the algorithm by Sedeño-Noda and González-Martín [146].

are utilized to reduce the number of nonbasic arcs in these candidate sets that are stored for further investigation. In each iteration, a new integer efficient flow x^{new} is obtained by adding a nonbasic arc (i, j) to the basic tree of a previously found efficient flow x^{old}. The left-to-right approach requires that Cx^{new} has the smallest c^1-value among all undiscovered nondominated criterion points. This is implemented by checking all known efficient flows with non-empty candidate sets. Among all combinations of known efficient flows and nonbasic candidate arcs, the one yielding the nondominated point with smallest c^1-value is chosen. The candidate set of x^{old} is assigned to x^{new}, while the arc (i, j) is deleted from the candidate set of x^{old}. Doing this ensures that combinations of nonbasic variables are taken into account when searching for unsupported nondominated points. The procedure stops when all candidate sets are empty.

Figure 7.4 illustrates the algorithm: Suppose that points y^1, y^4, and y^9 are the points obtained after Phase 1. The candidate nonbasic arcs associated with point y^1 yield points y^2, y^3, and y^7. In the first iteration, point y^2 is generated since it has smallest c^1-value among all undiscovered nondominated points. The candidate set of y^2 is the same as the one of y^1 as indicated by the dashed arrows. [146] do not investigate the points y^5, y^6, and y^8 any further, since these points are dominated.

In their numerical tests, 180 random samples are generated with 10 to 25 nodes and 20 to 100 arcs. Sedeño-Noda and González-Martín observe that the number of unsupported nondominated criterion points increases with the size of the networks. In their study, some 90% of all efficient flows are unsupported.

Przybylski, Gandibleux, and Ehrgott [132] show the incorrectness of the

approach suggested by Sedeño-Noda and González-Martín, since the latter authors implicitly assume that the adjacency graph is connected. But [132] gives a counterexample proving that the graph theoretical connectivity property does not hold for BiMCIF. They conclude that it is not possible to find all nondominated objective vectors for BiMCIF with a straightforward application of the network simplex algorithm. The result of [132] indicates that a different definition of an adjacency graph is needed to establish the graph theoretical connectivity result for MMCIF. One way to do this is based on Theorem 9.3 presented in Section 9.2. There it is proved that any efficient flow x can be obtained from another efficient flow x' by addition of an efficient incremental circulation from the incremental graph of x'. Furthermore, it is shown that no such addition of an efficient flow and an efficient circulation can return a non-efficient flow. Hence the modified adjacency graph containing all integer efficient flows is complete and thus trivially connected.

Figueira [47] proposes a branch-and-bound approach to find all nondominated criterion points for BiMCIF. Each pair of supported extreme nondominated points defines a triangle in the objective space where unsupported nondominated points may be located. Each of these triangles are searched for unsupported nondominated points in a left-to-right approach. The efficient flows found by the algorithm are stored in a list Φ with decreasing c^2-values. The ε-constraint problem (7.7) is solved using a branch-and-bound method to decide whether there are unsupported nondominated points having c^2-values less than the first element $\hat{x}$ and larger than the second element in the list Φ.

$$\begin{aligned} \min \ & c^1 x \\ \text{s.t.} \ & c^2 x \leq c^2 \hat{x} - 0.5 \\ & x \in \mathcal{X}_{flow} \end{aligned} \tag{7.7}$$

All subproblems occuring in the branch-and-bound procedure are stored in a list Ψ. Problem (7.7) is the current problem and its continuous relaxation is solved. Let x^{rel} denote the optimal value of the relaxed problem. If x^{rel} is integer, the optimal solution of (7.7) is found and x^{rel} is known to be efficient. If not, branching is performed to find an integer solution as described in the following. Note that $y^{rel} = Cx^{rel}$ is a vector located on the efficient frontier and can thus be determined by sending a fractional value along the cycle that connects the two nondominated extreme points in the triangle of interest. Let (s,t) denote the arc that defines this cycle. Two new problems are set up by introducing to (7.7) the constraints $l_{st} \leq x_{st} \leq \lfloor x^{rel}_{st} \rfloor$ and $\lfloor x^{rel}_{st} \rfloor + 1 \leq x_{st} \leq u_{st}$, respectively. The costs of the continuous relaxations of these subproblems are computed using the network simplex algorithm. Both problems are added to the list Ψ and branching is now performed on these two problems. Three rules are set up for fathoming: A subproblem is discarded from consideration when it has no feasible solution or when its cost is greater than the cost of the current best solution. If a subproblem has an integer

solution $\tilde{x}$, with a cost lower than the cost of the current best solution, then all subproblems having a cost greater than $c^1\tilde{x}$ are removed from Ψ and $\tilde{x}$ becomes the current best solution. The algorithm iterates until an optimal integer solution to problem (7.7) is found. Whenever an optimal solution of (7.7) is found, the first element of Φ is deleted. If the solution of (7.7) is a new efficient solution, it is inserted in the front of Φ and the algorithm proceeds.

7.3.2 Approximation methods

Two lines of thoughts are dividing the authors providing a representation set for the solutions of MMCIF. Some interpret this problem as the calculation of all integer efficient points on the efficient frontier [98, 117]. Others compute only a limited amount of compromise solutions [17, 41, 66, 108, 115, 116]. The latter group of papers will be reviewed in Section 7.4.

Compared with algorithms looking for all efficient solutions of BiMCIF, papers dealing with the determination of the integer flows on the efficient frontier of BiMCF face a conceptually easier task. In fact, all integer flows on the efficient frontier can be found by methods designed for solving the continuous BiMCF only involving considerations of network cycles already discussed in Section 7.2, in particular using the ideas of [98, 133].

Lee and Pulat [98] extend their algorithm for determining all efficient extreme points in decision space for the continuous BiMCF to find all integer efficient points on the efficient frontier. While moving from one efficient extreme solution, x^r with objective value y^r to the graph theoretically adjacent efficient extreme solution x^{r+1} with objective value y^{r+1}, an integer value δ_{st} is sent along the nonbasic arcs (s,t). All intermediate integer points on the frontier line piece $[y^r, y^{r+1}]$ are determined as

$$y^r + q \cdot \frac{y^{r+1} - y^r}{\delta_{st}}, \text{ where } q = 1, \ldots, \delta_{st} - 1 \; .$$

Nikolova [117] develops an algorithm finding all supported integer efficient solutions of BiMCF with a designated source node s and sink node t. A flow of ρ units are to be shipped from s to t. The following definition introduces some terminology also to be used in Section 9.2.

Definition 7.1 In a network with a designated source node s and sink node t, a feasible flow x sending ρ units from s to t has *(flow) value* $v(x) = \rho$.

Nikolova proves that each supported efficient solution of BiMCIF, with flow value ρ, can be obtained as a sum of a basic feasible solution x' with flow value $v(x) = \rho$ and an efficient circulation flow of value 0 in the incremental graph $G(x')$. Actually, a stronger result is presented in Theorem 9.3 in Section 9.2, from which the result by Nikolova can be derived easily.

7.4 Finding compromise solutions

A group of authors refer to the theoretical complexity of generating all efficient solutions of MMCF and MMCIF proven by Ruhe [134]. The difficulty faced by a decision maker to choose from a large set of offered efficient flows is also discussed. Based on this insight, several methods for generating only a limited amount of compromise solutions are proposed.

Obviously, it is easy to compute just a single efficient solution, since lexicographical minimum cost flows are known to be efficient. Although an arbitrary lexicographical solution is in general not a good compromise, **Calvete and Mateo** ([16, 17]) elaborate on this idea.

Calvete and Mateo [16] deduct complementary slackness optimality conditions for the lexicographical ordering and specify two algorithms for finding a lexicographical minimum cost flow: lexicographical generalizations of the out-of-kilter method and of the primal-dual method, (see [2]). Numerical comparisons (using test networks with $Q = 2$, 3, 5, and 15 objective functions and n and m varying between 27 and 102 and 250 and 1000, respectively) between these algorithms indicate that the out-of-kilter algorithm outperforms the primal-dual method. An application in water resource planning is discussed.

A sequential approach for solving the lexicographical MMCF problem is introduced in [17]. First, the network flow problem regarding the objective function with the highest priority is solved by using some MCF algorithm. The flow on arcs having non-zero reduced costs is fixed to the current value. This ensures that the current flow remains optimal for the objective function currently under consideration throughout the algorithm. The flow structure is still preserved when fixing variables. The reduced network flow problem is then solved regarding the objective function with the second highest priority, and so on. This iterative procedure terminates when all objective functions have been considered or when the flow is fixed to some value on every arc. Two different strategies for fixing flow on arcs are specified and numerically compared using the same instances as in Calvete and Mateo [16]. The solution obtained with either of the algorithms by Calvete and Mateo is a supported integer efficient flow.

Nikolova [115, 116] develops two interactive algorithms for finding an integer compromise solution of MMCF with side constraints. In [115] the side constraints are imposed initially by a decision maker but may be altered during the algorithm to obtain feasibility. In [116] the side constraints are imposed during the algorithm.

Nikolova [116] initially finds a solution of the weighted sum method with equal weights. In each iteration of the algorithm the decision maker is asked to classify which criteria should be improved, which criteria can be worsened and which criteria should be kept at the same value. Exploiting the same ideas of fixing flow on arcs and reducing the considered network as Calvete and Mateo [17], smaller MMCFs and BiMCFs are solved using the weighted sum method and a left-to-right approach resembling the one found in Pulat et al. [133], respectively.

Nikolova [115] solves in each iteration of the interactive algorithm a BiMCF by a left-to-right approach. The two objectives to be considered are a weighted sum of already considered objectives and an objective for which the decision maker's constraints have not been fulfilled, respectively. If infeasibility is revealed during the algorithm, the decision maker is asked to change the requirements. The solution obtained with either of the algorithms by Nikolova is a supported integer efficient flow.

Hamacher [66] develops a polynomial time algorithm for solving the K best integer MCF, (see also Chapter 5). This algorithm is extended to a solution procedure for the *max ordering* (MO) flow problem that minimizes the worst of the single objective functions. Considering as before Q objectives to be minimized simultaneously, the MO flow problem can formally be stated as

$$\min_{x \in \mathcal{X}_{flow}} \max_{q=1,\ldots,Q} c^q x \ .$$

It is noted that there exists at least one flow which is both a MO flow and efficient. The crucial part of the developed ranking algorithm is to obtain the second best solution x^2 for a MCF, given the best solution, x^1. This is achieved utilizing the network structure by determination of a proper minimal cycle O in the incremental graph of x^1, $G(x^1)$. This cycle is added to x^1 yielding x^2. The tree structure of the algorithm is built by consecutive changes of lower and upper arc bounds, ensuring intactness of the network structure. By applying the ranking method for a weighted sum of the Q objectives, a lower bound for the MO flow problem is improved iteratively until an integer MO solution has been identified.

Ehrgott [41] extends the ideas of Hamacher [66] in the search for a solution to the integer lexicographical max ordering network flow problem. A lexicographical max ordering solution is known to be a max ordering flow as well as an efficient flow. This is based on the fact that if $x^1, \ldots, x^K$ are the K best flows with respect to the first objective, i.e. $c^1x^1 \leq c^1x^2 \leq \cdots \leq c^1x^K \leq c^1x$, $\forall x \in \mathcal{X}_{flow} \backslash \{x^1, \ldots, x^K\}$, then $c^1x^K > \min_{x \in \mathcal{X}_{flow}} \max_{q=1,\ldots,Q} c^q x$ implies that all max ordering solutions are contained in $\{x^1, \ldots, x^K\}$. Therefore, Ehrgott proposes to identify the lexicographical max ordering solution by applying the ranking method for MCF of [66] followed by a suitable sorting and selection of the obtained solutions.

Mustafa and Goh [108] find an integer efficient compromise solution to BiMCIF and to the *tricriterion minimum cost integer flow problem (TMCIF)*, respectively. The software package DINAS [118], originally designed for solving transshipment problems involving facility location, is used to find a non-integral efficient compromise solution. An integral solution of BiMCIF is obtained by rounding all non-integral flow values on arcs to their nearest integer value. This is justified by the fact that non-integrality in an efficient solution for the bicriterion case can only occur on a unique cycle between two adjacent efficient extreme points with criterion points on the efficient frontier. Only in the case where the deviation from

the nearest integer value is 1/2, the cycle needs to be traced to get the correct flow direction on the arcs. The rounding process corresponds to sliding to the nearest integer point on the efficient frontier, and hence the identified solution is a supported integer efficient flow. The result is generalized to TMCIF obtaining also a supported efficient solution. The proposed algorithm is applied to the practical problem of assigning courses to faculty members.

Figueira, M'Silti, and Tolla [48] provide an interactive method for finding a robust solution of MMCIF. An initial solution is found applying the augmented weighted Tchebycheff method [152, 153]. When using the augmented weighted Tchebycheff method as a scalarization technique, a single criterion MCF with side constraints has to be solved. The obtained solution is presented to the decision maker who builds an indifference area using thresholds around the obtained nondominated point in the objective space. Further points located in the indifference area are identified by a heuristic method based on Lagrangian duality and subgradient techniques. The authors claim to be able to perform a test of the quality of the obtained solutions from the decision maker's point of view without explicitly knowing the decision maker's utility function. The outcome of the algorithm is the most robust solution of the ones in the indifference area. This solution need not be an efficient flow for MMCF. The algorithm was implemented and tested on bicriterion transportation problems with 200 nodes and the number of arcs ranging between 1308 and 2900.

The method by **Sedeño-Noda and González-Martín** [147] that finds all efficient extreme points for BiMCF (described in Section 7.2.1), consists of multiple applications of a subroutine performing guided search. As pointed out in [147], this subroutine, on its own, serves as a solution method for generating an efficient extreme point located in a desired region of the objective space.

Sun [155] investigates computational efficient algorithms for generating not only one but a finite number of nondominated solutions with the augmented weighted Tchebycheff method. The network simplex method with side constraints proposed by Chen and Saigal [21] is capable of solving the occuring MCFs with side constraints. Three strategies are proposed to produce good starting solutions for this specialized network simplex algorithm.

1. The weighted sum method.
2. The Tchebycheff method.
3. A sequential combination of the weighted sum method and the Tchebycheff method, i.e. a solution obtained with the weighted sum method is used as a starting solution for the Tchebycheff method.

The optimal solution of the three strategies is transformed into an initial basic feasible flow for the augmented weighted Tchebycheff problem. The network simplex method with side constraints uses the starting solution and solves the augmented weighted Tchebycheff problem to optimality.

The three strategies are numerically compared with the alternative of applying the network simplex method with side constraints directly. For testing, networks with $Q = 3$, 5, and 7 criteria and $n = 100$, 200, 300, 400, 500 and $m = 20n$ are generated. Costs are taken randomly from the interval $\{1, \ldots, 10\}$, while all lower capacities are set to be zero and the upper capacities are randomly chosen from the interval $\{100, \ldots, 500\}$. For each of the 15 parameter set-ups, 10 instances are solved. Strategy 3 outperforms the other algorithms with respect to CPU times in almost all test problems, which is due to a reduced number of iterations needed by the specialized simplex method for network problems with side constraints.

7.5 Schematic résumé of reviewed MMCF and MMCIF papers

To make a direct comparison of the solution procedures for the multicriteria minimum cost flow problem possible, the main features of the reviewed papers are concisely listed in Table 7.3 on page 109. For this purpose, I adapt the classification scheme of Ehrgott and Gandibleux [42]. The articles are grouped analogously as in Sections 7.2 to 7.4 and listed in alphabetical order within each group. Table 7.3 has six columns. In the first column, I refer to the article under consideration. The number of objective functions in this article is listed in the second column. Here, I distinguish between approaches designed for bicriterion (denoted by 2) and general multicriteria problems (denoted by Q). Note that all articles consider sum objectives. The third and fourth column provide information about the type of the problem and the applied solution methods, respectively. Tables 7.1 and 7.2 explain the abbreviations used in these columns. In the last two columns, a "+" indicates that examples are included in the paper, and that the algorithm has been implemented by the authors, respectively.

Entry	**Explanation**
E	Finding the efficient set
e	Finding a subset of the efficient set
SE	Finding the supported efficient solutions
$\hat{\text{E}}$	Finding an approximation of the efficient set
lex	Solving the lexicographical problem
MO	Solving the max ordering problem
lexMO	Solving the lexicographical max ordering problem
C	Finding a compromise solution

Table 7.1: Entries for *Problem Type*.

Entry	**Explanation**
SP	Exact algorithm specifically designed for the problem
BB	Algorithm based on branch and bound
εC	Algorithm based on the ε-constraint method
IA	Interactive method
A	Approximate algorithm with worst case performance bound
LP	Method based on linear programming

Table 7.2: Entries for *Solution Method*.

Paper	Number of objectives	Problem Type	Solution Method	Example	Implemented
MMCF					
Burkard et al. [13]	2	Ê	A	−	+
Fruhwirth et al. [50]	2	Ê	A	−	+
Lee and Pulat [98]	2	E	SP	+	−
Malhotra and Puri [101]	2	E	SP	+	−
Pulat et al. [133]	2	E	LP	+	+
Ruhe [136]	2	Ê	A	+	+
Ruhe [136]	2	E	SP	+	+
Ruhe and Fruhwirth [137]	2	Ê	A	−	+
Sedeño-Noda and González-Martín [145]	2	E	LP	+	+
Sedeño-Noda and González-Martín [147]	2	E	SP	+	+
MMCIF					
Figueira [47]	2	E	SP/BB/εC	+	−
Huarng et al. [79]	2	E	SP	+	+
Lee and Pulat [98]	2	SE	SP	−	−
Lee and Pulat [99]	2	E	SP	+	+
Nikolova [117]	2	SE	SP	−	−
Sedeño-Noda and González-Martín [146]	2	E	SP	+	+
Compromise solutions					
Calvete and Mateo [16]	Q	lex	SP	+	+
Calvete and Mateo [17]	Q	lex	SP	+	+
Ehrgott [41]	Q	lexMO	SP	+	−
Figueira et al. [48]	Q	C	IA	+	+
Hamacher [66]	Q	MO	SP	−	−
Mustafa and Goh [108]	2(3)	C	SP	+	−
Nikolova [115]	Q	C	SP/IA	−	−
Nikolova [116]	Q	C	SP/IA	−	−
Sedeño-Noda and González-Martín [147]	2	C	SP	+	−
Sun [155]	2	e	SP	−	+

Table 7.3: Classification of the reviewed papers.

Chapter 8

Representative system for bicriterion discrete optimization problems

The *bicriterion discrete optimization problem (BiDOP)* can be stated as

$$\min \{y(x) := (y_1(x), y_2(x)) \ : \ x \in \mathcal{X}\}$$

where $\mathcal{X}$ is the discrete feasible set and $y : \mathcal{X} \to \mathbb{Z}^2$ is a vector-valued objective or criterion function. $\mathcal{Y} := y(\mathcal{X})$ denotes the set of attainable criterion points. Recall that $\mathcal{X}_E$ and $\mathcal{Y}_N$ denotes the efficient solutions and the corresponding nondominated points, respectively.

Since the problem of finding $\mathcal{Y}_N$ is, in general, intractable for discrete as well as for continuous bicriterion optimization problems (see Ehrgott [40] and Ruhe [136]), the idea of computing a representation of $\mathcal{Y}_N$ arose. The term "representation" is not used consistently in literature. Here, a *representation* (or a *representative system*), *Rep*, is a set of points in $\mathcal{Y}$ that is computed instead of the nondominated set $\mathcal{Y}_N$. Since, in general, $Rep \neq \mathcal{Y}_N$, the question about the "quality" of such a representation naturally occurs. Although quality measures of a representation of the nondominated set were implicitly used in some algorithms (for example as stopping criteria), the article by Sayin [142] seems to be the first which explicitly deals with the question of how to measure the quality of a representation. On the case of discrete problems on which I focus in this chapter, the algorithms are evaluated based on the following quality attributes of a representative system.

1. Cardinality
2. Accuracy
3. Representation error
4. Cluster density

Here, *cardinality* refers to the fact that the representative system should be "reasonably" small, since the computation of every representing point requires a certain effort, usually related to the solution of a (single criterion) optimization problem. Furthermore, if too many points exist in a representative system, decision makers may find it impossible to consider all alternatives. An *accurate* representation should mirror the complete set of nondominated solutions satisfying some quality measure. Informally, this excludes a situation where large parts of $\mathcal{Y}_N$ are not represented by any point in the representative system. With *representation error* the maximal distance, $\max_{z \in Rep} \min_{y \in \mathcal{Y}_N} ||z - y||$, between the representing points and nondominated points is measured. If all representing points are contained in $\mathcal{Y}_N$, there is no representation error. The *cluster density* is closely related to the cardinality issue. It should be avoided that *Rep* contains large clusters of points, since all points in a cluster would be representatives for the same subset of nondominated points.

The reader should be aware that these quality issues are conflicting by nature. For example, the smaller the cardinality of a representation, the larger the "gaps", i.e. the more inaccurate the representation.

For continuous bicriterion optimization problems, the *sandwich algorithm* (see Section 7.2.2, Burkard et al. [13, 14], and Fruhwirth et al. [50]) and the *gauge algorithm* (see Klamroth, Tind, and Wiecek [88]) are among the most sophisticated methods for generating representations of $\mathcal{Y}_N$. Both algorithms iteratively refine a piecewise linear structure by generating new approximating points until this structure is close enough to $\mathcal{Y}_N$. For a more complete overview of methods for generating representations and for a discussion about which of the above quality attributes the sandwich and the gauge algorithm address, the reader is referred to the survey paper by Ruzika and Wiecek [139].

For bicriterion discrete optimization problems, the literature on exact approaches for computing representations is scarce. Among the few exact methods is the gauge algorithm, [88]. Although this algorithm is originally intended for continuous problems, it is also applicable to discrete problems. The papers by Kouvelis and Sayin [91] and Sayin and Kouvelis [143] address the same issue. To ensure that only (strongly) nondominated points are presented, they propose a two-stage subproblem approach to find a representation with provable quality features. The subproblems require solution of min-max problems resembling the approach for max-ordering problems.

On the other hand, there exists a large variety of heuristic methods for generating representations, (see e.g. the overview in Ehrgott and Gandibleux [42]). Note that these heuristic approaches often do not address the quality attributes mentioned above, in particular in regard to the error of the representation. However, Hansen and Jaszkiewicz [71] present a general framework using non-cardinal measures to compare two approximations of a given set of nondominated points.

Based on the preceding list of quality measures in this chapter, a generic method for computing representations of bicriterion discrete optimization prob-

lems is developed. This approach is referred to as the *box method*. The representation consists of a collection of nondominated points, i.e. $Rep \subseteq \mathcal{Y}_N$. By design, the algorithms will therefore satisfy the representation error quality criterion with a zero error.

Each point is associated with a rectangle (or *box*) and represents all nondominated points within this box. The geometrical features of the rectangles can thus be used to measure the preceding quality attributes.

At any stage of the algorithm, the collection of boxes contains all the nondominated set. Hence, the rectangles bound the nondominated set, and thus also show regions containing no nondominated points. Representing points are generated with a modification of the ε-constraint method. This method is capable of generating any nondominated point with appropriate choices of parameters. Furthermore, optimal solutions to the modified ε-constraint problem are known to be nondominated.

Rectangles have been used previously in the context of bicriterion optimization problems. An interactive method by Payne and Polak [125] constructs a nested family of rectangles based on a decision makers choice of most preferred rectangle during the algorithm. Payne [124] develops a chain of rectangles containing all nondominated points, ensuring a certain quality aspect to be fulfilled at termination of the algorithm. Both papers solve iteratively nonlinear programs. Barichard and Hao [7] consider the union of rectangles containing the nondominated set. During the execution of their evolutionary algorithm, these rectangles are updated. For the Q-dimensional ($Q > 2$) discrete optimization problem, Laumanns, Thiele, and Zitzler [94] solve a sequence of constrained single-objective problems to generate all nondominated points. This is done by adaptively updating a sequence of Q-dimensional boxes. Instead of focussing on the criterion space, one can also consider the decision space. The *Big Square – Small Square* (BSSS) method by Hansen [72] has been applied among others by Skriver and Andersen [150] to eliminate rectangles containing only non-Pareto locations for a bicriterion semi-obnoxious location problem.

The remainder of this chapter is organized as follows. In Section 8.1, a variant of the well-known ε-constraint method is introduced which will be employed for generating representing points. Two versions of the box algorithm for computing a discrete representation with provable quality features are proposed in Section 8.2. The algorithms are discussed and their quality properties are linked to existing literature in Section 8.3. In Section 8.4, possible future research directions are outlined for this topic and a few comments on a recent computational study of the box algorithms are given.

8.1 The lexicographical ε-constraint method

The scalarization that will be used intensively in the remainder of this paper is a lexicographical variant of the well-known ε-constraint scalarization (see e.g. Chankong and Haimes [18]). The following mathematical programme is referred to as the *lexicographical ε-constraint problem*.

$$\begin{aligned} &\operatorname{lex\,min}\,(y_2(x), y_1(x)) \\ &\text{s.t.} \qquad y_1(x) \leq \varepsilon \\ &\qquad\quad x \in \mathcal{X}. \end{aligned} \tag{8.1}$$

In (8.1), the vector $(y_2(x), y_1(x))$ is lexicographically minimized, and its feasible set is the feasible set of the bicriterion optimization problem $\mathcal{X}$ with an additional constraint bounding the y_1-values from above.

The lexicographical ε-constraint problem has several desirable properties which are very useful in constructing a representation of the nondominated set. In particular, the next result shows that the lexicographical objective function yields (strongly) efficient solutions – opposed to weakly efficient ones in the classical ε-constraint approach.

Theorem 8.1 *Let $\hat{x} \in \mathcal{X}$ be an optimal solution of* (8.1)*. Then $\hat{x} \in \mathcal{X}_E$.*

Proof. Assume that $x' \in \mathcal{X}$ with $y(x') \leq y(\hat{x})$. Therefore, $y_j(x') \leq y_j(\hat{x}),\ j = 1, 2$ with at least one strict inequality. In particular, $y_1(x') \leq y_1(\hat{x}) \leq \varepsilon$, such that x' is feasible for (8.1). Therefore, x' contradicts the optimality of $\hat{x}$ in (8.1). □

Conversely, any efficient solution can be obtained with the lexicographical ε-constraint scalarization with an adequate choice of ε.

Theorem 8.2 *Let $\hat{x} \in \mathcal{X}_E$. Then $\varepsilon := y_1(\hat{x})$, yields $\hat{x}$ as an optimal solution for* (8.1)*.*

Proof. The proof follows directly from the same result of the classical ε-constraint scalarization, see e.g. [18]. □

In particular, the one-to-one correspondence between solutions of (8.1) and $\mathcal{X}_E$ implies that we get supported as well as unsupported efficient solutions. This is not achievable by the weighted sum method (see page 28) which is by far the most utilized scalarization technique in literature.

An additional advantage of using the lexicographical ε-constraint method is that the original objective functions are maintained. This is an appealing feature, especially if the original objective functions possess structural properties like linearity or convexity. To the best of my knowledge, all scalarization techniques which preserve linearity of the objective functions and which are capable of generating all nondominated points for discrete problems, modify the constraint set by

at least one constraint. Therefore, the additional constraint induced by the lexicographical ε-constraint method can be viewed as a smallest possible modification. For a list of the most common scalarization techniques refer to Chapter 17 in [49].

8.2 Two versions of the box algorithm

In this section, two versions of the box algorithm for computing a representation (in $\mathcal{Y}$) of $\mathcal{Y}_N$ of a bicriterion discrete optimization problem are proposed. Both versions allow controlling the quality attributes of a representation as proposed in the introduction of this chapter.

8.2.1 Initialization and update

Before explaining the details, the major ideas of the box algorithm are outlined. Initially, the two lexicographical optimal solutions are computed. These points determine a rectangle (the *starting box*) containing the complete nondominated set. In the following, rectangular pieces of the starting box are discarded based on additional information obtained by iteratively solving (8.1) with adequately chosen values for ε. This yields a collection of rectangles or boxes containing the nondominated set in each stage of the algorithm. Furthermore, for each box, a nondominated point representing the set of nondominated points inside the box is known. The box algorithm stops based on a predetermined accuracy, which is measured by the largest area of the remaining boxes.

More detailed, the box algorithm works as follows. Initially, both lexicographical minima of the bicriterion optimization problem

$$y^{UL} := \operatorname{lex}\min_{x\in\mathcal{X}} (y_1(x), y_2(x)) \text{ and } y^{LR} := \operatorname{lex}\min_{x\in\mathcal{X}} (y_2(x), y_1(x))$$

are determined. Obviously, $\mathcal{Y}_N$ is a subset of $R(y^{UL}, y^{LR})$, the rectangle with upper left and lower right vertex y^{UL} and y^{LR}, respectively. This is the *starting box* of the algorithms. In general, $R(y^1, y^2)$ denotes the rectangle having $y^1 = (y_1^1, y_2^1) \in \mathbb{Z}^2$ as upper left and $y^2 = (y_1^2, y_2^2) \in \mathbb{Z}^2$ as lower right vertex. Moreover, let $a(R(y^1, y^2)) := (y_1^2 - y_1^1) \cdot (y_2^1 - y_2^2)$ denote the area of the rectangle $R(y^1, y^2)$.

Initially, the starting box $R(y^{UL}, y^{LR})$ contains the complete nondominated set $\mathcal{Y}_N$. Whenever additional nondominated points become known during the execution of the algorithm, one of the boxes will be split up into several smaller rectangles – while maintaining the inclusion property of $\mathcal{Y}_N$. The algorithm stops, when the area of the largest of these boxes is smaller than some given accuracy $\Delta > 0$. Such a collection of boxes is referred to as a Δ-*representation of* $\mathcal{Y}_N$. The lower right corner point of each of the rectangles is a *representing point* and will be added to the representing system.

In the following, we consider a box with $a(R(y^1, y^2)) > \Delta$, such that the representation has to be locally updated in $R(y^1, y^2)$. Consider (8.1) with $\varepsilon :=$

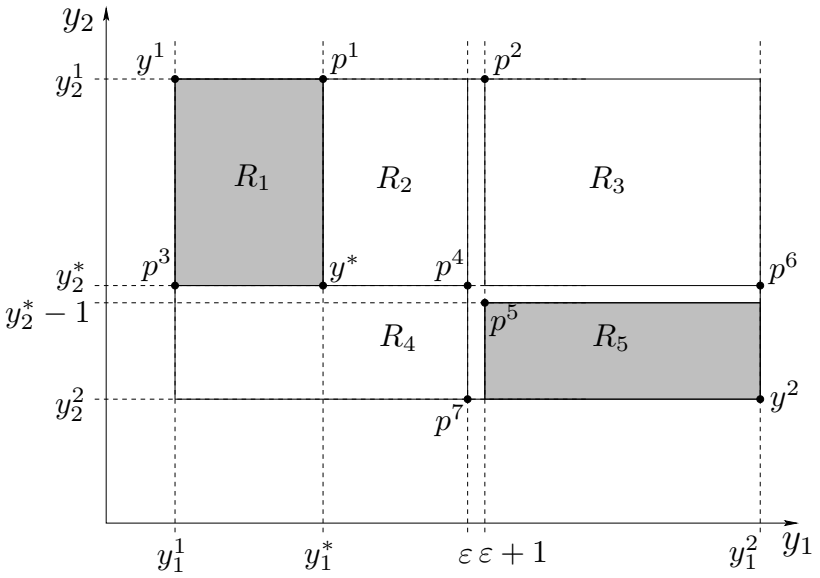


Figure 8.1: Updating of a rectangle $R(y^1, y^2)$.

$\lfloor \frac{y_1^1 + y_1^2}{2} \rfloor$. Let $x^* \in \mathcal{X}$ be optimal for (8.1) and let $y^* := y(x^*)$. Due to Theorem 8.1, y^* is nondominated. Using the point y^* and ε, $R(y^1, y^2)$ is divided into five rectangles

$$R_1 := R(y^1, y^*), \quad R_2 := R(p^1, p^4), \quad R_3 := R(p^2, p^6),$$
$$R_4 := R(p^3, p^7), \text{ and } R_5 := R(p^5, y^2)$$

(see Figure 8.1), where

$$p^1 := \left(y_1^*, y_2^1\right), \quad p^2 := \left(\varepsilon + 1, y_2^1\right), \quad p^3 := \left(y_1^1, y_2^*\right), \quad p^4 := \left(\varepsilon, y_2^*\right),$$
$$p^5 := \left(\varepsilon + 1, y_2^* - 1\right), \quad p^6 := \left(y_1^2, y_2^*\right), \text{ and } p^7 := \left(\varepsilon, y_2^2\right) .$$

Note that the points p^2 and p^5 have first coordinate one unit to the right of the ε-constraint. Furthermore, the point p^5 has second coordinate one smaller than the second objective value of y^*. This is due to the fact that no feasible criterion point can have non-integral coordinates. In general, $p^i \notin \mathcal{Y}$, $i = 1, \ldots, 7$, i.e. the points $\{p^i\}$ are not necessarily feasible criterion points. The rectangles R_1, R_2, and R_4 can degenerate to a line, in which case they are considered to have zero area.

Proposition 8.3 *R_2, R_3, and R_4 can be eliminated, since*

(a) $(R_2 \cup R_3) \cap \mathcal{Y}_N \subseteq \{y^*\}$, *and*

(b) $R_4 \cap \mathcal{Y}_N \subseteq \{y^*\}$.

Proof. (a) y^* is dominating any feasible point in $R_2 \cup R_3$. (b) Any feasible point in R_4 contradicts the optimality of x^* for (8.1). □

Due to the previous result, there does not exist nondominated points in R_2, R_3, R_4, other than y^*. Consequently, the following corollary holds.

Corollary 8.4

$$\mathcal{Y}_N \cap R(y^1, y^2) \subseteq R_1 \cup R_5$$

It is easy to derive the following result, comparing the size of rectangles R_1 and R_5 to the size of rectangle $R(y^1, y^2)$.

Proposition 8.5

$$a(R_1) + a(R_5) \leq \tfrac{1}{2} a(R(y^1, y^2))$$

After having obtained y^* as a new nondominated point, the representation is updated associating y^* with R_1 and y^2 with R_5. By Proposition 8.5, we see that computing y^* has locally improved the representation by a factor of 2.

Corollary 8.6 *After computing y^* and adding it to the representation, the representation's accuracy improved in $R(y^1, y^2)$ by a factor of 2.*

8.2.2 The a posteriori algorithm

In this section the updating procedure from Section 8.2.1 is applied iteratively to place ε-constraints in a rectangle having area bigger than Δ. Assume that Δ is given as input in the algorithm. Since Δ is the measure of the size of the boxes, its determination may also be left to the decision maker or be computed as a given percentage of the starting box.

Since the value of the ε-constraint is chosen *after* the given rectangle is identified, this method is called an *a posteriori* algorithm. A pseudo code description is given in Figure 8.2. Notice that the point y^* found by the procedure `SolveLexMin` is nondominated according to Theorem 8.1.

Note that the a posteriori algorithm is flexible in the sense that it always chooses the rectangle with the largest area, and the representation is hence only refined where it is needed. The following theorem establishes finiteness and the complexity of the a posteriori algorithm.

Theorem 8.7 *In finitely many steps, the a posteriori algorithm yields a Δ-representation of $\mathcal{Y}_N$ in which all representing points are nondominated. More specifically, the algorithm performs at most $\mathcal{O}(\frac{a(R(y^{UL}, y^{LR}))}{\Delta})$ iterations.*

```
1  algorithm APosteriori()
2     Input: A bicriterion discrete optimization problem, Δ > 0.
3     Output: A representation Rep ⊆ Y_N with accuracy Δ.
4     y^UL := FirstLexMin();
5     y^LR := SecondLexMin();
6     Rep := {y^UL, y^LR}, Ω := {R(y^UL, y^LR)};
7     while (Ω ≠ ∅) do
8        R(y^1, y^2) := ChooseLargestRectangle(Ω);
9        Ω := Ω \ {R(y^1, y^2)};
10       ε := ⌊(y_1^1 + y_1^2)/2⌋;
11       y* := SolveLexMin(ε); (solves (8.1))
12       Rep := Rep ∪ {y*};
13       if (a(R(y^1, y*)) > Δ) then Ω := Ω ∪ {R(y^1, y*)};
14       if (a(R(p^5, y^2)) > Δ) then Ω := Ω ∪ {R(p^5, y^2)};
15    end while
16  end algorithm
```

Figure 8.2: The a posteriori algorithm.

Proof. As long as the representation is not a Δ-representation, the algorithm processes in each iteration the largest rectangle that does not meet the accuracy of Δ. Due to Corollary 8.6, the resulting rectangle(s) have, after the update, an area which is at most half the size of the original area. The algorithm is therefore finite. Due to Theorem 8.1, all representing points are nondominated.

More specifically, the algorithm produces after 2^i-1 iterations no more than 2^i rectangles, each of which have an area of at most $\frac{a(R(y^{UL},y^{LR}))}{2^i}$. Since $i^* = \lceil \log_2(\frac{a(R(y^{UL},y^{LR}))}{\Delta})\rceil$ is the smallest integer satisfying $\frac{a(R(y^{UL},y^{LR}))}{2^{i^*}} \leq \Delta$, an upper bound on the number of iterations to obtain a Δ-representation is $2^{i^*}-1 = 2^{\lceil \log_2(\frac{a(R(y^{UL},y^{LR}))}{\Delta})\rceil}-1$, which is $\mathcal{O}(\frac{a(R(y^{UL},y^{LR}))}{\Delta})$. □

The following result is a direct consequence of Theorem 8.7.

Corollary 8.8 *The a posteriori algorithm runs in* $\mathcal{O}(T_1 + \frac{a(R(y^{UL},y^{LR}))}{\Delta}\, T_2)$ *where* T_1 *and* T_2 *is the time needed to compute a lexicographical minimum and to evaluate* (8.1)*, respectively.*

Observe that T_1 and T_2 depend on the specific problem under consideration.

8.2.3 The a priori algorithm

Rather than deciding on the value of ε for the next ε-constraint problem *after* each iteration, in the next algorithm a number of equidistant values for ε are computed

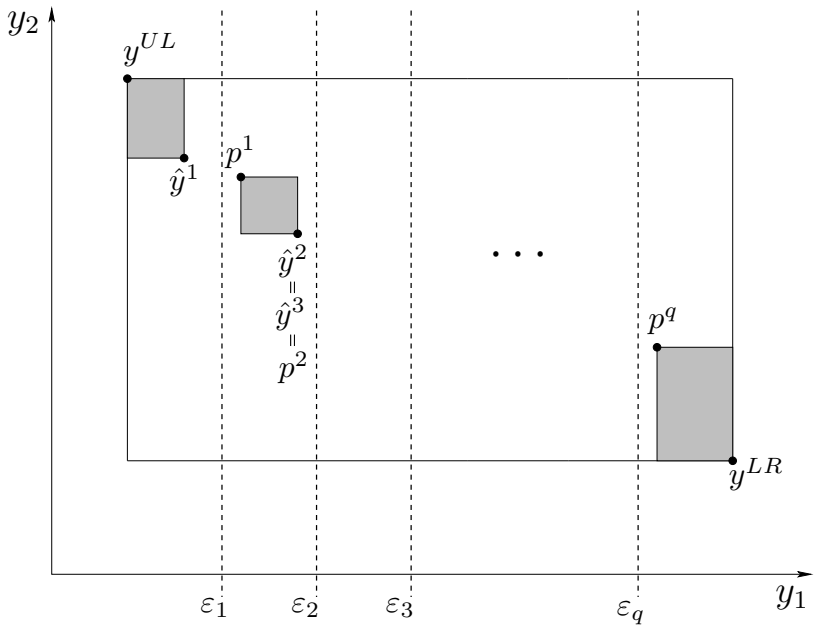


Figure 8.3: The representation after having solved (8.1) with $\varepsilon = \varepsilon_q, \varepsilon_{q-1}, \ldots, \varepsilon_1$.

a priori based on the value of Δ. The solution of the corresponding ε-constraint problems yields a Δ-accurate representation for any given Δ. Consider

$$\varepsilon_j := y_1^{UL} + \left\lfloor \frac{j}{q+1}(y_1^{LR} - y_1^{UL}) \right\rfloor, \; j = 1, \ldots, q \; .$$

Let $\hat{y}^j$ denote the image of the optimal solution to (8.1) with $\varepsilon := \varepsilon_j$. Note that the points $\hat{y}^j$, $j = 1, \ldots, q$, are not necessarily distinct. For $j = q-1, \ldots, 1$, define

$$p^j := \begin{cases} (\varepsilon_j + 1, \hat{y}_2^j - 1) & \text{if } \hat{y}^j \neq \hat{y}^{j+1} \\ \hat{y}^j & \text{otherwise} \end{cases} \; .$$

Set $p^q := (\varepsilon_q + 1, \hat{y}_2^q - 1)$. The points $\{p^j\}$ are used to define rectangles, but they are not necessarily feasible in $\mathcal{Y}$. Figure 8.3 illustrates these definitions.

The same reasoning concerning dominance and feasibility as in Proposition 8.3 and Corollary 8.4 applies here as well. Therefore,

$$\mathcal{Y}_N \subseteq R(y^{UL}, \hat{y}^1) \cup \bigcup_{j=1}^{q-1} R(p^j, \hat{y}^{j+1}) \cup R(p^q, y^{LR}) \; ,$$

and hence $\mathcal{Y}_N$ is contained in a collection of at most $q+1$ rectangles. For each rectangle, its lower right corner point is nondominated.

The following proposition evaluates the reduction rate of the unknown area of interest.

Proposition 8.9 *Suppose the algorithm solves q ε-constraint problems with equidistant ε-constraints. Then the sum of the areas of the resulting rectangles is at most $\frac{1}{q+1}a(R(y^{UL}, y^{LR}))$.*

Proof.

$$
\begin{aligned}
&a(R(y^{UL}, \hat{y}^1)) + \sum_{j=1}^{q-1} a(R(p^j, \hat{y}^{j+1})) + a(R(p^q, y^{LR})) \\
&\quad = (y_2^{UL} - \hat{y}_2^1)(\hat{y}_1^1 - y_1^{UL}) + \sum_{j=1}^{q-1} (p_2^j - \hat{y}_2^{j+1})(\hat{y}_1^{j+1} - p_1^j) + (p_2^q - y_2^{LR})(y_1^{LR} - p_1^q) \\
&\quad \leq \frac{1}{q+1}(y_1^{LR} - y_1^{UL})\Big(y_2^{UL} - \hat{y}_2^1 + \sum_{j=1}^{q-1}(p_2^j - \hat{y}_2^{j+1}) + p_2^q - y_2^{LR}\Big) \\
&\quad \leq \frac{1}{q+1}(y_1^{LR} - y_1^{UL})\Big(y_2^{UL} - \hat{y}_2^1 + \sum_{j=1}^{q-1}(\hat{y}_2^j - \hat{y}_2^{j+1}) + \hat{y}_2^q - y_2^{LR}\Big) \\
&\quad = \frac{1}{q+1}a(R(y^{UL}, y^{LR})) \qquad \square
\end{aligned}
$$

It should be emphasized that the *total* area of the rectangles is at most $\frac{1}{q+1}a(R(y^{UL}, y^{LR}))$. Hence the biggest rectangle in the representation has area of at most $\frac{1}{q+1}a(R(y^{UL}, y^{LR}))$. We thus get the next result.

Theorem 8.10 *To achieve an accuracy of Δ in the a priori algorithm, $q \leq \lceil \frac{a(R(y^{UL}, y^{LR}))}{\Delta} \rceil - 1$ solutions of (8.1) have to be computed.*

The algorithm is explicitly stated in Figure 8.4 and illustrated in Figure 8.3. Note that the lexicographical ε-constraint problems are solved from right to left (see line 8 in Figure 8.4). This is in practice faster than going from left to right, since the solution of the lexicographical ε-constraint problem for $\varepsilon = \varepsilon_j$ may be optimal for $\varepsilon_{j-1}, \ldots \varepsilon_{j-i}$, $i \geq 1$. Therefore, the solution of (8.1) with $\varepsilon = \varepsilon_{j-1}, \ldots, \varepsilon_{j-i}$ can be skipped.

8.3 Quality of the box representation

In this section, the quality issues stated in the introduction of this chapter are addressed for both algorithms.

Cardinality Due to Theorems 8.7 and 8.10, the number of representing points is at most $2^{\lceil \log_2(\frac{a(R(y^{UL}, y^{LR}))}{\Delta}) \rceil} + 1$ and $\lceil \frac{a(R(y^{UL}, y^{LR})}{\Delta} \rceil + 1$ for the a posteriori algorithm and the a priori algorithm, respectively. However, in both cases, a

```
1  algorithm APriori()
2     Input: A bicriterion discrete optimization problem, Δ > 0.
3     Output: A representation Rep ⊆ Y_N with accuracy Δ.
4     y^UL := FirstLexMin();
5     y^LR := SecondLexMin();
6     Rep := {y^UL, y^LR};
7     q := ⌈a(R(y^UL, y^LR)) / Δ⌉ − 1;
8     for (j := q to 1) do
9        ε_j := y_1^UL + ⌊(j/(q+1))(y_1^LR − y_1^UL)⌋;
10       ŷ^j := SolveLexMin(ε_j); (solves (8.1))
11       Rep := Rep ∪ {ŷ^j}.
12    end for
13 end algorithm
```

Figure 8.4: The a priori algorithm.

worst-case analysis is used to obtain these bounds. It can be expected, that the number of generated points is much smaller in practice, since solutions can be optimal for several ε-constraint problems.

Alternatively, the two algorithms can also be formulated in such a way that the cardinality Υ of the representative system is part of the input data instead of Δ. Then the accuracy Δ is a function of Υ, i.e. the representation is Δ-accurate with $\Delta = \frac{a(R(y^{UL},y^{LR})}{\Upsilon-1}$ and $\Delta = \frac{a(R(y^{UL},y^{LR})}{\Upsilon-2}$ for the a posteriori algorithm and the a priori algorithm, respectively.

Accuracy Both algorithms have as termination criterion a measure for the accuracy and therefore this issue has already been discussed in Sections 8.2.2 and 8.2.3.

Using the different accuracy measure of [142] (Sayin calls it "coverage error") it is easy to show that *Rep* is also a d_Δ-representation in the sense of [142] when using l_∞ to measure distances.

The ε-approximation , introduced in Definition 6.2 on page 67, is a further concept of accuracy from the literature, which measures the percentage distance ε, between any nondominated point and the nearest representing point in both coordinates. Even though both the concept of an ε-approximation and that of a representative system serve the same goal of providing the decision maker with a sufficiently good approximation of the nondominated points, these measures are not directly comparable to each other. However, the rectangle representation attained by any of the two algorithms in this paper yields an ϵ-approximation of

$$\varepsilon = \max_{y \in Rep, y \in R(z,y)} \left\{ \frac{y_1 - z_1}{y_1} \right\},$$

where y is a representing point in the rectangle $R(z, y)$.

Finally, it is obvious that both algorithms are exact if Δ is sufficiently small, e.g. $\Delta = 1$.

Representation error Extending ideas from [142], we can define the representation error formally as

$$error_{Rep} = \max_{z \in Rep} \min_{y \in \mathcal{Y}_N} ||z - y||_p \ ,$$

where $||.||_p$ denotes the l_p norm. In both algorithms, only nondominated points are generated and therefore the representation error is always zero for any p. Note that $\max\{accuracy, error_{Rep}\}$ is the Hausdorff distance (see (7.5) on page 96) which is used in a variety of algorithms for computing a representation of $\mathcal{Y}_N$ (for continuous problems), (see e.g. Section 7.2.2 and [50, 167]).

Cluster density Depending on the application, one can apply filtering procedures (see e.g. [154]) at the termination of either of the two algorithms to reduce the number of representing points in any rectangle of size Δ to a desirable amount of points. This filtering obviously reduces the cardinality of the representation while maintaining the Δ-accuracy.

8.4 Future research and computational results

Several variations and extentions of the box approach are possible. The first addresses the way accuracy is measured. Instead of using the area $a(R(y^1, y^2))$ of the boxes, other means, such as the maximal diagonal or the maximal side among the boxes, may prove useful in certain applications. This can be achieved with the same general idea as presented in Section 8.2. If the maximal length is attained in the second component, the ε-constraint is introduced for function y_2 instead of y_1. The corresponding analysis of the number of iterations is formally more complicated but rather straightforward.

An appealing extension is the generalization of the algorithms to discrete problems with more than two objective functions. If we assume that $Q > 2$ many objective functions are given, we can iteratively apply the bicriterion approach to pairs of objectives and in this get way a representative system. Another approach is to use Q-dimensional boxes instead of 2-dimensional in the bicriterion case. The quality of this approach depends on the choice of a good starting box which is easy for $Q = 2$ but, in general, very difficult for $Q > 2$.

The complexity of both versions of the box algorithm depends on the complexity of solving the ε-constraint problems which occur as subproblems. Interesting research questions concern the usage of the solution of one of these problems in solving another one with modified ε, and ways on how to measure the effect of an approximation of the ε-constraint problems inside the box algorithm.

8.4.1 Computational results

A recent computational study by Ruzika [138] compares the performance of the a posteriori algorithm presented in this chapter with the *algorithm robust* proposed by Sayin and Kouvelis [91, 143].

12 different set-ups of bicriterion knapsack problems were generated using either 500 or 1000 knapsack items. The weights of the knapsack items and the bicriterion cost vectors, all having a maximal value of 5000, were generated by six different methods. For each set-up, 100 instances were solved.

Both algorithms were implemented in C++ using CPLEX 9.1 [81] to solve the associated single criterion subproblems. The algorithms produced the same number of representative points for a given test instance.

On average, the a posteriori algorithm performed 2-3 times faster than algorithm robust. Both algorithms determine nondominated points and have therefore no representation error. By providing a more dispersed set of representing points the a posteriori algorithm outperformed algorithm robust on the other quality issues addressed in this chapter.

Chapter 9

Further developments on multicriteria network problems

In this chapter, I present some extensions of the ideas from Chapters 6 and 7. Due to time issues, these extensions have not yet been implemented and cannot be considered complete.

In Section 9.1, the bicriterion multi modal assignment problem is generalized to a bicriterion multi modal transportation problem and both problem classes are derived as subproblems of the bicriterion directed Chinese postman problem. In Section 9.2, ideas concerning an exact solution procedure for the multicriteria minimum cost integer flow problem are discussed and one of the core directions for further research on MMCIF is pointed out.

9.1 Extensions of BiMMAP

The transportation problem is an extension of the assignment problem. Here, a similar relevant extension of the bicriterion multi modal assignment problem which has, to the best of my knowledge, not yet been considered in literature is introduced.

Consider a company supplying a set W of n_1 retailers from a set V of n_2 production plants. Let s_i and d_j denote the units of goods supplied by plant i and demanded by retailer j, respectively. Assume that the plants are distributed across a wide area, and hence multiple routes and multiple modes of transportation can be chosen when shipping goods from a plant i to a retailer j. Let L_{ij} denote the number of distinct mode choices. To each mode choice $l = 1, \ldots, L_{ij}$, corresponds a *travel cost* c^1_{ijl} and a *travel time* c^2_{ijl}. In resemblance with BiMMAP, we have in the *transportation cost matrix* several two-dimensional cost vectors in each transportation cell (i, j). Obviously, the company wants to minimize the total cost of shipping goods from plants to retailers. However, also the total transportation

time needs to be minimized, so that the persons transporting the goods can be utilized for other normal daytime routines in idle periods. Hence, the objective in the *bicriterion multi modal transportation problem (BiMMTP)* is to identify all nondominated criterion points for the problem. Let x_{ijl} be an integer variable equal to the number of units of goods transported from plant i to retailer j using mode choice l. The mathematical formulation of BiMMTP is stated in (9.1), using again the three-indexed arc set $A := \{(i,j,l)\}_{ijl}$.

$$\begin{aligned} &\min \sum_{(i,j,l)\in A} c^1_{ijl} x_{ijl} \\ &\min \sum_{(i,j,l)\in A} c^2_{ijl} x_{ijl} \\ &\text{s.t.} \quad x \in \mathcal{X}_{MMTP} \end{aligned} \tag{9.1}$$

in which

$$\mathcal{X}_{MMTP} = \left\{ x \,:\, \sum_{j=1}^{n_2} \sum_{l=1}^{L_{ij}} x_{ijl} = s_i,\ \forall i \in W,\ \sum_{i=1}^{n_1} \sum_{l=1}^{L_{ij}} x_{ijl} = d_j,\ \forall j \in V, \right.$$
$$\left. x_{ijl} \in \mathbb{N}_0,\ \forall i,j,l \right\}. \tag{9.2}$$

Following the same ideas as in the solution method for BiMMAP, an exact two-phase procedure for BiMMTP could be employed. In phase one, a weighted sum scalarization technique, similar to the one presented in Section 6.1 utilizing a single criterion TP optimizer, finds all supported extreme nondominated points. Then, in the second phase, it suffices to use a ranking approach in each of the relevant triangles in the criterion space, to obtain the remaining nondominated points. This task could be performed extending the developments on ranking ordinary transportation solutions, presented in Section 5.3, to handle multi modal transportation solutions. Obviously, the effectiveness of this exact two-phase procedure depends heavily on the nature of reoptimization in ranking transportation solutions, and would have to be validated through extensive numerical testing.

Actually, both BiMMAP and BiMMTP developed as subproblems of a more complex problem class. Consider an instance of the Chinese postman problem (see Section 2.1.5), in which the optimization process involves both a time criterion *and* a cost criterion. Hence, the *bicriterion Chinese postman problem (BiCPP)* arises. Bicriterion versions of the three most classical CPPs, (the directed, undirected and mixed CPP), are, with obvious notation, denoted by *BiDCPP, BiUCPP and BiMCPP*. BiDCPP and BiUCPP can be interpreted as generalizations of the bicriterion assignment problem making them intractable and $\mathcal{NP}$-complete, even though their single criterion counterparts are polynomially-time solvable, [39, 148]. Obviously, BiMCPP is $\mathcal{NP}$-complete as its single criterion counterpart, and is also intractable.

```
1  procedure BiDCPP_1()
2     Input: A directed strongly connected non-Eulerian graph.
3     Output: 𝒴_N for BiDCPP.
4     W := {i ∈ N : s_i := d⁻(i) − d⁺(i) > 0};
5     V := {j ∈ N : d_j := d⁺(j) − d⁻(j) > 0};
6     for all ((i, j) ∈ W × V) do
7        {(c¹_ij1, c²_ij1), ..., (c¹_ijL_ij, c²_ijL_ij)} := BiSP(i, j);
8     end for all
9     C := {(c¹_ij1, c²_ij1), ..., (c¹_ijL_ij, c²_ijL_ij)}_ij;
10    𝒴_N := BiMMTP(W, V, C);
11 end procedure
```

Figure 9.1: An algorithm for the bicriterion directed Chinese postman problem.

Let me focus entirely on the bicriterion directed Chinese postman problem. Consider a strongly connected directed graph $G = (N, A)$, and let $W := \{i \in N : s_i := d^-(i) - d^+(i) > 0\}$ and $V := \{j \in N : d_j := d^+(j) - d^-(j) > 0\}$. Therefore, W and V are the sets of nodes from G with an excess of incoming and outgoing arcs, respectively. BiDCPP is to find all nondominated augmentations of G to make it Eulerian, which a directed graph is known to be if every node has in-degree equal to out-degree. Given an Eulerian graph, the actual traversal of the graph can be completed by the same algorithms applied for the single criterion case. As for the single criterion DCPP, the bicriterion augmentation problem can be solved with a two-phase method. In phase one, all efficient paths from any node in W to any node in V must be identified, after which phase two must determine all nondominated ways of matching these paths in order to cancel any node-deviations in the original graph.

Let L_{ij} denote the number of efficient paths in G from a node $i \in W$ to a node $j \in V$ each with two-dimensional cost vector (c^1_{ijl}, c^2_{ijl}), $l = 1, \ldots, L_{ij}$. Let x_{ijl} denote the number of times the arcs in path l between nodes $i \in W$ and $j \in U$ are added to the original graph G. An instance of the bicriterion multi modal transportation problem exactly solves the problem of finding all nondominated augmentations of G to make it Eulerian. The L_{ij} two-dimensional cost vectors in each transportation cell (i, j) correspond to the nondominated costs of the paths between nodes i and j in G. Also, the supply s_i corresponds to the excess amount of incoming arcs to node i, and d_j the excess amount of outgoing arcs from node j.

A complete description of the suggested solution procedure `BiDCPP_1` is given in Figure 9.1 in which `BiSP`(i, j) returns all nondominated cost vectors for the bicriterion shortest path problem between i and j in G, and `BiMMTP` is the exact two-phase solution procedure for BiMMTP.

In Figure 9.2, an alternative exact two-phase method, `BiDCPP_2`, for the bicriterion directed Chinese postman problem is suggested. Here, the parametric minimization problem of (3.4) with $\mathcal{X} := \mathcal{X}_{DCPP} := \mathcal{P}_{DCPP} \cap \mathbb{Z}^m$ is solved iteratively in phase one. Recall, that the directed Chinese postman problem is a

```
1 procedure BiDCPP_2()
2    Input: A directed strongly connected non-Eulerian graph.
3    Output: 𝒴_N for BiDCPP.
4    PhaseOne(); (see page 28) (solves iteratively (3.4)).
5    for all (△(y⁺, y⁻)) do
6        PhaseTwo(△(y⁺, y⁻)); (see page 30) (use function K-MCIF).
7    end for all
8  end procedure
```

Figure 9.2: An alternative BiDCPP algorithm.

special instance of the minimum cost integer flow problem. Therefore, in the second phase, one can search each triangle $\triangle(y^+, y^-)$ by utilizing the algorithm for ranking integer flows K-MCIF, presented in Chapter 5, as subroutine.

I allocated the initial implementational work to BiMMAP, since it is a special instance of BiMMTP. Actually, BiMMAP corresponds to an augmentation problem for a BiDCPP with node differences restricted to be no larger than one (referred to as BiDCCP$_R$ in Figure 9.3). Considering real-life road networks, the in- and out-degree of any node (corresponding to a crossing of roads) are practically always restricted by a small upper bound – as is then the node differences. However, in terms of real-life applications, it seems even more interesting to consider a mixed network yielding the more difficult problem BiMCPP.

In Figure 9.3, I present a graphical overview of the relation between the problems classes discussed in this section. A directed arrow indicates that a particular problem class is a subproblem of its ancestor. Figure 9.3 also offers insight in how the different ideas evolved and it gives a strategy plan for future research on this topic. For each problem class, the current status for that particular subproject is given. Notice that, for all-pairs BiSP, the status is stated as "yet to be investigated". Actually, one way to solve this problem is to apply a normal bicriterion shortest path algorithm a number of times, see e.g. [151].

9.2 The multicriteria minimum cost integer flow problem

In this section, a few novel and yet unpublished thoughts on the multicriteria minimum cost integer flow problem are presented. They constitute a natural extension of the ideas previously published on MMCIF, (reviewed in Chapter 7).

Consider the directed network $G = (N, A)$ and recall the definition of the flow integer lattice, $\mathcal{X}_{flow}$. To ease the disposition, assume that the network G has only one supply node and only one demand node. In general, any directed network can be transformed into such a network by addition of a *super-source* s, a *super-sink* t, an arc (s, i) of capacity b_i to every original supply node i, and an arc (i, t) of capacity $-b_i$ to every original demand node j. The entire supply and demand are

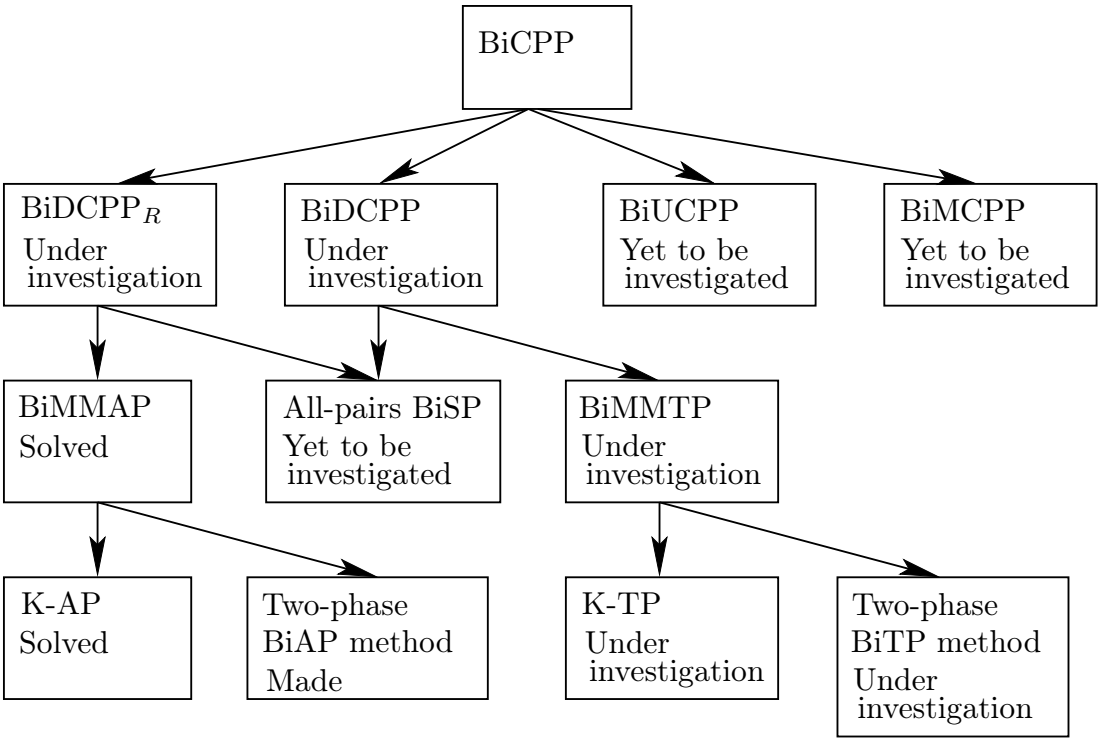


Figure 9.3: Connection between bicriterion network problems.

aggregated at s and t, respectively. Remember, from Definition 7.1 on page 103, that a feasible flow x sending ρ units from s to t has (flow) value $v(x) = \rho$.

The shortest augmenting path procedure for the single criterion minimum cost flow problem is based on a number of general concepts for network flows. Remember the definition of the flow addition $\oplus$ and the flow subtraction $\ominus$ operators from Definition 2.2 on page 11.

Proposition 9.1 ([2, 15, 85, 96])

(i) *Any feasible flow decomposes into directed cycles and directed paths from G.*

(ii) *If a feasible flow with value $\rho + q$ exists, any such flow $\hat{x}$ can be found as $\hat{x} = x \oplus \xi$, where ξ in $G(x)$ is an incremental flow of value q and x is a feasible flow of value ρ.*

(iii) *The difference of any two feasible flows $\hat{x}$ and x both of value ρ, can be decomposed into a sum of $r \leq m$ directed cycles,*

$$\xi := \hat{x} \ominus x = \sum_{k=1}^{r} O_k, \qquad \text{where } O_k \text{ is a directed cycle in } G(x).$$

Such a flow ξ of value 0 is referred to as a circulation.

(iv) *Adding to a feasible flow x of value ρ an optimal circulation from $G(x)$ yields an optimal flow of value ρ.*

(v) *Augmenting an optimal flow x of value ρ with q units along a shortest augmenting path ξ from $G(x)$ yields an optimal flow $\hat{x}$ of value $\rho + q$.*

The first three observations of Proposition 9.1 only utilize the network structure, and so they also hold true for a general MMCIF with any number of criteria. The following example shows that the fifth property of Proposition 9.1 cannot be extended to multicriteria instances, since adding q units along an efficient augmenting path to an efficient flow of value ρ does not necessarily generate an efficient flow of value $\rho + q$. Therefore, the example destroys the hope to develop a multicriteria version of the shortest augmenting path procedure along the same lines as in the single criterion case.

Example 9.2 Consider the graph in Figure 9.4(a) in which all arc bounds are equal to $[0, 1]$. This graph constitutes, with two objectives, an example of the class BiMCIF. Two efficient ways of sending $\rho = 1$ unit from s to t exist. Namely the two efficient paths $P_1 : s - 1 - 2 - 4 - t$ with cost vector $c(P_1) = (2, 1)$, and $P_2 : s-1-4-t$ with $c(P_2) = (1, 2)$. Consider the solution $x := P_1$ with incremental graph $G(x)$ as shown in Figure 9.4(b). In $G(x)$ two efficient augmenting s-t paths exist, namely $P_3 : s - 3 - t$ and $P_4 : s - t$. Notice that, this is also true, even if non-simple augmenting paths are allowed. Adding the efficient augmenting path, $\xi := P_3$ to x the solution $\hat{x} = x \oplus \xi$ (depicted in Figure 9.4(c)) is obtained with $y(\hat{x}) = (5, 6)$ and of flow value $v(\hat{x}) = 2$. However, $\hat{x}$ turns out to be dominated by the solution $\tilde{x}$ of flow value 2, shown in Figure 9.4(d) with $y(\tilde{x}) = (5, 4)$.

Setting $q = 0$ in the following theorem yields as an easy corollary a multicriteria version of the fourth property in Proposition 9.1.

Theorem 9.3 *Consider MMCIF with a designated source and a designated sink node. The following holds true.*

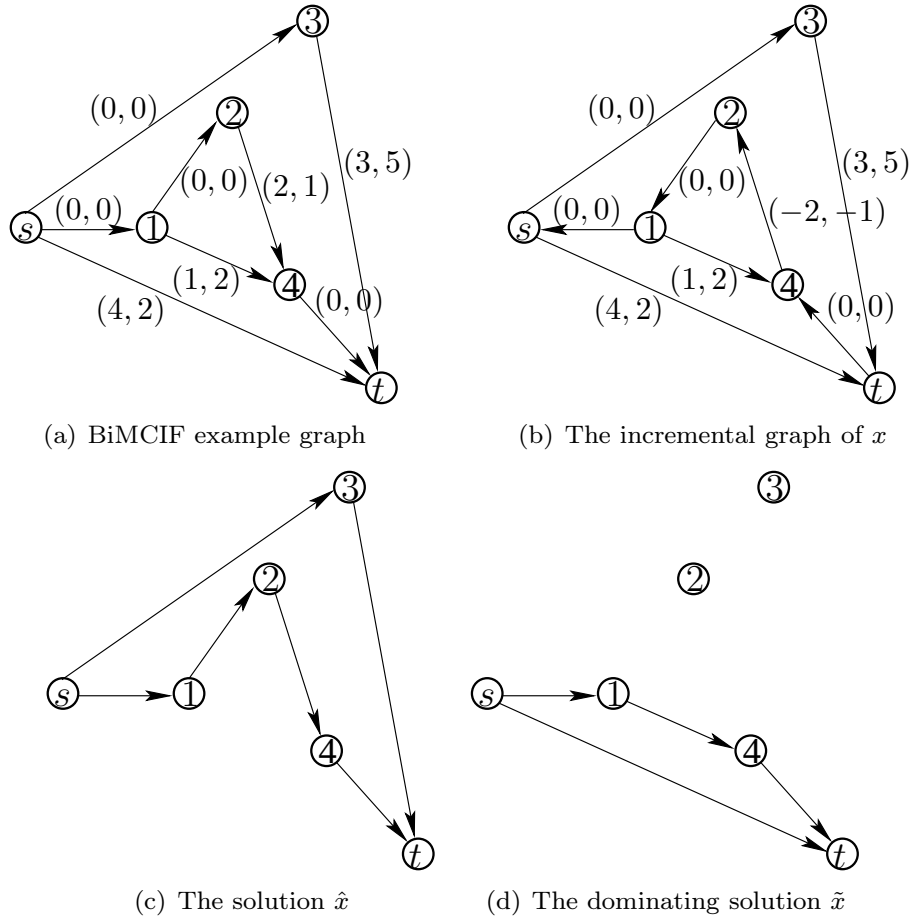
(i) *Let x be a feasible flow of value $v(x) = \rho$ and let ξ be an efficient incremental flow in $G(x)$ of value $v(\xi) = q \geq 0$. Then $\hat{x} := x \oplus \xi$ is an efficient flow of value $v(\hat{x}) = \rho + q$.*

(ii) *Let $\hat{x}$ be an efficient flow of value $v(\hat{x}) = \rho+q$, $q \geq 0$. Then for every feasible flow x with value $v(x) = \rho$, there exists an efficient incremental flow ξ in $G(x)$ of value $v(\xi) = q$ such that $\hat{x} = x \oplus \xi$.*

Proof. (i): Obviously, $v(\hat{x}) = \rho + q$. Suppose $\hat{x}$ is not an efficient flow. Then there exists a feasible flow x^* with value $v(x^*) = \rho + q$ such that $y(x^*) \leq y(\hat{x})$. Consider $\phi := x^* \ominus x$. Then ϕ is an incremental flow in $G(x)$ with value $v(\phi) = q$. Furthermore, letting $c^{G(x)}\phi$ denote the cost vector associated with ϕ in $G(x)$,

$$y(x) + c^{G(x)}\phi = y(x^*) \leq y(\hat{x}) = y(x) + c^{G(x)}\xi$$

is obtained, which implies $c^{G(x)}\phi \leq c^{G(x)}\xi$ – a contradiction, since ξ is efficient.
(ii): Let x be a feasible flow with value $v(x) = \rho$. Then $\xi := \hat{x} \ominus x$ is an incremental



(a) BiMCIF example graph (b) The incremental graph of x

(c) The solution $\hat{x}$ (d) The dominating solution $\tilde{x}$

Figure 9.4: Bicriterion minimum cost integer flow example.

flow in $G(x)$ with value $v(\xi) = q$. Suppose ξ is not efficient. Then there exists an incremental flow β in $G(x)$ such that $v(\beta) = q$ which dominates ξ, i.e. $c^{G(x)}\beta \leq c^{G(x)}\xi$. Let $\tilde{x} := x \oplus \beta$. Then

$$y(\tilde{x}) = y(x) + c^{G(x)}\beta \leq y(x) + c^{G(x)}\xi = y(\hat{x})$$

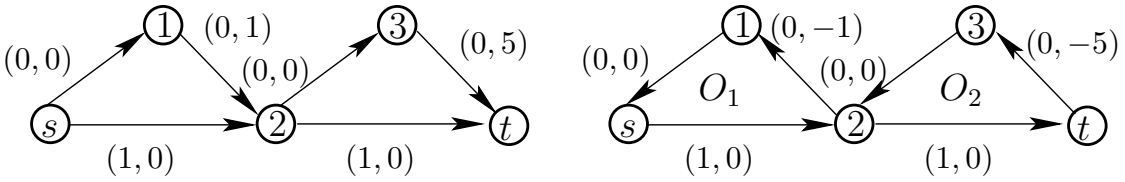
is obtained, which is a contradiction, since $\hat{x}$ is efficient. □

Example 9.2 (continued) Demonstrating the result in Theorem 9.3, the efficient flow $\tilde{x}$ of Figure 9.4(d) could only have been found from x by addition of the efficient incremental flow of value 1 composed of $P_4 \cup O$, where O is the cycle $O : 1 - 4 - 2 - 1$.

Corollary 9.4 *Let x be a feasible flow of value ρ. All efficient flows of value ρ can be found by identifying all efficient incremental flows of value 0 in $G(x)$.*

```
1 procedure MMCIF()
2    Input: A multicriteria minimum cost integer flow problem.
3    Output: X_E for the problem.
4    x := FeasibleSolution();
5    Ξ := EfficientCirculations(G(x));
6    X_E := x ⊕ Ξ;
7  end procedure
```

Figure 9.5: A multicriteria minimum cost integer flow algorithm.



(a) BMCIF example graph (b) The incremental graph of x

Figure 9.6: Bicriterion minimum cost integer flow example.

Following Corollary 9.4, the exact solution procedure for MMCIF of Figure 9.5 comes into mind. First, a feasible flow x of value $v(x)$ equal to the aggregated supply is identified in `FeasibleSolution`. Then, `EfficientCirculations` identifies the set Ξ of all efficient circulations in $G(x)$. Adding these to x yields all efficient solutions to the original problem.

An implementation of the essential subprocedure `EfficientCirculations` should exploit any kind of structural properties of efficient circulations. We know from the third property of Proposition 9.1, that circulations are decomposed of cycles in $G(x)$. Is it also true, that all such cycles must be efficient for the entire circulation to be efficient? The following example answers this questions in the negative.

Example 9.5 Consider the graph of Figure 9.6(a), where one unit of flow must be sent from s to t. All arc bounds are assumed to be $[0, 1]$. Four paths from s to t exist, of which the paths $P_1 : s-1-2-3-t$, $P_2 : s-1-2-t$ and $P_3 : s-2-t$ are efficient. In the incremental graph of $x := P_1$ (depicted in Figure 9.6(b)), two cycles O_1 and O_2 and three circulations $\xi_1 = O_1$, $\xi_2 = O_2$, and $\xi_3 = O_1 \cup O_2$ exist. The efficient circulation ξ_3 is seen to include the dominated cycle O_1.

With the negative result of the previous example, one core issue of multicriteria minimum cost integer flow optimization is still to characterize circulations in an efficient way. Obviously, finding a complete description of all efficient circulations may also provide vital information on the characterization of a minimal set of incremental flows which is sufficient to maintain connectivity of the adjacency graph (see Section 3.1).

IV

Job scheduling: A practical application

Chapter 10

Scheduling using tabu search

This chapter focuses on a specific practical problem faced by the Danish telecommunications net operator, Sonofon. By the end of each day a rather large number of jobs (End-of-Day jobs) have to be processed on three available servers. Each job is preassigned to one of the three servers and the objective is to schedule the jobs on the machines in order to minimize makespan. This task is complicated by the fact that a large number of precedence constraints among the jobs must be fulfilled, that time windows must be obeyed and that capacity limitations must be respected. In addition, the jobs are *elastic* which means that the duration of a particular job depends on the capacity assigned to the job. Elasticity of jobs complicates the problem considerably and has to the best of my knowledge not yet been considered in large-scale scheduling.

The applications of scheduling problems are wide-spread, and hence a considerable amount of promising research has been devoted to such problems both within the operations research literature and the computer science literature. Especially during the past decade, algorithms merging operations research techniques and constraint programming have proved efficient as exact solution methods for solving scheduling problems. Among a number of interesting constraint programming contributions to small- or medium-scaled scheduling problems, the work by Baptiste and Le Pape [5], Baptiste, Le Pape, and Nuijten [6], Hooker [77], Hooker and Ottosson [78], and Jain and Grossmann [84] should be mentioned. For large-scale problems, in particular metaheuristics have shown promising results.

One classical metaheuristic that has been successfully applied to scheduling problems is tabu search, due to Glover [60] and Glover and Laguna [61]. The papers on tabu search are numerous, but let me for brevity only mention a few which all appeared recently and consider scheduling problems. Grabowski and Wodecki [62] consider large-scale flow shop problems with makespan criterion and develop a very fast tabu search heuristic focusing on a lower bound for the makespan instead of the exact makespan value. Ferland, Ichoua, Lavoie, and Gagné [46] consider

a practical problem of scheduling internships for physician students and propose several variants of tabu search procedures. The last three papers all consider the problem of scheduling a number of jobs to a set of heterogeneous machines under precedence constraints, with the objective of minimizing makespan. In Porto, Kitajima, and Ribeiro [129] a parallel tabu search heuristic is developed and proved superior to a widely used greedy heuristic for the problem. In Chekuri and Bender [20] a new approximation algorithm is presented, but unfortunately, no computational results are reported. Finally, in Mansini, Speranza, and Tuza [102] jobs with up to three predecessors each are considered among groups of jobs requiring the same set of machines. The problem is formulated as a graph theoretical problem. In the paper a number of approximation results are provided, but no computational experience is reported.

Clearly, the vast solution space and the complexity of the present problem called for a heuristic procedure. Due to the high flexibility of tabu search and its promising results with scheduling problems, that method was chosen.

The remaining part of this chapter is organized as follows. In Section 10.1, I present the practical problem offered by Sonofon and derive a new approximate method for scheduling elastic jobs. The scheduling problem is then formulated using a hybrid integer programming and constraint programming model. In Section 10.2, I give a thorough introduction to the developed tabu search heuristic, and computational results are provided in Section 10.3.

10.1 Problem formulation

The Danish telecommunications net operator, Sonofon, faces a three-machine scheduling problem, with 346 End-of-Day jobs (EOD). Each job is dedicated to a particular server in advance and it has to be processed on that server without *preemption*[8]. This means that the allocation of jobs to machines is not part of the problem.

The scheduling time horizon runs from 7.00 pm to 8.00 am, and each job receives a time window in which it should be processed. The time windows are wide, leaving numerous feasible starting times for each job. Since most scheduling tools applying constraint programming rely heavily on propagation techniques, the wide time windows have a negative influence on the performance of such scheduling packages. The time windows will be explored further in Section 10.2.1. Since the servers immediately after completing the EOD-jobs are assigned to other operations, the objective is to minimize makespan.

Because of interrelations between jobs, a number of precedence constraints must be fulfilled. It might occur that a job needs information from a database to which another job (a predecessor) has written earlier.

[8] Preemption means that jobs can be interrupted during processing.

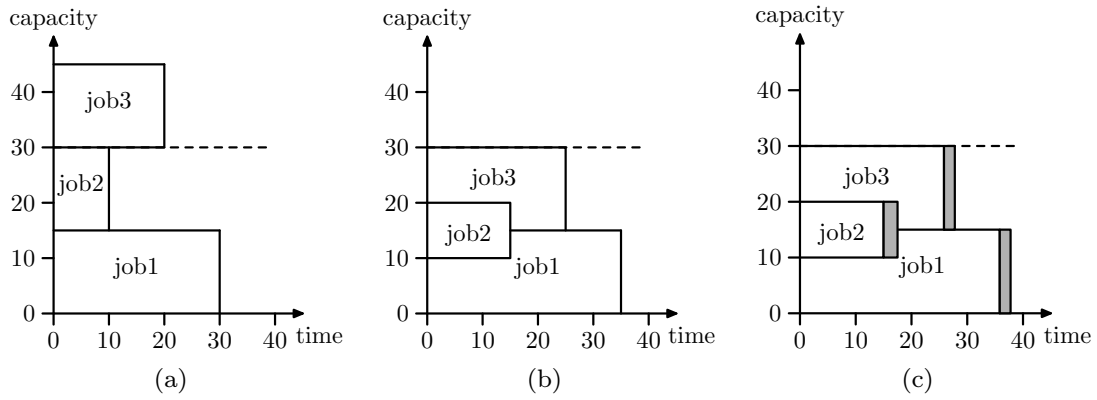


Figure 10.1: Relation between duration and capacity consumption of jobs.

In a real-world application, each job can be processed with varying capacity consumption during its runtime, as illustrated by job 1 in Figure 10.1(b). However, due to limitations of server exploitation, we can assume that each job has an upper bound of capacity consumption. In Figure 10.1(a) is illustrated a situation in which three jobs are placed at a machine to start processing at time 0. Each of the three jobs is assumed to have a maximal capacity consumption of 15 units, and the machine has capacity 30. Since all the jobs are scheduled to start at time 0, they must share the available capacity, as shown in Figure 10.1(b). Observe that, in Figures 10.1(a) and 10.1(b), the two corresponding boxes for a job have the same area. This would be an incorrect representation of a real-world application, since in general the product of duration and capacity consumption of a job increases with decreasing capacity due to lost server efficiency from *swapping*[9]. This fact is represented by the inclusion of the shaded area in Figure 10.1(c).

In this set-up it is assumed that the capacity consumption for a job remains constant during its runtime. Opposed to other literature on large-scale scheduling, time and capacity consumption are not restricted to be given beforehand. Instead, jobs are elastic, and hence the time and capacity consumption are allowed to be found during the optimization process. I deal with the non-linear functionality between time and capacity consumption by a rough approximation representing each job as a choice between three boxes, (see Figure 10.2).

The dimensions of the boxes for a given job j are explained in Figure 10.3, where cap_j ($time_j$) corresponds to the capacity (duration) for the job box having the least capacity consumption (and hence the longest duration)[10]. The second and third column gives the capacity and time consumption for a given box, and the last column gives the product of capacity consumption and duration. Notice

[9] Swapping or *trashing* means time being spent for reading jobs into and out of the temporary memory, not processing any job.

[10] Time constitutes an average longest runtime provided by Sonofon from historical data.

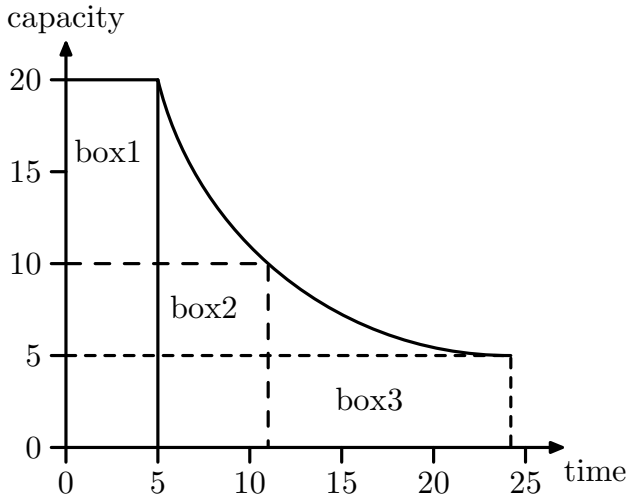


Figure 10.2: Three representations of a job.

b	cap_{jb}	$time_{jb}$	$(cap_{jb} \cdot time_{jb})$
1	$4 \cdot cap_j$	$25/121 \cdot time_j$	$100/121 \cdot cap_j \cdot time_j$
2	$2 \cdot cap_j$	$5/11 \cdot time_j$	$10/11 \cdot cap_j \cdot time_j$
3	cap_j	$time_j$	$cap_j \cdot time_j$

Figure 10.3: Dimensions of boxes representing job j.

that, by a 50% decrease in capacity, this product increases by 10%. This trade-off between capacity assigned to a particular job and its duration was determined in correspondence with Sonofon and reflects the specific problem rather closely.

Since the representation of scheduling problems is greatly simplified using the terminology from constraint programming, I too, shall adapt such a notation. Hence, for this scheduling problem, a hybrid integer programming and constraint programming model is derived, which is to be solved by a heuristic procedure, more specifically by a tabu search algorithm. The following notation is introduced.

$\mathcal{M}$ $:= \{1, 2, 3\}$ – Machines
$\mathcal{J}$ $:= \{1, \ldots, |\mathcal{J}|\}$ – Jobs
$\mathcal{I}$ $:= \{(j, k) \ : \$ job j shall precede job $k\}$ – Precedence constraints
$\mathcal{H}_m := \{j \ : \$ job j shall be processed on machine $m\}$ – Job-machine constraints
$\mathcal{B}$ $:= \{1, 2, 3\}$ – Boxes for each job

Notice that, $\cup_{m \in \mathcal{M}} \mathcal{H}_m = \mathcal{J}$ since all jobs are allocated to a particular server in

advance. For each job j, we introduce the four variables:

$j.start$ – Starting time of job j
$j.time$ – Duration of job j
$j.end$ – Completion time of job j
$j.cap$ – Capacity consumption of job j

connected by the implicit constraint $j.start + j.time = j.end$. In addition, we have the parameters:

R_m – Capacity available on machine m, $\forall m \in \mathcal{M}$
$[l_j, u_j]$ – Time window for job j, $\forall j \in \mathcal{J}$
$time_{jb}$ – Duration of the b'th box for job j, $\forall j \in \mathcal{J}$, $\forall\, b \in \mathcal{B}$
cap_{jb} – Capacity consumption of the b'th box for job j, $\forall j \in \mathcal{J}$, $\forall\, b \in \mathcal{B}$

Let x_{jb} denote a binary variable which is 1 if box b is chosen for job j and 0 otherwise. Introducing the artificial job *makespan* with zero duration, the model can be stated as follows:

$$\begin{aligned}
&\min \quad makespan.end \\
&\text{s.t.} \quad \sum_{b \in \mathcal{B}} x_{jb} = 1 && \forall j \in \mathcal{J} && \text{(10.1a)} \\
&\qquad j.time = \sum_{b \in \mathcal{B}} (x_{jb} \cdot time_{jb}) && \forall j \in \mathcal{J} && \text{(10.1b)} \\
&\qquad j.cap = \sum_{b \in \mathcal{B}} (x_{jb} \cdot cap_{jb}) && \forall j \in \mathcal{J} && \text{(10.1c)} \\
&\qquad l_j \leq j.start && \forall j \in \mathcal{J} && \text{(10.1d)} \\
&\qquad j.end \leq u_j && \forall j \in \mathcal{J} && \text{(10.1e)} \\
&\qquad j \; precedes \; makespan && \forall j \in \mathcal{J} && \text{(10.1f)} \\
&\qquad j \; precedes \; k && \forall\, (j,k) \in \mathcal{I} && \text{(10.1g)} \\
&\qquad cumulative \begin{pmatrix} \{j.start\}_{j \in \mathcal{H}_m} \\ \{j.time\}_{j \in \mathcal{H}_m} \\ \{j.cap\}_{j \in \mathcal{H}_m} \\ R_m \end{pmatrix} && \forall m \in \mathcal{M} && \text{(10.1h)} \\
&\qquad x_{jb} \in \{0,1\} && \forall j \in \mathcal{J},\ \forall b \in \mathcal{B} && \text{(10.1i)}
\end{aligned}$$

where *cumulative* is a *global constraint* in constraint programming stating that, at all times, the total capacity is not exceeded by the capacity consumption of

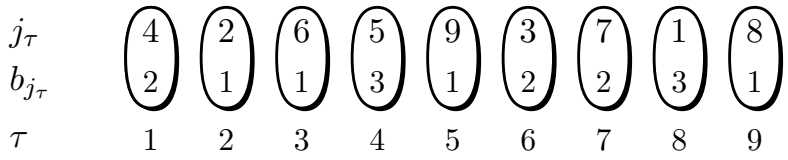


Figure 10.4: Sequence and box choices for example with 9 jobs.

running jobs. The constraint can be rewritten as

$$\begin{aligned} & cumulative\left((t_1,\ldots,t_q),(d_1,\ldots,d_q),(r_1,\ldots,r_q),R\right) \\ \Updownarrow & \\ & \sum_{j\,:\,t_j\le t\le t_j+d_j} r_j \le R,\ \ \forall t \end{aligned}$$

where the vector $(t_1,\ldots,t_q)$ represents starting times of jobs $1,\ldots,q$, with duration $(d_1,\ldots,d_q)$ and capacity consumption $(r_1,\ldots,r_q)$. Available capacity is R.

The above constraints (10.1a) choose a box for each job, yielding a specific duration and capacity consumption in cooperation with (10.1b) and (10.1c). Constraints (10.1d) and (10.1e) consider time windows. Constraints (10.1f) together with the objective function minimize the completion time of the last job. Constraints (10.1g) handle precedence constraints (j precedes k means $j.end \le k.start$), whereas constraints (10.1h) handle resource consumption for each machine.

10.2 Tabu search

To obtain a solution to the given problem, the starting time and the box size for each job are needed since then the completion times, the durations and the capacity consumptions are implicitly determined. However, due to wide time windows, numerous possible starting times exist for each job, which prevent me from using the starting times explicitly in the solutions. Instead, a solution is composed of a *list* Θ *of box choices* (one for each job) and a *sequence* Π specifying the order of the starting times. Given a sequence, j_τ denotes the number of the job at position τ in the following. The sequence specifies that since job j_1 is before job j_2 in the sequence, j_2 must start no earlier than j_1. A solution to a problem with 9 jobs is shown in Figure 10.4, where the sequence Π is defined by $j_1 \ldots j_9$ and the box choices by the box numbers b_{j_τ} stated below.

The tabu search procedure moves from one solution to the next either by changing the sequence or changing one of the box sizes. The neighbourhood structure is outlined in Section 10.2.4. Given a solution (Θ, Π) i.e. a box size for each job and a job sequence, the corresponding optimal starting times can be found or infeasibility can be proved. This means that the size of the solution space has

been dramatically decreased without excluding optimal solutions by considering a sequence instead of starting times. How to complete the solution to find the exact starting times for each job is discussed in Section 10.2.3.

A solution (Θ, Π) is feasible, if it is possible to schedule all jobs according to the sequence and to the box sizes and still satisfy all time windows, capacity constraints and precedence constraints. It turns out that the problem of finding an initial solution is very hard, but a heuristic method for solving this problem is presented in Section 10.2.2.

Elements and features of the tabu search such as the neighbourhood structure, tabu lists, intensification strategies and diversification strategies are discussed in Sections 10.2.4, 10.2.5, 10.2.6 and 10.2.7, respectively. Part of the notation is inhereted from Chiang and Russell [22].

10.2.1 Preprocessing

In order to detect infeasible solutions quickly, the time windows are tightened by considering precedence constraints. If a job j, has a time window $[0, u]$, but at the same time is a successor of another job $\hat{j}$, then the time window can be adjusted to start at the earliest completion time for job $\hat{j}$. To do this, a *precedence graph* $G^{\mathcal{I}}$ is constructed where all jobs are represented by a node, and all precedence constraints by a directed arc between the two nodes involved, pointing away from the predecessor.

For all connected components in the precedence graph, the following procedure adjusts the left end points of the time windows. Let $C \subseteq G^{\mathcal{I}}$ be a connected component, and let $j \in C$ be a job in C. Then l_j denotes the earliest starting time, and $time_{j1}$ denotes the minimal duration for job j. Let $\mathcal{I}_j$ denote all predecessors of job j and note that $\mathcal{I}_j \subset C$. The earliest starting times for the jobs in C are now adjusted by setting $l_j = \max\{l_j, l_i + time_{i1},\ \forall i \in \mathcal{I}_j\}$ for all $j \in C$, but in an order such that all predecessors of j have been adjusted before j. Such an order exists, since otherwise a directed cycle would exist in C, and the jobs would be impossible to schedule. The latest completion times can be adjusted in a similar manner by starting with the jobs in C having no successors.

10.2.2 Initial solution

Garey and Johnson [56] have shown that, for a similar set-up, the decision problem on determining the existence of a feasible schedule with a makespan less than a given deadline (in the present application 8.00 am) is $\mathcal{NP}$-complete. A heuristic procedure to generate an initial feasible solution for this particular instance is outlined in Figure 10.5. The procedure is divided into three parts, where the first part (lines 2–4) uses the precedence graph to generate a sequence Π. The second part (line 5) chooses a list of box sizes Θ, after which feasibility of (Θ, Π) is checked in the third part (lines 6–8). This check is performed during the process of

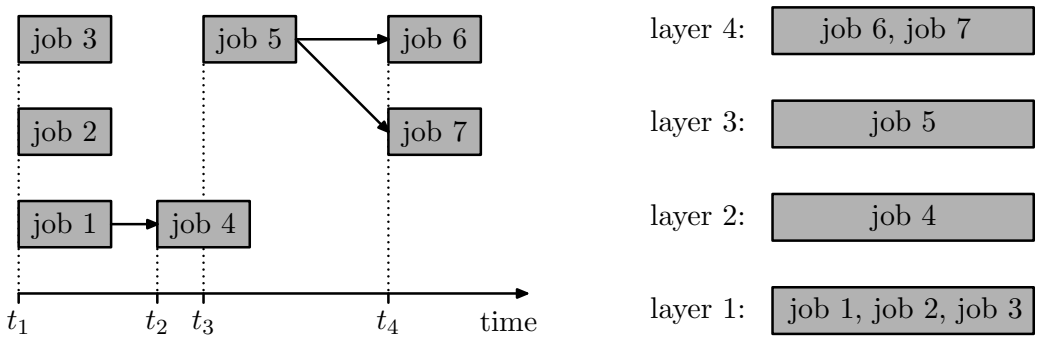
```
1 procedure InitialSolution()
2     Π_f := GenerateForwardSequence(G^I, {[l_j, u_j], time_j1}_{j∈J});
3     Π_b := GenerateBackwardSequence(G^I, {[l_j, u_j], time_j1}_{j∈J});
4     Π := GenerateSequence(Π_f, Π_b);
5     Θ := ChooseBoxes({R_m}_{m∈M}, {cap_j3}_{j∈J});
6     if ((Θ, Π) is infeasible) then
7         (Θ, Π) := FindInitialSequenceUsingTabuSearch((Θ, Π));
8     end if
9   end procedure
```

Figure 10.5: Finding an initial feasible solution for the scheduling problem.

completing the solution as described in Section 10.2.3. If (Θ, Π) is infeasible, the tabu search procedure is used to find a feasible solution from (Θ, Π). Subsequently, I describe the procedures of Figure 10.5.

`GenerateForwardSequence` Notice, to obtain a feasible solution, three groups of constraints must be fulfilled simultaneously, namely precedence constraints, time window constraints and capacity constraints. To ensure fulfilment of the precedence constraints, the precedence graph $G^{\mathcal{I}}$ described in Section 10.2.1 is used to divide the jobs into *layers*. The successor of a job will always be in a higher layer than the job itself, and the jobs in one layer cannot start before all jobs in preceding layers have started.

Figure 10.6(a) illustrates an example with seven jobs which must be divided into layers. In the figure the jobs are placed at their earliest starting time indicated by dotted lines and precedence constraints between jobs are indicated by arrows. In this situation, job 4 must be in a higher layer than job 1 since it is a successor of job 1. However, if jobs are only divided according to the precedence constraints, the process faces the risk of assigning jobs with late time windows to an early processing layer. This could happen if an entire component of the precedence graph has to be processed after a certain time, but the first job is assigned to layer 1. Jobs from other components, which could be processed early, would then be stalled if they were in layer 2, and the entire schedule would be delayed. This corresponds to assigning job 5 from Figure 10.6(a) to layer 1. If that happens, job 4 must be scheduled after time t_3 since it would be allocated to a higher layer than job 5. Hence in the derivation of layers, a variable *start* is introduced and initialized to 0. Then jobs that have no predecessors and are able to start before or at time *start* are assigned. In Figure 10.6(a), job 1, job 2 and job 3 would hence be assigned to layer 1 as seen in Figure 10.6(b). All their successors, having no other predecessors and being able to start before or at time *start*, are then scheduled in the next layer etc. When no more jobs can be assigned due to either time window constraints or precedence constraints, the variable *start* is increased



(a) Jobs to schedule (b) Layers

Figure 10.6: Illustration of how jobs are divided into layers.

by one time unit, and a new level of layers can be derived with jobs being able to start before the new limit.

The jobs are numbered consecutively, starting with the jobs on the lowest layer. The difference between the width of the time window and the duration of the smallest box expresses a degree of freedom for a given job. The higher this difference is, the higher degree of freedom the job possesses. Within each layer the jobs are numbered in an increasing order of this degree of freedom. This continues in an iterative fashion, until all jobs are numbered and we have a sequence containing all jobs. After the first three jobs from Figure 10.6(a) have been assigned to layer 1, the variable *start* is increased until it is equal to t_2 at which point job 4 can be assigned to layer 2. As the algorithm continues, the remaining jobs will be assigned as shown in Figure 10.6(b).

`GenerateBackwardSequence` This function is similar to `GenerateForwardSequence`, except the layers are generated backwards. This means that the layer containing the last jobs are generated first, and then the preceding layers are generated one by one. Again the successor of a job will always be in a higher layer than the job itself, and the job in one layer cannot start before all jobs in the preceding layer have started.

`GenerateSequence` The forward sequence Π_f has the disadvantage that all jobs without precedence constraints and time windows are scheduled in the first layer, e.g. job 2 and job 3 in Figure 10.6(a). This means that jobs which could have been scheduled later might delay some of the large components of the precedence graph. The backward sequence Π_b has the opposite problem since, in this case, the jobs with few constraints are scheduled in the last layer and might cause jobs to break their time windows. Hence, with `GenerateSequence`, a new sequence Π is obtained by taking a convex combination of the two sequences Π_f and Π_b. This is

done by calculating the convex combination of the positions in the two sequences for each job and then generating a sequence according to these numbers. Ties are broken arbitrarily. Notice that the new sequence still satisfies all precedence constraints.

`ChooseBoxes` A list of box sizes Θ is chosen considering each job j on machine m according to the following scheme.

$$\begin{aligned} x_{j1} &= 1 \quad \text{if } 0 < cap_{j3} \leq \frac{R_m}{10} \\ x_{j2} &= 1 \quad \text{if } \frac{R_m}{10} < cap_{j3} \leq \frac{R_m}{4} \\ x_{j3} &= 1 \quad \text{if } \frac{R_m}{4} < cap_{j3} \end{aligned} \tag{10.2}$$

These choices have proved efficient in the particular problem.

`FindInitialSequenceUsingTabuSearch` If the solution from (Θ, Π) is infeasible, the tabu search procedure is used to find a feasible solution. The problem is relaxed by setting $u_j = \infty$ for all j, i.e. the time windows have no upper limit. Notice that this problem always has a feasible solution when the capacity requirement for each job is less than the capacity on the corresponding machine. The objective in this part of the tabu search procedure is to minimize the number of jobs which violate their original time windows, and the search stops when a solution with value 0 has been found.

The implemented idea corresponds to running a tabu search procedure in two phases – one phase ensuring feasibility and another phase minimizing makespan. These phases could alternatively be merged by using weights to yield an objective function composed of the makespan criterion and a penalty for the broken time windows, see Chiang and Russell [22]. However, since an implementation of this idea showed poor performance compared to the two-phase tabu search procedure, focus is on the two-phase approach.

10.2.3 Completing a solution

As mentioned, the solutions used in the tabu search procedure consist only of a list of box choices Θ and a job sequence Π which determines the order of the starting times. This solution must be completed to include the exact starting and completion times for each job, since fulfilment of time windows and capacity constraints must be checked in order to prove feasibility of the solution. Since this check is done for all considered moves in each iteration, the efficiency of this subroutine has great influence on the overall performance of the tabu search procedure.

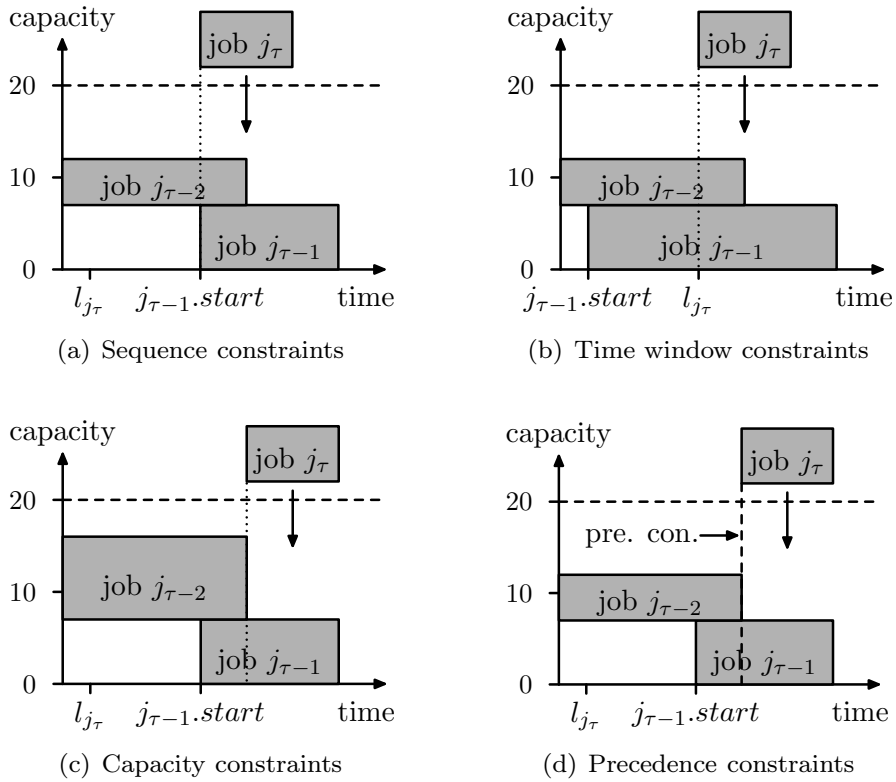
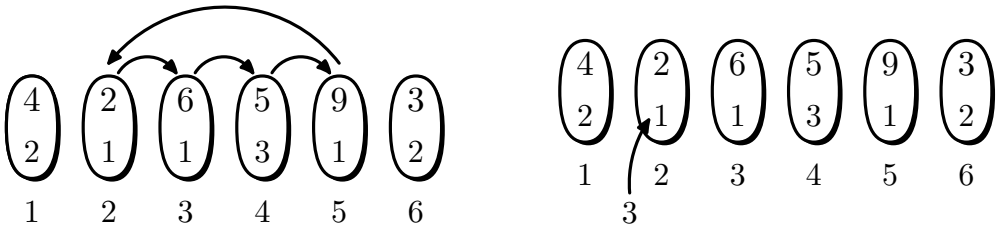


Figure 10.7: Scheduling the job j_τ.

Before the procedure is outlined, let me mention that the sequences given to the procedure always satisfy the precedence constraints, i.e. if j must be completed before $\hat{j}$ can start, then j will always precede $\hat{j}$ in the sequence.

The procedure exploits that an optimal schedule with respect to the given sequence and box choices can be generated by scheduling one job at a time in the order of the sequence without backtracking. Since j_τ is the job at position τ in the sequence, we know that when j_τ is about to be scheduled, all jobs $j_{\bar{\tau}}$ with $\bar{\tau} < \tau$ have been scheduled and $j_{\tau-1}.start \leq j_\tau.start$ due to the sequence. Furthermore, all the jobs that have been scheduled so far start before or at $j_{\tau-1}.start$ and therefore the capacity consumption on each machine must be decreasing in time after $j_{\tau-1}.start$. The optimal starting time for j_τ will hence be the first time after $\max\{l_{j_\tau},\ j_{\tau-1}.start\}$ and after $\max\{j_{\bar{\tau}}.end\ :\ (j_{\bar{\tau}}, j_\tau) \in \mathcal{I}\}$ for which the capacity consumption on the machine m used to process j_τ is less than or equal to $R_m - j_\tau.cap$. This means that a job is started the first time the four conditions shown in Figure 10.7 are fulfilled.

When the starting time of j_τ has been determined, the procedure checks if $j_\tau.end \leq u_{j_\tau}$ to see if the time window constraint is satisfied. If so, $j_{\tau+1}$ is scheduled and otherwise the solution is infeasible and the procedure stops. If all



(a) Position move (b) Box move

Figure 10.8: Illustration of moves in the tabu search procedure.

jobs are scheduled, we have a feasible solution since all constraints are satisfied and the makespan is equal to $\max\{j.end | j \in \mathcal{J}\}$.

10.2.4 Neighbourhood

To characterize the neighbourhood of a given solution (Θ, Π), two kinds of moves are defined. A *position move* moves a job to a new position while the box sizes in Θ are kept constant, whereas a *box move* maintains the job sequence Π but changes the box choice for a single job. Figure 10.8 illustrates both kinds of moves. Notice in Figure 10.8(a) that, when job 9 at position 5 in the job sequence is moved to position 2, not only does job 9 get a new position, but the jobs at position 2, 3 and 4 are moved to the subsequent position.

The position move described above has been chosen instead of alternatives, such as exchanging two jobs, since the precedence constraints do not limit the flexibility of this move. Consider the move from Figure 10.8(a) and imagine that job 2 must precede job 6 and job 6 must precede job 5. In that case three "exchanges" of jobs are needed to perform the single position move shown in the figure.

The neighbourhood for solution (Θ, Π) can be characterized as the union of solutions obtained by a single box move and solutions obtained by a single position move which fulfils the precedence constraints. The cardinality of the neighbourhood is $O(|\mathcal{J}|^2)$ due to the large number of position moves, and in the present implementation we must consider approximately 120,000 moves (some are ignored due to violation of the precedence constraints) for each solution. The ability to select only part of the neighbourhood for examination is therefore crucial. I use two methods for limiting the number of possible moves.

Restricting position moves

By introducing a limit *moveLimit* on how far a job can move, the number of considered position moves are reduced. This leads to faster iterations but might restrict the search from choosing some very good solutions. To avoid the search

from stalling due to this restriction, the entire neighbourhood is examined every time the algorithm has performed non-improving moves for a predefined number of iterations. This makes the search capable of performing a single time consuming move and then a number of fast iterations to exploit the new conditions.

Candidate lists

The *Elite candidate list* approach (see [61]), is used to limit the number of position moves by only evaluating moves belonging to candidate lists. In this set-up two lists $Cand_1$ and $Cand_2$ are used, and they are constructed by evaluating the neighbourhood of the initial solution. All moves which lead to an improving makespan are stored in $Cand_1$, and all moves leading to the same makespan are stored in $Cand_2$. In the following iterations only moves from the two candidate lists are considered. First the moves in $Cand_1$ are evaluated and, if one of these moves leads to an improving makespan, the best move is chosen. If $Cand_1$ does not contain an improving move, the moves in $Cand_2$ are evaluated and the best move considering both $Cand_1$ and $Cand_2$ is chosen.

When a move has been chosen from one of the candidate lists, both lists are updated by deleting all moves conflicting with the chosen one. This means that, if a position move for job j is chosen, then all other position moves for job j are deleted from the candidate lists and correspondingly for box moves. The candidate lists are used until no improving move has been found in the lists. When this happens, both lists are deleted and two new lists are generated by examining the possible moves of the current solution. Notice that this might not be an evaluation of all possible moves, since the position moves might be restricted as explained above. The underlying assumption of the strategy is that a move which performs well in the current solution will probably also lead to improvements in the following iterations.

10.2.5 Tabu list

The corner stone in tabu search is the use of short-term memory by generating a tabu list $Tabu$. The tabu list stores the move from an iteration and keeps it for $timeTabu$ iterations. This is done by keeping the iteration number $\hat{i}$ from the iteration in which the move is made tabu and deleting the move from $Tabu$ when the iteration number exceeds $\hat{i} + timeTabu$. The tabu list differentiates between the two kinds of moves, but the number of the job involved is always stored. If a box move is performed for job j, the tabu list restricts job j from performing a new box move in the following $timeTabu$ iterations, unless the *aspiration criterion* is satisfied. If a position move is moving job j from position τ, the tabu list restricts the search from performing a new position move taking job j to a position $\bar{\tau}$ where $|\tau - \bar{\tau}| \leq tabuPosLimit$ in the following $timeTabu$ iterations, unless the aspiration criterion is satisfied.

The aspiration criterion checks if an improved makespan can be obtained by performing a forbidden move. If this is the case, the tabu restriction is suspended and the search is allowed to perform the move.

The tabu search implemented here has the ability to dynamically adjust the variable *timeTabu* which determines the number of iterations for which a move is tabu. Variable *timeTabu* is decreased by the parameter $z_{\downarrow} = 0.9$ every time the search is trapped in a solution without a non-tabu or feasible neighbour and increased by $z^{\uparrow} = 1.1$ when the same makespan has been found in many successive iterations.

In addition a variable *steps* is counting the number of moves without a change in *timeTabu*, and *timeTabu* is decreased by $z_{\downarrow}$ if *steps* exceeds a fixed threshold *movingAverage*. This adjustment helps the search to avoid a lot of bad moves which could be the result of a long tabu list.

10.2.6 Intensification strategy

A list $IntenArray$ holds moves which have led to improvements of the makespan. The moves are kept for *intensize* iterations and corresponding moves for the same job are not allowed while the move is in the $IntenArray$. For example, if a position move is performed for job j in iteration $\hat{i}$, a new position move cannot be performed for j before iteration $\hat{i} + intensize$. However, the intensification status is not considered if a job satisfies the aspiration criterion. In this case the job can be chosen even though the move is in the intensification array.

10.2.7 Diversification strategies

The algorithm contains two kinds of diversification strategies. The first strategy is active throughout the search and helps the algorithm to perform a thorough search in the current region of the solution space, while the other strategy forces the search to change the region.

Penalized move value

The quality of a move is measured by $moveValue$, which gives the difference between the current makespan and the makespan obtained by performing the move, $moveValue = newTime - curTime$. This $moveValue$ could be used to guide the search but, in order to implement the first diversification strategy, a *penalized move value* *pmv* is introduced. The *pmv* takes into account how many times the job has been moved before:

$$pmv = \begin{cases} moveValue + \alpha \cdot Move[j] & \text{if } moveValue \geq 0 \\ moveValue & \text{if } moveValue < 0 \end{cases}$$

where $Move[j]$ counts the number of moves performed by job j and α is a non-negative parameter to adjust the penalty. By choosing moves according to lowest pmv, the algorithm automatically follows the diversification strategy.

Escape procedure

In order to move the search from one region of the solution space to another, an escape procedure is invoked when too many successive iterations have resulted in the same makespan. The procedure makes a number of random moves which lead the algorithm away from the current region. During the escape procedure, only feasible moves are allowed, since a feasible solution must be available when all the moves are performed.

The general tabu search procedure adjusted according to the strategies above can be seen in Figure 10.9. Notice that the functions `FindImprovingMove` and `FindBestMove` include the procedure to complete a solution, and they also check if the moves are allowed due to both the tabu list $Tabu$ and the intensification strategy controlled by $IntenArray$.

10.3 Computational results

In this section, I present the computational results of the tabu search procedure. In addition to solving the problem faced by Sonofon, extensive testings on random large-scale scheduling instances are performed. The results for the practical application show that significant improvements can be gained within a short amount of time, while the additional tests show the robustness and speed of the algorithm to instances with varying structure.

The results obtained by the tabu search procedure are compared to a lower bound which is found by disregarding the precedence constraints. Therefore the three machines can be scheduled independently. Then for each job total capacity consumption is assumed to be the product of capacity consumption and duration for the smallest possible box (box 1). Now, for a particular machine a sequence is constructed by ordering the jobs according to the starting time of their time windows, with ties broken arbitrarily. When the jobs are scheduled according to this sequence and treated as totally elastic without variation of the total capacity consumption, a lower bound on the makespan is obtained.

The algorithm was implemented in C++ and compiled with the GNU C++ compiler using optimize option `-O`. Moreover, all computations were performed on an Intel Xeon 2.67 GHz computer with 4 GB RAM using operating system Red Hat Linux version 9.0.

```
 1 procedure TabuSearch()
 2     time := 0;
 3     {[l_j, u_j]}_{j∈J} := AdjustTimeWindows(); [10.2.1]
 4     (Θ, Π) := InitialSolution(); [10.2.2]
 5     iteration := 0;
 6     while ((iteration < maxIteration) & (time < timeLimit)) do
 7         curMove := ∅;
 8         Tabu := UpdateTabuList();
 9         IntenArray := UpdateIntenArray();
10         if (Cand_1 ∪ Cand_2 = ∅) then
11             (Cand_1, Cand_2) := CandLists((Θ, Π)); [10.2.4]
12         end if
13         curMove := FindImprovingMove(Cand_1 ,(Θ, Π));
14         if (curMove = ∅) then
15             curMove := FindBestMove(Cand_1 ,Cand_2 ,(Θ, Π));
16         end if
17         if (curMove = ∅) then
18             timeTabu := z_↓ · timeTabu , steps := 0; [10.2.5]
19         end if
20         else
21             (Θ, Π) := UpdateCurrentSol((Θ, Π),curMove);
22             Tabu := AddMove(curMove); [10.2.5]
23             IntenArray := AddImprovingMove(curMove); [10.2.6]
24         end else
25         (Cand_1, Cand_2) := UpdateCandLists((Θ, Π)); [10.2.4]
26         if (sameMakespanIterations = escapeRepetion) then
27             (Θ, Π) := EscapeProcedure((Θ, Π)); [10.2.7]
28         end if
29         iteration++
30     end while
31   end procedure
```

Figure 10.9: Pseudo code for the tabu search procedure. Numbers in the square brackets refer to sections in which the corresponding explanations are given.

10.3.1 The practical application

The problem faced by Sonofon consists of 346 jobs and 587 precedence constraints. The average makespan reported by Sonofon[11] is 821 minutes, hence today the average completion time exceeds the deadline by 41 minutes. This means that with the existing scheduling strategy, new hardware needs to be purchased in order to keep satisfying the given requirements.

The algorithm presented in this chapter yields a makespan of 615 minutes, which is only 4.06 percent above the lower bound computed to 591 minutes. The makespan obtained by Sonofon is 38.92 percent above the lower bound. By a direct comparison of the two makespans, it can be seen that the new schedule saves 25.09 percent of scheduling time compared to the strategy implemented by Sonofon. It is important that this pratical project has shown that the existing hardware is, in fact, sufficient to complete the jobs in time and, indeed, spare capacity is available when a good schedule is chosen.

The best solution was found in 56 min 31 sec, and hence the algorithm can be used on a daily basis to schedule the jobs which have to be processed during the night. Furthermore, Figure 10.10 shows that the significant improvements are obtained in a rather short amount of computation time, and afterwards only small improvements are made. This means that the algorithm is still applicable even though the job specifications are unknown until just prior to the actual scheduling process. The jumps for the current solution reported in Figure 10.10 are due to the escape procedure used in the diversification strategy.

In addition, the solution of the algorithm can be used to examine how the available capacity is used. Figure 10.11 shows a very uneven server exploitation during the night, and in particular if jobs were moved from machines 1 and 3 to machine 2, the makespan could be reduced.

For additional testing the problem was implemented in OPL Studio 3.7 (by ILOG [80]) where it was provided with the search strategy to start with box choices according to the scheme in (10.2) on page 144. OPL Studio with default setting was unable to solve the problem in 24 hours. In fact, within 24 hours, OPL Studio was unable even to find a feasible solution to the problem, whereas the algorithm presented in this chapter provided a feasible solution in 1 min 1 sec. The efficiency of the tabu search procedure is in particular due to the speed of the procedure explained in Section 10.2.3 which schedules the jobs when a job sequence and a list of box sizes are given. The order of magnitude for the average time used to run this procedure is 10^{-4} seconds.

I have also tested the benefits of scheduling all jobs on one large server instead of three separate ones. Within 3 hours of computation time the algorithm yields a makespan of 526 minutes and therefore supports such an implementation. This

[11] The average makespan was found using the historical data that constituted the specifications for the jobs.

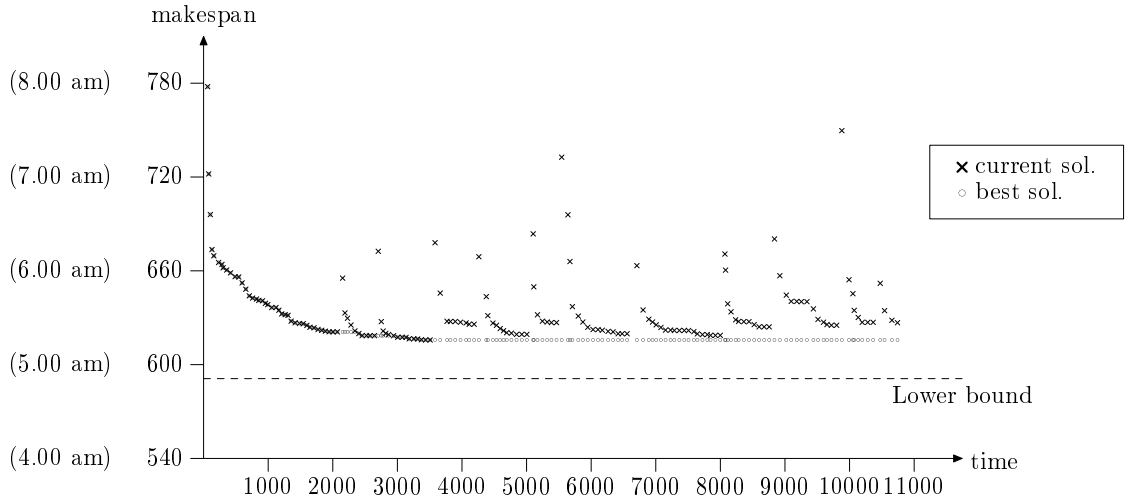
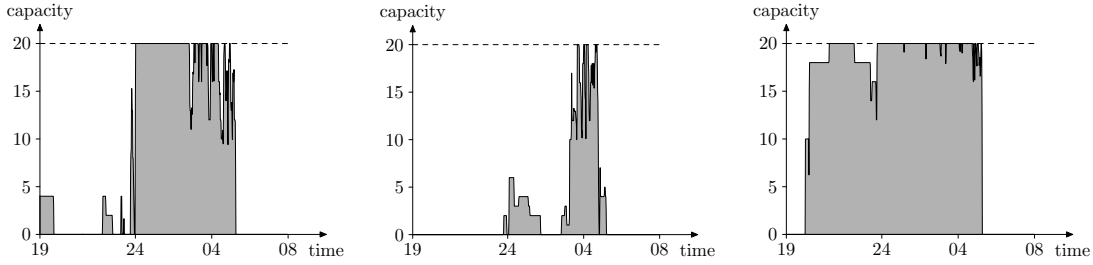


Figure 10.10: Makespan obtained by the tabu search procedure for every 50 iterations.



(a) Machine 1 (b) Machine 2 (c) Machine 3

Figure 10.11: Capacity consumption on the three servers.

scenario has been considered by Sonofon but is not implementable with their current hardware.

10.3.2 General large-scale scheduling instances

In order to test the robustness of the tabu search procedure, numerous random instances were generated. The first half of these instances has a structure resembling the structure seen in the data from Sonofon, while the structure in the rest of the instances are random.

The tabu search procedure was tested on instances with $|\mathcal{J}| = 150, 300$, and 400 jobs. For each number of jobs, the number of precedence constraints is either $\frac{1}{2} \cdot |\mathcal{J}|$, $|\mathcal{J}|$ or $2 \cdot |\mathcal{J}|$. Furthermore, since the jobs are randomly generated, 10 instances have been solved for each specific number of jobs and number of prece-

		CPU (avg.)		Makespan		Deviation (%)		
$\|\mathcal{J}\|$	$\|\mathcal{I}\|$	Initial	Best	min	max	min	avg.	max
150	75	0.00	68.26	335	355	0.00	0.06	0.30
150	150	0.00	17.64	334	433	0.00	0.11	0.82
150	300	0.00	3.04	480	630	0.00	0.16	0.80
300	150	0.00	1170.54	342	386	0.00	1.27	3.40
300	300	0.00	599.25	369	486	0.00	0.60	2.41
300	600	0.50	86.99	623	767	0.00	0.20	1.14
400	200	0.00	1419.62	382	430	0.23	5.60	9.42
400	400	0.00	1250.09	399	458	0.00	2.20	5.00
400	800	51.51	482.98	618	777	0.00	0.15	1.07

Table 10.1: Computational results for instances resembling Sonofon data.

dence constraints to give a general idea of the performance of the tabu search procedure. The algorithm was allowed to run for 30 minutes on each instance. The CPU time used to find an initial feasible solution and the time used to find the best solution within the 30 minutes time limit are reported in seconds. To give an idea of the size of the makespan, I give the minimum and maximum makespan for the 10 instances and, finally, report both minimum, average and maximum deviation from the lower bound in per cent.

The servers at Sonofon face two peaks during the night. One at the beginning of the process where a large number of jobs are allowed to start, and one at 24:00 where a second group of jobs are allowed to start due to the change of date. The instances resembling data from Sonofon adopt this structure in the sense that one quarter of the jobs must be finished before 24:00, one quarter of the jobs must start after 24:00 and the rest of the jobs are not restricted by time windows. Furthermore, three quarters of the jobs are small with duration randomly chosen between 1 and 5 minutes, while the rest are large jobs with duration randomly chosen between 5 and 40 minutes. The capacity consumption is random and so are the precedence constraints.

The results for the instances resembling data from Sonofon are presented in Table 10.1. We see that the average time for finding the best solution increases with the number of jobs while it decreases with the number of precedence constraints. The latter observation shows that the precedence constraints actually constrain the problem in a way that makes it easier to solve. The tabu search procedure was able to find a feasible solution in all instances within the time limit and we see that the best solution is very close to the lower bound in almost all instances. In the worse case the best obtained solution only exceeds the lower bound with 9.42 per cent.

		CPU (avg.)		Makespan		Deviation (%)		
$\|\mathcal{J}\|$	$\|\mathcal{I}\|$	Initial	Best	min	max	min	avg.	max
150	75	0.00	107.48	514	547	0.00	0.30	0.96
150	150	0.00	43.76	521	568	0.00	0.19	0.75
150	300	0.00	3.80	568	668	0.00	0.10	0.32
300	150	0.00	1296.62	520	542	0.00	1.47	4.13
300	300	3.84	574.26	538	617	0.00	0.76	4.00
300	600	8.16	75.94	610	705	0.00	0.13	0.33
400	200	1.80	1607.16	536	563	1.90	5.21	8.35
400	400	135.35	1357.75	549	630	0.32	2.01	5.82
400	800*	212.85	553.53	609	690	0.00	0.26	1.00

*In only 8 out of the 10 instances a feasible solution was found.

Table 10.2: Computational results for random instances.

For the instances with random data structure, precedence constraints are random and all jobs have a random time window, a random capacity consumption and a random duration. The results for these instances are reported in Table 10.2, and again we see that the computation time to obtain the best solution increases with the number of jobs and decreases with the number of precedence constraints. On the other hand, this table shows that additional precedence constraints make it harder to find a feasible solution and, in the case with 400 jobs and 800 precedence constraints, the tabu search procedure was unable to find a feasible solution for 2 of the 10 instances within the time limit.

References

[1] E.H.L. Aarts and J.K. Lenstra, editors. *Local Search in Combinatorial Optimization*. John Wiley & Sons, Chichester, 1997.

[2] R.K. Ahuja, T.L. Magnanti, and J.B. Orlin. *Network Flows: Theory, Algorithms, and Applications*. Prentice Hall, New Jersey, 1993.

[3] E. Alba, El-G. Talbi, and A.J. Nebro, editors. *Special Issue on Advances in Metaheuristics for Multiobjective Optimization*. 2006. Special issue of Journal of Heuristics.

[4] Y.P. Aneja and K.P.K. Nair. Bicriteria transportation problem. *Management Science*, 25(1):73–78, 1979.

[5] P. Baptiste and C. Le Pape. Constraint propagation and decomposition techniques for highly disjunctive and highly cumulative project scheduling problems. *Constraints*, 5(1–2):119–139, 2000.

[6] P. Baptiste, C. Le Pape, and C. Nuijten. *Constraint-Based Scheduling: Applying Constraint Programming to Scheduling Problems*. Kluwer Academic Publishers, Boston, 2001.

[7] V. Barichard and J.-K. Hao. A population and interval constraint propagation algorithm. In C.M. Fonseca, P.J. Fleming, E. Zitzler, K. Deb, and L. Thiele, editors, *Evolutionary Multi-Criterion Optimization. Second International Conference, EMO 2003*, volume 2632 of *Lecture Notes in Computer Science*, pages 88–101, Faro, Portugal, 2003. Springer Verlag.

[8] J.E. Beasley. Linear programming on cray supercomputers. *The Journal of the Operational Research Society*, 41(2):133–139, 1990.

[9] J.E. Beasley. OR-library: Distributing test problems by electronic mail. *The Journal of the Operational Research Society*, 41(2):1069–1072, 1990.

[10] R. Bellman. On a routing problem. *Quarterly of Applied Mathematics*, 16: 87–90, 1958.

[11] F. Bock, H. Kantner, and J. Haynes. An algorithm (the r-th best path algorithm) for finding and ranking paths through a network. Research report, Armour Research Foundation of Illinois Institute of Technology, Chicago, Illinois, 1957.

[12] J.A. Bondy and U.S.R. Murty. *Graph Theory with Applications*. Macmillan, London, 1976.

[13] R.E. Burkard, G. Rote, G. Ruhe, and N. Sieber. Algorithmische Untersuchungen zu bikriteriellen kostenminimalen Flüssen in Netzwerken. *Wissenschaftliche Zeitschrift der Technischen Hochschule Leipzig*, 13(6):333–341, 1989.

[14] R.E. Burkard, H.W. Hamacher, and G. Rote. Sandwich approximation of univariate convex functions with an application to separable convex programming. *Naval Research Logistics Quarterly*, 38(6):911–924, 1991.

[15] R.G. Busacker and P.J. Gowen. A procedure for determining a family of minimal-cost network flow patterns. Technical Report 15, John Hopkins University, Operations Research Office, 1961.

[16] H.I. Calvete and P.M. Mateo. An approach for the network flow problem with multiple objectives. *Computers and Operations Research*, 22(9):971–983, 1995.

[17] H.I. Calvete and P.M. Mateo. A sequential network-based approach for the multiobjective network flow problem with preemptive priorities. In M. Tamiz, editor, *Multi-Objective Programming and Goal Programming – Theory and Applications*, volume 432 of *Lecture Notes in Economics and Mathematical Systems*, pages 74–86. Springer Verlag, Berlin, 1996.

[18] V. Chankong and Y. Haimes. *Multiobjective Decision Making: Theory and Methodology*. Elsevier Science, New York, 1983.

[19] C.R. Chegireddy and H.W. Hamacher. Algorithms for finding K-best perfect matchings. *Discrete Applied Mathematics*, 18(2):155–165, 1987.

[20] C. Chekuri and M. Bender. An efficient approximation algorithm for minimizing makespan on uniformly related machines. *Journal of Algorithms*, 41 (2):212–224, 2001.

[21] S. Chen and R. Saigal. A primal algorithm for solving a capacitated network flow problem with additional linear constraints. *Networks*, 7(1):59–79, 1977.

[22] W.-C. Chiang and R.A. Russell. A reactive tabu search metaheuristic for the vehicle routing problem with time windows. *INFORMS Journal on Computing*, 9(4):417–430, 1997.

[23] J.L. Cohon. *Multiobjective Programming and Planning*. Academic Press, New York, 1978.

[24] J.R. Current and M. Marsh. Multiobjective transportation network design and routing problems: Taxonomy and annotation. *European Journal of Operational Research*, 65:4–19, 1993.

[25] J.R. Current and H. Min. Multiobjective design of transportation networks: taxonomy and annotation. *European Journal of Operational Research*, 26 (2):187–201, 1986.

[26] C.G. da Silva, J.C.N. Clímaco, and J. Figueira. A scatter search method for bi-criteria $\{0, 1\}$-knapsack problems. *European Journal of Operational Research*, 169(2):373–391, 2006.

[27] G.B. Dantzig. Programming in a linear structure. Technical report, Comptroller, United States Air Force, Washington, D.C., 1948.

[28] G.B. Dantzig. Application of the simplex method to a transportation problem. In T.C. Koopmans, editor, *Activity Analysis of Production and Allocation*, pages 359–373. John Wiley & Sons, New York, 1951.

[29] M. Dell'Amico and S. Martello. Linear assignment. In M. Dell'Amico, F. Maffioli, and S. Martello, editors, *Annotated Bibliographies in Combinatorial Optimization*, pages 355–371. Wiley, Chichester, 1997.

[30] M. Dell'Amico and P. Toth. Algorithms and codes for dense assignment problems: the state of the art. *Discrete Appl. Math.*, 100(1):17–48, 2000.

[31] M. Dell'Amico, F. Maffioli, and S. Martello, editors. *Annotated Bibliographies in Combinatorial Optimization*. Wiley, Chichester, 1997.

[32] C. Dhaenens, P. Siarry, and El-G. Talbi, editors. *Special Issue on Cooperative methods for Multiobjective Optimization*. 2006. Special issue of RAIRO Operations Research.

[33] E.W. Dijkstra. A note on two problems in connexion with graphs. *Numerische Mathematik*, 1:269–271, 1959.

[34] E.A. Dinic. Algorithm porazryadnogo sokrashcheniya nevyazok i transportnye zadachi [Russian; the method of scaling and transportation problems]. In A.A. Fridman, editor, *Issledovaniya po Diskretnoi Matematike [Russian; Studies in Discrete Mathematics]*. Science, Moscow, 1973.

[35] M. Dror. *Arc Routing: Theory, Solutions and Applications*. Kluwer Academic Publishers, Boston, Massachusetts, 2000.

[36] J. Edmonds and E.L. Johnson. Matching, Euler tours and the Chinese postman. *Mathematical Programming*, 5(1):88–124, 1973.

[37] J. Edmonds and R.M. Karp. Theoretical improvements in algorithmic efficiency for network flow problems. *Journal of the Association for Computing Machinery*, 19(2):248–264, 1972.

[38] E. Egerváry. Matrixok kombinatorius tulajdonságairól [Hungarian with German summary]. *Matematikai és Fizikai Lapok*, 38:16–28, 1931. [English translation [by H.W. Kuhn]: On combinatorial properties of matrices, Logistics Papers, George Washington University, issue 11 (1955), paper 4, pp. 1–11].

[39] M. Ehrgott. *Multicriteria optimization*, volume 491 of *Lecture Notes in Economics and Mathematical Systems*. Springer-Verlag, Berlin, 2000.

[40] M. Ehrgott. *Multicriteria Optimization*. Springer-Verlag, Berlin, second edition, 2005.

[41] M. Ehrgott. Integer solutions of multicriteria network flow problems. *Investigação Operacional*, 19:229–243, 1999.

[42] M. Ehrgott and X. Gandibleux. *Multiple Criteria Optimization: State of the Art Annotated Bibliographic Surveys*. Kluwer Academic Publishers, Boston, Massachusetts, 2002.

[43] M. Ehrgott and K. Klamroth. Connectedness of efficient solutions in multiple criteria combinatorial optimization. *European Journal of Operational Research*, 97:159–166, 1997.

[44] M. Ehrgott, J. Figueira, and X. Gandibleux, editors. *Multiple Objective Discrete and Combinatorial Optimization*. 2005. Special issue of Annals of Operations Research.

[45] H.A. Eiselt, M. Gendreau, and G. Laporte. Arc routing problems, part I: The Chinese postman problem. *Operations Research*, 43(2):231–242, 1995.

[46] J.A. Ferland, S. Ichoua, A. Lavoie, and E. Gagné. Scheduling using tabu search methods with intensification and diversification. *Computers and Operations Research*, 28(11):1075–1092, 2001.

[47] J. Figueira. On the integer bi-criteria network flow problem: A branch-and-bound approach. Cahier du LAMSADE N° 191, Université Paris-Dauphine, France, 2002.

[48] J. Figueira, H. M'Silti, and P. Tolla. Using mathematical programming heuristics in a multicriteria network flow context. *The Journal of the Operational Research Society*, 49(8):878–885, 1998.

[49] J. Figueira, S. Greco, and M. Ehrgott, editors. *Multiple Criteria Decision Analysis – State of the Art Surveys*. Springer, New York, 2004.

[50] B. Fruhwirth, R.E. Burkard, and G. Rote. Approximation of convex curves with application to the bicriterial minimum cost flow problem. *European Journal of Operational Research*, 42(3):326–338, 1989.

[51] D.R. Fulkerson. An out-of-kilter method for minimal-cost flow problems. *Journal of the Society for Industrial and Applied Mathematics*, 9(1):18–27, 1961.

[52] D.R. Fulkerson. Flow networks and combinatorial operations research. *American Mathematical Monthly*, 73(2):115–138, 1966.

[53] Z. Galil and É. Tardos. An $\mathcal{O}(n^2(m + n \log n) \log n)$ min-cost flow algorithm. In *27th Annual Symposium on Foundations of Computer Science (*27th FOCS, Toronto, Ontario, 1986*)*, IEEE Computer Society Press, pages 1–9. Washington, D.C., 1986.

[54] X. Gandibleux, H. Morita, and N. Katoh. Use of genetic heritage for solving the assignment problem with two objectives. In C.M. Fonseca, P.J. Fleming, E. Zitzler, K. Deb, and L. Thiele, editors, *Evolutionary Multi-Criterion Optimization. Second International Conference, EMO 2003*, volume 2632 of *Lecture Notes in Computer Science*, pages 43–57, Faro, Portugal, 2003. Springer Verlag.

[55] X. Gandibleux, H. Morita, and N. Katoh. A population-based metaheuristic for solving assignment problems with two objectives. To appear in Journal of Mathematical Modelling and Algorithms, Revised 2005.

[56] M.R. Garey and D.S. Johnson. *Computers and Intractability. A Guide to the Theory of NP-Completeness*. W.H. Freeman and Company, New York, 1979.

[57] S. Gass and T. Saaty. The computational algorithm for the parametric objective function. *Naval Research Logistics Quarterly*, 2(1):39–45, 1955.

[58] A.M. Geoffrion. Solving bicriterion mathematical programs. *Operations Research*, 15(1):39–54, 1967.

[59] A.M. Geoffrion. Proper efficiency and the theory of vector maximization. *Journal of Mathematical Analysis and Applications*, 22(3):618–630, 1968.

[60] F. Glover. Tabu search – part I. *ORSA Journal on Computing*, 1(3):190–206, 1989.

[61] F. Glover and M. Laguna. *Tabu Search*. Kluwer Academic Publishers, Norwell, Massachusetts, 1997.

[62] J. Grabowski and M. Wodecki. A very fast tabu search algorithm for the permutation flow shop problem with makespan criterion. *Computers and Operations Research*, 31(11):1891–1909, 2004.

[63] M. Grötschel, L. Lovász, and A. Schrijver. *Geometric Algorithms in Combinatorial Optimization*. Springer, New York, 1988.

[64] M. Guan. Graphic programming using odd and even points. *Chinese Mathematics*, 1:273–277, 1962.

[65] H.W. Hamacher. Algebraic flows in regular matroids. *Discrete Applied Mathematics*, 2(1):27–38, 1980.

[66] H.W. Hamacher. A note on K best network flows. *Annals of Operations Research*, 57:65–72, 1995.

[67] H.W. Hamacher and C. Hüsselman. Ranking approach to max-ordering combinatorial optimization and network flows. In E. Schock, editor, *Beiträge zur Angewandten Analysis und Informatik – Helmut Brakhage zu Ehren*, pages 97–111. Shaker Verlag, Aachen, 1994.

[68] H.W. Hamacher and M. Queyranne. K best solutions to combinatorial optimization problems. *Annals of Operations Research*, 6(4):123–143, 1985.

[69] H.W. Hamacher, C.R. Pedersen, and S. Ruzika. Multiple objective minimum cost flow problems: A review. *European Journal of Operational Research,* in press, 2005.

[70] H.W. Hamacher, C.R. Pedersen, and S. Ruzika. Finding representative systems for discrete bicriterion optimization problems. *Operations Research Letters,* in press, 2006.

[71] M.P. Hansen and A. Jaszkiewicz. Evaluating the quality of approximations to the non-dominated set. IMM Technical Report IMM-REP-1998-7, Institute of Mathematical Modelling, Technical University of Denmark, 1998.

[72] P. Hansen. Bicriterion path problems. In G. Fandel and T. Gal, editors, *Multiple criteria decision making – Theory and Applications*, volume 177 of *Lecture Notes in Economics and Mathematical Systems*, pages 109–127. Springer Verlag, Heidelberg, 1979.

[73] C. Hierholzer. Über die möglichkeit, einen linienzug ohne wiederholung und ohne unterbrechung zu umfahren. *Mathematische Annalen*, 6(1):30–32, 1873.

[74] W.M. Hirsch and G.B. Dantzig. The fixed charge problem. Research Memorandum RM-1383, The Rand Corporation, Santa Monica, California, 1954.

[75] F.L. Hitchcock. The distribution of a product from several sources to numerous localities. *Journal of Mathematical Physics*, 20:224–230, 1941.

[76] W. Hoffman and R. Pavley. A method for the solution of the N'th best path problem. *Journal of the Association for Computing Machinery*, 6:506 – 514, 1959.

[77] J.N. Hooker. Logic-based benders methods for planning and scheduling, 2003. Lecture given at *ISMP 2003*, Lyngby, Denmark.

[78] J.N. Hooker and G. Ottosson. Logic-based Benders decomposition. *Mathematical Programming*, 96(1, ser. A):33–60, 2003.

[79] F. Huarng, P.S. Pulat, and A. Ravindran. An algorithm for bicriteria integer network flow problem. In *Proceedings of the 10th International Conference on Multiple Criteria Decision Making, Taipei, Taiwan*, volume 3, pages 305–318, 1992.

[80] ILOG. OPL Studio 3.7, Language Manual, 2003. `http://www.ilog.com`.

[81] ILOG. CPLEX 9.1, User's Manual and Reference Manuals, 2004. `http://www.ilog.com`.

[82] H. Isermann. Proper efficiency and the linear vector maximum problem. *Operations Research*, 22:189–191, 1974.

[83] H. Isermann. The enumeration of the set of all efficient solutions for a linear multiple-objective program. *Naval Research Logistics Quaterly*, 26(1):123–139, 1979.

[84] V. Jain and I.E. Grossmann. Algorithms for hybrid MILP/CP models for a class of optimization problems. *INFORMS Journal on Computing*, 13(4): 258–276, 2001.

[85] W.S. Jewell. Optimal flows through networks. Interim Technical Report No. 8, Massachusetts Institute of Technology, 1958.

[86] R. Jonker and A. Volgenant. A shortest augmenting path algorithm for dense and sparse linear assignment problems. *Computing*, 38(4):325–340, 1987.

[87] L.V. Kantorovich. Mathematical methods in the organization and planning of production. Technical report, Publication House of the Leningrad University, 1939. [Translated in *Management Science* 6 (1960), pp. 366–422].

[88] K. Klamroth, J. Tind, and M.M. Wiecek. Unbiased approximation in multicriteria optimization. *Mathematical Methods of Operations Research*, 56(3): 413–437, 2002.

[89] D. Klingman, A. Napier, and J. Stutz. NETGEN: A program for generating large scale capacitated assignment, transportation, and minimum cost flow network problems. *Management Science*, 20(5):814–821, 1974.

[90] T.C. Koopmans. Optimum utilization of the transportation system. In D.H. Leavens, editor, *The Econometric Society Meeting*, Proceedings of the International Statistical Conferences – Volume V, pages 136–146. Washington, D.C., 1947. [reprinted in: *Econometrica* 17, 1949].

[91] P. Kouvelis and S. Sayin. Algorithm Robust for the bicriteria discrete optimization problem. To appear in Annals of Operations Research, Revised 2004.

[92] H.W. Kuhn. The Hungarian method for the assignment problem. *Naval Research Logistics Quarterly*, 2:83–97, 1955.

[93] H.W. Kuhn. Variants of the Hungarian method for the assignment problem. *Naval Research Logistics Quarterly*, 3:253–258, 1956.

[94] M. Laumanns, L. Thiele, and E. Zitzler. An efficient, adaptive parameter variation scheme for metaheuristics based on the epsilon-constraint method. *European Journal of Operational Research*, 169(3):932–924, 2006.

[95] E.L. Lawler. A procedure for computing the K best solutions to discrete optimization problems and its application to the shortest path. *Management Science*, 18(7):401–405, 1972.

[96] E.L. Lawler. *Combinatorial Optimization: Networks and Matroids.* Holt, Rinehart and Winston, New York, 1976.

[97] H. Lee. *Integer solutions of bicriteria network flow problems.* PhD thesis, University of Oklahoma, Norman, OK, 1991.

[98] H. Lee and P.S. Pulat. Bicriteria network flow problems: Continuous case. *European Journal of Operational Research*, 51(1):119–126, 1991.

[99] H. Lee and P.S. Pulat. Bicriteria network flow problems: Integer case. *European Journal of Operational Research*, 66(1):148–157, 1993.

[100] J.Y-T. Leung, editor. *Handbook of Scheduling: Algorithms, Models, and Performance Analysis.* Chapman & Hall/CRC Press, Boca Raton, Florida, 2004.

[101] R. Malhotra and M.C. Puri. Bicriteria network problem. *Cahiers du Centre d'Études Recherche Opérationelle*, 26(1/2):95–102, 1984.

[102] R. Mansini, M.G. Speranza, and Z. Tuza. Scheduling groups of tasks with precedence constraints on three dedicated processors. *Discrete Applied Mathematics*, 134(1–3):141–168, 2004.

[103] P. McKeown. Solving incremental quantity discounted transportation problems by vertex ranking. *Naval Research Logistics Quarterly*, 27(3):437–445, 1980.

[104] G.J. Minty. Monotone networks. *Proceedings of the Royal Society. London. Series A.*, 257:194–212, 1960.

[105] J. Mote, I. Murthy, and D.L. Olson. A parametric approach to solving bicriterion shortest path problems. *European Journal of Operational Research*, 53:81–92, 1991.

[106] K.G. Murty. An algorithm for ranking all the assignments in order of increasing cost. *Operations Research*, 16(3):682–687, 1968.

[107] K.G. Murty. Solving the fixed charge problem by ranking the extreme points. *Operations Research*, 16(2):286–279, 1968.

[108] A. Mustafa and M. Goh. Finding integer efficient solutions for bicriteria and tricriteria network flow problems using DINAS. *Computers and Operations Research*, 25(2):139–157, 1998.

[109] P. Naccache. Connectedness of the set of nondominated outcomes in multicriteria optimization. *Journal of Optimization Theory and Applications*, 25 (3):459–467, 1978.

[110] G.L. Nemhauser and L.A. Wolsey. *Integer and Combinatorial Optimization*. Wiley & Sons, New York, 1999.

[111] Netlib. The performance database server, 2006. `http://netlib.org/`.

[112] L.R. Nielsen. *Route Choice in Stochastic Time-Dependent Networks*. PhD thesis, Department of Operations Research, University of Aarhus, 2004.

[113] L.R. Nielsen and C.R. Pedersen. APGen – an assignment problem generator. Reference manual `http://www.research.relund.dk`, 2006.

[114] L.R. Nielsen, K.A. Andersen, and D. Pretolani. Bicriterion shortest hyperpaths in random time-dependent networks. *IMA Journal of Management Mathematics*, 14(3):271–303, 2003.

[115] M. Nikolova. An approach for finding Pareto optimal solutions of the multicriteria network flow problem. *Cybernetics and Information Technologies*, 1(2):56–62, 2001.

[116] M. Nikolova. A classification based approach for finding Pareto optimal solutions of the multicriteria network flow. *Cybernetics and Information Technologies*, 3(1):11–17, 2003.

[117] M. Nikolova. Properties of the effective solutions of the multicriteria network flow problem. *Problems of Engineering Cybernetics and Robotics*, 47:104–111, 1998.

[118] W. Ogryczak, K. Studziński, and K. Zorychta. DINAS: a computer-assisted analysis system for multiobjective transshipment problems with facility location. *Computers and Operations Research*, 19(7):637–647, 1992.

[119] J.B. Orlin. A faster strongly polynomial minimum cost flow algorithm. *Operations Research*, 41(2):338–350, 1993.

[120] C.H. Papadimitriou. On the complexity of edge traversing. *Journal of the Association for Computing Machinery*, 23(3):544–554, 1976.

[121] C.H. Papadimitriou and K. Steiglitz. *Combinatorial Optimization: Algorithms and Complexity*. Prentice Hall, Englewood Cliffs, N.J., 1982.

[122] M.M.B. Pascoal, 2006. Personal communication.

[123] M.M.B. Pascoal, M.E. Captivo, and J.C.N. Clímaco. A note on a new variant of Murty's ranking assignments algorithm. *4OR: Quarterly Journal of the Belgian, French and Italian Operations Research Societies*, 1(3):243–255, 2003.

[124] A.N. Payne. Efficient approximate representation of bi-objective tradeoff sets. *Journal of the Franklin Institute*, 330(6):1219–1233, 1993.

[125] A.N. Payne and E. Polak. An interactive rectangle elimination method for biobjective decision making. *IEEE Transactions on Automatic Control*, 25 (3):421–432, 1980.

[126] C.R. Pedersen, L.R. Nielsen, and K.A. Andersen. On the bicriterion multi modal assignment problem. Working Paper No. 2005/3, Department of Operations Research, University of Aarhus, 2005. Submitted.

[127] C.R. Pedersen, L.R. Nielsen, and K.A. Andersen. A note on ranking assignments using reoptimization. Working Paper No. 2005/2, Department of Operations Research, University of Aarhus, 2005. Will be submitted in revised form.

[128] C.R. Pedersen, R.V. Rasmussen, and K.A. Andersen. Solving a large-scale precedence constrained scheduling problem with elastic jobs using tabu search. *Computers and Operations Research,* in press, 2005.

[129] S.C.S. Porto, J.P.F.W. Kitajima, and C.C. Ribeiro. Performance evaluation of a parallel tabu search task scheduling problem. *Parallel Computing*, 26 (1):73–90, 2000.

[130] A. Przybylski, 2006. Personal communication.

[131] A. Przybylski, X. Gandibleux, and M. Ehrgott. The biobjective assignment problem. Research Report N° 05.07, lina – Laboratoire d'Informatique de Nantes Atlantique, 2005. Submitted.

[132] A. Przybylski, X. Gandibleux, and M. Ehrgott. The biobjective integer minimum cost flow problem – incorrectness of Sedeño-Noda and González-Martín's algorithm. *Computers and Operations Research*, 33(5):1459–1463, 2006.

[133] P.S. Pulat, F. Huarng, and H. Lee. Efficient solutions for the bicriteria network flow problem. *Computers and Operations Research*, 19(7):649–655, 1992.

[134] G. Ruhe. Complexity results for multicriterial and parametric network flows using a pathological graph of Zadeh. *Zeitschrift für Operations Research*, 32 (1):9–27, 1988.

[135] G. Ruhe. *Flüsse in Netzwerken – Komplexität und Algorithmen.* PhD thesis, Technische Hochschule Leipzig, Sektion Mathematik und Informatik, 1988.

[136] G. Ruhe. *Algorithmic Aspects of Flows in Networks.* Kluwer Academic Publishers, Dordrecht, 1991.

[137] G. Ruhe and B. Fruhwirth. ε-optimality for bicriteria programs and its application to minimum cost flows. *Computing*, 44(1):21–34, 1990.

[138] S. Ruzika. Finding representations of the nondominated set for discrete bicriteria optimization problems by box algorithms, 2006. Lecture given at *MOPGP 2006*, Tours, France.

[139] S. Ruzika and M.M. Wiecek. Approximation methods in multiobjective programming. *Journal of Optimization Theory and Applications*, 126(3): 473–501, 2005.

[140] S. Sadagopan and A. Ravindran. A vertex ranking algorithm for the fixed-charge transportation problem. *Journal of Optimization Theory and Applications*, 37(2):221–230, 1982.

[141] Y. Saruwatari and T. Matsui. A note on K-best solutions to the Chinese postman problem. *SIAM Journal on Optimization*, 3(4):726–733, 1993.

[142] S. Sayin. Measuring the quality of discrete representations of efficient sets in multiple objective mathematical programming. *Mathematical Programming*, 87:543–560, 2000.

[143] S. Sayin and P. Kouvelis. The multiobjective discrete optimization problem: A weighted min-max two-stage optimization approach and a bicriteria algorithm. *Management Science*, 51(10):1572–1581, 2005.

[144] A. Schrijver. *Combinatorial Optimization: Polyhedra and Efficiency*, volume 24 of *Algorithms and Combinatorics*. Springer-Verlag, Berlin, 2003.

[145] A. Sedeño-Noda and C. González-Martín. The biobjective minimum cost flow problem. *European Journal of Operational Research*, 124(3):591–600, 2000.

[146] A. Sedeño-Noda and C. González-Martín. An algorithm for the biobjective integer minimum cost flow problem. *Computers and Operations Research*, 28(2):139–156, 2001.

[147] A. Sedeño-Noda and C. González-Martín. An alternative method to solve the biobjective minimum cost flow problem. *Asia-Pacific Journal of Operational Research*, 20(2):241–260, 2003.

[148] P. Serafini. Some considerations about computational complexity for multiobjective combinatorial problems. In J. Jahn and W. Krabs, editors, *Recent Advances and Historical Development of Vector Optimization*, volume 294 of *Lecture Notes in Economics and Mathematical Systems*, pages 222–232. Springer, Berlin, 1986.

[149] A.J.V. Skriver. A classification of bicriterion shortest path (BSP) algorithms. *Asia-Pacific Journal of Operational Research*, 17(2):199–212, 2000.

[150] A.J.V. Skriver and K.A. Andersen. The bicriterion semi-obnoxious location (BSL) problem solved by an ε-approximation. *European Journal of Operational Research*, 146(3):517–528, 2003.

[151] A.J.V. Skriver and K.A. Andersen. A label correcting approach for solving bicriterion shortest-path problems. *Computers and Operations Research*, 27 (6):507–524, 2000.

[152] R.E. Steuer. *Multiple Criteria Optimization: Theory, Computation, and Application*. Wiley Series in Probability and Mathematical Statistics: Applied Probability and Statistics. John Wiley & Sons, New York, 1986.

[153] R.E. Steuer and E.U. Choo. An interactive weighted Tchebycheff procedure for multiple objective programming. *Mathematical Programming*, 26(3):326–344, 1983.

[154] R.E. Steuer and F.W. Harris. Intra-set point generation and filtering in decision and criterion space. *Computers and Operations Research*, 7(1):41–53, 1980.

[155] M. Sun. Procedures for finding nondominated solutions for multiple objective network programming problems. *Transportation Science*, 37(2):139–152, 2003.

[156] É. Tardos. A strongly polynomial minimum cost circulation algorithm. *Combinatorica*, 5(3):247–255, 1985.

[157] R.E. Tarjan. *Data Structures and Network Algorithms*, volume 44 of *CBMS-NSF Regional Conference Series in Applied Mathematics.* SIAM, Philadelphia, PA, 1983.

[158] N. Tomizawa. On some techniques useful for solution of transportation network problems. *Networks*, 1:173–194, 1971.

[159] D. Tuyttens, J. Teghem, Ph. Fortemps, and K. Van Nieuwenhuyze. Performance of the MOSA method for the bicriteria assignment problem. *Journal of Heuristics*, 6(3):295–310, 2000.

[160] E.L. Ulungu and J. Teghem. Multi-objective combinatorial optimization problems: A survey. *Journal of Multi-Criteria Decision Analysis*, 3:83–104, 1994.

[161] E.L. Ulungu and J. Teghem. The two phases method: An efficient procedure to solve bi-objective combinatorial optimization problems. *Foundations of Computing and Decision Sciences*, 20(2):149–165, 1995.

[162] T. van Aardenne-Ehrenfest and N.G. de Bruijn. Circuits and trees in oriented linear graphs. *Simon Stevin*, 28:203–217, 1951.

[163] M. Visée, J. Teghem, M. Pirlot, and E.L. Ulungu. Two-phases method and branch and bound procedures to solve the bi-objective knapsack problem. *Journal of Global Optimization*, 12(2):139–155, 1998.

[164] A. Warburton. Quasiconcave vector maximization: Connectedness of the sets of Pareto-optimal and weak Pareto-optimal alternatives. *Journal of Optimization Theory and Applications*, 40(4):537–557, 1983.

[165] A. Warburton. Approximation of Pareto optima in multiple-objective, shortest-path problems. *Operations Research*, 35(1):70–79, 1987.

[166] D.J. White. A bibliography on the applications of mathematical programming multiple-objective methods. *The Journal of the Operational Research Society*, 41(8):669–691, 1990.

[167] X.Q. Yang and C.J. Goh. A method for convex curve approximation. *European Journal of Operational Research*, 97(1):205–212, 1997.

[168] J.Y. Yen. Finding the K shortest loopless paths in a network. *Management Science*, 17(11):712–716, 1971.

[169] N. Zadeh. A bad network for the simplex method and other minimum cost flow algorithms. *Mathematical Programming*, 5:255–266, 1973.

List of figures

List of tables

List of notation

This is a list of the notation used in this thesis. In cases of ambiguity, the proper use should be clear from the context in which it appears.

Symbols

Δ accuracy measure, 115

Θ list of box choices, 140

Π sequence of jobs, 140

Φ candidate set in ranking procedure, 37

$\delta_{1,k}$ relative increase in cost, 42

λ weighting vector in parametric minimization problem (search/ranking direction in two-phase method), 27

τ position of a job, 140

$<$ binary relation between two vectors, 22

$\leq$ binary relation between two vectors, 22

$\leqq$ binary relation between two vectors, 22

$<^{\text{lex}}$ lexicographical less than, 52

$\ominus$ subtraction of flows, 11

$\oplus$ addition of flows (also used for direct sum), 11

$\triangle(y^+, y^-)$ second phase triangle, 29

A

AP the assignment problem, 13

A arc set of a graph, 8

A_b set of backward arcs in residual network, 11

A_f set of forward arcs in residual network, 11

$AP(\mathcal{X})$ reduced AP for subset $\mathcal{X} \subset {}_a\mathcal{X}_{AP}$, 38

a an assignment, 35

$a(R(y^1, y^2))$ area of rectangle $R(y^1, y^2)$, 115

B

BiAP the bicriterion assignment problem, 65

BiCPP the bicriterion Chinese postman problem, 126

BiDCPP the bicriterion directed Chinese postman problem, 126

BiDOP the bicriterion discrete optimization problem, 111

BiMCF the bicriterion minimum cost flow problem, 90

BiMCIF the bicriterion minimum cost integer flow problem, 97

BiMCPP the bicriterion mixed Chinese postman problem, 126

BiMMAP the bicriterion multi modal assignment problem, 66

BiMMTP the bicriterion multi modal transportation problem, 126

BiUCPP the bicriterion undirected Chinese postman problem, 126

$\mathcal{B}$ boxes representing jobs, 138

b demand vector, 9

b_i demand/supply at node i, 9

C

CPP the Chinese postman problem, 15

C cost/objective matrix, 22

$Cand_1$ candidate list 1, 147

$Cand_2$ candidate list 2, 147

c cost vector, 7

$c(\cdot)$ cost of $\cdot$, 11

$c^\pi(\cdot)$ reduced cost of $\cdot$, 11

c^π_{ij} reduced cost for arc (i,j), 11

$c^{\hat{\pi}}_{\cdot j}$ minimum reduced cost in column j, 74

$c^{\hat{\pi}}_{i\cdot}$ minimum reduced cost in row i, 74

c^1_{ijl} assignment cost, 66

c^2_{ijl} assignment time, 66

c_{ij} unit flow cost of arc (i,j), 9

cap_j least capacity consumption of job j, 137

D

DCPP the directed Chinese postman problem, 15

$d^+(i)$ outdegree of node i, 8

$d^-(i)$ indegree of node i, 8

d_j demand of node j, 15

E

EOD End-of-Day jobs, 136

$\mathcal{E}$ edge set of the adjacency graph, 24

E edge set in a graph, 9

$error_{Rep}$ representation error, 122

F

$f(\lambda)$ lower envelope of lines in the parametric space, 27

f_λ parametric objective function, 27

G

$\mathcal{G}$ the adjacency graph, 24

G a graph, 8

$G(x)$ Residual network/incremental graph given flow x, 10

$G^{\mathcal{I}}$ precedence graph, 141

$G^\pi(x)$ reduced cost residual network given flow x, 11

g univariate convex function, 94

H

$\mathcal{H}_m$ job-machine constraints, 138

I

$\mathcal{I}$ precedence constraints, 138

$\widetilde{I}^{(p,q)}$ index set for variables strictly larger than their lower bound and lexicographically smaller than (p,q), 57

$I^{(p,q)}$ index set for variables lexicographically smaller than (p,q), 52

$I^{\text{fix}(\mathcal{X})}$ set of fixed variables in subset $\mathcal{X} \subset \mathcal{X}_{flow}$, 54

$IntenArray$ list of moves that led to improvements in tabu search procedure, 148

(i,j) arc/edge in a graph, 8

J

$\mathcal{J}$ jobs in scheduling problem, 138

$j.cap$ capacity consumption of job j, 139

$j.end$ completion time of job j, 139

$j.start$ starting time of job j, 139

$j.time$ duration of job j, 139

K

K number of solutions to rank, 17

k number of currently identified solutions in ranking procedure, 17

L

$\mathcal{L}$ lower approximation of $\mathcal{Y}_N$, 96

L_{ij} number of mode choices for assignment (i,j), 65

LB lower bound in two-phase procedure, 29

l mode choice in mathematical formulation of BiMMAP, 66

$l(z)$ piecewise linear lower approximating function of convex univariate function, 94

l_{ij} lower bound of arc flow, 9

$l^{\star}_{ij}(\lambda)$ optimal mode choice in a given cell for a specific λ, 68

$[l_j, u_j]$ time window for job j, 139

M

MCF the minimum cost flow problem, 10

MCIF the minimum cost integer flow problem, 10

MCPP the mixed Chinese postman problem, 15

MMCF the multicriteria minimum cost flow problem, 90

MMCIF the multicriteria minimum cost integer flow problem, 90

MOCO the multicriteria combinatorial optimization problem, 22

MOLIP the multicriteria linear integer problem, 22

MOLP the multicriteria linear problem, 22

$\mathcal{M}$ machines, 138

$MCIF(\mathcal{X})$ reduced MCIF for subset $\mathcal{X} \subset \mathcal{X}_{flow}$, 52

m number of arcs in a graph, 8

$maxCost$ maximal cost value in BiMMAP, 76

$maxEnt$ maximal number of

entries in an assignment cell, 76

method parameter for BiMMAP instances, 76

minEnt minimal number of entries in an assignment cell, 76

N

$\mathcal{N}$ node set of the adjacency graph, 24

N node set of a graph, 8

n number of nodes in a graph, 8

n_1 number of nodes in W, 15

n_2 number of nodes in V, 15

O

O cycle, 11

P

$\mathcal{P}_{AP}$ the assignment polyhedron, 14

$\mathcal{P}_{DCPP}$ the directed Chinese postman polyhedron, 16

$\mathcal{P}_{flow}$ the flow polyhedron, 10

$\mathcal{P}_{MMAP}$ the multi modal assignment polyhedron, 67

$\mathcal{P}_{path}$ the path polyhedron, 13

$\mathcal{P}_{TP}$ the transportation polyhedron, 15

P path, 11

π set of node potentials, 11

$\pi(i)$ node potential for node i, 11

Q

Q number of objectives, 22

R

$\mathbb{R}^Q_{\geqq}$ the Pareto Cone, 23

$R(y^1, y^2)$ rectangle in representation algorithm, 115

R_m capacity of machine m, 139

Rep representative system, 111

r_{ij} residual capacity of arc (i, j), 11

S

SP the shortest path problem, 13

$\mathcal{S}$ decision space/general feasible set, 22

$\mathcal{S}_E$ the efficient solutions, 23

$\mathcal{S}_{wE}$ the weakly efficient solutions, 23

s source node, 13

s_i supply of node i, 14

shape parameter for BiMMAP instances, 76

steps number of moves without a change in *timeTabu*, 148

T

TMCIF the tricriterion minimum cost integer flow problem, 105

TP the transportation problem, 14

$Tabu$ the tabu list, 147

$TP(\mathcal{X})$ reduced TP for subset $\mathcal{X} \subset \mathcal{X}_{TP}$, 57

t sink node, 13

$time_j$ longest duration of job j, 137

timeTabu number of iterations a move is tabu, 147

U

UCPP the undirected Chinese postman problem, 15

$\mathcal{U}$ upper approximation of $\mathcal{Y}_N$, 96

UB upper bound in two-phase procedure, 29

UB^{IP} integer based upper bound in two-phase procedure, 69

$UB^{IP}(\varepsilon)$ integer based upper bound in approximation two-phase procedure, 71

$u(z)$ piecewise linear upper approximating function of convex univariate function, 94

u_{ij} upper bound of arc flow, 9

V

V node set in a graph, 9

$v(x)$ flow value of x, 103

W

W node set in a graph, 9

X

$\mathcal{X}$ feasible set defined by an integer lattice, 7

$\mathcal{X}_{AP}$ the assignment integer lattice, 14

${}_a\mathcal{X}_{AP}$ set of feasible assignment using network formulation, 37

$\mathcal{X}_E$ the efficient solutions in a discrete feasible set $\mathcal{X}$, 23

$\mathcal{X}_{flow}$ the flow integer lattice, 10

$\mathcal{X}_{MMAP}$ the multi modal integer lattice, 66

${}_a\mathcal{X}_{MMAP}$ set of feasible multi modal assignment using network formulation, 72

$\mathcal{X}_{TP}$ the transportation integer lattice, 15

$\mathcal{X}_{wE}$ the weakly efficient solutions in a discrete feasible set $\mathcal{X}$, 23

$\mathcal{X}^i$ subset of assignments, 37

$\mathcal{X}^{pq}$ subset of integer flows/transportation solutions, 52

${}_d\mathcal{X}^{pq}$ downward branch of integer flows/transportation solutions, 52

${}_u\mathcal{X}^{pq}$ upward branch of integer flows/transportation solutions, 52

x decision vector, 7

x^{LR} efficient solution to lower right nondominated point, 28

x^{UL} efficient solution to upper left nondominated point, 28

x_{ijl} binary decision variable in BiMMAP, 66

x_{ij} flow on arc (i, j), 9

x_{jb} binary decision variable in scheduling problem, 139

Y

$\mathcal{Y}$ criterion space, 22

$\mathcal{Y}^{\geq}$ auxiliary set yielded by the nondominated criterion points, 23

$\mathcal{Y}_N$ the nondominated (criterion) points, 23

$\mathcal{Y}_{sN}$ the supported nondominated (criterion) points, 82

$\mathcal{Y}_{usN}$ the unsupported nondominated (criterion) points, 82

$\mathcal{Y}_{wN}$ the weakly nondominated (criterion) points, 23

$y(x)$ criterion value of decision vector x, 7

y^+ extreme nondominated point – upper left in triangle, 28

y^- extreme nondominated point – lower right in triangle, 28

y^{LR} lower right nondominated point, 28

y^{UL} upper left nondominated point, 28

$y_j(x)$ the jth objective value for the decision vector x, 21

Author index

The numbers in the square brackets refer to entries in the reference list.

Subject index