Studies in Fuzziness and Soft Computing, Volume 228

Further volumes of this series can be found on our homepage: springer.com

Vol. 212. Elisabeth Rakus-Andersson
Fuzzy and Rough Techniques in Medical Diagnosis and Medication, 2007
ISBN 978-3-540-49707-3

Vol. 213. Peter Lucas, José A. Gàmez, Antonio Salmerón (Eds.)
Advances in Probabilistic Graphical Models, 2007
ISBN 978-3-540-68994-2

Vol. 214. Irina Georgescu
Fuzzy Choice Functions, 2007
ISBN 978-3-540-68997-3

Vol. 215. Paul P. Wang, Da Ruan, Etienne E. Kerre (Eds.)
Fuzzy Logic, 2007
ISBN 978-3-540-71257-2

Vol. 216. Rudolf Seising
The Fuzzification of Systems, 2007
ISBN 978-3-540-71794-2

Vol. 217. Masoud Nikravesh, Janusz Kacprzyk, Lofti A. Zadeh (Eds.)
Forging New Frontiers: Fuzzy Pioneers I, 2007
ISBN 978-3-540-73181-8

Vol. 218. Masoud Nikravesh, Janusz Kacprzyk, Lofti A. Zadeh (Eds.)
Forging New Frontiers: Fuzzy Pioneers II, 2007
ISBN 978-3-540-73184-9

Vol. 219. Roland R. Yager, Liping Liu (Eds.)
Classic Works of the Dempster-Shafer Theory of Belief Functions, 2007
ISBN 978-3-540-25381-5

Vol. 220. Humberto Bustince, Francisco Herrera, Javier Montero (Eds.)
Fuzzy Sets and Their Extensions: Representation, Aggregation and Models, 2007
ISBN 978-3-540-73722-3

Vol. 221. Gleb Beliakov, Tomasa Calvo, Ana Pradera
Aggregation Functions: A Guide for Practitioners, 2007
ISBN 978-3-540-73720-9

Vol. 222. James J. Buckley, Leonard J. Jowers
Monte Carlo Methods in Fuzzy Optimization, 2008
ISBN 978-3-540-76289-8

Vol. 223. Oscar Castillo, Patricia Melin
Type-2 Fuzzy Logic: Theory and Applications, 2008
ISBN 978-3-540-76283-6

Vol. 224. Rafael Bello, Rafael Falcón, Witold Pedrycz, Janusz Kacprzyk (Eds.)
Contributions to Fuzzy and Rough Sets Theories and Their Applications, 2008
ISBN 978-3-540-76972-9

Vol. 225. Terry D. Clark, Jennifer M. Larson, John N. Mordeson, Joshua D. Potter, Mark J. Wierman
Applying Fuzzy Mathematics to Formal Models in Comparative Politics, 2008
ISBN 978-3-540-77460-0

Vol. 226. Bhanu Prasad (Ed.)
Soft Computing Applications in Industry, 2008
ISBN 978-3-540-77464-8

Vol. 227. Eugene Roventa, Tiberiu Spircu
Management of Knowledge Imperfection in Building Intelligent Systems, 2008
ISBN 978-3-540-77462-4

Vol. 228. Adam Kasperski
Discrete Optimization with Interval Data, 2008
ISBN 978-3-540-78483-8

Adam Kasperski

Discrete Optimization with Interval Data

Minmax Regret and Fuzzy Approach

Springer

Author

Dr. Adam Kasperski
Institute of Industrial Engineering and Management
Wroclaw University of Technology
Wybrzeze Wyspianskiego 27
Wroclaw, 50-370
Poland
Email: adam.kasperski@pwr.wroc.pl

ISBN 978-3-540-78483-8 e-ISBN 978-3-540-78484-5

DOI 10.1007/978-3-540-78484-5

Studies in Fuzziness and Soft Computing ISSN 1434-9922

Library of Congress Control Number: 2008922195

Typeset & Cover Design: Scientific Publishing Services Pvt. Ltd., Chennai, India.

Printed in acid-free paper

9 8 7 6 5 4 3 2 1

springer.com

Foreword

Operations research often solves deterministic optimization problems based on elegant and concise representations where all parameters are precisely known. In the face of uncertainty, probability theory is the traditional tool to be appealed for, and stochastic optimization is actually a significant sub-area in operations research. However, the systematic use of prescribed probability distributions so as to cope with imperfect data is partially unsatisfactory.

First, going from a deterministic to a stochastic formulation, a problem may become intractable. A good example is when going from deterministic to stochastic scheduling problems like PERT. From the inception of the PERT method in the 1950's, it was acknowledged that data concerning activity duration times is generally not perfectly known and the study of stochastic PERT was launched quite early. Even if the power of today's computers enables the stochastic PERT to be addressed to a large extent, still its solutions often require simplifying assumptions of some kind.

Another difficulty is that stochastic optimization problems produce solutions in the average. For instance, the criterion to be maximized is more often than not expected utility. This is not always a meaningful strategy. In the case when the underlying process is not repeated a lot of times, let alone being one-shot, it is not clear if this criterion is realistic, in particular if probability distributions are subjective. Expected utility was proposed as a rational criterion from first principles by Savage. In his view, the subjective probability distribution was basically an artefact useful to implement a certain ordering of solutions. However, subsequent research has shown that, in some situations, people having incomplete information on the problem at hand, or no information at all, consistently avoid making decisions by means of this criterion. There is a wealth of research papers in decision theory, in the last twenty years, that depart from the expected utility tradition.

And indeed, the use of single probability distributions in order to model imperfect data leads to another, more basic difficulty, namely it creates a confusion between values that are non-deterministic due to known variability, and values that are just unknown or ill-known while being possibly constant. The

use of probability distributions is perfectly legitimate if sufficient statistics are available. In the face of mere ignorance, a probability distribution is either unreachable (when variability is present) or misleading. Subjective probability is a very rich concept that, due to its operational semantics, deliberately gives up the distinction between variability and ignorance.

In the recent past, a simpler idea has emerged, namely that partial ignorance on the value of a numerical parameter can be modelled by means of an interval. Intervals were used for a long time in physics for modelling measurement errors. But it never was the highlight in this area. Interval mathematics were further developed in the late sixties under the impulse given by Ramon Moore, motivated by the propagation of rounding errors made by digital computers. Since then, it has becomes a full-fledged approach to constraint-based problem solving and to the handling of uncertainty based on a best and worst case analysis. The advantage of this modeling is that it is not so much data-demanding as the probabilistic approach. The draw-back is that it is a cruder representation of uncertainty.

On this basis, there is no surprise that the interval-based approach became more popular in operations research in the 1990's (even if linear programming with interval coefficients was known for some time with the works of Ralph Steuer). Prior to devising algorithms adapted to this framework, a basic question to be addressed by optimization specialists is : what should be optimized if the criterion takes on interval values? Again decision theory has ready-made answers. One of the oldest such proposal is the maximin criterion: choose the solution that optimizes the criterion in the worst situation compatible the available incomplete information, i.e. maximize utility in the worst case. This criterion is very well known as being overpessimistic, as the worst situation may actually be very unlikely.

Interestingly, the same person who advocated expected utility, namely Leonard Savage, introduced the regret criterion, consisting of minimizing a loss function computed for each decision. The maximal regret associated with a decision is computed as the maximal difference between the figure of merit of the best decision that could have been taken had the underlying circumstances been known and the figure of merit of the considered decision in these circumstances. It turns out this is the most instrumental approach to state optimization problem under incomplete knowledge of data, so as to ensure a form of robustness to the selected solution.

This book considers solving combinatorial problems with incomplete data modelled by intervals using the regret criterion. The material is heavily based on the author's findings in his team in Wrocław, and focuses on some classical OR problems, such as knapsack, spanning tree, shortest paths, assignment, and minimal cut, and various sequencing problems in production research. Each OR researcher can then learn the regret minimization approach on her or his favourite problem. In this sense, the book makes an important contribution to the dissemination of more realistic approaches to combinatorial optimization, dropping the assumption of perfect knowledge of the data, without presupposing statistical information is

available. The price paid is that, in general, the regret criterion leads to an increase in computational complexity, even if this increase is not so dramatic as in some stochastic approaches. From this point of view this book offers a detailed analysis to the complexity issues in each problem considered and proposes algorithms that cope with this difficulty.

Finally a significant merit of this book is to devote a chapter to the extension of the framework to the case where information is a little richer than what mere intervals allow to represents, namely when fuzzy intervals are allowed to model ill-known data. This step is very important for several reasons: first it introduces a formalism and a level of sophistication comparable with the probabilistic representations. It is the framework of possibility theory. Next, it is a direct extension, since a fuzzy interval is a collection of nested intervals. Another point is that the pessimism maximin criterion is tamed and becomes more sensible.

More often than not in the past, fuzzy versions of optimization problems were regarded with some doubt, because the obtained solutions were hard to interpret. This book highlights the fact that the fuzzy counterparts to operations research problems should be viewed as extensions of interval-valued formulations, and not of deterministic formulations. In other words if you are tempted to make your favorite method fuzzy because of imperfect information, try to state and solve the interval-valued problem first, and only then can you state and solve its fuzzy version on a safe ground. This book has the great merit to lay bare the importance of this methodology, thus opening a path to the rigorous, meaningful statement of fuzzy operations research problems in the setting of possibility theory, pursuing the pioneering works by the author's mentor, the late Stefan Chanas, and by Inuiguchi and Sakawa in Japan. No doubt this book is a very useful and original contribution to modern approaches in problem solving.

Toulouse
January 2008

Didier Dubois
Directeur de Recherche au CNRS

Preface

In operations research applications we are often faced with the problem of incomplete or uncertain data. This is caused by a lack of knowledge about the considered system or by a varying nature of the world. Uncertainty is a basic structural feature of the technological and business environment and it must be regarded as a part of the decision making process. In contrast, the classical mathematical models generally deal with the deterministic data. The uncertain parameters are often estimated to be the mean or the worst-case values over all possible realizations. The main weakness of the deterministic approach is that it does not take into account many possible realizations of the input data. In consequence, the obtained decision may be unacceptable in a varying and uncertain environment.

The *stochastic optimization* is a natural approach to model the uncertainty. It requires specifying a value of probability of every instance of the input data that may be realized. Then typically a decision is generated that minimizes or maximizes an expected performance measure. The stochastic approach has several disadvantages. It may be impossible or very expensive to estimate the probability distributions for the unknown data. The assumption of the distributional independence, which is often made to simplify the model, may be not justified. On the other hand the correlations between the parameters of the problem may be hard to identify. However, the most important failure of the stochastic optimization, pointed out by Kouvelis and Yu [87], is that it typically optimizes the expected system performance. But decision makers are reasonably more interested in hedging against the risk of poor (worst case) performance over all possible data realizations. This is particularly important for decisions that are encountered only once.

In this monograph we deal with an alternative approach to modeling the incomplete knowledge, which we refer to as the *robust optimization*. The idea and the framework of the robust optimization were described in a book by Kouvelis and Yu [87] and in the influential paper by Mulvey *et al* [107]. The robust approach is based on the concept of a *scenario*, which expresses a realization of the input data which may occur with some positive, but perhaps unknown

probability. Before computing a solution the set of all scenarios, denoted by Γ, is identified. The aim is to find a solution that performs reasonably well under any scenario. Hence, contrary to the stochastic approach, in the robust optimization we minimize the worst-case system performance rather than the expected one.

There are two methods of determining the scenario set Γ. In the first case, called a *discrete scenario representation*, we explicitly list all scenarios, that is $\Gamma = \{S_1, S_2, \ldots, S_K\}$, where $K \geq 1$. In the second case, called *interval scenario representation*, for every parameter in the problem there is a closed interval given and it is assumed that the value of the parameter may fall within this interval regardless of the values of the other parameters. In the interval scenario representation the scenario set Γ is the Cartesian product of all uncertainty intervals. Observe, that in this case Γ contains an infinite number of scenarios. The discrete scenario representation allows us to model the correlations among the input data. On the other hand, the interval representation assumes that the parameters in the problem are unrelated, that is the value of every parameter does not depend on the values of the remaining ones.

There are two types of uncertainty connected with the robust approach. The first appears when the feasibility of a solution depends on a particular scenario and the second is when the value of the cost (or the profit) of a given solution depends on a particular scenario. These two types of uncertainty were discussed by Mulvey *et al.* [107] and by by Kouvelis and Yu [87]. A solution that is "close" to optimal for any scenario is called *solution robust* and a solution that remains "almost" feasible for all scenarios is termed *model robust*. The notions of "close" and "almost" depend on the assumed optimization criterion. In this monograph we deal only with the first type of uncertainty, that is the one connected with the value of the objective.

In order to choose a solution for a given scenario set Γ several *robust criteria* were proposed in literature. Under *minmax criterion*, called also *absolute robust*, we seek a solution that minimizes the highest cost or maximizes the lowest profit over all scenarios. We thus assume that the worst will happen and we obtain in consequence a conservative, free of risk decision. Another criterion, first described by Savage [115], is that of *minmax regret*. The *maximal regret* (called also *robust deviation*) of a given solution expresses the maximal deviation of the cost (profit) of this solution from optimum over all scenarios. We seek a solution that minimizes the maximal regret. Using this criterion we obtain a solution which is less conservative and which minimizes the magnitude of missing opportunities. The maximal regret criterion is appropriate when the obtained decision is evaluated *ext post* and it is compared to the best decision that could have been made. For a deeper discussion on different robust criteria and the motivation of the robust approach we refer the reader to the book of Kuovelis and Yu [87] and to the paper of Mulvey *et al.* [107].

The main part of this monograph is devoted to the following class of robust optimization problems, namely *minmax regret discrete optimization problems*:

- A solution (decision) must be deriven from a finite set and it is either a subset or a permutation of a given, finite core set. Hence we consider the class of discrete optimization problems.
- The uncertainty appears only in the value of the objective. The feasibility of a solution does not depend on a scenario.
- The interval scenario representation is adopted. Thus for every uncertain parameter an interval of possible values, without a specific probability distribution, is given.
- In order to choose a solution the maximal regret criterion is applied. We thus seek a solution that minimizes the maximal regret.

The minmax regret approach to discrete optimization problems was discussed in book [87]. However, the major part of [87] is devoted to discrete scenario representation of uncertainty. Since the 1997s a lot of papers devoted to the interval uncertainty representation have appeared and the aim of this monograph is to present the state of the art in this field. This state of the art is still far from being complete and some open problems and questions are also addressed.

This monograph is divided into two parts. In the first part, composed of Chapters 1-11, we consider a class of discrete optimization problems in which the feasible solutions are formed by subsets of a given, finite core set of elements. A problem of this type is called a *combinatorial optimization problem*. A well known example is Shortest Path in which we wish to find a path of the minimal total length between two distinguished nodes of a given graph. The uncertainty, modeled by closed intervals, appears in the weights associated to elements. These weights express the element costs, lengths, times *etc*. In Chapter 1 we provide the general formulation of the *minmax regret combinatorial optimization problem* and we show its general properties. In Chapter 1 we also discuss some related problems, in particular we discuss some other robustness criteria. Chapter 2 is devoted to a problem which is closely related to the minmax regret approach. In this chapter we introduce the notions of *possible* and *necessary* optimality of solutions and elements and we show some relationships between these concepts and the robust approach. The next two chapters, that is chapters 3 and 4 are devoted to general exact and approximation methods for solving the minmax regret combinatorial optimization problems. Among the exact methods are the mixed integer programming (MIP) formulation and the branch and bound algorithm. It turns out that a simple 2-approximation algorithm for this class of problems can also be designed. This algorithm, together with some of its extensions, are presented in Chapter 4. Chapters 5-9 are devoted to particular problems. We consider the very basic problems like Minimum Selecting Items, Minimum Spanning Tree, Shortest Path, Minimum Assignment and Minimum Cut. For every problem we characterize its computational complexity and provide some exact and approximation algorithms.

In Chapter 10 we discuss a generalization of the minmax regret approach to combinatorial optimization problems. We show how the optimality evaluation and the solution concept can be generalized by extending the notion of the classical closed interval to a fuzzy one. We provide an interpretation of the fuzzy

problem in terms of possibility theory. Namely, by introducing fuzzy intervals we can define a possibility distribution over scenario set. We show that all results obtained for interval problems can be applied to the fuzzy problems as well.

The second part of this monograph, composed of Chapters 12-16, is devoted to another class of discrete optimization problems, namely *sequencing problems*. In a sequencing problem every solution is a permutation of a given core set of elements, called *jobs*. Contrary to the combinatorial optimization problems considered in the first part, there may be several uncertain parameters associated with an element. In Chapter 12 we provide a general formulation of the *minmax regret sequencing problem*. In Chapters 13-15 we discuss three very basic problems.

In this monograph we consider only a particular class of discrete optimization problems. However, the minmax regret approach was also applied to the linear programming problem with interval coefficients in the objective function. For the results concerning this important problem we refer the reader to the papers of Inuiguchi and Sakawa [65], Mausser and Laguna [99, 100] and Averbakh and Lebedev [19].

In this monograph we do not discuss in detail the discrete scenario representation of uncertainty, which forms a separate and large class of problems. In Appendix A we only review briefly the known results and provide some new ones in this field. We compare them to the results known for the interval scenario representation.

Acknowledgments

I would like to dedicate this monograph to Professor Stefan Chanas, who is sadly no longer with us. Professor Chanas was not just my teacher; he was also my mentor who has played a great part in guiding me. Without his inspiration this monograph could not have been completed. I wish to express my gratitude to Professor Jacek Mercik and Professor Janusz Kacprzyk for their helpful suggestions and active interest in the publication of this monograph. I am indebted to Professor Didier Dubois who agreed to write the Foreword. I wish to express my thanks to all colleagues from my University, especially to Paweł Zieliński who has contributed to many results described in this book.

Wrocław
January 2008 Adam Kasperski

Contents

1 Problem Formulation

We start this chapter by recalling the formulation of a deterministic combinatorial optimization problem, that is the one in which all the input parameters are precisely known. We also present some well known, special cases of this problem like minimum spanning tree, minimum selecting items, shortest path, minimum assignment and minimum cut. We introduce then the minmax regret combinatorial optimization problem with interval data, which can be viewed as a generalization of the deterministic problem where the imprecision is taken into account. We also discuss some closely related problems. In particular we consider some other robustness criteria which can be applied to the interval uncertainty representation.

1.1 Deterministic Combinatorial Optimization Problem

Let $E = \{e_1, \ldots, e_n\}$ be a finite ground set and let Φ be a set of subsets of E called the set of the *feasible solutions*. For every element $e \in E$ there is a nonnegative weight w_e given, which expresses a certain parameter like cost, length, time *etc.* associated with e. A *deterministic combinatorial optimization problem*, denoted by $\mathcal{P}$, consists in finding a feasible solution $X \in \Phi$ whose total weight is minimal, that is

$$\mathcal{P}: \min_{X \in \Phi} \sum_{e \in X} w_e. \tag{1.1}$$

The problem $\mathcal{P}$ can be alternatively represented as the following 0-1 programming problem with a linear objective function:

$$\min \textstyle\sum_{i=1}^{n} w_i x_i$$
$$(x_1, \ldots, x_n) \subset \{0,1\}^n,$$

where x_i is a binary variable associated with element e_i and w_i is the weight of e_i. The set of feasible solutions is represented as a set of *characteristic vectors* of elements of Φ and it is typically described in a compact form by some set of constraints.

A. Kasperski: Discrete Optimization with Interval Data, STUDFUZZ 228, pp. 3–16, 2008.
springerlink.com

Formula (1.1) encompasses a wide range of problems. The following special versions of problem $\mathcal{P}$ play an important role in theory and applications:

- MINIMUM SPANNING TREE: E is a set of edges of a given undirected graph $G = (V, E)$ and Φ consists of all spanning trees of G. For this problem we wish to find a spanning tree of G whose total weight is minimal. In a typical application we wish to connect a set of points (for instance terminals) using the least weight collection of edges.
- MINIMUM SELECTING ITEMS: E is a set of items and Φ consists of all subsets of E, whose cardinalities are exactly p, where $0 < p \leq n$ is a fixed integer. The problem consists in finding a subset of exactly p items whose total weight is minimal. This problem can be viewed as a basic resource allocation one or as the 0-1 KNAPSACK problem with unit capacities of the items.
- SHORTEST PATH: E is a set of edges (arcs) of a given undirected (directed) graph G. Set Φ consists of all paths between two distinguished nodes s and t in G. We wish to find a path whose total weight (length) is minimal. This problem arises in plenty of applications. The most popular one is to design the shortest or the quickest tour between two cities. Hence V models a set of cities and E is a set of possible connections between them with specified lengths or traveling times.
- MINIMUM ASSIGNMENT: E is a set of edges of a given bipartite graph $G = (V \cup W, E)$, $|V| = |W|$, and Φ is the set of all assignments (perfect matchings) in G. We seek an assignment whose total weight is minimal. This problem arises if we wish to pair some objects, for instance tasks and machines, to achieve the minimum total cost.
- MINIMUM CUT: E is a set of edges (arcs) of a given undirected (directed) graph G and s, t are two given nodes of G. Set Φ consists of all subsets of arcs that form $s - t$ cuts in G. An $s - t$ cut is a subset of arcs (edges) whose deletion disconnects s and t. For this problem we seek a minimum weighted $s - t$ cut. The MINIMUM CUT problem arises for instance in the analysis of traffic networks.

All the special versions of problem $\mathcal{P}$ described above are polynomially solvable and they have emerged as one of the major research topics in operations research since the 1950s. They arise frequently in practice and often lie at the heart of more complex problems.

In problem $\mathcal{P}$ we have assumed that we seek a solution that minimizes the total weight. Obviously, we can also consider a problem of determining a solution whose total weight is maximal. Hence we may consider problems MAXIMUM SPANNING TREE, MAXIMUM SELECTING ITEMS, LONGEST PATH, MAXIMUM ASSIGNMENT and MAXIMUM CUT. Later in this monograph we will show that under a general assumption it is enough to consider only minimization problems.

Sometimes objective (1.1) is replaced with the *bottleneck* one, that is $\sum_{e \in X} w_e$ is replaced with $\max_{e \in X} w_e$. We will discuss the problems with the bottleneck objective in one of the next sections in this chapter.

Since $\mathcal{P}$ is a computational problem, we need to describe a way by which it is represented as the input to a computer algorithm. Obviously, listing all feasible

solutions from Φ is very inefficient because Φ may contain up to $2^{|E|}$ solutions. Therefore, we will represent Φ in a compact form of a size polynomial in n. For instance, in terms of a polynomial algorithm, which decides whether a subset $X \subseteq E$ is a feasible solution (or it is a part of a feasible solution). For example in MINIMUM SPANNING TREE the input consists of a graph $G = (V, E)$ together with $|E|$ numbers denoting the weights of edges. We additionally provide an algorithm, which decides whether a given subset of edges does not contain a cycle.

1.1.1 Matroidal Problem

We distinguish now an important class of the deterministic combinatorial optimization problems namely *matroidal problems*. Let us recall that a *matroid* is a system $(E, \mathcal{I})$, where E is a finite set of elements and $\mathcal{I}$ is a set of subsets of E which fulfills the following two axioms:

(A1) if $A \subseteq B$ and $B \in \mathcal{I}$, then $A \in \mathcal{I}$,
(A2) if $A, B \in \mathcal{I}$ and $|A| < |B|$ then there is $e \in B \setminus A$ such that $A \cup \{e\} \in \mathcal{I}$.

The maximal under inclusion elements in $\mathcal{I}$ are called *bases* and the minimal under inclusion elements not in $\mathcal{I}$ are called *circuits*. It can be easily shown that all bases of a given matroid have the same cardinality. Perhaps the best known example of a matroid is a *graphic matroid* in which E is the set of edges of a given undirected graph and $\mathcal{I}$ consists of all acyclic subgraphs of G. A base of this matroid is a spanning tree of G. Another example is a *uniform matroid* in which E is a set of elements and $\mathcal{I}$ consists of all subsets of E whose cardinalities are less than or equal to a fixed integer p. A base in this matroid is a subset whose cardinality is exactly p

In the *matroidal combinatorial optimization problem* the set of feasible solutions Φ consists of all bases of a given matroid. In MINIMUM SPANNING TREE, Φ consists of the bases of a graphic matroid and in MINIMUM SELECTING ITEMS Φ consists of the bases of an uniform matroid. The remaining three problems considered in the previous section, that is SHORTEST PATH, MINIMUM ASSIGNMENT and MINIMUM CUT are not matroidal ones. What makes the matroidal problems very specific is that they are precisely the ones for which a simple `Greedy Algorithm` works. This algorithm is shown in Figure 1.1.

The `Greedy Algorithm` first arranges the elements in nondecreasing order of their weights. It picks then the elements one by one and tries to add it to the constructed solution X. The only case in which it excludes an element is when this element makes X infeasible. As the result it returns a solution, that is a base of a matroid X, whose total weight is minimal. The greedy algorithm for MINIMUM SPANNING TREE is known in literature as Kruskal's algorithm. For MINIMUM SELECTING ITEMS the greedy algorithm simply outputs p elements of the smallest weights. Due to their special structure, the matroidal problems will play an important role in this monograph.

```
Greedy Algorithm
Require: A matroidal combinatorial optimization problem P
Ensure: An optimal solution X ∈ Φ
Order elements so that w_{e_1} ≤ w_{e_2} ≤ ··· ≤ w_{e_n}
X ← ∅
for i ← 1 to n do
    if X ∪ {e_i} ∈ I then X ← X ∪ {e_i}
end for
return X
```

Fig. 1.1. The greedy algorithm that computes an optimal solution for a matroidal combinatorial optimization problem

1.2 Minmax Regret Combinatorial Optimization Problem with Interval Data

In practical applications the exact values of input data like costs, times, lengths *etc.*, are often not known in advance. It is caused by a lack of knowledge about a considered system or by the varying nature of the world. Perhaps, the simplest form of the uncertainty representation is to assume that the value of a given parameter may fall within a given range, independently on the values taken by the other parameters. Suppose that the values of the weights in problem $\mathcal{P}$ are only known to belong to the closed intervals $\tilde{w}_e = [\underline{w}_e, \overline{w}_e]$, where $\underline{w}_e \geq 0$ for all $e \in E$. If $\underline{w}_e = \overline{w}_e$, then the value of the weight of e is *precise* and in this case interval $\tilde{w}_e$ is called *degenerate*.

A particular realization of the weights $S = (w_e^S)_{e \in E}$ such that $w_e^S \in \tilde{w}_e$ for all $e \in E$ is called a *scenario*. Thus every scenario represents a certain state of the world, that is a configuration of the weights, which may occur with a positive, but perhaps unknown probability. We will denote by Γ the set of all scenarios, that is Γ is the Cartesian product of all the uncertainty intervals, namely $\Gamma = \times_{e \in E} \tilde{w}_e$. Among the scenarios we will distinguish the *extreme* ones, in which all the weights take the extreme values $\underline{w}_e$ or $\overline{w}_e$. Let $A \subseteq E$ be a given subset of E. We will use S_A^+ to denote the extreme scenario in which the elements $e \in A$ have weights $\overline{w}_e$ and all the other elements have weights $\underline{w}_e$. Similarly, we define scenario S_A^-, in which the elements $e \in A$ have weights $\underline{w}_e$ and all the other elements have weights $\overline{w}_e$. The particular scenarios S_A^+ and S_A^- will play a crucial role in further considerations.

The weight of a given solution $X \in \Phi$ under scenario $S \in \Gamma$ is defined as follows:

$$F(X, S) = \sum_{e \in X} w_e^S. \tag{1.2}$$

Let us denote by $F^*(S)$ the weight of the optimal solution under scenario S, that is

$$F^*(S) = \min_{X \in \Phi} F(X, S). \tag{1.3}$$

In order to obtain the value of $F^*(S)$ we must solve the deterministic combinatorial optimization problem $\mathcal{P}$ for the fixed scenario $S \in \Gamma$. The *maximal regret* of a given solution X is defined as follows:

$$Z(X) = \max_{S \in \Gamma} \{F(X,S) - F^*(S)\}. \tag{1.4}$$

It is clear that $Z(X) \geq 0$ for all solutions $X \in \Phi$. A scenario S that maximizes the right hand side of (1.4) is called the *worst case scenario for* X. We now prove the following proposition, which will be crucial in further considerations:

Proposition 1.1. *The scenario* S_X^+ *is the worst case scenario for solution* X.

Proof. Consider a scenario $S \in \Gamma$ and denote by Y the optimal solution under S, that is $F(Y,S) = F^*(S)$. It holds

$$F(X,S) - F^*(S) = \sum_{e \in X \setminus Y} w_e^S - \sum_{e \in Y \setminus X} w_e^S \leq \sum_{e \in X \setminus Y} \overline{w}_e - \sum_{e \in Y \setminus X} \underline{w}_e =$$
$$= F(X,S_X^+) - F(Y,S_X^+) \leq F(X,S_X^+) - F^*(S_X^+).$$

Hence S_X^+ maximizes the right hand side of (1.4) and S_X^+ is the worst case scenario for X. □

By Proposition 1.1 we can express the maximal regret of a given solution X in the following way:

$$Z(X) = F(X,S_X^+) - F^*(S_X^+) \tag{1.5}$$

or alternatively, making use of (1.2) and (1.5):

$$Z(X) = \max_{Y \in \Phi} \{ \sum_{e \in X \setminus Y} \overline{w}_e - \sum_{e \in Y \setminus X} \underline{w}_e \} = \sum_{e \in X \setminus X^*} \overline{w}_e - \sum_{e \in X^* \setminus X} \underline{w}_e, \tag{1.6}$$

where X^* is the optimal solution under scenario S_X^+. The solution X^* is called the *worst case alternative for* X.

Equation (1.5) allows us to derive an important general conclusion. If the underlying deterministic combinatorial optimization problem $\mathcal{P}$ is polynomially solvable, then the maximal regret of a given solution X can be computed in polynomial time. It follows from the fact that in this case both scenario S_X^+ and the value of $F^*(S_X^+)$ can be obtained in polynomial time.

In this part of the monograph we focus on the following *minmax regret combinatorial optimization problem* $\mathcal{P}$:

$$\text{Minmax Regret } \mathcal{P}: \min_{X \in \Phi} Z(X).$$

Thus we seek a feasible solution for which the maximal regret is minimal. We will call the optimal solution to Minmax Regret $\mathcal{P}$ the *optimal robust solution*. Observe that Minmax Regret $\mathcal{P}$ is a natural generalization of problem $\mathcal{P}$ under uncertainty. If all the weight intervals are degenerate (in other words,

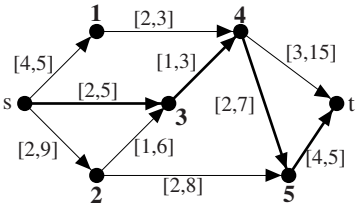


Fig. 1.2. An example of Minmax Regret Shortest Path

their values are precisely known), then there is only one scenario and Minmax Regret $\mathcal{P}$ is equivalent to $\mathcal{P}$. It is worth pointing out here that among the minmax regret versions of the problems considered in Section 1.1, only Minmax Regret Minimum Selecting Items is known to be polynomially solvable. All the remaining problems turned out to be NP-hard. Thus Minmax Regret $\mathcal{P}$ may be much harder to solve than $\mathcal{P}$.

Example 1.2. Let us illustrate the Minmax Regret $\mathcal{P}$ problem by an example. Consider an instance of Minmax Regret Shortest Path presented in Figure 1.2. An input is a directed graph G with interval weights that represent uncertain arc lengths. The path X composed of arcs $(s,3)$, $(3,4)$, $(4,5)$, $(5,t)$ is the optimal robust path. The worst case scenario S_X^+ for X is obtained by setting the weights of arcs in X to their upper bounds and the weights of the remaining arcs to their lower bounds. One can check that the maximal regret of path X is equal to 11. □

In some problems set Φ may contain two solutions X and Y such that $X \subset Y$. This is the case for example in Minmax Regret Shortest Path or Minmax Regret Minimum Cut if we do not make the assumption that only simple paths and simple cuts are considered. However, the following proposition shows that this assumption is not necessary:

Proposition 1.3. *Let X be a feasible solution such that $X \subset A$, where $A \subseteq E$. Then the following inequality holds*

$$Z(X) \leq F(S_A^+, A) - F^*(S_A^+). \tag{1.7}$$

Proof. Since the weights under all scenarios are nonnegative and $X \subset A$ we have

$$F(A, S_A^+) = F(X, S_X^+) + \sum_{e \in A \setminus X} \overline{w}_e. \tag{1.8}$$

and

$$F^*(S_A^+) \leq F(X^*, S_A^+) = F(X^*, S_{X \cup (A \setminus X)}^+) \leq F(X^*, S_X^+) + \sum_{e \in A \setminus X} \overline{w}_e. \tag{1.9}$$

Subtracting (1.9) from (1.8) yields (1.7). □

Proposition 1.3 will also be used in Chapter 3, where a mixed integer programming model for MINMAX REGRET $\mathcal{P}$ will be constructed. We finish this section by proving the following auxiliary proposition, that will be used later in this monograph:

Proposition 1.4. *Let X and Y be two solutions such that $F(X, S_X^-) = F(Y, S_X^-)$. Then the following two statements are true:*

1) $F(X, S) \geq F(Y, S)$ for all scenarios $S \in \Gamma$;
2) if X is an optimal robust solution, then Y is also an optimal robust solution.

Proof. We first prove 1). For any scenario $S \in \Gamma$ it holds $F(X, S) - F(Y, S) \geq F(X, S_X^-) - F(Y, S_X^-) = 0$. Thus $F(X, S) \geq F(Y, S)$ for all scenarios $S \in \Gamma$. Using this fact we also get

$$Z(Y) = \max_{S \in \Gamma} \{F(Y, S) - F^*(S)\} = F(Y, S_Y^+) - F^*(S_Y^+) \leq$$
$$\leq F(X, S_Y^+) - F^*(S_Y^+) \leq \max_{S \in \Gamma} \{F(X, S) - F^*(S)\} = Z(X)$$

Hence 2) follows. □

1.3 Some Related Problems

In the previous section we have introduced the problem MINMAX REGRET $\mathcal{P}$, which is the central object of study in this part of the monograph. In this section we briefly describe some problems, which are closely related to MINMAX REGRET $\mathcal{P}$. We discuss first some other robustness criteria, that is *absolute robust* and *relative regret*. We consider also the minmax regret versions of the deterministic problems with the objective function other than the minimization of the total weight. We also present a problem which can be viewed as a special case of MINMAX REGRET $\mathcal{P}$, namely a *central combinatorial optimization problem*.

1.3.1 Other Robustness Criteria

The maximal regret criterion is one of several criteria used in robust optimization. One can also evaluate a solution by computing its largest weight over all scenarios, namely

$$A(X) = \max_{S \in \Gamma} F(X, S).$$

In the MINMAX $\mathcal{P}$ problem we seek a solution $X \in \Phi$ that minimizes the value of $A(X)$. The criterion $A(X)$ is called in literature *absolute robust*. It is easy to see that for the interval representation of uncertainty this problem boils down to determining an optimal solution for problem $\mathcal{P}$ under scenario S_E^+. It follows from the fact that $\max_{S \in \Gamma} F(X, S) = F(X, S_E^+)$. Hence the complexity of MINMAX $\mathcal{P}$ is the same as the complexity of problem $\mathcal{P}$. Any algorithm for solving problem $\mathcal{P}$ can be used to solve MINMAX $\mathcal{P}$ as well.

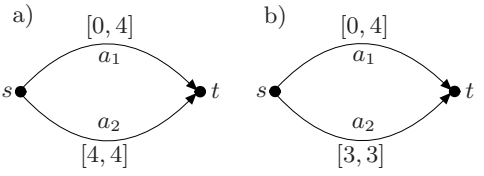


Fig. 1.3. Two sample SHORTEST PATH problems with interval weights

Let us now compare the two robust criteria, that is the maximal regret and the absolute robust, by demonstrating a very simple example. Consider a problem shown in Figure 1.3, in which we wish to choose one of two possible paths from s to t. The intervals associated with arcs express the uncertain costs of traversing these arcs. In the first case, shown in Figure 1.3a, there are no doubts that path a_1 should be chosen. Under every scenario, path a_1 has the cost less than or equal to the cost of path a_2. However, applying the absolute robust criterion we may obtain the worse path a_2, while the maximal regret yields path a_1 (observe that the maximal regret of path a_1 equals 0). In this case the maximal regret criterion is clearly better than the absolute robust one.

Consider the case shown in Figure 1.3b. In this problem, applying the absolute robust criterion we get path a_2 and applying the maximal regret criterion we obtain path a_1. However, it is not obvious which path is now better. Path a_2 guarantees that in the worst case we pay less than if we had chosen path a_1. Hence, path a_2 seems to be a better choice. However, suppose that we have a competitor who has chosen path a_1. In the worst case we pay 3, while the competitor will pay 0, thus the competitor will win the prize of 3. If we chose path a_1, then the competitor could win only the prize of 1 in the worst case. Hence choosing path a_2 we minimize the maximal superiority of our competitor, which may be more important than minimizing the maximal cost.

Another criterion used in robust optimization is a *relative regret*. The relative regret of a solution $X \in \Phi$ is defined as follows:

$$R(X) = \max_{S \in \Gamma} \left\{ \frac{F(X,S) - F^*(S)}{F^*(S)} \right\}.$$

We assume that $F^*(S) > 0$ for all scenarios $S \in \Gamma$. In the RELATIVE REGRET $\mathcal{P}$ problem we wish to find a feasible solution X which minimizes the value of $R(X)$. It was pointed out by Kouvelis and Yu [87], that the relative regret should be used when the performance of a decision across scenarios is highly variable and for most cases the relative regret and the maximal regret criteria tend to favor similar decisions (see [87]).

The RELATIVE REGRET $\mathcal{P}$ problem has been studied by Averbakh [20] and it turns out that it is closely related to MINMAX REGRET $\mathcal{P}$. Averbakh has shown that under some mild conditions MINMAX REGRET $\mathcal{P}$ can be reduced to solve the RELATIVE REGRET $\mathcal{P}$ problem. Therefore RELATIVE REGRET $\mathcal{P}$ is at least as hard as MINMAX REGRET $\mathcal{P}$. Moreover, similar techniques that lead

to an algorithm for MINMAX REGRET $\mathcal{P}$ allow us to construct an algorithm for RELATIVE REGRET $\mathcal{P}$. In particular a polynomial time algorithm for RELATIVE REGRET MINIMUM SELECTING ITEMS can be designed, which is similar to the polynomial algorithm for MINMAX REGRET MINIMUM SELECTING ITEMS. In this monograph we do not consider the relative regret and for a deeper discussion on this criterion we refer the reader to Averbakh's paper [20].

1.3.2 Problem with Maximization Objective Function

In Sections 1.1 and 1.2 we have assumed that the underlying deterministic combinatorial optimization problem $\mathcal{P}$ is a minimization problem (see (1.1)). However, some important problems are often formulated as the maximization ones, that is

$$\mathcal{Q}: \max_{X\in\Phi}\sum_{e\in X} w_e.$$

A well known example of problem $\mathcal{Q}$ is LONGEST PATH, in which there is an acyclic directed graph $G=(V,A)$ given and we wish to find a path between two distinguished nodes of G of the maximal total weight (length). Of course, the minmax regret version of problem $\mathcal{Q}$ can also be considered. A slight modification of the definition of the maximal regret is only required. First, for a given scenario $S\in\Gamma$, we define

$$F^*(S)=\max_{X\in\Phi} F(X,S).$$

The maximal regret of a given solution X is defined as follows:

$$Z(X)=\max_{S\in\Gamma}\{F^*(S)-F(X,S)\}.$$

It is easy to show that S_X^- is now the worst case scenario for a given solution X and the proof goes the same as the proof of Proposition 1.1. Therefore, we can express the maximal regret of a given solution as follows:

$$Z(X)=F(X^*,S_X^-)-F(X,S_X^-),$$

where $F(X^*,S_X^-)=F^*(S_X^-)$ or alternatively

$$Z(X)=\max_{Y\in\Phi}\{\sum_{e\in Y\setminus X}\overline{w}_e-\sum_{e\in X\setminus Y}\underline{w}_e\}=\sum_{e\in X^*\setminus X}\overline{w}_e-\sum_{e\in X\setminus X^*}\underline{w}_e. \tag{1.10}$$

In the MINMAX REGRET $\mathcal{Q}$ problem we seek a solution X that minimizes the maximal regret. We now show that under a general assumption, MINMAX REGRET $\mathcal{Q}$ can be efficiently transformed into MINMAX REGRET $\mathcal{P}$ and vice versa.

Theorem 1.5. *If all feasible solutions in set Φ have the same cardinality, that is $|X|=k$ for all $X\in\Phi$, then problems* MINMAX REGRET $\mathcal{Q}$ *and* MINMAX REGRET $\mathcal{P}$ *can be transformed one to the other in $\mathcal{O}(n)$ time.*

Proof. Consider the MINMAX REGRET $\mathcal{Q}$ problem, which consists of sets E, Φ and interval weights $[\underline{w}_e, \overline{w}_e]$ specified for all $e \in E$. Let us define problem MINMAX REGRET $\mathcal{P}$ that has the same sets E and Φ. The interval weights in MINMAX REGRET $\mathcal{P}$ are $[M - \overline{w}_e, M - \underline{w}_e]$, $e \in E$, where $M = \max_{e \in E} \overline{w}_e$. Let $Z_{\mathcal{Q}}(X)$ and $Z_{\mathcal{P}}(X)$ denote the maximal regrets of a solution X in MINMAX REGRET $\mathcal{Q}$ and MINMAX REGRET $\mathcal{P}$ respectively. It holds (see formula (1.6))

$$Z_{\mathcal{P}}(X) = \max_{Y \in \Phi} \{ \sum_{e \in X \setminus Y} (M - \underline{w}_e) - \sum_{e \in Y \setminus X} (M - \overline{w}_e) \}.$$

Since $|X| = |Y| = k$ it holds $\sum_{e \in X \setminus Y} M = \sum_{e \in Y \setminus X} M$ and, by (1.10), we get

$$Z_{\mathcal{P}}(X) = \max_{Y \in \Phi} \{ \sum_{e \in Y \setminus X} \overline{w}_e - \sum_{e \in X \setminus Y} \underline{w}_e \} = Z_{\mathcal{Q}}(X).$$

Thus the maximal regrets of a given solution in MINMAX REGRET $\mathcal{Q}$ and MINMAX REGRET $\mathcal{P}$ are equal and both problems have the same optimal solutions. The transformation can be done in $\mathcal{O}(n)$ time required to find the value of M and to transform the weight intervals. Exactly the same method can be applied to perform the opposite transformation from MINMAX REGRET $\mathcal{P}$ to MINMAX REGRET $\mathcal{Q}$. □

Theorem 1.5 can be applied immediately to MINMAX REGRET MAXIMUM SELECTING ITEMS, MINMAX REGRET MAXIMUM SPANNING TREE and MINMAX REGRET MAXIMUM ASSIGNMENT. After an additional transformation it can also be applied to MINMAX REGRET LONGEST PATH in directed acyclic graphs (we will show this transformation in Chapter 7). For all of these problems it is enough to consider only the case in which the deterministic counterpart is a minimization problem.

Let us also point out that the assumption that all feasible solutions have the same cardinality is crucial. Consider for example MINMAX REGRET MAXIMUM CUT. There is no polynomial transformation of this problem to MINMAX REGRET MINIMUM CUT if P≠NP. It follows from the fact that MINIMUM CUT is polynomially solvable while MAXIMUM CUT is known to be NP-hard. Obviously, in MAXIMUM CUT not all feasible solutions ($s-t$ cuts) have the same cardinality and Theorem 1.5 cannot be applied to this problem.

1.3.3 Problem with Bottleneck Objective Function

Consider again the deterministic combinatorial optimization problem discussed in Section 1.1. Let us replace the objective function (1.1) with the *bottleneck* one, and consider the following deterministic optimization problem:

$$\mathcal{R}: \min_{X \in \Phi} \max_{e \in X} w_e.$$

For the interval uncertainty representation, the weight of a solution X in scenario S is as follows:

$$F(X, S) = \max_{e \in X} w_e^S.$$

The maximal regret of X is defined exactly the same as in MINMAX REGRET $\mathcal{P}$, that is by means of formula (1.4). The minmax regret version of problem $\mathcal{R}$, namely MINMAX REGRET $\mathcal{R}$, consists in finding a feasible solution that minimizes the maximal regret. For this problem a general result can be proved. It turns out that if problem $\mathcal{R}$ is polynomially solvable, then its minmax regret version is polynomially solvable as well. We will show the proof of this result, which is similar to the one presented by Averbakh [15]. The following proposition allows us to calculate the maximal regret of a given solution X:

Proposition 1.6. *Let X be a feasible solution. Then*

$$Z(X) = \max_{e \in X} \{\max\{\overline{w}_e - F^*(S^+_{\{e\}}), 0\}\}.$$

Proof. Let S be a worst case scenario for X, that is S maximizes the value of $F(X,S) - F^*(S)$, and let $f \in X$ be the element such that

$$F(X,S) = \max_{e \in X} w_e^S = w_f^S.$$

It holds

$$Z(X) = w_f^S - F^*(S) \leq \overline{w}_f - F^*(S^+_{\{f\}}) \leq \max_{e \in X} \{\max\{\overline{w}_e - F^*(S^+_{\{e\}}), 0\}\}. \quad (1.11)$$

For every element $e \in X$ it holds $Z(X) \geq F(X, S^+_{\{e\}}) - F^*(S^+_{\{e\}}) \geq \overline{w}_e - F^*(S^+_{\{e\}})$ and $Z(X) \geq \max\{\overline{w}_e - F^*(S^+_{\{e\}}), 0\}$ because the maximal regret is nonnegative. Consequently

$$Z(X) \geq \max_{e \in X} \{\max\{\overline{w}_e - F^*(S^+_{\{e\}}), 0\}\},$$

which together with (1.11) completes the proof. □

Let us define $w_e^* = \max\{\overline{w}_e - F^*(S^+_{\{e\}}), 0\}$ for every element $e \in E$. According to Proposition 1.6, MINMAX REGRET $\mathcal{R}$ is equivalent to the following deterministic problem:

$$\mathcal{R}^* : \min_{X \in \Phi} \max_{e \in X} w_e^*.$$

We have thus established the following result (see also [15]):

Theorem 1.7 ([15]). *If the deterministic problem $\mathcal{R}$ can be solved in $\mathcal{O}(f(n))$ time, then the corresponding* MINMAX REGRET $\mathcal{R}$ *problem can be solved in $\mathcal{O}(nf(n))$ time.*

Proof. The algorithm for MINMAX REGRET $\mathcal{R}$ proceeds as follows. First, the values of $F^*(S^+_{\{e\}})$ and the modified weights w_e^* for all $e \in E$ are computed, which requires $\mathcal{O}(nf(n))$ time. Next, problem $\mathcal{R}^*$ is solved, which requires $\mathcal{O}(f(n))$ time. Thus the overall running time is $\mathcal{O}(nf(n))$. □

`Algorithm Minmax Regret` $\mathcal{R}$
Require: An instance of MINMAX REGRET $\mathcal{R}$
Ensure: The optimal robust solution X
1: Compute the optimal solution Y for problem $\mathcal{R}$ under S_E^-
2: $l \leftarrow F(Y, S_E^-)$
3: **for all** $e \in E$ **do**
4: **if** $e \in Y$ **then** $w_e^* \leftarrow \max\{\overline{w}_e - F^*(S_{\{e\}}^+), 0\}$ **else** $w_e^* \leftarrow \max\{\overline{w}_e - l, 0\}$
5: **end for**
6: Output the optimal solution for problem $\mathcal{R}$ for weights $(w_e^*)_{e \in E}$

Fig. 1.4. Algorithm for solving MINMAX REGRET $\mathcal{R}$ (see also [15])

From Theorem 1.7 it follows that if problem $\mathcal{R}$ is polynomially solvable, then the corresponding MINMAX REGRET $\mathcal{R}$ problem is polynomially solvable as well. The running time of the algorithm for solving MINMAX REGRET $\mathcal{R}$ can be additionally refined by observing that it is not necessary to calculate the values of $F^*(S_{\{e\}}^+)$ for all elements $e \in E$. It is a consequence of the following proposition:

Proposition 1.8. *Let Y be the optimal solution for problem $\mathcal{R}$ under scenario S_E^-. Then for every element $e \in E \setminus Y$ equality $F^*(S_{\{e\}}^+) = F^*(S_E^-)$ holds.*

Proof. It is clear that $F^*(S_{\{e\}}^+) \geq F^*(S_E^-)$ for all $e \in E$. If additionally $e \notin Y$, then

$$F^*(S_{\{e\}}^+) \leq F(Y, S_{\{e\}}^+) = F(Y, S_E^-) = F^*(S_E^-).$$

Hence $F^*(S_{\{e\}}^+) = F^*(S_E^-)$ for all $e \notin Y$. □

From Proposition 1.8 it follows that we shall first solve problem $\mathcal{R}$ for scenario S_E^-, obtaining a solution Y. Problem $\mathcal{R}$ must be then additionally solved for all scenarios $S_{\{e\}}^+$, $e \in Y$. Thus all the modified weights w_e^* can be computed in $\mathcal{O}(kf(n))$ time, where k is the maximal cardinality of a solution in Φ. The algorithm for solving MINMAX REGRET $\mathcal{R}$ is shown in Figure 1.4, under the assumption that we have an algorithm for solving the deterministic problem $\mathcal{R}$.

1.3.4 Central Combinatorial Optimization Problem

Assume that all the feasible solutions, that is the elements of Φ, have the same cardinality equal to k. This is the case for instance in MINIMUM SPANNING TREE or MINIMUM ASSIGNMENT. We define a *distance* between two solutions X and Y as

$$d(X, Y) = |X \setminus Y| = |Y \setminus X|,$$

so, the distance equals the cardinality of the difference between X and Y. Consider the following *central combinatorial optimization problem* $\mathcal{P}$:

$$\text{CENTRAL } \mathcal{P}: \min_{X \in \Phi} \max_{Y \in \Phi} d(X, Y).$$

In this problem we seek a solution whose maximal distance to all other solutions is minimal. The optimal solution of CENTRAL $\mathcal{P}$ is called the *central solution*. An example of the CENTRAL $\mathcal{P}$ problem is CENTRAL SPANNING TREE in which Φ consists of all spanning trees of a given graph G and we seek a spanning tree for whose maximal distance to all other spanning trees is minimal. This problems arises in some applications, for instance in the analysis of transportation networks. We now show that problem CENTRAL $\mathcal{P}$ can be viewed as a special case of problem MINMAX REGRET $\mathcal{P}$. To see this, let us associate interval weight $\tilde{w}_e = [0, 1]$ with every element $e \in E$. The following relation between the maximal regret and the distance holds:

Proposition 1.9. *For every solution X it holds*

$$Z(X) = \max_{Y \in \Phi} d(X, Y).$$

Proof. Applying formula (1.6) and using the fact that $\underline{w}_e = 0$ and $\overline{w}_e = 1$ for all $e \in E$ we obtain $Z(X) = |X \setminus X^*| = d(X, X^*)$, where X^* is the worst case alternative for X. Hence, from the definition of the maximal regret we get $Z(X) = \max_{Y \in \Phi} d(X, Y)$. □

A direct consequence of Proposition 1.9 is the following theorem:

Theorem 1.10. *A solution X is a central solution in* CENTRAL $\mathcal{P}$ *if and only if it is an optimal robust solution in the corresponding* MINMAX REGRET $\mathcal{P}$ *problem with* $[0, 1]$ *weight intervals.*

1.4 Notes and References

The deterministic combinatorial optimization problems have been extensively studied by researchers since the 1950s. The books by Ahuja *et al.* [2], Lawler [92], Papadimitriou and Steiglitz [112] provide an excellent review of the known algorithms for this class of problems. Matroids were first introduced by Whitney [122]. The application of matroid theory to combinatorial optimization was first shown by Edmonds [48]. The greedy algorithm for MINIMUM SPANNING TREE was constructed by Kruskal [89].

The MINMAX REGRET $\mathcal{P}$ problem is one of the robust approaches for modeling uncertainty in discrete optimization. The framework of the robust approach was described in a book by Kouvelis and Yu [87]. In [87] as well as in some papers the maximal regret criterion is also called a *robust deviation* or an *absolute robust deviation*. The maximal regret as a criterion of choosing a decision under uncertainty was introduced by Savage [115].

Proposition 1.1, which describes the worst case scenario for a given solution X, was proved for some particular problems like MINMAX REGRET MINIMUM SPANNING TREE (Yaman *et al.* [124]) and MINMAX REGRET SHORTEST PATH (Karasan *et al.* [70]). The generalization to all MINMAX REGRET $\mathcal{P}$ problems is straightforward. The fact that the maximal regret of a given solution

can be computed in polynomial time, if problem $\mathcal{P}$ is polynomially solvable, is an important property of this class of problems. This property is not always true. For instance, if the minmax regret approach is applied to linear programming with interval coefficients in the objective function, then even calculation of the maximal regret of a given feasible solution is NP-hard (Averbakh and Lebedev [19]).

The criteria used in the robust optimization were described by Kuovelis and Yu in book [87]. The application of the robust absolute criterion to problem $\mathcal{P}$ with interval weights is trivial. This criterion becomes nontrivial for the discrete scenario representation, where most problems turned out to be NP-hard even if $\mathcal{P}$ is polynomially solvable. Most of the results concerning the absolute robust criterion can be found in [87, 129, 128]. The relative regret criterion is closely related to the maximal regret. The relationships between these two criteria were discussed by Averbakh [20].

The results concerning the Minmax Regret $\mathcal{R}$ problem (in particular Theorem 1.7 and the algorithm shown in Figure 1.4) were first obtained by Averbakh [15]. The notion of a central spanning tree was introduced by Deo [41] and the Central Spanning Tree problem was studied by Deo [41], Amoia and Cottafava [10], Bezrukow and Poguntke [25]. Bezrukov and Poguntke [25] proved that this problem is strongly NP-hard. The connection between Central $\mathcal{P}$ and Minmax Regret $\mathcal{P}$ problems was first established by Aron and van Hentenryck [13], while studying the complexity of Minmax Regret Minimum Spanning Tree. In fact, Theorem 1.10 together with the result obtained by Bezrukow and Poguntke [25], immediately imply that the Minmax Regret Minimum Spanning Tree problem is strongly NP-hard. We will discuss this result more deeply in Chapter 6.

2 Evaluation of Optimality of Solutions and Elements

In this chapter we discuss a problem, which is closely related to the minmax regret approach. Namely, we provide a method of characterization of optimality of a given solution or a given element under uncertainty. In order to approach this topic, consider a deterministic combinatorial optimization problem $\mathcal{P}$. The solution set Φ in this problem can be partitioned into two disjoint subsets: the optimal and the non-optimal solutions. A similar partition can be done for the elements. An element $e \in E$ is called *optimal* if it belongs to an optimal solution to problem $\mathcal{P}$. Now set E can also be partitioned into the optimal and non-optimal elements.

If the weights in the problem are uncertain, then the optimality of solutions and elements may be not known in advance and it may depend on scenario $S \in \Gamma$. However, the solutions and elements now form three groups: those which are optimal for sure (they are optimal for all scenarios $S \in \Gamma$), those that are non-optimal for sure (they are non-optimal for all scenarios $S \in \Gamma$) and those whose optimality is unknown (they are optimal for some scenarios but non-optimal for the other ones). The aim of this chapter is to provide a method of finding this tri-partition of solutions and elements. We also show that this partition has some important connections with the minmax regret approach. It allows us to preprocess a given instance of the problem by adding or excluding some elements while constructing the solution. It has been reported in literature that the preprocessing may significantly speed up the calculations of the optimal robust solution. In Section 2.1 we discuss the evaluation of optimality of solutions and in Section 2.2 the evaluation of optimality of elements. While the optimality of a solution can be easily characterized, the characterization of optimality of an element may be much more complex, even if problem $\mathcal{P}$ is polynomially solvable. Some additional results for particular problems will be presented in the next chapters of this monograph.

The characterization of optimality has some tradition in discrete optimization. Apart from its usefulness in the minmax regret approach, it also allows us to obtain some information about the solutions and elements under uncertainty. Furthermore, a classical interval can be viewed as a special case of a fuzzy

A. Kasperski: Discrete Optimization with Interval Data, STUDFUZZ 228, pp. 17–30, 2008.
springerlink.com

interval that allows more sophisticated, possibilistic uncertainty evaluation. The generalization of the optimality evaluation to the fuzzy case will be discussed in Chapter 10.

2.1 Evaluation of Optimality of Solutions

Suppose we are given a combinatorial optimization problem $\mathcal{P}$ with interval weights of the elements. A solution $X \in \Phi$ is called *possibly optimal* if it is an optimal solution to problem $\mathcal{P}$ for at least one scenario $S \in \Gamma$. A solution $X \in \Phi$ is called *necessarily optimal* if its an optimal solution to problem $\mathcal{P}$ for all scenarios $S \in \Gamma$. Obviously, a necessarily optimal solution is possibly optimal but the converse is not true in general. When one deals with the interval uncertainty representation, it is clear that a necessarily optimal solution is the best choice. That is because this solution is optimal regardless of the realization of the weights. On the other hand, the possible optimality of a solution is the minimum requirement that should be satisfied. A possibly optimal solution always exists because we can choose any scenario $S \in \Gamma$ and compute the optimal solution under S. For that reason, the number of possibly optimal solutions may be very large. A necessarily optimal solution, however, rarely exists. Detecting a necessarily optimal solution is more tricky and we postpone this problem to Chapter 4. We only mention here, that it is possible to detect a necessarily optimal solution in polynomial time if problem $\mathcal{P}$ is polynomially solvable. We thus can see that the possible optimality is too weak criterion for choosing a solution, while the necessary optimality seems to be too strong. In this section we show that every optimal robust solution is possibly optimal and this is precisely the solution that minimizes the distance to the necessary optimality.

Let $S \in \Gamma$ be a given scenario. We define a *deviation* of solution X under S in the following way:

$$\delta_X(S) = F(X,S) - F^*(S). \tag{2.1}$$

We can define the widest interval $\Delta_X = [\underline{\delta}_X, \overline{\delta}_X]$ of possible values of deviation of solution X, where

$$\underline{\delta}_X = \min_{S\in\Gamma} \delta_X(S), \quad \overline{\delta}_X = \max_{S\in\Gamma} \delta_X(S).$$

The following fact is easy to prove:

Proposition 2.1. *A solution $X \in \Phi$ is possibly optimal if and only if $\underline{\delta}_X = 0$ and it is necessarily optimal if and only if $\overline{\delta}_X = 0$.*

So, the concept of a deviation allows us to express both possible and necessary optimality of a solution. Moreover, the deviation indicates how far this solution is from being possibly and necessarily optimal. We now focus on the problem of computing Δ_X for a given solution X.

Proposition 2.2. *Let X be a given feasible solution. Then*

$$\underline{\delta}_X = \delta_X(S_X^-), \tag{2.2}$$

$$\overline{\delta}_X = \delta_X(S_X^+) = Z(X). \tag{2.3}$$

Proof. The equality (2.3) follows directly from the fact that $\overline{\delta}_X$ is the maximal value of (2.1) over all scenarios $S \in \Gamma$ and it is obviously the maximal regret $Z(X)$ of X (see (1.4)). The scenario which maximizes $\overline{\delta}_X(S)$ is the worst case scenario for X and it is S_X^+ (see Proposition 1.1). In order to prove (2.2), suppose that S minimizes $\delta_X(S)$ and denote by Y the optimal solution under S_X^-. It holds $\underline{\delta}_X = F(X,S) - F^*(S) \geq F(X,S) - F(Y,S) \geq F(X,S_X^-) - F(Y,S_X^-) = F(X,S_X^-) - F^*(S_X^-) = \delta_X(S_X^-)$. Consequently, scenario S_X^- also minimizes $\delta_X(S)$. □

Propositions 2.1 and 2.2 lead to the following characterization of the possibly and necessarily optimal solutions:

Proposition 2.3. *A solution X is possibly optimal if and only if it is the optimal solution to problem $\mathcal{P}$ under scenario S_X^- and it is necessarily optimal if and only if it is the optimal solution to problem $\mathcal{P}$ under scenario S_X^+.*

Hence we can characterize the optimality of solution X in terms of two extreme scenarios S_X^+ and S_X^-. If problem $\mathcal{P}$ is polynomially solvable, then checking whether a given solution X is possibly or necessarily optimal can be done in polynomial time as well. It is enough to compute the values of $F^*(S_X^-)$ and $F^*(S_X^+)$, which requires polynomial time. A solution X is then possibly optimal if and only if $F(X,S_X^-) = F^*(S_X^-)$ and it is necessarily optimal if and only if $F(X,S_X^+) = F^*(S_X^+)$. The following property of every optimal robust solution is true:

Proposition 2.4. *Every optimal robust solution is possibly optimal.*

Proof. Let X be an optimal robust solution. Suppose, by contradiction, that X is not possibly optimal. Therefore, by Proposition 2.3, X is not optimal under scenario S_X^-. Let Y be the optimal solution under S_X^-. It holds $F(Y,S_X^-) < F(X,S_X^-)$. It is easily seen that $F(Y,S) < F(X,S)$ for all scenarios $S \in \Gamma$. In consequence, we obtain

$$Z(Y) = F(Y,S_Y^+) - F^*(S_Y^+) < F(X,S_Y^+) - F^*(S_Y^+) \leq$$

$$\leq \max_{S\in\Gamma}\{F(X,S) - F^*(S)\} = Z(X),$$

which contradicts the assumption that X is the optimal robust solution. □

Now we have a link between necessarily optimal solutions and optimal robust solutions.

Proposition 2.5. *Every necessarily optimal solution is the optimal robust solution with the maximal regret equal to 0.*

Proof. Let X be a necessarily optimal solution. From Proposition 2.3, it follows that X is the optimal solution to problem $\mathcal{P}$ under scenario S_X^+. Furthermore, scenario S_X^+ is the worst case scenario for X so

$$Z(X) = F(X,S_X^+) - F^*(S_X^+) = 0.$$

Obviously, X is the optimal robust solution since the maximal regret is nonnegative. □

We can see that every optimal robust solution X is such that $\underline{\delta}_X = 0$ (it is optimal under some scenario) and $\overline{\delta}_X = Z(X)$ has the smallest (but perhaps positive) value among all solutions in Φ. Hence X can be viewed as a solution that has the smallest distance to the necessary optimality.

We now show another consequence of the relationships between the optimality evaluation and the optimal robust solutions. If X is an optimal robust solution, then it is possibly optimal and, by Proposition 2.3, it is the optimal solution to problem $\mathcal{P}$ under extreme scenario S_X^-. Furthermore, Proposition 1.4 implies that all optimal solutions under S_X^- are the optimal robust solutions. Hence the optimal robust solution can be found by computing the optimal solutions for $\mathcal{P}$ under all extreme scenarios and choosing the one with the smallest maximal regret. Clearly, this procedure is exponential in the general case since the number of extreme scenarios may grow exponentially with n. However, the number of distinct extreme scenarios depends on the number of nondegenerate weight intervals. If the number of nondegenerate intervals is equal to r, then the number of distinct extreme scenarios equals 2^r. Therefore, if r is bounded by the logarithm of a polynomial function of n, then the number of distinct extreme scenarios is bounded by a polynomial function of n. In this case an optimal robust solution can be found in polynomial time. We have thus obtained the following general result, first observed by Averbakh and Lebedev [17]:

Theorem 2.6 ([17]). *If problem $\mathcal{P}$ is polynomially solvable and the number of nondegenerate intervals is bounded by the logarithm of a polynomial function of n, then* Minmax Regret $\mathcal{P}$ *is polynomially solvable.*

2.2 Evaluation of Optimality of Elements

An element $e \in E$ is said to be *optimal* under scenario S if e belongs to an optimal solution under S. An element $e \in E$ is called *possibly optimal* if it is optimal under at least one scenario $S \in \Gamma$. An element $e \in E$ is called *necessarily optimal* if it is optimal under all scenarios $S \in \Gamma$.

There are some obvious relationships between the optimality of solutions and elements. A possibly optimal solution is composed of possibly optimal elements. However, the converse statement is not true since there may exist a nonpossibly optimal solution entirely composed of possibly optimal elements. An example is shown in Figure 2.1a. Similarly, a necessarily optimal solution is composed of necessarily optimal elements but there may exist a nonnecessarily optimal solution entirely composed of necessarily optimal elements. Furthermore, a necessarily optimal solution may not exist, even if there are some necessarily optimal elements. The examples are demonstrated in Figure 2.1b and Figure 2.2. Finally, it is easily seen that every necessarily optimal solution (element) is possibly optimal but the converse is not true.

Similarly to the solutions, we can express the optimality of an element by means of the concept of a deviation. Let $S \in \Gamma$ be a given scenario. A *deviation* of element e under S is defined as follows:

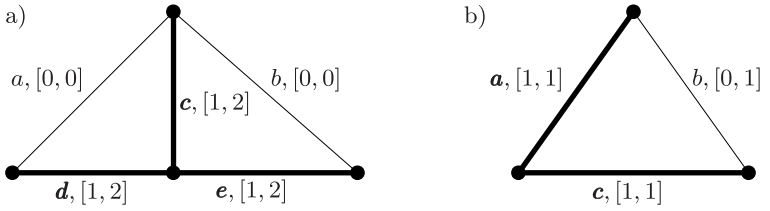


Fig. 2.1. a) An example of Minimum Spanning Tree, where possibly optimal edges $\{c, d, e\}$ (in bold) do not form a possibly optimal spanning tree. b) An example of Minimum Spanning Tree, where necessarily optimal edges $\{a, c\}$ (in bold) do not form a necessarily optimal spanning tree.

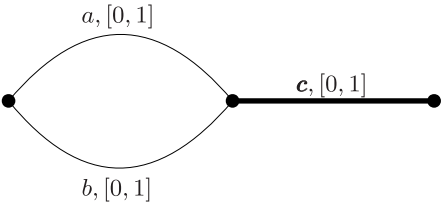


Fig. 2.2. An example of Minimum Spanning Tree with isolated necessarily optimal edge c (in bold)

$$\delta_e(S) = F_e^*(S) - F^*(S) = \min_{X \in \Phi_e} F(X, S) - F^*(S), \tag{2.4}$$

where $\Phi_e \subseteq \Phi$ is the set of all feasible solutions that contain e. The widest interval of possible values of the deviation of e is $\Delta_e = [\underline{\delta}_e, \overline{\delta}_e]$, where

$$\underline{\delta}_e = \min_{S \in \Gamma} \delta_e(S), \quad \overline{\delta}_e = \max_{S \in \Gamma} \delta_e(S).$$

The following fact is obvious:

Proposition 2.7. *An element $e \in E$ is possibly optimal if and only if $\underline{\delta}_e = 0$ and it is necessarily optimal if and only if $\overline{\delta}_e = 0$.*

Consider now the problem of computing Δ_e for a given element e. According to Proposition 2.7, if we could compute this interval in polynomial time, then we would be able to decide efficiently whether e is possibly or necessarily optimal. However, as we will see in the next part of this monograph, this task is hard in general case. Hence there is no hope to provide formulae equivalent to (2.2) and (2.3) to compute $\underline{\delta}_e$ and $\overline{\delta}_e$. We can only prove the following fact:

Proposition 2.8. *Let $e \in E$ be a given element. The value of $\delta_e(S)$ attains minimum (maximum) in an extreme scenario.*

Proof. Let $S \in \Gamma$ be a scenario that minimizes $\delta_e(S)$. Denote by X the solution such that $e \in X$ and $F(X, S) = F_e^*(S)$ and by Y the solution such

that $F(Y,S_X^-)=F^*(S_X^-)$. Thus $\underline{\delta}_e=F(X,S)-F^*(S)\geq F(X,S)-F(Y,S)\geq F(X,S_X^-)-F(Y,S_X^-)=F(X,S_X^-)-F^*(S_X^-)\geq F_e^*(S_X^-)-F^*(S_X^-)=\delta_e(S_X^-)$. Consequently, the extreme scenario S_X^- also minimizes $\delta_e(S)$.

Suppose that $S\in\Gamma$ is a scenario that maximizes $\delta_e(S)$. Denote by Y a solution such that $F(Y,S)=F^*(S)$ and by X a solution such that $e\in X$ and $F(X,S_Y^-)=F_e^*(S_Y^-)$. Thus $\overline{\delta}_e=F_e^*(S)-F^*(S)\leq F(X,S)-F^*(S)=F(X,S)-F(Y,S)\leq F(X,S_Y^-)-F(Y,S_Y^-)\leq F(X,S_Y^-)-F^*(S_Y^-)=\delta_e(S_Y^-)$. Consequently, the extreme scenario S_Y^- also maximizes the value of $\delta_e(S)$. □

We now see that in order to compute the bounds of interval Δ_e we must identify two extreme scenarios that minimize and maximize deviation $\delta_e(S)$. The number of distinct extreme scenarios is finite. It may, however, grow exponentially with n. Contrary to the solutions there is not an easy characterization of the possible and necessary optimality of elements. There is not an equivalent of Proposition 2.3 for elements and the complexity of the optimality evaluation strongly depends on the combinatorial structure of problem $\mathcal{P}$. Let us define the following two decision problems:

Pos $\mathcal{P}$
Input: Problem $\mathcal{P}$ with interval weights, an element $f\in E$.
Question: Is element f possibly optimal?

Nec $\mathcal{P}$
Input: Problem $\mathcal{P}$ with interval weights, an element $f\in E$.
Question: Is element f necessarily optimal?

Both problems Pos $\mathcal{P}$ and Nec $\mathcal{P}$ are far from being trivial even if problem $\mathcal{P}$ is polynomially solvable. It turns out that the general problem Pos $\mathcal{P}$ is strongly NP-complete even if $\mathcal{P}$ is polynomially solvable (in one of the next chapters we will show that it is strongly NP-complete when $\mathcal{P}$ is Shortest Path). However, in this case the general problem Nec $\mathcal{P}$ remains open. It turns out that we can characterize the possible and necessary optimality of a given element in terms of two extreme scenarios if $\mathcal{P}$ is a matroidal problem. This case is discussed in Section 2.2.3. We will explore problems Pos $\mathcal{P}$ and Nec $\mathcal{P}$ in detail while considering particular problems later in this monograph. In the next two sections we show some important relationships between the possibly and necessarily optimal elements and the optimal robust solutions.

2.2.1 Possibly Optimal Elements and the Optimal Robust Solutions

It is clear that element $e\in E$ is possibly optimal if and only if it is a part of a possibly optimal solution. Hence, by Proposition 2.4, a nonpossibly optimal element cannot be a part of an optimal robust solution. Observe also that a nonpossibly optimal element cannot be a part of a worst case alternative X^* of any solution X, since X^* is optimal under S_X^+. Thus removing a nonpossibly optimal element e from E does not change the value of the maximal regret of any

solution from Φ that does not contain e. We have thus established the following result:

Theorem 2.9. *Every optimal robust solution is composed of possibly optimal elements and all nonpossibly optimal elements can be removed from E without violating the optimal robust solution.*

Theorem 2.9 has some important consequences. It states that all nonpossibly optimal elements can be removed from set E without violating the optimal robust solution of the original problem. We have thus established a method of preprocessing an instance of Minmax Regret $\mathcal{P}$. Before applying an algorithm for solving this problem we can remove from set E all nonpossibly optimal elements. This may reduce the size of the problem and, in consequence, it may speed up the calculation of the optimal robust solution. In the next chapters we will show that the number of nonpossibly optimal elements may be quite large and the preprocessing may significantly reduce the problem size.

2.2.2 Necessarily Optimal Elements and the Optimal Robust Solutions

Let us now focus on the necessarily optimal elements. One may conjecture that all necessarily optimal elements are always contained in an optimal robust solution. It is not difficult to see that this conjecture is false. To see this consider an example of Minmax Regret Minimum Spanning Tree shown in Figure 2.1b. Edges a and c are necessarily optimal, but $\{a, c\}$ is not the optimal robust spanning tree. Observe, however, that the interval weights of edges a and c are degenerate and this is precisely the obstacle. We will show that in the absence of degeneracy, there exists an optimal robust solution that contains all necessarily optimal elements. Moreover, even if there are some degenerate intervals, a single necessarily optimal element is always a part of an optimal robust solution. The following theorem characterizes a single necessarily optimal element:

Theorem 2.10. *A necessarily optimal element is a part of an optimal robust solution.*

Proof. Let $e \in E$ be a necessarily optimal element. Assume, by contradiction, that no optimal robust solution contains e. Let us denote by X an optimal robust solution. By Proposition 2.4, X must be possibly optimal and, by Proposition 2.3, X is optimal under scenario S^-_X. From the fact that e is necessarily optimal we conclude that there must be a solution Y such that $e \in Y$ and Y is optimal under scenario S^-_X. Thus we get $F(Y, S^-_X) = F(X, S^-_X)$. Now Proposition 1.4 implies that Y is also the optimal robust solution, which is a contradiction. □

From Theorem 2.10 we conclude that a single necessarily optimal element can always be added to the constructed solution. However, there may exist more necessarily optimal elements in the problem. Unfortunately, in the general case we cannot add all of them to the constructed solution. We show that this is

only the case if there are some degenerate intervals associated with elements. We start by proving the following proposition:

Proposition 2.11. *If all weight intervals are nondegenerate, then for every two distinct solutions X and Y there exists a scenario S such that $F(X,S) \neq F(Y,S)$.*

Proof. Consider scenario S^-_E in which the weights of all elements are at their lower bounds. If $F(X,S^-_E) \neq F(Y,S^-_E)$, then we are done. Otherwise, we can choose element $e \in Y \setminus X$ and consider scenario $S^+_{\{e\}}$. Since interval $\tilde{w}_e$ is nondegenerate, the weight of e under $S^+_{\{e\}}$ is strictly greater than under S^-_E. Hence, $F(X,S^+_{\{e\}}) < F(Y,S^+_{\{e\}})$ and the proof is completed. □

Let us introduce the following characterization of a solution. A solution X is said to be an *unique solution* under scenario S if X is the only optimal solution under S.

Proposition 2.12. *If all weight intervals are nondegenerate, then there exists an optimal robust solution X that is unique under scenario S^-_X.*

Proof. If X_1 is an optimal robust solution, then it is possibly optimal and it is optimal under scenario $S^-_{X_1}$ (see Proposition 2.3). If X_1 is unique under $S^-_{X_1}$, then we are done. If not, then there is another solution X_2 that is optimal under $S^-_{X_1}$, thus $F(X_1,S^-_{X_1}) = F(X_2,S^-_{X_1})$. Proposition 1.4 now implies that X_2 is also the optimal robust solution. Moreover $F(X_1,S) \geq F(X_2,S)$ for all scenarios $S \in \Gamma$. Again solution X_2 is possibly optimal and it must be optimal under $S^-_{X_2}$. We can repeat this argument obtaining a sequence $X_1, X_2, \ldots, X_k$ of optimal robust solutions such that $F(X_1,S) \geq F(X_2,S) \geq \cdots \geq F(X_k,S)$ for all $S \in \Gamma$. No solution in this sequence can be repeated. Indeed, suppose $X_i = X_j$ for some $i \neq j$. From the construction of the sequence it follows that $j \neq i+1$. Now we have $F(X_i,S) \geq F(X_{i+1},S) \geq F(X_j,S) = F(X_i,S)$ for all scenarios S. Thus we have two distinct solutions X_i and X_{i+1} that have the same weight under all scenarios. This is not possible if all the weight intervals are nondegenerate (see Proposition 2.11). Since the number of feasible solutions is finite, the following two cases are possible: we meet a solution X that is unique under S^-_X and we are done, or we enumerate all feasible solutions. In the second case, let $X_{|\Phi|}$ be the last solution enumerated. From Proposition 2.11 we obtain that there is scenario S such that $F(X_{|\Phi|-1},S) > F(X_{|\Phi|},S)$. It holds $F(X_1,S) \geq F(X_2,S) \geq \cdots \geq F(X_{|\Phi|-1},S) > F(X_{|\Phi|},S) \geq F(X_{|\Phi|},S^-_{X_{|\Phi|}})$ and $X_{|\Phi|}$ is the unique solution under $S^-_{X_{|\Phi|}}$. □

We are ready to prove the following theorem:

Theorem 2.13. *If all weight intervals are nondegenerate, then there exists an optimal robust solution that contains all necessarily optimal elements.*

Proof. From Proposition 2.12 it follows that there is an optimal robust solution X that is unique under S^-_X. It is obvious that all necessarily optimal elements must belong to X. □

2.2.3 Matroidal Problems

In this section we show that all possibly and necessarily optimal elements can be detected efficiently if $\mathcal{P}$ has a matroidal structure. Assume that $\mathcal{P}$ is a matroidal combinatorial optimization problem, that is set Φ consists of all bases of a given matroid $(E, \mathcal{I})$. Recall that the elements of $\mathcal{I}$ are called *independent sets* and they fulfill two axioms **(A1)** and **(A2)** presented in Section 1.1.1. The maximal independent sets in $\mathcal{I}$ are called *bases* and they form set Φ. The minimal subsets of E that are not in $\mathcal{I}$ are called *circuits*. The deterministic matroidal problem $\mathcal{P}$ can be solved by means of `Greedy Algorithm` shown in Figure 1.1. Let us introduce the following notations:

- σ specifies a sequence of elements of E;
- $pred(e, \sigma)$ is the set of elements that precede element e in sequence σ;
- $\sigma(S, e)$ denotes a special sequence of elements in which the elements are sorted in nondecreasing order of their weights under scenario S; moreover, if $w_e^S = w_f^S$, $e \neq f$, then element e precedes element f in this sequence;
- B_σ stands for the base (solution) constructed by `Greedy Algorithm` if the sequence of elements in line 1 is specified by σ (see algorithm shown in Figure 1.1); observe that for any $e \in E$, base $B_{\sigma(S,e)}$ is the optimal solution for problem $\mathcal{P}$ under scenario S;
- B is a *maximal independent subset* of A if, $B \subseteq A$, $B \in \mathcal{I}$ and there is no $e \in A \setminus B$ such that $B \cup \{e\} \in \mathcal{I}$.

Proposition 2.14. *Let $e \in E$ be a given element. Suppose that σ is a sequence such that $e \notin B_\sigma$. If ρ is a sequence such that $pred(e, \sigma) \subseteq pred(e, \rho)$, then $e \notin B_\rho$.*

Proof. Let us denote by U the elements of σ chosen and added to B_σ by `Greedy Algorithm` before encountering element e. Similarly, let V denote the elements of ρ chosen and added to B_ρ by `Greedy Algorithm` before encountering element e. Of course, U is a maximal independent subset of $pred(e, \sigma)$ and V is a maximal independent subset of $pred(e, \rho)$. Since $pred(e, \sigma) \subseteq pred(e, \rho)$, $U, V \subseteq pred(e, \rho)$. Assume on the contrary that $e \in B_\rho$.

Consider the case $|U| > |V|$. Both U and V are independent sets, therefore there exists an element $f \in U \setminus V$ such that set $V \cup \{f\}$ is independent (axiom **(A2)**). Obviously, $f \in pred(e, \rho)$, which contradicts the fact that V is the maximal independent subset of $pred(e, \rho)$.

Consider the case $|U| \leq |V|$. This gives $|U| \leq |V| < |V \cup \{e\}|$. Set $V \cup \{e\}$ is independent, which is due to the assumption that $e \in B_\rho$. Since U is an independent set, we can extend U to an independent set U^* by adding elements from $(V \cup \{e\}) \setminus U$ to U so that $|U^*| = |V \cup \{e\}|$ (axiom **(A2)**). From the assumption $e \notin B_\sigma$, we conclude that $U \cup \{e\}$ contains a circuit and in consequence $e \notin U^*$. Hence $|U^*| > |V|$. Again, both U^* and V are independent sets and there exists an element $f \in U^* \setminus V$, $f \neq e$, such that set $V \cup \{f\}$ is independent. Clearly, $f \in pred(e, \rho)$, which contradicts the fact that V is the maximal independent subset of $pred(e, \rho)$. □

Proposition 2.15. *Let $e \in E$ be a given element. Then e is optimal under scenario S if and only if $e \in B_{\sigma(S,e)}$.*

Proof. ($\Rightarrow$) It suffices to show that if $e \notin B_{\sigma(S,e)}$, then e is not optimal under S. To prove this, assume on the contrary that e is optimal under S. Therefore, there is another base B such that $e \in B$ and B is optimal under S. We can represent both bases as the subsequences of elements of E with respect to sequence $\sigma(S,e)$, that is $B_{\sigma(S,e)} = (u_1, \ldots, u_l)$, $B = (v_1, \ldots, v_l)$. Suppose that $u_i = v_i$ for $i = 1, \ldots, k-1$ and $u_k \neq v_k$. Since the number of optimal bases under S is finite, we can choose base B so that $e \in B$ and k is as large as possible. It is clear that $k < l$. Otherwise $B_{\sigma(S,e)} = B$ and in consequence $e \in B_{\sigma(S,e)}$, which contradicts the assumption. Thus the relation between B and $B_{\sigma(S,e)}$ is as follows:

$$\begin{aligned} B_{\sigma(S,e)} &= (u_1 \ \ldots \ u_{k-1} \ \boldsymbol{u_k} \quad \ldots \quad u_l) \\ &= \cdots = \neq \\ B = \quad &(v_1 \ \ldots \ v_{k-1} \ v_k \ \ldots \ \boldsymbol{f} \ \ldots v_l). \end{aligned}$$

Consider element $u_k \in B_{\sigma(S,e)} \setminus B$. There exists an element f of $B \setminus B_{\sigma(S,e)}$ such that $B^* = B \setminus \{f\} \cup \{u_k\}$ is a base. A proof of this fact can be found in [111, Theorem 1.2.3]. It must hold $w_f^S = w_{u_k}^S$. Otherwise ($w_f^S > w_{u_k}^S$) set B^* would be a base with a weight less than the weight of B under S. This would contradict the optimality of B under S. From equality $w_f^S = w_{u_k}^S$, condition $u_k \neq e$ and the construction of sequence $\sigma(S,e)$, we get $f \neq e$. Thus after removing element f and adding u_k to B, base B^* contains e and B^* is optimal under S. This contradicts the maximality of k.

($\Leftarrow$) It is a direct consequence of the fact that $B_{\sigma(S,e)}$ is the optimal base in scenario S. □

We are ready to prove the following result:

Theorem 2.16. *Let $\mathcal{P}$ be a matroidal problem with interval weights and let $e \in E$ be a given element. Then*

$$\underline{\delta}_e = \delta_e(S^-_{\{e\}}), \tag{2.5}$$

$$\overline{\delta}_e = \delta_e(S^+_{\{e\}}). \tag{2.6}$$

Proof. Let S be a scenario that minimizes deviation $\delta_e(S)$. Let Y be the solution in Φ_e of the smallest value of $F(Y,S)$. Note that $e \in Y$. Thus $\underline{\delta}_e = F(Y,S) - F^*(S)$. Let us subtract $\underline{\delta}_e$ from w_e^S in S and denote the resulting realization of the weights by S'. It is clear that solution Y is optimal under S' and, consequently, element e is optimal under S'. Thus, by Proposition 2.15, the greedy algorithm applied to sequence $\sigma(S',e)$ chooses element e. Let us now transform S' into S'' by increasing the weights of all elements $f \neq e$ to $\overline{w}_f$ and by decreasing the weight of e to $\underline{w}_e - \underline{\delta}_e$. The greedy algorithm applied to $\sigma(S'',e)$ also chooses element e. It follows from the fact, that all elements that precede e in $\sigma(S'',e)$ also precede e

in $\sigma(S', e)$ (see Proposition 2.14). Denote by X the solution computed by greedy algorithm applied to sequence $\sigma(S'', e)$. Clearly $e \in X$ and X is optimal under S''. Furthermore, it holds $F(X, S'') = F(X, S^-_{\{e\}}) - \underline{\delta}_e$ and $F^*(S'') \leq F^*(S^-_{\{e\}})$. We obtain

$$F(X, S^-_{\{e\}}) - \underline{\delta}_e - F^*(S^-_{\{e\}}) \leq F(X, S'') - F^*(S'') = 0,$$

which implies

$$\delta(S^-_{\{e\}}) = \min_{Y \in \Phi_e} F(Y, S^-_{\{e\}}) - F^*(S^-_{\{e\}}) \leq F(X, S^-_{\{e\}}) - F^*(S^-_{\{e\}}) \leq \underline{\delta}_e$$

and (2.5) follows. The proof of (2.6) is very similar. □

The following theorem characterizes a possibly optimal element:

Theorem 2.17. *Element $e \in E$ is possibly optimal if and only if e belongs to base $B_{\sigma(S^-_{\{e\}}, e)}$.*

Proof. Element e is possibly optimal if and only if $\underline{\delta}_e = \delta_e(S^-_{\{e\}}) = 0$. In other words e is possibly optimal if and only if it is optimal under scenario $S^-_{\{e\}}$. Consequently, by Proposition 2.15, the element e is possibly optimal if and only if $e \in B_{\sigma(S^-_{\{e\}}, e)}$. □

The following theorem characterizes a necessarily optimal element and its proof is similar to the proof of Theorem 2.17:

Theorem 2.18. *Element $e \in E$ is necessarily optimal if and only if e belongs to base $B_{\sigma(S^+_{\{e\}}, e)}$.*

We have thus shown that the optimality of a given element e in a matroidal problem can be characterized by means of two extreme scenarios $S^-_{\{e\}}$ and $S^+_{\{e\}}$.

```
Detect Pos (Nec)
```

Require: A matroidal problem $\mathcal{P}$ with interval weights, element $f \in E$.
Ensure: **true** if f is possibly (necessarily) optimal and **false** otherwise.

```
1: Order the elements in E with respect to σ(S^-_{f}, f) (σ(S^+_{f}, f))
2: B ← ∅
3: for i ← 1 to n do
4:    if B ∪ {e_i} ∈ I then
5:       B ← B ∪ {e_i}
6:       if e_i = f return true
7:    end if
8: end for
9: return false
```

Fig. 2.3. An algorithm for solving problem Pos (Nec) $\mathcal{P}$ if $\mathcal{P}$ is a matroidal problem

Applying Theorems 2.17 and 2.18 we can design efficient algorithms for checking whether a given element is possibly (necessarily) optimal (see Figure 2.3).

Algorithm `Detect Pos (Nec)`, shown in Figure 2.3, is a slightly modified version of the greedy algorithm shown in Figure 1.1. It returns **true** if and only if the specified element f is added to the constructed optimal base. It is evident that the algorithm runs in $\mathcal{O}(n \log n + nf(n))$ time, where $f(n)$ is the time required for asserting whether $B \cup \{e_i\} \in \mathcal{I}$ (see line 4 of the algorithm). Hence both problems POS $\mathcal{P}$ and NEC $\mathcal{P}$ can be solved in $\mathcal{O}(n \log n + nf(n))$ time. Obviously, executing the algorithm n times we can detect all possibly (necessarily) optimal elements in $\mathcal{O}(n^2 \log n + n^2 f(n))$ time.

Detection of all possibly optimal elements

The detection of all possibly optimal elements can be additionally refined. We prove the following theorem:

Theorem 2.19. *Let B^+ be an optimal base under scenario S_E^+ and assume that $e \notin B^+$. Let $f \neq e$ be an element in the unique circuit $C \subset B^+ \cup \{e\}$ of the maximal value of $\overline{w}_f$. Then e is possibly optimal if and only if $\underline{w}_e \leq \overline{w}_f$.*

Proof. Denote by σ^+ the sequence of elements ordered in nondecreasing order of the weights under scenario S_E^+. This sequence is formed by `Greedy Algorithm` while constructing the optimal solution under S_E^+.

($\Rightarrow$) Suppose, by contradiction, that e is possibly optimal and $\underline{w}_e > \overline{w}_f$. Consider sequence $\sigma(S^-_{\{e\}}, e)$. This sequence is obtained from σ^+ by moving element e zero or more positions to the left because we only decrease the weight of e. Since $\underline{w}_e > \overline{w}_f$ for all $f \in C \setminus \{e\}$, we conclude that $C \setminus \{e\} \subseteq pred(e, \sigma(S^-_{\{e\}}, e))$. All the elements of $C \setminus \{e\}$ were chosen by `Greedy Algorithm` applied to sequence σ^+. Therefore, the same must hold if `Greedy Algorithm` is applied to the sequence $\sigma(S^-_{\{e\}}, e)$. This means that $e \notin B_{\sigma(S^-_{\{e\}}, e)}$ and Theorem 2.17 now yields that e is not possibly optimal, which is a contradiction.

($\Leftarrow$) Suppose, by contradiction, that $\underline{w}_e \leq \overline{w}_f$ and e is not possibly optimal. Hence, by Theorem 2.17, e does not belong to an optimal solution under scenario $S^-_{\{e\}}$. It is easy to see that in this case B^+ must be optimal in scenario $S^-_{\{e\}}$. But now we can construct another solution $B = B^+ \cup \{e\} \setminus \{f\}$, which by our assumption that $\underline{w}_e \leq \overline{w}_f$, has a weight not greater than B^+ in scenario $S^-_{\{e\}}$. Therefore, base B is optimal in $S^-_{\{e\}}$ and element $e \in B$ is possibly optimal which is a contradiction. □

Using Theorem 2.19 we can construct an algorithm for detecting all possibly optimal elements, which is significantly faster than executing n times algorithm `Detect Pos`. This algorithm is shown in Figure 2.4.

First the optimal solution B^+ under scenario S_E^+ is computed by `Greedy Algorithm`. This step requires $\mathcal{O}(n \log n + nf(n))$ time. All elements $e \in B^+$ are then possibly optimal by definition. Now, for every element $e \notin B^+$ the circuit $C \subset B^+ \cup \{e\}$ and the element $f \in C$, $f \neq e$, of the maximal value of $\overline{w}_f$ are

`Detect All Pos`

Require: A matroidal problem $\mathcal{P}$ with interval weights.
Ensure: Array $Pos[1...n]$ such that $Pos[i] =$ **true** if e_i is possibly optimal and $Pos[i] =$ **false** otherwise.
1: **for** $i \leftarrow 1$ **to** n **do** $Pos[i] \leftarrow$ **false**
2: Compute the optimal solution B^+ under scenario S_E^+ using `Greedy Algorithm`
3: **forall** $e_i \in B^+$ **do** $Pos[i] \leftarrow$ **true**
4: **for all** $e_i \notin B^+$ **do**
5: Compute the unique circuit $C \subset B^+ \cup \{e_i\}$
6: $max \leftarrow \max_{f \in C \setminus \{e_i\}} \overline{w}_f$
7: **if** $\underline{w}_{e_i} \leq max$ **then** $Pos[i] \leftarrow$ **true else** $Pos[i] \leftarrow$ **false**
8: **end for**
9: **return** $Pos[1...n]$

Fig. 2.4. The algorithm for detecting all possibly optimal elements in a matroidal combinatorial optimization problem

determined, which requires $nf(n)$ time. Thus all the possibly optimal elements can be detected in $\mathcal{O}(n \log n + nf(n))$ time.

Unfortunately, we are not able here to construct a similar algorithm for detecting all necessarily optimal elements. One must execute n times algorithm `Detect Nec` to perform this task.

2.3 Notes and References

The relationships between the optimality evaluation and the minmax regret approach were first established by Yaman *et al.* [124] and by Karasan *et al.* [70] while considering the Minmax Regret Minimum Spanning Tree and Minmax Regret Shortest Path problems. It must be mentioned that the authors in [124] and [70] used a different terminology. They called a possibly optimal solution (element) *weak* and a necessarily optimal solution (element) *strong*. We prefer to use the notions of possible and necessary optimality since they have a clear possibilistic interpretation and thus may be used in a wider context. The observation that all nonpossibly optimal arcs can be removed from the input graph in Minmax Regret Shortest Path was established by Karasan *et al.* [70]. Similarly, Yaman *et al.* [124] observed that all nonpossibly optimal edges can be removed from graph G in Minmax Regret Minimum Spanning Tree. In this chapter these two results are generalized to all minmax regret combinatorial optimization problems (see also [82]). Moreover, Yaman *et al.* [124] proved that under the assumption that all interval weights are nondegenerate, all necessarily optimal edges can be added to the constructed robust spanning tree. Theorem 2.13 generalizes this result to all problems. Theorem 2.6, which establishes a polynomially solvable class of problems was proved by Averbakh and Lebedev in [17]. Here it is shown as a direct consequence of the fact that every optimal robust solution is possibly optimal. In paper [124] some efficient methods of detecting the possibly and necessarily optimal edges in Minmax Regret

Minimum Spanning Tree were proposed. Being inspired by [124], Kasperski and Zieliński [78] showed that these methods can be generalized to all matroidal problems. Most of the results presented in Section 2.2.3, devoted to matroidal problems, were obtained by Kasperski and Zieliński in [78]. If $\mathcal{P}$ is not a matroidal problem, then evaluating the optimality of a given element may be much more complex. This case will be discussed more deeply in the next chapters of this monograph.

3 Exact Algorithms

In this chapter two general methods for solving the MINMAX REGRET $\mathcal{P}$ problem are presented. The first one, discussed in Section 3.1, is based on a *mixed integer programming formulation* (shortly MIP) which is a standard tool for solving discrete optimization problems. We show that under some additional and general assumptions it is possible to construct a MIP model with linear constraints and linear objective function for MINMAX REGRET $\mathcal{P}$. This model is compact in the sense that it involves a polynomial number of variables and constraints. A considerable number of excellent software packages are available to solve the constructed MIP model. One that is widely regarded is CPLEX, which will be used to test the constructed MIP formulations for particular problems. After constructing a MIP model we can apply the results obtained in the previous chapter to speed up calculations. That is, we can remove from the model all variables that correspond to nonpossibly optimal elements and fix some variables that correspond to the necessarily optimal ones. After relaxing the integrality constraints the MIP formulation may also be used to compute the lower bound on the value of the maximal regret of the optimal robust solution.

The second method, discussed in Section 3.2, is based on a branch and bound technique, which is commonly used to solve hard optimization problems. We present a general framework of the branch and bound algorithm and a general method of calculating the lower bound on the maximal regret of the optimal robust solution. The application of the branch and bound algorithm to particular problems will be shown in the next chapters of this monograph.

3.1 Mixed Integer Programming Formulation

Let us introduce a binary variable $x_i \in \{0,1\}$ for every element $e_i \in E$. This variable will express whether element e_i is a part of the constructed optimal robust solution. A *characteristic vector* of a given subset of elements $A \subseteq E$ is a binary vector $\boldsymbol{x} = (x_i)_{i=1}^n$ such that $x_i = 1$ if and only if $e_i \in A$. Let us associate with the set of feasible solutions Φ a set of binary vectors $ch(\Phi) \subseteq \{0,1\}^n$, which satisfies the following two conditions:

A. Kasperski: Discrete Optimization with Interval Data, STUDFUZZ 228, pp. 31–38, 2008.
springerlink.com

- if $\boldsymbol{x}$ is a characteristic vector of a feasible solution $X \in \Phi$, then $\boldsymbol{x} \in ch(\Phi)$;
- if $\boldsymbol{y}$ is a characteristic vector a subset $Y \subseteq E$ and $\boldsymbol{y} \in ch(\Phi)$, then there exists $X \subseteq Y$ such that $X \in \Phi$.

Hence set $ch(\Phi)$ contains all characteristic vectors of the feasible solutions in Φ. It may also contain a characteristic vector of a subset $Y \subseteq E$ such that $Y \notin \Phi$. However, in this case case Y must contain a feasible solution, that is there must be $X \subseteq Y$ such that $X \in \Phi$.

We will assume that $ch(\Phi)$ can be described by a set of linear constraints of the form

$$ch(\Phi) = \{\boldsymbol{x} \in \{0,1\}^n : \ \boldsymbol{A}[\boldsymbol{x}, \boldsymbol{x'}]^T = \boldsymbol{b}\},$$

where $\boldsymbol{x'}$ is a vector of auxiliary variables (if required), $\boldsymbol{A}$ is a matrix and $\boldsymbol{b}$ is a vector of fixed coefficients. We also allow the signs $\leq$ or $\geq$ in some constraints instead of equality. We will also assume that matrix $\boldsymbol{A}$ is *totally unimodular*. Recall that a matrix $\boldsymbol{A}$ is totally unimodular if every square submatrix of $\boldsymbol{A}$ has determinant 0, +1 or -1. The assumption that matrix $\boldsymbol{A}$ is totally unimodular restricts the class of considered problems. However, if problem $\mathcal{P}$ is polynomially solvable, then it can be often formulated as a linear programming problem with totally unimodular constraint matrix. As we will see in the next chapters, this is the case for all min-max regret combinatorial optimization problems considered in this monograph.

Assume that $[\underline{w}_i, \overline{w}_i]$ is the interval weight of element $e_i \in E$. Define function

$$\phi(\boldsymbol{x}, \boldsymbol{y}) = \sum_{i=1}^{n} (\overline{w}_i x_i + \underline{w}_i (1 - x_i)) y_i.$$

Proposition 3.1. *The* MINMAX REGRET $\mathcal{P}$ *problem can be expressed as the following mathematical programming problem:*

$$\min_{\boldsymbol{x} \in ch(\Phi)} \left\{ \sum_{i=1}^{n} \overline{w}_i x_i - \min_{\boldsymbol{y} \in ch(\Phi)} \phi(\boldsymbol{x}, \boldsymbol{y}) \right\}. \tag{3.1}$$

Proof. Let $\boldsymbol{x} = (x_i)_{i=1}^n$ be an optimal solution to (3.1). Suppose that $\boldsymbol{x}$ is a characteristic vector of a feasible solution $X \in \Phi$. Then $F(X, S_X^+) = \sum_{i=1}^n \overline{w}_i x_i$ and $F^*(S_X^+) = \min_{\boldsymbol{y} \in ch(\Phi)} \phi(\boldsymbol{x}, \boldsymbol{y})$, which follows from the structure of the worst case scenario S_X^+. Hence

$$Z(X) = \sum_{i=1}^{n} \overline{w}_i x_i - \min_{\boldsymbol{y} \in ch(\Phi)} \phi(\boldsymbol{x}, \boldsymbol{y}).$$

Since $ch(\Phi)$ contains the characteristic vectors of all feasible solutions, X must be the optimal robust solution. On the other hand, if $\boldsymbol{x}$ is a characteristic vector of $X \subseteq E$ such that $X \notin \Phi$, then from the definition of $ch(\Phi)$ there is $X_1 \in \Phi$ such that $X_1 \subset X$. We can apply Proposition 1.3 obtaining

$$Z(X_1) \leq F(X, S_X^+) - F^*(S_X^+) = \sum_{i=1}^{n} \overline{w}_i x_i - \min_{\boldsymbol{y} \in ch(\Phi)} \phi(\boldsymbol{x}, \boldsymbol{y}), \tag{3.2}$$

thus X_1 is the optimal robust solution. □

Problem (3.1) is not linear. However, we will show that if matrix $\boldsymbol{A}$ is totally unimodular, then it can be transformed into a linear problem with binary variables. Let us fix $\boldsymbol{x}$ and consider subproblem

$$\min_{\boldsymbol{y}\in ch(\Phi)} \phi(\boldsymbol{x},\boldsymbol{y}),$$

which can be alternatively rewritten as follows:

$$\begin{aligned} &\min \sum_{i=1}^{n} (\overline{w}_i x_i + \underline{w}_i(1-x_i))y_i \\ &\boldsymbol{A}[\boldsymbol{y},\boldsymbol{y'}]^T = \boldsymbol{b} \\ &y_i \in \{0,1\} \qquad\qquad \text{for } i = 1,\ldots,n \end{aligned} \tag{3.3}$$

From the total unimodularity of $\boldsymbol{A}$, we can replace constraints $y_i \in \{0,1\}$ with $0 \le y_i \le 1$, obtaining the following problem that has the same minimal value of the objective function:

$$\begin{aligned} &\min \sum_{i=1}^{n} (\overline{w}_i x_i + \underline{w}_i(1-x_i))y_i \\ &\boldsymbol{A}[\boldsymbol{y},\boldsymbol{y'}]^T = \boldsymbol{b} \\ &0 \le y_i \le 1 \qquad\qquad \text{for } i = 1,\ldots,n \end{aligned} \tag{3.4}$$

For a fixed $\boldsymbol{x}$ problem (3.4) is a linear one. We can construct a dual problem to (3.4) with vector of dual variables $\boldsymbol{\lambda}$ associated with every constraint of (3.4). Denote by $\phi^*(\boldsymbol{x},\boldsymbol{\lambda})$ the objective of the dual and by $\Lambda(\Phi)$ the set of feasible dual vectors. It is easy to see that the dual problem is linear with respect to both $\boldsymbol{x}$ and $\boldsymbol{y}$. Moreover, the well known strong duality theorem implies:

$$\min_{\boldsymbol{y}\in ch(\Phi)} \phi(\boldsymbol{x},\boldsymbol{y}) = \max_{\boldsymbol{\lambda}\in\Lambda(\Phi)} \phi^*(\boldsymbol{x},\boldsymbol{\lambda}). \tag{3.5}$$

From (3.1) and (3.5) we obtain:

$$\min_{\boldsymbol{x}\in ch(\Phi)} \left\{ \sum_{i=1}^{n} \overline{w}_i x_i - \min_{\boldsymbol{y}\in ch(\Phi)} \phi(\boldsymbol{x},\boldsymbol{y}) \right\} = \min_{\boldsymbol{x}\in ch(\Phi)} \left\{ \sum_{i=1}^{n} \overline{w}_i x_i - \max_{\boldsymbol{\lambda}\in\Lambda(\Phi)} \phi^*(\boldsymbol{x},\boldsymbol{\lambda}) \right\} =$$

$$= \min_{\boldsymbol{x}\in ch(\Phi)} \left\{ \sum_{i=1}^{n} \overline{w}_i x_i + \min_{\boldsymbol{\lambda}\in\Lambda(\Phi)} -\phi^*(\boldsymbol{x},\boldsymbol{\lambda}) \right\},$$

which is equivalent to:

$$\min_{\boldsymbol{x}\in ch(\Phi)} \min_{\boldsymbol{\lambda}\in\Lambda(\Phi)} \left\{ \sum_{i=1}^{n} \overline{w}_i x_i - \phi^*(\boldsymbol{x},\boldsymbol{\lambda}) \right\}. \tag{3.6}$$

Problem (3.6) is a mixed integer linear programming problem with binary variables (x_i) and real variables (λ_i). It can be solved by means of a standard

software. Moreover, relaxing constraints $x_i \in \{0,1\}$ with $0 \leq x_i \leq 1$ we obtain a polynomially solvable linear programming problem. Solving this problem we get a lower bound for the maximal regret of the optimal robust solution.

In the previous section we have shown some relationships between the possible and necessary optimality of elements and the minmax regret approach. These relationships can now be applied to preprocessing an instance of the problem before solving the MIP model. Namely, every nonpossibly optimal element $e_i \in E$ cannot be a part of any optimal robust solution and we can remove variable x_i that corresponds to e_i from the model. Similarly, if we detect a necessarily optimal element $e_i \in E$ we can fix $x_i = 1$. This can only be done for one element in general case. But, if all the weight intervals are nondegenerate, then we can set $x_i = 1$ for all necessarily optimal elements $e_i \in E$. It has been reported in literature that such a preprocessing of the MIP model may significantly speed up the calculations.

Example 3.2. In order to illustrate the presented framework we construct a MIP model for Minmax Regret Minimum Selecting Items. Set $ch(\Phi)$ in this problem can be described by the following constraints:

$$\begin{aligned} & x_1 + \cdots + x_n = p \\ & x_i \in \{0,1\} \qquad \text{for } i = 1, \ldots, n \end{aligned} \tag{3.7}$$

In this case the description of $ch(\Phi)$ is very simple. Every characteristic vector of a feasible solution fulfills (3.7) and, conversely, every feasible solution to (3.7) is a characteristic vector of a feasible solution. There are no auxiliary variables involved. Obviously, the constraints matrix $\boldsymbol{A} = [1, 1, \ldots, 1]$ is totally unimodular and our assumption is thus fulfilled. The relaxed subproblem $\min_{\boldsymbol{y} \in ch(\Phi)} \phi(\boldsymbol{x}, \boldsymbol{y})$ is the following one:

$$\begin{aligned} & \min \sum_{i=1}^{n} (\overline{w}_i x_i + \underline{w}_i (1 - x_i)) y_i \\ & y_1 + \cdots + y_n = p \\ & 0 \leq y_i \leq 1 \qquad \text{for } i = 1, \ldots, n \end{aligned}$$

and the dual $\max_{\boldsymbol{\lambda} \in \Lambda(\Phi)} \phi^*(\boldsymbol{x}, \boldsymbol{\lambda})$ is of the following form:

$$\begin{aligned} & \max \lambda p - \lambda_1 - \ldots \lambda_n \\ & \lambda - \lambda_i \leq \overline{w}_i x_i + \underline{w}_i (1 - x_i) \text{ for } i = 1, \ldots, n \\ & \lambda_i \geq 0 \qquad \text{for } i = 1, \ldots, n \end{aligned}$$

Hence the final MIP is

$$\begin{aligned} & \min \sum_{i=1}^{n} \overline{w}_i x_i - \lambda p + \lambda_1 + \cdots + \lambda_n \\ & x_1 + \cdots + x_n = p \end{aligned}$$

$$\lambda - \lambda_i \leq \overline{w}_i x_i + \underline{w}_i(1 - x_i) \text{ for } i = 1, \ldots, n$$
$$\lambda_i \geq 0 \qquad \text{for } i = 1, \ldots, n$$
$$x_i \in \{0, 1\} \qquad \text{for } i = 1, \ldots, n$$

Solving the above MIP we get an optimal robust solution to MINMAX REGRET MINIMUM SELECTING ITEMS. We will show in Chapter 5 that this particular model can be solved in polynomial time. □

3.2 Branch and Bound Algorithm

A *branch and bound algorithm* is one of the most widely used tools for solving NP-hard combinatorial optimization problems. It searches the complete solutions space using the bounds for the value of the objective function combined with the value of the current best solution. The search process is represented as a dynamically generated rooted tree whose leaves describe some unexplored subsets of the solutions space. Initially, the root of this tree represents the whole solution space. At each iteration a certain unexplored node is selected and the subspace represented by this node is divided into two or more subspaces. The procedure of dividing a node is called *branching*. For every new node d a *lower bound* $LB(d)$ on the value of the objective function in the subspace $\Phi_d \subseteq \Phi$ represented by d is computed. The value of the lower bound is then compared to the value of the *upper bound*, which equals the objective value of the current best solution. If the upper bound is less than or equal to the lower bound $LB(d)$, then the subspace represented by d cannot contain a better solution and it can be discarded.

In this section we show a general framework of the branch and bound algorithm for MINMAX REGRET $\mathcal{P}$. We start by introducing the following notations:

- Every node d of the search tree is represented by structure $< Q, R >$, where $Q \subseteq E$ is a subset of *selected elements* and $R \subseteq E$ is a subset of *rejected elements*. Thus structure $< Q, R >$ represents the following subset of the feasible solution:

$$\Phi_{<Q,R>} = \{X \in \Phi : \ Q \subseteq X, X \cap R = \emptyset\}.$$

  Clearly, structure $< \emptyset, \emptyset >$ represents the whole solution space Φ and it is the initial node of the search tree.
- $LB(< Q, R >)$ denotes a lower bound on the value of the maximal regret of the solutions from set $\Phi_{<Q,R>}$.
- $X_{<Q,R>}$ denotes a feasible solution deriven from $\Phi_{<Q,R>}$.

A general scheme of the branch and bound algorithm is shown in Figure 3.1.

Initially, the search tree contains the single node $< \emptyset, \emptyset >$ representing the whole solution space. The current best solution is a feasible solution deriven from

```
Branch and Bound
```

Require: The MINMAX REGRET $\mathcal{P}$ problem with scenario set Γ.
Ensure: The optimal robust solution X_{opt}.

1: $N \leftarrow \{< \emptyset, \emptyset >\}$, $X_{opt} \leftarrow X_{<\emptyset,\emptyset>}$, $UB \leftarrow Z(X_{opt})$
2: **while** $N \neq \emptyset$ **do**
3: Select node $< Q, R > \in N$ of the smallest value of $LB(< Q, R >)$ and remove $< Q, R >$ from N
4: Select $e \in E \setminus \{Q \cup R\}$
5: Create two nodes $d_1 =< Q \cup \{e\}, R >$ and $d_2 =< Q, R \cup \{e\} >$
6: Apply some reduction rules to d_1 and d_2 obtaining d_1' and d_2'
7: Derive $X_{d_1'}$ from d_1' and $X_{d_2'}$ from d_2'
8: **if** $Z(X_{d_1'}) < UB$ **then**
9: $X_{opt} \leftarrow X_{d_1'}$, $UB \leftarrow Z(X_{opt})$
10: **end if**
11: **if** $Z(X_{d_2'}) < UB$ **then**
12: $X_{opt} \leftarrow X_{d_2'}$, $UB \leftarrow Z(X_{opt})$
13: **end if**
14: **if** $LB(d_1') < UB$ **then** $N \leftarrow N \cup \{d_1'\}$
15: **if** $LB(d_2') < UB$ **then** $N \leftarrow N \cup \{d_2'\}$
16: **end while**
17: **return** X_{opt}

Fig. 3.1. The branch and bound algorithm for MINMAX REGRET $\mathcal{P}$

$< \emptyset, \emptyset >$ and its maximal regret is the upper bound on the maximal regret of the optimal robust solution. In line 3 a node $d =< Q, R >$ of the minimal value of the lower bound is selected to be branched. The branching consists in selecting an element e whose status is unknown, that is the one from $E \setminus \{R \cup Q\}$, and creating two subproblems in which element e is respectively selected and rejected.

In line 6 the obtained two subproblems can be additionally modified by applying some *reduction rules*. There are two types of reduction rules. The first consists in pruning infeasible solutions. Some elements from $E \setminus \{R \cup Q\}$ cannot form a feasible solution with elements from Q and they can be added to R. The second rule is based on the concepts of possible and necessary optimal elements. For instance, the nonpossibly optimal elements are always rejected. The reduction rules allow reduction in the size of the search tree, which speeds up calculations. We will show some examples of the reduction rules while considering the particular problems later in this monograph.

In lines 7 - 13 of the algorithm some candidate solutions are deriven from both subproblems and the current best solution together with the current upper bound are possibly updated. In order to derive a feasible solution some heuristics that will be designed in Chapter 4 can be applied. Finally, in lines 14 - 15 the subproblems whose lower bound is greater than or equal to the maximal regret of the current best solution are discarded.

We now show a general method of obtaining the value of a lower bound $LB(< Q, R >)$ for a given node $< Q, R >$. Let us define

$$F^*_{<Q,R>}(S) = \min_{X \in \Phi_{<Q,R>}} F(X, S).$$

The following theorem establishes a lower bound $LB(< Q, R >)$ on the maximal regret of a solution from $\Phi_{<Q,R>}$:

Theorem 3.3. *For every $X \in \Phi_{<Q,R>}$ it holds*

$$Z(X) \geq F^*_{<Q,R>}(S^+_E) - F^*(S^+_{E \setminus R}).$$

Proof. Let X be a solution from set $\Phi_{<Q,R>}$. It holds $F(X, S^+_X) = F(X, S^+_E)$. From the fact that $R \cap X = \emptyset$ we have $F^*(S^+_X) \leq F^*(S^+_{E \setminus R})$. Hence

$$Z(X) = F(X, S^+_X) - F^*(S^+_X) \geq F(X, S^+_E) - F^*(S^+_{E \setminus R}). \tag{3.8}$$

Since $X \in \Phi_{<Q,R>}$, it holds $F^*_{<Q,R>}(S^+_E) \leq F(X, S^+_E)$, which together with (3.8) yields $Z(X) \geq F^*_{<Q,R>}(S^+_E) - F^*(S^+_{E \setminus R})$. □

Computing the lower bound given by Theorem 3.3 requires solving two deterministic problems. In the first problem we need to compute the value of $F^*(S^+_{E \setminus R})$. This can be performed in polynomial time if the deterministic problem $\mathcal{P}$ is polynomially solvable. Solving the second problem, that is computing the value of $F^*_{<Q,R>}(S^+_E)$, may be more tricky and it depends on the particular problem to which the algorithm is applied.

In order to apply the branch and bound algorithm to a particular problem we must fill some of its details. We must specify a method of branching, that is choosing an element $e \in E \setminus (Q \cup R)$ to be added to Q and R. It is well known that a good branching method may significantly improve the performance of the algorithm. We must also specify reduction rules and a method of deriving a feasible solution from set $\Phi_{<Q,R>}$. We will show all these details while considering the particular problems in the next chapters of this monograph.

3.3 Notes and References

A mixed integer programming model for Minmax Regret $\mathcal{P}$, based on the idea shown in Section 3.1, was first constructed for Minmax Regret Shortest Path by Karasan *et al.* [70]. Using a similar idea the MIP models for Minmax Regret Minimum Spanning Tree was constructed by Yaman *et al.* [124] and for Minmax Regret Minimum Assignment by Kasperski and Zieliński [74]. All these models will be shown in the next chapters of this monograph. The preprocessing performed before solving the MIP model, based on the idea of possibly and necessarily optimal elements, was applied in [70] and [124]. A good introduction to linear programming can be found for instance in books by Schrijver [116] or Bazaraa *et al.* [21].

The general framework of the branch and bound algorithm, presented in Section 3.2, shares some common elements with the branch and bound algorithms constructed for Minmax Regret Minimum Spanning Tree by Aron and van Hentenryck [12] and Montemanni and Gambardella [104] and for Minmax Regret Shortest Path by Montemanni *et al.* [103]. In particular, the algorithms designed in [12, 103, 104] use the lower bound shown in Theorem 3.3. We will fill some details of the branch and bound algorithm and explore its efficiency while considering the particular problems in the next chapters.

4 Approximation Algorithms

One of the methods of dealing with hard optimization problems is applying approximation algorithms. An *approximation algorithm* is a suboptimal procedure, which provably works fast and provably returns a feasible solution of a high quality. Denote by OPT the maximal regret of the optimal robust solution in Minmax Regret $\mathcal{P}$. Suppose that we have an algorithm $\mathtt{A}$ for the problem that outputs a feasible solution $Y \in \Phi$. This algorithm runs in polynomial time and for all instances of the problem it returns a solution Y such that $Z(Y) \leq kOPT$, where $k > 1$ is a fixed constant. Thus the algorithm $\mathtt{A}$ always returns a solution that is at most k-times worse than optimum. The constant k is called a *performance ratio* of algorithm $\mathtt{A}$ and $\mathtt{A}$ is called a *k-approximation algorithm* for Minmax Regret $\mathcal{P}$.

It is sometimes possible to obtain a stronger approximation result. Let ϵ be a given number from interval $(0, 1)$. A family of $(1+\epsilon)$-approximation algorithms $\mathtt{A}_\epsilon$, such that every $\mathtt{A}_\epsilon$ runs in polynomial time for fixed ϵ, is called a *polynomial time approximation scheme* (shortly PTAS). If the running time of every $\mathtt{A}_\epsilon$ is polynomial both in the input size and the value of $1/\epsilon$, then the family of $\mathtt{A}_\epsilon$ is called a *fully polynomial time approximation scheme* (shortly FPTAS). If P$\neq$NP, then the existence of an FPTAS is the best that can be expected if the optimization problem is NP-hard.

In Section 4.1 we show that under the assumption that problem $\mathcal{P}$ is polynomially solvable, it is possible to design a 2-approximation algorithm for Minmax Regret $\mathcal{P}$. This result is general and, in the simplest case, the approximation algorithm outputs an optimal solution for problem $\mathcal{P}$ under a particular scenario. In Section 4.1 we propose some extensions of this simple 2-approximation algorithm. In Section 4.2 we show that under some additional assumptions it is possible to construct an FPTAS for some special cases of Minmax Regret $\mathcal{P}$.

4.1 2-Approximation Algorithms

In this section, we construct a 2-approximation algorithm for Minmax Regret $\mathcal{P}$ under the assumption that the underlying deterministic problem $\mathcal{P}$

A. Kasperski: Discrete Optimization with Interval Data, STUDFUZZ 228, pp. 39–50, 2008.
springerlink.com

is polynomially solvable. In consequence, we obtain an efficient approximation algorithm for all particular minmax regret combinatorial optimization problems considered in this monograph. We then propose some extensions of this algorithm, which also have the worst case performance ratio of 2, but which may be expected to perform better in the average case.

4.1.1 Algorithm for Midpoint Scenario

We construct now a simple 2-approximation algorithm for MINMAX REGRET $\mathcal{P}$, which is surprisingly general. It runs in polynomial time if problem $\mathcal{P}$ is polynomially solvable and returns solution X such that $Z(X) \leq 2OPT$. The idea of this algorithm is to solve problem $\mathcal{P}$ for the *midpoint scenario* S, that is the one in which $w_e^S = \frac{1}{2}(\underline{w}_e + \overline{w}_e)$ for all $e \in E$. Assume that we have an algorithm, denoted by `AlgOpt`(S), that outputs an optimal solution for problem $\mathcal{P}$ under a fixed scenario S. The approximation algorithm, denoted by `Algorithm AM`, is presented in Figure 4.1.

`Algorithm AM`
Require: Problem MINMAX REGRET $\mathcal{P}$.
Ensure: A feasible solution Y such that $Z(Y) \leq 2OPT$.
1: **for all** $e \in E$ **do**
2: $\quad w_e^S \leftarrow \frac{1}{2}(\underline{w}_e + \overline{w}_e)$
3: **end for**
4: $Y \leftarrow$`AlgOpt`(S)
5: **return** Y

Fig. 4.1. A 2-approximation algorithm for MINMAX REGRET $\mathcal{P}$

We now characterize the solution computed by `Algorithm AM`.

Proposition 4.1. *Let $X \in \Phi$ and $Y \in \Phi$ be two feasible solutions. Then the following inequality holds:*

$$Z(X) \geq \sum_{e \in X \setminus Y} \overline{w}_e - \sum_{e \in Y \setminus X} \underline{w}_e. \tag{4.1}$$

Proof. From the definition of the maximal regret we obtain:

$$Z(X) = F(X, S_X^+) - F^*(S_X^+) \geq F(X, S_X^+) - F(Y, S_X^+). \tag{4.2}$$

It is easily seen that

$$F(X, S_X^+) - F(Y, S_X^+) = \sum_{e \in X \setminus Y} \overline{w}_e - \sum_{e \in Y \setminus X} \underline{w}_e,$$

which, together with (4.2), imply (4.1). □

Proposition 4.2. *Let $X \in \Phi$ and $Y \in \Phi$ be two feasible solutions. Then the following inequality holds:*

$$Z(Y) \leq Z(X) + \sum_{e \in Y \setminus X} \overline{w}_e - \sum_{e \in X \setminus Y} \underline{w}_e. \tag{4.3}$$

Proof. The following equality is easy to verify:

$$F(Y, S_Y^+) = F(X, S_X^+) + \sum_{e \in Y \setminus X} \overline{w}_e - \sum_{e \in X \setminus Y} \overline{w}_e. \tag{4.4}$$

We now show that

$$F^*(S_Y^+) \geq F^*(S_X^+) - \sum_{e \in X \setminus Y} (\overline{w}_e - \underline{w}_e). \tag{4.5}$$

Suppose that inequality (4.5) is false. Let Y^* be a feasible solution such that $F(Y^*, S_Y^+) = F^*(S_Y^+)$. We thus get

$$F^*(S_X^+) > F(Y^*, S_Y^+) + \sum_{e \in X \setminus Y} (\overline{w}_e - \underline{w}_e) \geq F(Y^*, S_{X \cup Y}^+) \geq F(Y^*, S_X^+),$$

which contradicts the definition of $F^*(S_X^+)$. Now subtracting (4.5) from (4.4) yields (4.3). □

Proposition 4.3. *Let Y be the solution returned by* `Algorithm AM`*. Then for every $X \in \Phi$ it holds $Z(Y) \leq 2Z(X)$.*

Proof. Since Y is the solution constructed by `Algorithm AM`, it fulfills inequality

$$\frac{1}{2} \sum_{e \in Y} (\underline{w}_e + \overline{w}_e) \leq \frac{1}{2} \sum_{e \in X} (\underline{w}_e + \overline{w}_e).$$

Hence

$$\sum_{e \in Y \setminus X} (\underline{w}_e + \overline{w}_e) \leq \sum_{e \in X \setminus Y} (\underline{w}_e + \overline{w}_e),$$

which is equivalent to:

$$\sum_{e \in X \setminus Y} \overline{w}_e - \sum_{e \in Y \setminus X} \underline{w}_e \geq \sum_{e \in Y \setminus X} \overline{w}_e - \sum_{e \in X \setminus Y} \underline{w}_e. \tag{4.6}$$

Using inequality (4.3) we get:

$$Z(Y) \leq Z(X) + \sum_{e \in Y \setminus X} \overline{w}_e - \sum_{e \in X \setminus Y} \underline{w}_e. \tag{4.7}$$

Inequality (4.1) together with (4.6) yield:

$$Z(X) \geq \sum_{e \in X \setminus Y} \overline{w}_e - \sum_{e \in Y \setminus X} \underline{w}_e \geq \sum_{e \in Y \setminus X} \overline{w}_e - \sum_{e \in X \setminus Y} \underline{w}_e. \tag{4.8}$$

Now conditions (4.7) and (4.8) imply $Z(Y) \leq 2Z(X)$ which completes the proof. □

If X is an optimal robust solution, that is $Z(X) = OPT$, then it holds $Z(Y) \leq 2OPT$. Hence the worst case ratio of `Algorithm AM` is at most 2. In order to show that this bound is tight consider an example of Minmax Regret Shortest Path presented in Figure 4.2.

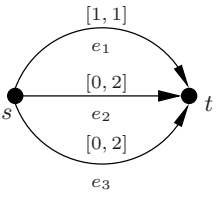


Fig. 4.2. An example of Minmax Regret Shortest Path for which `Algorithm AM` achieves the worst case ratio of 2

Set Φ in this problem consists of three paths $\Phi = \{\{e_1\}, \{e_2\}, \{e_3\}\}$ with maximal regrets as follows: $Z(\{e_1\}) = 1$, $Z(\{e_2\}) = Z(\{e_3\}) = 2$. It is easy to check that `Algorithm AM` may return each of these paths. The worst ones are $\{e_2\}$ or $\{e_3\}$ and they give the bound of 2. Proposition 4.3 and the presented example lead to the following theorem:

Theorem 4.4. *If problem $\mathcal{P}$ is polynomially solvable, then* `Algorithm AM` *is a 2-approximation algorithm for* Minmax Regret $\mathcal{P}$.

Proposition 4.3 is still true even if problem $\mathcal{P}$ is NP-hard. In this case `Algorithm AM` can also be applied and it still has the worst case ratio of 2. However, it may not run in polynomial time which is due to the fact that the optimal solution to problem $\mathcal{P}$ under the midpoint scenario S is hard to compute.

We show now an additional consequence of Proposition 4.3. Let us recall from Section 2.1 that a necessarily optimal solution is the solution with the maximal regret equal to 0. It follows from Proposition 4.3 that if there exists a necessarily optimal solution X, with $Z(X) = 0$, then `Algorithm AM` must return solution Y such that $Z(Y) \leq 2Z(X) = 0$. Hence this algorithm detects a necessarily optimal solution if it exists. We have thus established the following result:

Theorem 4.5. `Algorithm AM` *returns a necessarily optimal solution if such a solution exists.*

So, if problem $\mathcal{P}$ is polynomially solvable, then a necessarily optimal solution can be detected in polynomial time as well.

4.1.2 Algorithm for Midpoint and Upper Scenarios

In this section we propose a simple extension of `Algorithm AM`. We do not improve the worst case ratio of 2. However, the computational experiments have shown that it is possible to improve the average quality of the solutions returned by `Algorithm AM` with a small additional computational effort. The results of these experiments will be presented later in this monograph. Consider an algorithm, denoted by `Algorithm AMU`, shown in Figure 4.3.

`Algorithm AMU`

Require: Problem Minmax Regret $\mathcal{P}$.
Ensure: A feasible solution Y such that $Z(Y) \leq 2OPT$.
1: **for all** $e \in E$ **do**
2: $\quad w_e^{S_1} \leftarrow \frac{1}{2}(\underline{w}_e + \overline{w}_e)$
3: $\quad w_e^{S_2} \leftarrow \overline{w}_e$
4: **end for**
5: $Y_1 \leftarrow$`AlgOpt`(S_1)
6: $Y_2 \leftarrow$ `AlgOpt`(S_2)
7: **if** $Z(Y_1) < Z(Y_2)$ **then return** Y_1 **else return** Y_2

Fig. 4.3. A refinement of `Algorithm AM`

`Algorithm AMU` solves problem $\mathcal{P}$ for two scenarios: the midpoint scenario and the scenario in which all the weights are set to their upper bounds. It returns then the better of these two solutions. It is obvious that `Algorithm AMU` has the worst case performance ratio at most 2 and it runs in polynomial time if problem $\mathcal{P}$ is polynomially solvable. We show now some sample instances of Minmax Regret Shortest Path and Minmax Regret Minimum Spanning Tree for which the bound of 2 is attained. It is not difficult to give similar examples for Minmax Regret Minimum Assignment and Minmax Regret Minimum Cut. Therefore, the theoretical worst case performance of `Algorithm AMU` is the same as `Algorithm AM`.

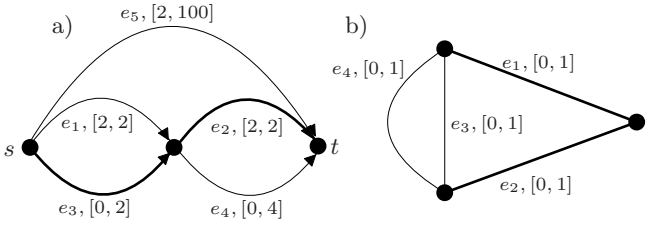


Fig. 4.4. Examples of a) Minmax Regret Shortest Path and b) Minmax Regret Minimum Spanning Tree for which `Algorithm AMU` achieves a ratio of 2. The optimal robust solutions are shown in bold.

Consider an instance of Minmax Regret Shortest Path shown in Figure 4.4a. There are five paths from s to t in the sample graph. The algorithm may compute $Y_2 = \{e_1, e_2\}$ with $Z(Y_2) = 4$ and $Y_1 = \{e_3, e_4\}$ with $Z(Y_1) = 4$. Thus it returns a path with the maximal regret equal to 4 while there is path $X = \{e_2, e_3\}$ with $Z(X) = 2$. The graph shown in Figure 4.4a has a special series-parallel topology (see Section 7.1 for description of this class of graphs). Hence `Algorithm AMU` achieves a ratio of 2 even for series-parallel graphs.

Consider an instance of Minmax Regret Minimum Spanning Tree shown in Figure 4.4b. Since all edges have the same uncertainty intervals, the algorithm may return any spanning tree of G, say $\{e_2, e_3\}$ with the maximal regret equal

to 2 while spanning tree $\{e_1, e_2\}$ has the maximal regret equal to 1. The graph shown in Figure 4.4b is a multigraph since there is more than one edge joining some two nodes. In Section 6.5 we present a more complicated example of MINMAX REGRET MINIMUM SPANNING TREE for which `Algorithm AMU` also attains the worst case ratio of 2 and the input graph does not contain multiedges.

4.1.3 Algorithm Enumerate

We construct now another approximation algorithm for problem MINMAX REGRET $\mathcal{P}$, which is also based on `Algorithm AM`. We use an idea proposed by Montemanni and Gambardella in [102], which was applied to MINMAX REGRET SHORTEST PATH. This idea can be easily generalized to any MINMAX REGRET $\mathcal{P}$ problem. Let us choose any scenario $S \in \Gamma$ and consider the sequence of feasible solutions $X_1, X_2, \ldots, X_{|\Phi|}$ such that

$$F(X_1, S) \leq F(X_2, S) \leq \cdots \leq F(X_{|\Phi|}, S). \tag{4.9}$$

Hence the solutions are ordered with respect to nondecreasing total weights under S. Observe that solution X_1 is optimal under S. Such an enumeration of the feasible solutions is not a trivial task but it can be done for all problems considered in this monograph (see notes and references at the end of this chapter). The algorithms that enumerate the feasible solutions usually efficiently construct the next solution X_{i+1} using the fact that solutions $X_1, \ldots, X_i$ have been already enumerated. Obviously, enumerating all solutions may require exponential time. However, we hope that only a small part of set Φ must be enumerated to meet a good solution. We prove the following two propositions:

Proposition 4.6. *For every scenario $S \in \Gamma$ and for any $k \in \{1, \ldots, |\Phi|\}$ it holds*

$$Z(X_k) \geq F(X_k, S) - F(X_1, S).$$

Proof. From the definition of the maximal regret we get $Z(X_k) \geq F(X_k, S) - F^*(S)$ for every particular scenario S. But, $F^*(S) = F(X_1, S)$, which completes the proof. □

Proposition 4.7. *Let X be a solution from set $\{X_1, \ldots, X_k\}$ of the minimal value of the maximal regret, where $k \in \{1, \ldots, |\Phi|\}$ and let S be a given scenario. If*

$$Z(X) \leq F(X_k, S) - F(X_1, S),$$

then X is the optimal robust solution.

Proof. Consider a solution X_j such that $j > k$. By Proposition 4.6 and inequalities (4.9) it holds $Z(X_j) \geq F(X_j, S) - F(X_1, S) \geq F(X_k, S) - F(X_1, S) \geq Z(X)$. Thus all solutions from set $\{X_j, \ldots, X_{|\Phi|}\}$ have the maximal regrets not less than $Z(X)$. Hence the optimal robust solution is one of the $X_1, \ldots, X_k$ and this is precisely X. □

```
Algorithm Enumerate
Require: Problem MINMAX REGRET P, K > 0, scenario S ∈ Γ.
Ensure: A feasible solution Y.
 1: i ← 1, Y ← X_1, Z* ← Z(X_1)
 2: while i ≤ K do
 3:    i ← i + 1
 4:    if Z(X_i) < Z* then
 5:       Y ← X_i
 6:       Z* ← Z(X_i)
 7:    end if
 8:    if F(X_i, S) − F(X_1, S) ≥ Z* then
 9:       return Y {Y is the optimal robust solution}
10:    end if
11: end while
12: return Y {Y is a heuristic solution}
```

Fig. 4.5. An algorithm that enumerates the feasible solutions

Assume now that scenario S is fixed and solutions from Φ are enumerated according to (4.9). Consider `Algorithm Enumerate` presented in Figure 4.5.

`Algorithm Enumerate` requires a scenario S and a parameter $K \geq 1$, which limits the maximal number of solutions that are enumerated. If we set $K = |\Phi|$, then the algorithm returns an optimal robust solution since, in the worst case, it enumerates all feasible solutions. Clearly, it is prohibitive in the general case since the number of feasible solutions may be very large. If $K < |\Phi|$, then the algorithm may return a solution that is not optimal. However, due to Proposition 4.6, we always get the lower bound on the value of the optimal robust solution, that is $OPT \geq Z(X_K, S) - Z(X_1, S)$. Moreover, if the condition in line 8 is fulfilled, then we get the optimal robust solution.

Observe that if we choose the midpoint scenario S as the input of `Algorithm Enumerate`, then the returned solution Y is always such that $Z(Y) \leq 2OPT$. It follows from the fact that the first examined solution X_1 is optimal under the midpoint scenario. Thus in this case the algorithm can be viewed as an extension of the 2-approximation `Algorithm AM`.

We show now a sample problem for which `Algorithm Enumerate` behaves poorly. Consider an instance of MINMAX REGRET SHORTEST PATH shown in Figure 4.6.

In the problem shown in Figure 4.6 there are four arcs that have interval weights $[0, 1]$ and all the remaining arcs have interval weights $[0, 0]$ (these weights are not shown in the figure). It is easy to see that the path in bold is the only optimal robust path with the maximal regret equal to 1 and all the remaining paths have the maximal regret equal to 2. Let us choose the midpoint scenario S. Under this scenario all paths from s to t in the sample graph have the same length equal to 1. Thus the optimal robust path may be enumerated as the last path in the sequence (4.9). The number of paths in the sample graph is exponential, since the layered graph between nodes o and v has an exponential number of subpaths. Thus if $K < |\Phi|$, then the algorithm may run in exponential

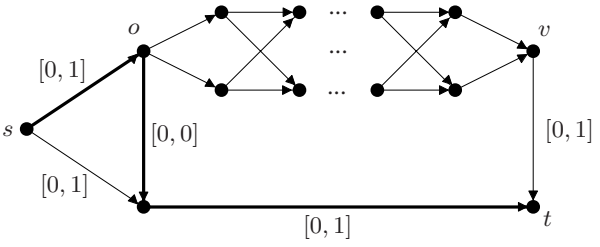


Fig. 4.6. Example of Minmax Regret Shortest Path. All arcs in the subgraph between o and v have interval weights $[0, 0]$.

time and return path Y such that $Z(Y) = 2OPT$ which gives the worst case ratio of 2. Observe also that $F(X_i, S) - F(X_1, S) = 0$ for all paths X_i, thus only a trivial lower bound equal to 0 is obtained during the execution of the algorithm.

4.2 Fully Polynomial Time Approximation Scheme

Let us recall that a fully polynomial time approximation scheme, shortly FPTAS, is a family of $(1 + \epsilon)$-approximation algorithms that run in $\mathcal{O}(q(n, 1/\epsilon))$ time, where q is a bivariate polynomial. It is well known that if P$\neq$NP, there cannot be an FPTAS for any strongly NP-hard problem with polynomially bounded, integral objective values. On the other hand, the existence of a pseudopolynomial algorithm for a hard optimization problem does not guarantee the existence of an FPTAS for it. In this section we prove a general result that allows construction of an FPTAS for some special cases of Minmax Regret $\mathcal{P}$. We will use the scaling and rounding technique, which was successfully adopted for some hard optimization problems.

In the previous section we have constructed `Algorithm AM` that outputs a feasible solution Y such that $Z(Y) \leq 2OPT$. We can thus set the upper bound $UB = Z(Y)$ and the lower bound $LB = Z(Y)/2$ on OPT. Clearly, both bounds can be computed in polynomial time if problem $\mathcal{P}$ is polynomially solvable. Assume now that we have an algorithm, denoted by `Exact`(Γ), that outputs an optimal robust solution to Minmax Regret $\mathcal{P}$ for a given scenario set Γ. Consider an algorithm $(1 + \epsilon)$-`Approx`, shown in Figure 4.7. In this algorithm we apply a scaling and rounding technique, that is we scale the weights using the value of the lower bound on OPT and we invoke then algorithm `Exact`$(\hat{\Gamma})$, where $\hat{\Gamma}$ is the scenario set for the scaled weights.

We prove the following theorem:

Theorem 4.8. *Given a problem* Minmax Regret $\mathcal{P}$*, if*

1. *$\mathcal{P}$ is polynomially solvable*
2. *and algorithm* `Exact`(Γ) *runs in time $O(q(n, UB))$, where q is a bivariate polynomial (the bounds UB and LB are within a factor 2 of each other),*

$(1+\epsilon)$-`Approx`
Require: Problem MINMAX REGRET $\mathcal{P}$ with scenario set Γ, $\epsilon \in (0,1)$.
Ensure: Solution $Y \in \Phi$ such that $Z(Y) \leq (1+\epsilon)OPT$.
1: Compute the lower bound LB on OPT by invoking `Algorithm AM` using Γ
2: **for all** $e \in E$ **do**
3: $\hat{\underline{w}}_e \leftarrow \lfloor 2\underline{w}_e n/(LB\epsilon) \rfloor$ {Determine scenario set $\hat{\Gamma}$ by scaling weights}
4: $\hat{\overline{w}}_e \leftarrow \lfloor 2\overline{w}_e n/(LB\epsilon) \rfloor$
5: **end for**
6: $Y \leftarrow$ EXACT$(\hat{\Gamma})$
7: **return** Y

Fig. 4.7. An application of scaling and rounding technique for MINMAX REGRET $\mathcal{P}$

then `Exact`$(\hat{\Gamma})$ *outputs a solution* Y *such that* $Z(Y) \leq (1+\epsilon)OPT$ *in time* $O(q(n, n/\epsilon))$ *and* $(1+\epsilon)$-`Approx` *is an FPTAS for* MINMAX REGRET $\mathcal{P}$.

Proof. Let X denote the optimal robust solution for scenario set Γ, that is $Z(X) = OPT$. We will also denote by X^* and Y^* the worst case alternatives of X and Y for scenario set Γ respectively, and similarly, by $\hat{X}^*$ and $\hat{Y}^*$ the worst case alternatives of X and Y for the scaled scenario set $\hat{\Gamma}$. Equality (1.6) and Proposition 4.1 imply

$$\hat{Z}(Y) = \sum_{e \in Y \setminus \hat{Y}^*} \hat{\overline{w}}_e - \sum_{a \in \hat{Y}^* \setminus Y} \hat{\underline{w}}_e \geq \sum_{e \in Y \setminus Y^*} \hat{\overline{w}}_e - \sum_{e \in Y^* \setminus Y} \hat{\underline{w}}_e.$$

By the definitions of $\hat{\underline{w}}_e$ and $\hat{\overline{w}}_e$ and the fact that $a \geq \lfloor a \rfloor \geq a - 1$ we obtain

$$\hat{Z}(Y) \geq \sum_{e \in Y \setminus Y^*} \frac{2\overline{w}_e n}{LB\epsilon} - n - \sum_{e \in Y^* \setminus Y} \frac{2\underline{w}_e n}{LB\epsilon}. \tag{4.10}$$

The following inequality holds:

$$\hat{Z}(X) = \sum_{e \in X \setminus \hat{X}^*} \hat{\overline{w}}_e - \sum_{e \in \hat{X}^* \setminus X} \hat{\underline{w}}_e \leq \sum_{e \in X \setminus \hat{X}^*} \frac{2\overline{w}_e n}{LB\epsilon} - \sum_{e \in \hat{X}^* \setminus X} \frac{2\underline{w}_e n}{LB\epsilon} + n. \tag{4.11}$$

Now, from (4.10), (4.11) and the fact that $\hat{Z}(Y) \leq \hat{Z}(X)$ (Y is the optimal robust solution for scenario set $\hat{\Gamma}$) we have

$$\sum_{e \in Y \setminus Y^*} \frac{2\overline{w}_e n}{LB\epsilon} - n - \sum_{e \in Y^* \setminus Y} \frac{2\underline{w}_e n}{LB\epsilon} \leq \sum_{e \in X \setminus \hat{X}^*} \frac{2\overline{w}_e n}{LB\epsilon} - \sum_{e \in \hat{X}^* \setminus X} \frac{2\underline{w}_e n}{LB\epsilon} + n.$$

Multiplying both sides of the above inequality by $LB\epsilon/(2n)$, we get

$$\sum_{e \in Y \setminus Y^*} \overline{w}_e - \sum_{e \in Y^* \setminus Y} \underline{w}_e - \frac{LB\epsilon}{2} \leq \sum_{e \in X \setminus \hat{X}^*} \overline{w}_e - \sum_{e \in \hat{X}^* \setminus X} \underline{w}_e + \frac{LB\epsilon}{2}. \tag{4.12}$$

Since

$$Z(Y) = \sum_{e \in Y \setminus Y^*} \overline{w}_e - \sum_{e \in Y^* \setminus Y} \underline{w}_e,$$

we can rewrite (4.12) as

$$Z(Y) \leq \sum_{e \in X \setminus \hat{X}^*} \overline{w}_e - \sum_{e \in \hat{X}^* \setminus X} \underline{w}_e + LB\epsilon. \tag{4.13}$$

From Proposition 4.1, it follows that

$$Z(X) \geq \sum_{e \in X \setminus \hat{X}^*} \overline{w}_e - \sum_{e \in \hat{X}^* \setminus X} \underline{w}_e.$$

Thus from (4.13) and the assumption $Z(X) = OPT$ we finally obtain

$$Z(Y) \leq Z(X) + \epsilon LB \leq Z(X) + \epsilon Z(X) = (1+\epsilon)OPT. \tag{4.14}$$

The running time of algorithm $\texttt{Exact}(\hat{\Gamma})$ is $\mathcal{O}(q(n, \widehat{UB}))$, where $\widehat{UB}$ is the upper bound for scenario set $\hat{\Gamma}$ computed by `Algorithm AM`. Hence $\widehat{UB} \leq 2\hat{Z}(Y) \leq 2\hat{Z}(X)$. From inequality (4.11) we get

$$\hat{Z}(X) \leq \frac{2n}{LB\epsilon} \left(\sum_{e \in X \setminus \hat{X}^*} \overline{w}_e - \sum_{e \in \hat{X}^* \setminus X} \underline{w}_e \right) + n$$

and consequently

$$\widehat{UB} \leq \frac{4n}{LB\epsilon} \left(\sum_{e \in X \setminus \hat{X}^*} \overline{w}_e - \sum_{e \in \hat{X}^* \setminus X} \underline{w}_e \right) + 2n. \tag{4.15}$$

The application of Proposition 4.1 to (4.15) gives

$$\widehat{UB} \leq \frac{4n}{LB\epsilon} Z(X) + 2n \leq \frac{4n}{LB\epsilon} UB + 2n. \tag{4.16}$$

Bounds LB and UB are computed by `Algorithm AM`, which gives $UB/LB = 2$. Thus from (4.16) we obtain $\widehat{UB} \leq \frac{8n}{\epsilon} + 2n$. Therefore, the running time of $\texttt{Exact}(\hat{\Gamma})$ is $\mathcal{O}(q(n, n/\epsilon))$. From the assumption that problem $\mathcal{P}$ is polynomially solvable it follows that line 1 of $(1+\epsilon)$-`Approx` can be performed in polynomial time. Lines 2-5 require $\mathcal{O}(n)$ time. Finally, line 6 requires $\mathcal{O}(q(n, n/\epsilon))$ time. Hence the overall running time of $(1+\epsilon)$-`Approx` is bounded by a polynomial in n and $1/\epsilon$. This, together with condition (4.14), means that $(1+\epsilon)$-`Approx` is an FPTAS for Minmax Regret $\mathcal{P}$. □

We will show some applications of Theorem 4.8 in the next chapters. Notice that we must first design a pseudopolynomial algorithm for the problem before applying the theorem. We will do this for some special cases of Minmax Regret Shortest Path and Minmax Regret Minimum Cut.

4.3 Notes and References

The first approximation result for MINMAX REGRET $\mathcal{P}$ was `Algorithm AM` designed by Kasperski and Zieliński [80]. `Algorithm AMU` being a slight modification of `Algorithm AM` was proposed by Kasperski and Zieliński [76] and Kasperski *et al.* [77]. Some computational experiments carried out in [76, 77] showed that for the tested instances `Algorithm AMU` returns solutions significantly better than `Algorithm AM`. Nevertheless, the worst case behavior of both algorithms is the same.

`Algorithm Enumerate` is based one the idea proposed by Montemanni and Gambardella [102]. Propositions 4.6 and 4.7 are also generalizations of the results obtained in [102]. Montemanni and Gambardella applied a ranking algorithm to solve MINMAX REGRET SHORTEST PATH. They used the ordering of paths under scenario S_E^+, that is the one in which all the element weigths are set to their upper bounds. However, it is not difficult to show that the maximal regret of a path that is optimal under scenario S_E^+ may be arbitrarily far from optimum.

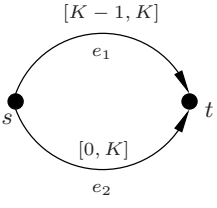


Fig. 4.8. A sample problem

Consider the problem shown in Figure 4.8. There are two paths in G, $\{e_1\}$ and $\{e_2\}$ that have the same weights under S_E^+. The maximal regrets of these paths are as follows: $Z(\{e_1\}) = K$, $Z(\{e_2\}) = 1$. Thus $Z(\{e_1\})/Z(\{e_2\}) = K$, which may be arbitrarily large. This simple example demonstrates that a path of the minimal weight under scenario S_E^+ may be arbitrarily far from the optimal robust path. Therefore, from the theoretical point of view, it is better to use the midpoint scenario while ranking the paths, since we get at once the worst case ratio of 2 of the algorithm. The experiments carried out in [102] showed that the performance of the algorithm based on the idea of enumerating the solutions is good if the number of solutions is not very large. The poorest performance was reported for layered graphs in which the number of paths grows exponentially and where a large number of paths of the same weight under the specified scenario exist.

`Algorithm Enumerate` requires a method of ranking the solutions under a given scenario S. This is not a trivial task in general. However, for the problems considered in this monograph, such a ranking can be done. For MINIMUM SPANNING TREE a ranking algorithm was designed by Gabow [55]. For SHORTEST PATH the ranking algorithm designed by Eppstein [49] can by used. For MINIMUM ASSIGMENT a ranking algorithm was designed by Murty [108] and it was refined by Pascoal *et al.* [113].

Theorem 4.8 that allows us to construct an FPTAS for a particular class of problems was proved by Kasperski and Zieliński [83]. Later in this monograph we will show some applications of this theorem to some particular problems. Obviously, a pseudopolynomial algorithm should be designed first and this can be done only if the problem is not strongly NP-hard.

The 2-approximation `Algorithm AM` and Theorem 4.8 are the only known approximation results for Minmax Regret $\mathcal{P}$. There is an important open question whether there is an approximation algorithm with a performance ratio lower than 2 for the general problem or for some of its special versions. It is also possible that the problem admits a polynomial time approximation scheme. The questions concerning the approximation of Minmax Regret $\mathcal{P}$ are among the most important open problems in the minmax regret approach.

5 Minmax Regret Minimum Selecting Items

In this chapter we discuss the Minmax Regret Minimum Selecting Items problem. We are given a set of elements, called *items*, $E = \{e_1, \ldots, e_n\}$ with interval weights and we wish to choose a subset of exactly p items that minimizes the maximal regret. Thus the set of feasible solutions in this problem is the following:

$$\Phi = \{X \subseteq E : \ |X| = p\},$$

where p is a fixed integer such that $1 \leq p \leq n$. The problem can also be viewed as a special case of 0-1 Knapsack in which the capacities of all items are equal to 1. The optimal solution to the deterministic Minimum Selecting Items problem can be obtained by selecting p items of the smallest weights. This can be done in $\mathcal{O}(n)$ time by first selecting p-th smallest weighted item f and by selecting then $p-1$ items of the weight less than or equal to the weight of f.

It turns out that Minmax Regret Minimum Selecting Items is the only polynomially solvable minmax regret combinatorial optimization problem considered in this monograph. In Section 5.2 we give an algorithm with running time $\mathcal{O}(n \min\{p, n-p\})$ for this problem, which is due to Conde [37]. In Section 5.1 we show that all possibly and necessarily optimal items can be detected very efficiently in $\mathcal{O}(n)$ time. Hence for large problems it may be advantageous to perform the preprocessing, described in Section 2.2, before applying the exact algorithm.

Since all feasible solutions in the problem have the same cardinality, all results presented in this chapter can also be applied to the Minmax Regret Maximum Selecting Items problem. It is enough to use the transformation shown in the proof of Theorem 1.5.

5.1 Evaluation of Optimality

In this section we use the fact, that the deterministic Minimum Selecting Items problem has a matroidal structure. To see this, consider system $(E, \mathcal{I})$, where E is a given set of items and $\mathcal{I}$ is the set of subsets of E, whose cardinalities are less than or equal to p, that is $\mathcal{I} = \{X \subseteq E : \ |X| \leq p\}$. It is easy to

A. Kasperski: Discrete Optimization with Interval Data, STUDFUZZ 228, pp. 51–60, 2008.
springerlink.com

verify that system $(E, \mathcal{I})$ is a matroid and in literature it is called a *uniform matroid*. The bases of this matriod are those subsets in $\mathcal{I}$ whose cardinalities are exactly p. Hence the set of feasible solutions Φ consists of all bases of a uniform matroid. The greedy algorithm applied to the problem simply outputs p items of the smallest weights. If the item weights are specified as intervals, then we can apply the results from Section 2.2.3 to detect all the possibly and necessarily optimal items. In this section we show that this can be done in linear time $\mathcal{O}(n)$.

5.1.1 Possibly Optimal Items

The following proposition characterizes a possibly optimal item:

Proposition 5.1. *Let f be the p-th smallest weighted item under scenario S_E^+. Then $e \in E$ is possibly optimal if and only if $\underline{w}_e \leq \overline{w}_f$.*

Proof. ($\Rightarrow$) Suppose, by contradiction, that e is possibly optimal and $\underline{w}_e > \overline{w}_f$. So, there are at least p items whose weights are strictly less than the weight of e in scenario $S_{\{e\}}^-$. In consequence, e cannot be a part of an optimal solution under $S_{\{e\}}^-$ and e is not possibly optimal (see Theorem 2.17), which is a contradiction.
($\Leftarrow$) If $\underline{w}_e \leq \overline{w}_f$, then e is one of the p items of the smallest weights in scenario $S_{\{e\}}^-$. Thus e is a part of an optimal solution under $S_{\{e\}}^-$ and, consequently, it is possibly optimal. □

Using Proposition 5.1 we can detect all possibly optimal items in $\mathcal{O}(n)$ time. Indeed, the p-th smallest weighted item f in scenario S_E^+ can be found in $\mathcal{O}(n)$ time. We can detect then all possibly optimal items by comparing the values of $\underline{w}_e$ and $\overline{w}_f$ for all $e \in E$, which also requires $\mathcal{O}(n)$ time. Applying Theorem 2.9, we can remove all items which are nonpossibly optimal from set E, which may reduce the size of the problem.

5.1.2 Necessarily Optimal Items

The following proposition characterizes a necessarily optimal item:

Proposition 5.2. *Let f be the p-th smallest weighted item and let g be the $(p+1)$-th smallest weighted item under scenario S_E^-. Then $e \in E$ is necessarily optimal if and only if $\underline{w}_e \leq \underline{w}_f$ and $\overline{w}_e \leq \underline{w}_g$.*

Proof. ($\Rightarrow$) Suppose, by contradiction, that item e is necessarily optimal and $\underline{w}_e > \underline{w}_f$ or $\overline{w}_e > \underline{w}_g$. In the first case ($\underline{w}_e > \underline{w}_f$) item e cannot be a part of the optimal solution under S_E^- (since there are p items of the weights strictly smaller than the weight of e under S_E^-) and in the second case ($\overline{w}_e > \underline{w}_g$) it cannot be a part of the optimal solution under $S_{\{e\}}^+$ (since there are p items of the weights strictly smaller than the weight of e under $S_{\{e\}}^+$). This contradicts the assumption that e is necessarily optimal.
($\Leftarrow$) Conditions $\underline{w}_e \leq \underline{w}_f$ and $\overline{w}_e \leq \underline{w}_g$ assure that e is one of the p items of the smallest weights under $S_{\{e\}}^+$. Thus, e is a part of the optimal solution in $S_{\{e\}}^+$ and, by Theorem 2.18, it is necessarily optimal. □

Using Proposition 5.2 we can detect all necessarily optimal items in $\mathcal{O}(n)$ time. We must first detect the p-th and $(p+1)$-th smallest weighted items f and g in scenario S_E^-, which requires $\mathcal{O}(n)$ time. We can detect then all necessarily optimal items by comparing $\underline{w}_e$ to $\underline{w}_f$ and $\overline{w}_e$ to $\underline{w}_g$ for all $e \in E$, which also requires $\mathcal{O}(n)$ time. Observe that if all weight intervals are nondegenerate, then Proposition 5.2 can be simplified. Indeed, conditions $\overline{w}_e \leq \underline{w}_g$ and $\underline{w}_e < \overline{w}_e$ imply $\underline{w}_e < \underline{w}_g$ and e must be among the p smallest weighted items under S_E^-. This forces $\underline{w}_e \leq \underline{w}_f$. In consequence, condition $\underline{w}_e \leq \underline{w}_f$ is redundant and it can be removed from Proposition 5.2.

We now show that in the absence of degeneracy, the detection of all necessarily optimal items allows us to reduce the problem to a smaller one. We first prove the following proposition:

Proposition 5.3. *If all weight intervals are nondegenerate, then for every scenario $S \in \Gamma$ there exists an optimal solution that contains all necessarily optimal items.*

Proof. Let us denote the set of all necessarily optimal items by N. Observe, that it is enough to prove the proposition for the particular scenario S_N^+. Indeed, if all items from N are among the p smallest weighted items in S_N^+, then this is also the case under all scenarios $S \in \Gamma$. Let g be the $(p+1)$-th smallest weighted item in scenario S_E^-. By Proposition 5.2 and the assumption that all intervals are nondegenerate, for every item $e \in N$ it holds $\underline{w}_e < \overline{w}_e \leq \underline{w}_g$. In consequence, all items from N are among the p smallest weighted items in S_N^+. □

By Theorem 2.13, in the absence of degeneracy, there is at least one optimal robust solution X that contains all necessarily optimal items. Hence X is of the form $Y \cup N$, where $|Y| = p - |N|$. From Proposition 5.3 it follows that for every scenario S there is an optimal solution X^* that contains all necessarily optimal items. Thus X^* is of the form $Y^* \cup N$, where $Y^* = p - |N|$. Moreover, Y^* must be the optimal solution for the problem of selecting $p - |N|$ items from $E \setminus N$ under scenario S. Now we see that

$$F(X,S) - F^*(S) = F(X,S) - F(X^*,S) = F(Y,S) - F(Y^*,S).$$

In consequence, if Y is the optimal robust solution in the problem of selecting $p - |N|$ items from set $E \setminus N$, then the corresponding solution $Y \cup N$ is the optimal robust solution of the original problem.

We have thus established another fast and efficient method of preprocessing an instance of the problem under the assumption that all the weight intervals are nondegenerate. We detect first in $\mathcal{O}(n)$ time all the necessarily optimal items obtaining set N. We remove then all necessarily optimal items from E and we solve a smaller subproblem obtaining a solution Y. The solution $N \cup Y$ is then the optimal robust solution of the original problem.

Example 5.4. We now illustrate the preprocessing by an example. Let $E = \{e_1, \ldots, e_9\}$ and $p = 5$. The interval weights of the items are shown in Table 5.1a.

Table 5.1. a) The interval item weights in the sample problem. b) The problem after removing the nonpossibly optimal item e_5. c) The problem after detecting necessarily optimal items e_3 and e_8; now it is enough to select 3 items from set $\{e_1, e_2, e_4, e_6, e_7, e_9\}$.

a)

e_i	$\underline{w}_{e_i}$	$\overline{w}_{e_i}$
e_1	1	6
e_2	6	9
e_3	2	3
e_4	4	5
e_5	9	9
e_6	7	8
e_7	2	10
e_8	1	2
e_9	3	9

b)

e_i	$\underline{w}_{e_i}$	$\overline{w}_{e_i}$
e_1	1	6
e_2	6	9
e_3	2	3
e_4	4	5
e_6	7	8
e_7	2	10
e_8	1	2
e_9	3	9

c)

e_i	$\underline{w}_{e_i}$	$\overline{w}_{e_i}$
e_1	1	6
e_2	6	9
e_4	4	5
e_6	7	8
e_7	2	10
e_9	3	9

We start by detecting all nonpossibly optimal items making use of Proposition 5.1. The p-th smallest weighted item under scenario S_E^+ is e_6 with $\overline{w}_{e_6} = 8$. Now, applying Proposition 5.1, we get that element e_5 is nonpossibly optimal (because $\underline{w}_{e_5} > 8$). Therefore we can remove e_5 from the problem and consider further selecting 5 items from set $E \setminus \{e_5\}$. The data of this problem are shown in Table 5.1b. Now, applying Proposition 5.2, we detect the necessarily optimal elements. Observe that all the intervals in Table 5.1b are nondegenerate. Thus it is enough to find the $(p+1)$-th smallest weighted item under scenario S_E^- that is the item e_4 with $\underline{w}_{e_2} = 4$. Now, applying Proposition 5.2, we get that items e_3 and e_8 are necessarily optimal (because $\overline{w}_{e_3} \leq 4$ and $\overline{w}_{e_8} \leq 4$). Thus, we have $N = \{e_3, e_8\}$. We remove both items from E obtaining set $E_1 = \{e_1, e_2, e_4, e_6, e_7, e_9\}$. Since all the weight intervals in Table 5.1b are nondegenerate, the problem is now to select 3 items from set E_1. The data of this final problem are shown in Table 5.1c. □

In the next section we construct a polynomial time algorithm for the problem whose running time is $\mathcal{O}(n \min\{p, n-p\})$. Hence the preprocessing, which requires only $\mathcal{O}(n)$ time, may be attractive from the computational point of view.

5.2 Polynomial Algorithm for the Problem

In this section we construct a polynomial algorithm for solving Minmax Regret Minimum Selecting Items, which is based on the idea of Conde [37]. The starting point of this algorithm is the MIP model constructed in Section 3.1. We show that this model can be solved in polynomial time. Recall that the integer program has variable $x_i \in \{0, 1\}$ associated to every item $e_i \in E$, which expresses whether e_i is a part of the constructed solution. Assume that $[\underline{w}_i, \overline{w}_i]$ is the interval weight of element $e_i \in E$, $i = 1, \ldots, n$. The MIP model for Minmax Regret Minimum Selecting Items is the following (see Section 3.1):

$$\begin{aligned}
&\min \sum_{i=1}^{n} \overline{w}_i x_i - \lambda p + \lambda_1 + \cdots + \lambda_n \\
&x_1 + \cdots + x_n = p \\
&\lambda - \lambda_i \leq \overline{w}_i x_i + \underline{w}_i (1 - x_i) \quad i = 1, \ldots, n \\
&\lambda_i \geq 0 \qquad\qquad\qquad\qquad\quad i = 1, \ldots, n \\
&x_i \in \{0, 1\} \qquad\qquad\qquad\qquad\quad i = 1, \ldots, n
\end{aligned} \tag{5.1}$$

Observe that the objective of (5.1) is an increasing function of all λ_i, $i = 1, \ldots, n$. Thus every optimal solution to (5.1) must satisfy

$$\lambda_i = \max\{0, \lambda - \overline{w}_i x_i - \underline{w}_i (1 - x_i)\}, \ i = 1, \ldots, n.$$

Hence model (5.1) is equivalent to the following one:

$$\begin{aligned}
&\min \left(\sum_{i=1}^{n} \overline{w}_i x_i - \lambda p + \sum_{i=1}^{n} \max\{0, \lambda - (\overline{w}_i x_i + \underline{w}_i (1 - x_i))\} \right) \\
&x_1 + \cdots + x_n = p \\
&x_i \in \{0, 1\} \qquad i = 1, \ldots, n
\end{aligned} \tag{5.2}$$

Consider the ordered sequences of the bounds of the interval weights

$$\underline{w}_{[1]} \leq \underline{w}_{[2]} \leq \cdots \leq \underline{w}_{[n]},$$

$$\overline{w}^{[1]} \leq \overline{w}^{[2]} \leq \cdots \leq \overline{w}^{[n]}.$$

We prove the following proposition:

Proposition 5.5. *The objective function of (5.2) attains minimum for*

$$\lambda \in \{\underline{w}_{[p]}, \ldots, \underline{w}_{[2p]}\} \cup \{\overline{w}^{[1]}, \ldots, \overline{w}^{[p]}\}.$$

Proof. Let $(x_i^*)_{i=1}^n$, λ^*, be the optimal solution to (5.2). Hence $(x_i^*)_{i=1}^n$ is a characteristic vector of the optimal robust solution X and the value of the objective of (5.2) is equal to $Z(X)$. The worst case scenario $S \equiv S_X^+$ is such that

$$w_i^S = \overline{w}_i x_i^* + \underline{w}_i (1 - x_i^*), \ i = 1, \ldots, n. \tag{5.3}$$

From (5.3) and (5.2) we obtain:

$$Z(X) = \sum_{i=1}^{n} \overline{w}_i x_i^* - \lambda^* p + \sum_{i=1}^{n} \max\{0, \lambda^* - w_i^S\}.$$

Consider the ordered sequence of weights under the worst case scenario S

$$w_{[1]}^S \leq w_{[2]}^S \leq \cdots \leq w_{[p]}^S \leq \cdots \leq w_{[n]}^S.$$

Let us now replace λ^* with $w^S_{[p]}$. We get:

$$Z(X) \leq \sum_{i=1}^{n} \overline{w}_i x_i^* - w^S_{[p]} p + \sum_{i=1}^{n} \max\{0, w^S_{[p]} - w^S_i\} =$$

$$= \sum_{i=1}^{n} \overline{w}_i x_i^* - w^S_{[p]} p + w^S_{[p]} p - \sum_{i=1}^{p} w^S_{[i]} = \sum_{i=1}^{n} \overline{w}_i x_i^* - \sum_{i=1}^{p} w^S_{[i]} = Z(X).$$

The last equality follows from the fact that $F(X,S) = \sum_{i=1}^{n} \overline{w}_i x_i^*$ and $F^*(S) = \sum_{i=1}^{p} w^S_{[i]}$. Hence, $\lambda = w^S_{[p]}$ also minimizes the objective of (5.2).

Because $S = S^+_X$ is an extreme scenario, it is clear that $\lambda = w^S_{[p]}$ is one of the endpoints $\underline{w}_i$ or $\overline{w}_i$ of the weight intervals. Since it is the p-th smallest weighted item in scenario S^+_X and precisely p items have weights set to their upper bounds in S^+_X, it follows that either $\lambda \in \{\underline{w}_{[p]}, \ldots, \underline{w}_{[2p]}\}$ or $\lambda \in \{\overline{w}^{[1]}, \ldots, \overline{w}^{[p]}\}$. □

Let us now additionally transform problem (5.2). Since $p = \sum_{i=1}^{n} x_i$, it can be rewritten as follows:

$$\begin{aligned} &\min \sum_{i=1}^{n} (\overline{w}_i x_i + \max\{0, \lambda - \overline{w}_i x_i - \underline{w}_i (1 - x_i)\} - \lambda x_i) \\ &x_1 + \cdots + x_n = p \\ &x_i \in \{0,1\} \qquad \text{for } i = 1, \ldots, n \end{aligned} \tag{5.4}$$

which is equivalent to:

$$\begin{aligned} &\min \sum_{i=1}^{n} \max\{(\overline{w}_i - \lambda) x_i, (\lambda - \underline{w}_i)(1 - x_i)\} \\ &x_1 + \cdots + x_n = p \\ &x_i \in \{0,1\} \qquad \text{for } i = 1, \ldots, n \end{aligned} \tag{5.5}$$

It holds:

$$\max\{(\overline{w}_i - \lambda) x_i, (\lambda - \underline{w}_i)(1 - x_i)\} = a_i(\lambda) x_i + b_i(\lambda),$$

where

$$\begin{aligned} a_i(\lambda) &= \max\{0, \overline{w}_i - \lambda\} - \max\{0, \lambda - \underline{w}_i\} \\ b_i(\lambda) &= \max\{0, \lambda - \underline{w}_i\} \end{aligned},$$

which can be easily verified by checking two cases $x_i = 1$ or $x_i = 0$. Finally, problem (5.5) can be rewritten as follows:

$$\begin{aligned} &\min \textstyle\sum_{i=1}^{n} (a_i(\lambda) x_i + b_i(\lambda)) \\ &x_1 + \cdots + x_n = p \\ &x_i \in \{0,1\} \end{aligned} \tag{5.6}$$

Proposition 5.6. *The problem (5.6) for a fixed λ can be solved in $\mathcal{O}(n)$ time.*

Proof. Indeed, if λ is fixed, then the part $\sum_{i=1}^{n} b_i(\lambda)$ of the objective of (5.6) is a constant and problem (5.6) is equivalent to the deterministic MINIMUM SELECTING ITEMS in which the item weights are $a_i(\lambda)$. This problem can be solved in $\mathcal{O}(n)$ time by selecting p items of the smallest values of $a_i(\lambda)$. □

Propositions 5.5 and 5.6 immediately suggest an $\mathcal{O}(pn)$ algorithm for the problem. It is enough to solve problem (5.6) for $\mathcal{O}(p)$ values of λ as shown in Proposition 5.5. Since every problem can be solved in $\mathcal{O}(n)$ time, the overall complexity of the algorithm is $\mathcal{O}(pn)$. We now show, following Averbakh [16], that this complexity can be additionally refined.

Proposition 5.7. *If $p > n/2$ then the problem can be transformed in $\mathcal{O}(n)$ time to an equivalent problem with $p' \leq n/2$.*

Proof. Consider the problem with scenario set Γ in which $p > n/2$. Let us define a new scenario set Γ', which is the cartesian product of intervals $[M-\overline{w}_e, M-\underline{w}_e]$, $e \in E$, where $M = \max_{e\in E} \overline{w}_e$. Every scenario $S = (w_e^S)_{e\in E}$ in Γ has the corresponding scenario $S' = (M - w_e^S)_{e\in E}$ in Γ'. Consider a new problem with scenario set Γ' in which we wish to select a subset of $n-p$ items that minimizes the maximal regret. Denote by $F'(X, S')$ the weight of a solution X under $S' \in \Gamma'$, by $F^{*'}(S')$ the value of the optimal solution under $S' \in \Gamma'$ and by $Z'(X)$ the maximal regret of solution X in the new problem. Let X be a solution such that $|X| = n - p$. Define $\overline{X} = E \setminus X$, thus $|\overline{X}| = p$. It holds

$$F'(X, S') - F(\overline{X}, S) = \sum_{e\in X}(M - w_e^S) - \sum_{e\in\overline{X}} w_e^S = (n-p)M - \sum_{e\in E} w_e^S. \quad (5.7)$$

Consider the ordered sequence of elements $e_{[1]}, \ldots, e_{[n]}$ such that

$$w^S_{e_{[1]}} \leq w^S_{e_{[2]}} \leq \ldots \leq w^S_{e_{[n]}}.$$

From the relation between S and S' we have

$$w^{S'}_{e_{[n]}} \leq w^{S'}_{e_{[n-1]}} \leq \ldots \leq w^{S'}_{e_{[1]}}.$$

Hence the value of $F^*(S)$ can be obtained by adding the weights of elements $e_{[1]}, \ldots, e_{[p]}$ under S and the value of $F^{*'}(S')$ can be obtained by adding the weights of the remaining elements $e_{[p+1]}, \ldots, e_{[n]}$ under S'. Now it is easily seen that

$$F^{*'}(S') - F^*(S) = (n-p)M - \sum_{e\in E} w_e^S. \quad (5.8)$$

Equalities (5.7) and (5.8) imply

$$F'(X, S') - F^{*'}(S') = F(\overline{X}, S) - F^*(S),$$

which means that $Z'(X) = Z(\overline{X})$. Hence if $p > n/2$, then we can transform the original problem into the one with scenario set Γ' and $p' = n - p \leq n/2$. Solving

```
Algorithm Selecting Items
Require: n, p, ([w_i, w̄_i])_{i=1}^n.
Ensure: The optimal robust solution X.
 1: if p > n/2 then transform the problem according to Proposition 5.7
 2: Determine set Λ = {w_[p], ..., w_[2p]} ∪ {w̄^[1], ..., w̄^[p]}
 3: X ← ∅, OPT ← ∞
 4: for all λ ∈ Λ do
 5:    b ← 0
 6:    for i ← 1 to n do
 7:       a_i ← max{0, w̄_i − λ} − max{0, λ − w_i}
 8:       b ← b + max{0, λ − w_i}
 9:    end for
10:    Y ← arg min_{U:|U|=p} Σ_{e_i∈U} a_i {Solve determ. problem with weights a_i}
11:    val ← Σ_{e_i∈Y} a_i + b
12:    if val < OPT then
13:       X ← Y, OPT ← val
14:    end if
15: end for
16: return X
```

Fig. 5.1. The algorithm for Minmax Regret Minimum Selecting Items (see also 37)

the new problem we obtain the optimal robust solution X, which immediately gives the optimal robut solution $\overline{X} = E \setminus X$ of the original problem. It is clear that the transformation requires $\mathcal{O}(n)$ time, which is necessary to transform Γ into Γ'. □

The algorithm for solving the problem is shown in Figure 5.1.

First, if $p > n/2$, then we transform the problem so that $p' \leq n/2$, as shown in the proof of Proposition 5.7. Next we determine set Λ that contains all candidate values of λ. By Proposition 5.5, one of these values minimizes the objective of (5.6). The set Λ can be constructed in $\mathcal{O}(np)$ time. For every $\lambda \in \Lambda$ we then solve problem (5.6) and we choose the solution of the minimal value of the objective function. For a fixed λ problem (5.6) can be solved in $\mathcal{O}(n)$ time, therefore the optimal solution can be found in $\mathcal{O}(np)$ time. Since $p \leq n/2$, the overall running time of the algorithm is $\mathcal{O}(n \min\{p, n-p\})$. We have thus established the following result (see also [37]):

Theorem 5.8. *The* Minmax Regret Minimum Selecting Items *problem can be solved in* $\mathcal{O}(n \min\{p, n-p\})$ *time.*

Let us illustrate the calculations of the optimal robust solution by an example.

Example 5.9. Let $E = \{e_1, \ldots, e_{10}\}$ and $p = 4$. The interval weights of items are shown in Table 5.2. We first determine set Λ, which by Proposition 5.5 is the following:

$$\Lambda = \{1, 5, 7, 12\} \cup \{26, 27, 28, 32\}.$$

Table 5.2. The interval weights in the sample problem. The underlined bounds belong to Λ.

e_i	$\underline{w}_{e_i}$	$\overline{w}_{e_i}$
e_1	$\underline{12}$	33
e_2	18	$\underline{26}$
e_3	19	$\underline{27}$
e_4	$\underline{7}$	$\underline{28}$
e_5	$\underline{1}$	$\underline{32}$
e_6	1	32
e_7	1	34
e_8	0	35
e_9	$\underline{5}$	32
e_{10}	0	34

Table 5.3. The calculations in the sample problem. The optimal robust solution is marked in bold.

λ	solution	$f(\lambda)$
1	$\{e_2, e_3, e_4, e_5\}$	111
5	$\{e_2, e_3, e_4, e_5\}$	111
7	$\{e_2, e_3, e_5, e_6\}$	111
12	$\{\mathbf{e_5}, \mathbf{e_6}, \mathbf{e_7}, \mathbf{e_{10}}\}$	**108**
26	$\{e_5, e_6, e_7, e_{10}\}$	123
27	$\{e_5, e_6, e_7, e_{10}\}$	125
28	$\{e_5, e_6, e_7, e_{10}\}$	127
32	$\{e_5, e_6, e_7, e_{10}\}$	135

Now we solve problem (5.6) for all $\lambda \in \Lambda$. The optimal solutions for every λ, together with the minimal value of the objective $f(\lambda)$, are shown in Table 5.3. We can see that the optimal robust solution is $X = \{e_5, e_6, e_7, e_{10}\}$ with the maximal regret $Z(X) = 108$. This is precisely the solution which minimizes (5.6). □

5.3 Notes and References

The deterministic Minimum Selecting Items problem can be viewed as a basic resource allocation problem [63] or a special version of 0-1 Knapsack with unit capacities of items. The proof of the fact that the solutions are bases of a uniform matroid is easy and it was given for instance by Welsh [121]. Computing the p-th smallest weighted item can be done in linear time [26, 38]. Consequently, the optimal solution for the deterministic Selecting Items problem can also be obtained in linear time.

The evaluation of optimality of items in the problem was studied by Kasperski and Zieliński [78, 81]. Propositions 5.1 and 5.2 were first proved in [81]. The first polynomial algorithm for Minmax Regret Minimum Selecting Items, with

running time $\mathcal{O}((\min\{p, n-p\})^2 n)$, was designed by Averbakh [16]. A faster algorithm, with running time $\mathcal{O}(n \min\{p, n-p\})$, was designed by Conde [37]. In Section 5.2 the algorithm based on the Conde's idea is presented.

An interesting result concerning the problem was obtained by Averbakh [16]. He showed, that the problem with discrete scenario representation of uncertainty is NP-hard, even if the number of scenarios is equal to 2. This result suggests that the discrete scenario representation forms a different class of problems than the interval one. Perhaps, this second class is simpler from the computational point of view. For a deeper discussion on this fact we refer the reader to Appendix A, where we prove that for the discrete scenario representation the problem is strongly NP-hard and it is not approximable within $2-\epsilon$ for any $\epsilon > 0$ if P$\neq$NP.

The Minmax Regret Minimum Selecting Items problem can be generalized by introducing capacities c_e for all items $e \in E$ and defining $\Phi = \{X \subseteq E : \sum_{e \in X} c_e \geq C\}$, where C is a fixed integer. We obtain in this way the 0-1 Knapsack problem. The deterministic 0-1 Knapsack is known to be NP-hard so the minmax regret version of this problem with interval weights is NP-hard as well. However the deterministic problem can be solved by a pseudopolynomial algorithm with complexity $\mathcal{O}(nC)$ and admits an FPTAS. The existence of a pseudopolynomial algorithm (and FPTAS) for Minmax Regret 0-1 Knapsack is an interesting open problem.

6 Minmax Regret Minimum Spanning Tree

In this chapter we discuss Minmax Regret Minimum Spanning Tree. In this problem set E consists of all edges of a given undirected and connected graph $G = (V, E)$. The set of feasible solutions Φ contains all *spanning trees* of G, that is the subsets of $|V| - 1$ edges that form acyclic subgraphs of G. For every edge $e \in E$ there is an interval weight $[\underline{w}_e, \overline{w}_e]$ given. We wish to find a spanning tree that minimizes the maximal regret. A sample problem is shown in Figure 6.1. The spanning tree in bold is the optimal robust solution in the sample problem and it has the maximal regret equal to 8.

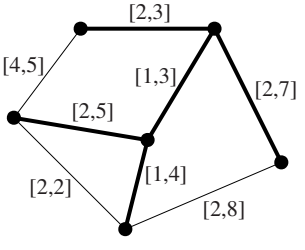


Fig. 6.1. An example of Minmax Regret Minimum Spanning Tree

The deterministic Minimum Spanning Tree problem is among the most extensively studied problems in combinatorial optimization and many efficient polynomial time algorithms for it have been developed. The fastest algorithm up to now was developed by Chazelle [35]. Its running time is $\mathcal{O}(|E|\alpha(|E|, |V|))$, where α is the inverse-Ackermann function. This running time is very close to $\mathcal{O}(|E|)$. However, in practical applications, Kruskal's or Prim's simpler algorithms are often used.

Contrary to the deterministic case, Minmax Regret Minimum Spanning Tree turned out to be strongly NP-hard. The proof of this fact, according to Aron and van Hentenryck [13], is presented in Section 6.1. In Section 6.2 we describe an efficient method of detecting all possibly and necessarily optimal edges,

A. Kasperski: Discrete Optimization with Interval Data, STUDFUZZ 228, pp. 61–79, 2008.
springerlink.com

which is a consequence of the matroidal structure of the problem. Among the exact methods for solving the problem there are a MIP formulation (described in Section 6.3) and a branch and bound algorithm (described in Section 6.4). Both techniques allow us to solve the problem for rather not very large graphs. In Section 6.5 we propose a local search algorithm, which is based on a natural neighborhood structure. We also provide the worst case analysis of the local minima returned by the local search. For large graphs the simple 2-approximation algorithms designed in Section 4.1 can also be used.

All spanning trees of a given graph $G = (V, E)$ have the same number of edges equal to $|V| - 1$. Therefore, due to Theorem 1.5, all the results presented in this chapter are also valid for Minmax Regret Maximum Spanning Tree.

6.1 Computational Complexity

In Section 1.3.4 the Central $\mathcal{P}$ problem has been described. Consider a special version of this problem, namely Central Spanning Tree. In this problem, there is given a connected, undirected graph $G = (V, E)$ and we wish to find a *central spanning tree* of G, that is a spanning tree whose maximal distance to all other spanning trees in G is minimal (recall that the distance between trees T_1 and T_2 is $|T_1 \setminus T_2|$). The following result characterizes the complexity of Central Spanning Tree:

Theorem 6.1 ([25]). *The* Central Spanning Tree *problem is strongly NP-hard.*

From Theorem 1.10 it follows that if we introduce the interval weight $[0, 1]$ for every edge e of G, then Central Spanning Tree becomes equivalent to Minmax Regret Minimum Spanning Tree. That is an optimal robust spanning tree in G is the central spanning tree in G and vice versa. Hence, from Theorem 6.1, we immediately get the following result, which was first established by Aron and van Hentenryck [13] and independently by Averbakh and Lebedev [17]:

Theorem 6.2 ([13, 17]). *Problem* Minmax Regret Minimum Spanning Tree *is strongly NP-hard even if all edges have interval weights equal to* $[0, 1]$.

Thus the problem is computationally intractable even if all edges have the same uncertainty intervals equal to $[0, 1]$. Following Aron and van Hentenryck [13] we prove the following result:

Theorem 6.3. *Problem* Minmax Regret Minimum Spanning Tree *is strongly NP-hard in complete graphs, even if all bounds of the weight intervals are 0,1 or 2.*

Proof. Consider a connected graph $G = (V, E)$ with interval weights of edges equal to $[0, 1]$. By Theorem 6.2, computing the robust spanning tree in this graph is strongly NP-hard. Let us transform graph G into complete graph G' in the following way: for all $i, j \in V$, $i \neq j$, if $\{i, j\} \notin G$, then we add to G a *dummy* edge $\{i, j\}$ with interval weight equal to $[2, 2]$.

It is easy to see that every dummy edge is nonpossibly optimal. In other words, it cannot be a part of a minimum spanning tree under any scenario $S \in \Gamma$. In consequence, the optimal robust spanning tree in G' cannot use a dummy edge and the optimal robust spanning tree in G' is the same as in G (see Theorem 2.9). Hence adding dummy edges does not change the optimal robust solution in the original graph G and the problem remains strongly NP-hard in complete graphs. Observe that all bounds of weight intervals in the constructed graph are 0, 1 or 2. □

Applying the transformation of Minmax Regret Minimum Spanning Tree to Minmax Regret Maximum Spanning Tree, given in the proof of Theorem 1.5, we can see that Minmax Regret Maximum Spanning Tree is strongly NP-hard as well. This is also the case if all elements have interval weights equal to $[0, 1]$. These negative results show that Minmax Regret $\mathcal{P}$ is strongly NP-hard even if $\mathcal{P}$ has a matroidal structure.

6.2 Evaluation of Optimality

It is well known that the deterministic Minimum Spanning Tree is a special version a matroidal problem. Let $\mathcal{I}$ be the set of all forests of G. Recall that a *forest* is a subset of edges of G that forms an acyclic subgraph of G. System $(E, \mathcal{I})$ is a matroid and it is called in literature a *graphic matriod*. It is easy to see that a base of this matroid, that is the maximal acyclic subgraph of G, is a spanning tree of G. Therefore, if we introduce interval weights for the edges of G, we can use the results presented in Section 2.2.3 to detect efficiently the possibly and necessarily optimal edges. We can remove then all nonpossibly optimal edges from graph G and, in the absence of degeneracy, we can add all necessarily optimal edges to the constructed robust spanning tree.

The preprocessing, which consists in removing nonpossibly optimal edges and including the necessarily optimal ones, can be applied before solving the MIP formulation or as a reduction rule in the branch and bound algorithm. We will describe this preprocessing in the next sections.

6.2.1 Possibly and Necessarily Optimal Edges

Algorithms `Detect Pos` and `Detect Nec`, designed in Section 2.2.3, which assert whether a given edge is possibly or necessarily optimal, can be easily implemented to run in $\mathcal{O}(|E| \log |E|)$ time. In order to detect all possibly optimal edges we can use `Algorithm Detect All Pos`, which can be implemented to run in $\mathcal{O}(|E| \log |E| + |V|^2)$ time. Observe, that in dense graphs this running time becomes $\mathcal{O}(|E| \log |E|)$.

It is interesting to check how many nonpossibly and necessarily optimal edges may appear in a given graph. To answer this question we carried out some computational experiments. We used a family of graphs $G(|V|, \delta, u, l)$, where:

- $|V|$ denotes the number of nodes of G;
- $\delta \in (0,1]$ is the edge density, that is an edge between two nodes exists with probability δ. If $\delta = 1$, then G is a complete graph;
- for every edge $e \in E$ the value of $\underline{w}_e$ is a randomly selected integer from interval $[0, l)$ and the value of $\overline{w}_e$ is a randomly selected integer from interval $(\underline{w}_e, u]$.

A similar family of graphs was used by researches in [12, 104, 124]. Observe that all intervals in the sample graphs are nondegenerate. We carried out the computational experiments for graphs with $|V| \in \{10, 15, 20, 25\}$ and $\delta \in \{1, 0.8, 0.5\}$. For every combination of $|V|$ and δ we generated randomly five instances from every family: $G(|V|, \delta, 10, 10)$, $G(|V|, \delta, 15, 15)$, $G(|V|, \delta, 20, 20)$, $G(|V|, \delta, 10, 20)$, $G(|V|, \delta, 15, 30)$ and $G(|V|, \delta, 20, 40)$. Thus we generated 30 instances for every combination of $|V|$ and δ. The obtained results are shown in Table 6.1.

Table 6.1. The number of nonpossibly and necessarily optimal edges

			Nonposs.				Nec.			
$\|V\|$	δ	Edges	Min	Max	Aver.	Perc.	Min	Max	Aver.	Perc.
10	1	45	0	26	12	26.6%	0	3	1.06	2.3%
15	1	105	7	59	36.8	35.0%	0	4	0.8	0.7%
20	1	190	45	110	78.2	41.1%	0	3	0.76	0.4%
25	1	300	84	196	145.2	48.4%	0	4	0.7	0.2%
10	0.8	35	0	16	6.7	19.1%	0	5	1.2	3.4%
15	0.8	84.5	7	44	26.5	31.3%	0	4	1.1	1.3%
20	0.8	153.5	12	85	57.4	37.4%	0	3	0.9	0.5%
25	0.8	236	40	145	100	42,3%	0	4	1.3	0.5%
10	0.5	23	0	13	2.9	12.6%	0	6	1.9	8.2%
15	0.5	53	0	26	9.8	18.4%	0	4	1.7	3.2%
20	0.5	93.5	4	49	23.9	25.6%	0	5	1.9	2.0%
25	0.5	150	12	81	48.1	32.0%	0	5	1.8	1.2%

The third column of Table 6.1 contains the average number of edges in the generated graphs. Obviously, if $\delta = 1$, then every graph is complete and the number of edges is $|V|(|V|-1)/2$. For δ equal to 0.8 and 0.5 the number of edges may be different for every graph. For every class of graphs the reported minimal, maximal and average number of nonpossibly and necessarily optimal edges are shown. The average number is also expressed as a percent of the average number of the edges of a graph.

Notice that the number of nonpossibly optimal edges is quite large and we can expect that it is between 18%-48% of the edges of a graph. Removing all these edges may signicantly reduce the size of the problem and in consequence speed up the calculations of the optimal robust solution. We may also expect that a

few edges (typically one or two) are necessarily optimal and, in the absence of degeneracy, we can add all of them to the constructed robust spanning tree.

6.3 Mixed Integer Programming Formulation

In this section we construct a mixed integer programming model for solving MINMAX REGRET MINIMUM SPANNING TREE. This model is according to Yaman *et al.* [124] and is based on the general idea shown in Section 3.1. Let us assign numbers $1, 2 \ldots, |V|$ to the nodes of G. Define binary variable $x_{\{i,j\}} \in \{0,1\}$ for every edge $e = \{i,j\} \in E$, which expresses whether edge e is contained in the constructed spanning tree. Let us create for every edge $\{i,j\}$ two arcs (i,j) and (j,i) and denote by A the set of all such arcs. Define also $V^* = V \setminus \{1\}$. Consider the following set of constraints:

$$
\begin{aligned}
&\sum_{\{j:\,(1,j)\in A\}} f_{1j} - \sum_{\{j:\,(j,1)\in A\}} f_{j1} = |V| - 1 && \\
&\sum_{\{j:\,(i,j)\in A\}} f_{ij} - \sum_{\{j:\,(j,i)\in A\}} f_{ji} = -1 && \text{for } i \in V^* \\
&f_{ij} \le (|V|-1)x_{\{i,j\}} && \text{for } \{i,j\} \in E \\
&f_{ji} \le (|V|-1)x_{\{i,j\}} && \text{for } \{i,j\} \in E \\
&\sum_{\{i,j\}\in E} x_{\{i,j\}} = |V| - 1 && \\
&f_{ij} \ge 0 && \text{for } (i,j) \in A \\
&x_{\{i,j\}} \in \{0,1\} && \text{for } \{i,j\} \in E
\end{aligned}
\tag{6.1}
$$

Constraints (6.1) describe a *single commodity model*, which was discussed by Magnanti and Wosley [98]. In this model we wish to send one unit of flow from node 1 (chosen arbitrarily) to every other node of G. Hence node 1 serves as a source node and it has $|V| - 1$ units of flow that must be sent. The variable f_{ij} denotes the flow on edge $\{i,j\}$ in direction (i,j). The constraints of (6.1) assure that the subgraph induced by every feasible solution $(x_e)_{e\in E}$ is connected and has exactly $|V|-1$ edges. Thus it is a spanning tree of G. Conversely, every solution $(x_e)_{e\in E}$ that represents a spanning tree of G is a feasible solution to (6.1). Hence the constraints (6.1) define set $ch(\Phi)$ in the problem (see Section 3.1). Unfortunately, formulation (6.1) is not quite appropriate for our purpose since the constraints matrix $\boldsymbol{A}$ of (6.1) is not totally unimodular. However, we can still use formulation (6.1) do describe set $ch(\Phi)$.

We present now another, equivalent formulation, which is also due to Magnanti and Wolsey [98]. This formulation is called a *directed multicommodity flow* model. Every node $k \neq 1$ of graph G defines now a commodity. One unit of commodity $k \in V^*$ originates at root node 1 and must be delivered to node k. Variable f_{ij}^k denotes the flow of commodity k on edge $\{i,j\} \in E$ in direction from i to j. The formulation is as follows:

$$
\begin{aligned}
&\sum_{\{j:(j,1)\in A\}} f^k_{j1} - \sum_{\{j:(1,j)\in A\}} f^k_{1j} = -1 && \text{for } k \in V^* \\
&\sum_{\{j:(j,i)\in A\}} f^k_{ji} - \sum_{\{j:(i,j)\in A\}} f^k_{ij} = 0 && \text{for } i,k \in V^* \text{ and } i \neq k \\
&\sum_{\{j:(j,k)\in A\}} f^k_{jk} - \sum_{\{j:(k,j)\in A\}} f^k_{kj} = 1 && \text{for } k \in V^* \\
&f^k_{ij} \leq x'_{ij} && \text{for } (i,j) \in A \\
&\sum_{(i,j)\in A} x'_{ij} = |V| - 1 \\
&f^k_{ij} \geq 0 && \text{for } (i,j) \in A, k \in V^* \\
&x_{\{i,j\}} = x'_{ij} + x'_{ji} && \text{for } \{i,j\} \in E \\
&x'_{ij} \in \{0,1\} && \text{for } (i,j) \in A
\end{aligned}
\tag{6.2}
$$

It can be shown (see Magnanti and Wolsey [98]) that a feasible vector $(x_e)_{e\in E}$ to (6.2) is a characteristic vector of a spanning tree of G. Conversely, suppose $(x_e)_{e\in E}$ is a characteristic vector of a spanning tree of G. We can direct the edges of the tree so that the path from the root node (node 1) to any other node is directed. Setting $x'_{ij} = 1$ for the arcs that correspond to this directed tree and $x'_{ij} = 0$ for all remaining arcs we obtain a feasible solution to (6.2). That is a commodity k is delivered from node 1 to node k on a directed path induced by variables x'_{ij}. The constraints matrix $\boldsymbol{A}$ of (6.2) is now totally unimodular and we can use formulation (6.2) to construct the relaxed subproblem $\min_{\boldsymbol{y}} \phi(\boldsymbol{x}, \boldsymbol{y})$, which takes the following form:

$$
\begin{aligned}
&\min \sum_{\{i,j\}\in E} (\overline{w}_{\{i,j\}} x_{\{i,j\}} + \underline{w}_{\{i,j\}}(1 - x_{\{i,j\}}))(y'_{ij} + y'_{ji}) \\
&\sum_{\{j:(j,1)\in A\}} f^k_{j1} - \sum_{\{j:(1,j)\in A\}} f^k_{1j} = -1 && \text{for } k \in V^* \\
&\sum_{\{j:(j,i)\in A\}} f^k_{ji} - \sum_{\{j:(i,j)\in A\}} f^k_{ij} = 0 && \text{for } i,k \in V^* \text{ and } i \neq k \\
&\sum_{\{j:(j,k)\in A\}} f^k_{jk} - \sum_{\{j:(k,j)\in A\}} f^k_{kj} = 1 && \text{for } k \in V^* \\
&f^k_{ij} \leq y'_{ij} && \text{for } (i,j) \in A \\
&\sum_{(i,j)\in A} y'_{ij} = |V| - 1 \\
&f^k_{ij} \geq 0 && \text{for } (i,j) \in A, k \in V^* \\
&y'_{ij} \geq 0 && \text{for } (i,j) \in A
\end{aligned}
\tag{6.3}
$$

Observe, that we have replaced $y'_{ij} \in \{0,1\}$ with $y'_{ij} \geq 0$. This follows from the fact that constraints of (6.3) imply $y'_{ij} \leq 1$ for all $(i,j) \in A$ in every optimal solution to (6.3). We have also substituted $y_{\{i,j\}} = y'_{ij} + y'_{ji}$ in the objective

of (6.3). Now, assigning dual variables $\boldsymbol{\lambda}$ to the constraints of (6.3) we get the following dual to (6.3), that is $\max_{\boldsymbol{\lambda}} \phi^*(\boldsymbol{x}, \boldsymbol{\lambda})$:

$$
\begin{aligned}
&\max \sum_{k \in V^*} (\alpha_k^k - \alpha_1^k) + (|V| - 1)\mu \\
&\sigma_{ij}^k \geq \alpha_j^k - \alpha_i^k && \text{for } k \in V^*, (i,j) \in A \\
&\sum_{k \in V^*} \sigma_{ij}^k + \mu \leq \overline{w}_{\{i,j\}} x_{\{i,j\}} + \underline{w}_{\{i,j\}} (1 - x_{\{i,j\}}) && \text{for } \{i,j\} \in E \\
&\sum_{k \in V^*} \sigma_{ji}^k + \mu \leq \overline{w}_{\{i,j\}} x_{\{i,j\}} + \underline{w}_{\{i,j\}} (1 - x_{\{i,j\}}) && \text{for } \{i,j\} \in E \\
&\sigma_{ij}^k \geq 0 && \text{for } k \in V^*, (i,j) \in A \\
&\alpha_i^k \geq 0 && \text{for } i, k \in V^*
\end{aligned}
\tag{6.4}
$$

Now, from (3.6), (6.1) and (6.4) we get the following MIP formulation for Minmax Regret Minimum Spanning Tree (see also [124]):

$$
\begin{aligned}
&\min \sum_{\{i,j\} \in E} \overline{w}_{\{i,j\}} x_{\{i,j\}} - \sum_{k \in V^*} (\alpha_k^k - \alpha_1^k) - (|V| - 1)\mu \\
&\sigma_{ij}^k \geq \alpha_j^k - \alpha_i^k && \text{for } k \in V^*, (i,j) \in A \\
&\sum_{k \in V^*} \sigma_{ij}^k + \mu \leq \overline{w}_{\{i,j\}} x_{\{i,j\}} + \underline{w}_{\{i,j\}} (1 - x_{\{i,j\}}) && \text{for } \{i,j\} \in E \\
&\sum_{k \in V^*} \sigma_{ji}^k + \mu \leq \overline{w}_{\{i,j\}} x_{\{i,j\}} + \underline{w}_{\{i,j\}} (1 - x_{\{i,j\}}) && \text{for } \{i,j\} \in E \\
&\sum_{\{j:(1,j) \in A\}} f_{1j} - \sum_{\{j:(j,1) \in A\}} f_{j1} = |V| - 1 \\
&\sum_{\{j:(i,j) \in A\}} f_{ij} - \sum_{\{j:(j,i) \in A\}} f_{ji} = -1 && \text{for } i \in V^* \\
&f_{ij} \leq (|V| - 1) x_{\{i,j\}} && \text{for } \{i,j\} \in E \\
&f_{ji} \leq (|V| - 1) x_{\{i,j\}} && \text{for } \{i,j\} \in E \\
&\sum_{\{i,j\} \in E} x_{\{i,j\}} = |V| - 1 \\
&\sigma_{ij}^k \geq 0 && \text{for } k \in V^*, (i,j) \in A \\
&\alpha_i^k \geq 0 && \text{for } i, k \in V^* \\
&f_{ij} \geq 0 && \text{for } (i,j) \in A \\
&x_{\{i,j\}} \in \{0,1\} && \text{for } \{i,j\} \in E
\end{aligned}
\tag{6.5}
$$

Model (6.5) can be solved by means of a standard software. In order to speed up calculations we can detect first all nonpossibly optimal edges and set $x_{\{i,j\}} = 0$ for all variables that correspond to them. Moreover, in the absence of degeneracy we can set $x_{\{i,j\}} = 1$ for all necessarily optimal edges $\{i,j\} \in E$.

Table 6.2. The average computational times in seconds required for solving the problem

$\|V\|$	δ	CPU Time
10	1	3.31
15	1	143.44
20	1	2258.8
10	0.8	2.83
15	0.8	44.90
20	0.8	1406.87
10	0.5	1.137
15	0.5	62.63
20	0.5	797.34

6.3.1 Computational Results

In this section we present some computational results using the constructed MIP formulation. In our tests we used the same set of problems as in Section 6.2.1. All instances were first preprocessed by removing all nonpossibly optimal edges and by adding the necessarily optimal ones. Table 6.2 demonstrates the average CPU times in seconds required to solve the problem for every combination of $|V|$ and δ. In order to solve the problems we used CPLEX 8.0 and a Pentium III 866MHz computer. Observe that the required CPU time grows very quickly with the problem size. For $|V| = 25$ only one instance was solved and it required 15744 seconds (about 4.3 hours). Some results of computational tests on the MIP formulation can also be found in the paper by Yaman *et al.* [124].

The obtained results suggest that the MIP formulation allows us to solve the problem for rather small graphs. In the next two sections we construct two algorithms that allow us to solve larger problems. The first one is the branch and bound, which can be applied to the problem for graphs having up to 40 nodes and the second is a local search algorithm, which can be applied to larger graphs.

6.4 Branch and Bound Algorithm

In this section we fill all details of the branch and bound algorithm, described in Section 3.2, to solve the problem. In order to do this we must specify a branching method, reduction rules and a method of deriving a feasible solution from a given subset $\Phi_{<Q,R>}$. Recall that $\Phi_{<Q,R>}$ contains all spanning trees T of G such that $Q \subseteq T$ and no edge from R is in T. Structure $< Q, R >$ describes a node d of the search tree.

6.4.1 Branching Method

Assume that we have chosen a node $d =< Q, R >$ of the search tree to be branched. We describe a method of branching, which has been proposed by

Aron and van Hentenryck [12]. Let us denote $L = E \setminus (Q \cup R)$, that is L is the subset of edges that are neither in Q nor in R. We need to derive an edge from L to branch node d. Let T be a minimum spanning tree under scenario $S^+_{E\setminus R}$. If $L \cap T \neq \emptyset$, then we chose element $e \in L \cap T$ of the maximal value of $\overline{w}_e - \underline{w}_e$; otherwise ($L \cap T = \emptyset$) we choose element $e \in L$ of the maximal value of $\overline{w}_e - \underline{w}_e$. We branch d by creating two nodes $< Q \cup \{e\}, R >$ and $< Q, R \cup \{e\} >$. Using this method of branching together with the lower bound given by Theorem 3.3, we can expect that the lower bounds for the two new nodes will be greater than the lower bound of the parent node d.

An alternative and more sophisticated method of branching was proposed by Montemanni an Gambardella [104] and we refer the reader to [104] for details.

6.4.2 Reduction Rules

The first reduction rule follows directly from Theorem 2.9 and Theorem 2.13. We can detect in polynomial time the set $A \subseteq E$ of all nonpossibly optimal edges and the set $B \subseteq E$ of all necessarily optimal edges. If all weight intervals are nondegenerate, then we can define the root node of the search tree as $< B, A >$, that is we select all necessarily optimal edges and reject all nonpossibly optimal ones. If there are some degenerate intervals in the problem, then we can start with root node $< \{e\}, A >$, where $e \in B$, since we can always select a single necessarily optimal edge. Hence the first reduction rule can be applied to the root node of the search tree.

Let $d =< Q, R >$ be an internal node of the search tree. Consider edge $f \in E \setminus (Q \cup R)$. If set $Q \cup \{f\}$ contains a cycle, then no spanning tree can contain all edges from $Q \cup \{e\}$. Therefore, edge f can be added to R. Hence we have established another reduction rule, which prunes the infeasible solutions. This reduction rule can be additionally strengthened. Suppose $Q \cup \{f\}$ does not contain a cycle. However, there may be no possibly optimal spanning tree that contains all edges from $Q \cup \{f\}$. In this case (see Proposition 2.4) there is no optimal robust solution that contains all edges from $Q \cup \{f\}$ and edge f can be added to R. The problem of checking whether there is a possibly optimal solution that contains all edges from $Q \cup \{f\}$ can be solved in $\mathcal{O}(|E| \log |E|)$ time by a slightly modified algorithm `Detect Pos` designed in Section 2.2.3. We only need to modify line 2 of this algorithm by initializing $X \leftarrow Q$. The proof of the correctness of this method was given by Montemanni and Gambardella in [104].

6.4.3 Deriving a Feasible Solution

For every node $d =< Q, R >$ of the search tree we seek a feasible solution $T_{<Q,R>}$ that contains all edges from Q and does not contain any edge of R. This solution can be found by solving the deterministic Minimum Spanning Tree problem for some fixed scenario $S \in \Gamma$. One of the candidate scenarios is the midpoint scenario, that is the one in which $w^S_e = \frac{1}{2}(\underline{w}_e + \overline{w}_e)$ for all $e \in E$. Some other scenarios, like S^+_E, can also be used. Having fixed a scenario S we proceed as follows: first we initialize $T_{<Q,R>} \leftarrow Q$; then we greedily add the edges from $E \setminus R$ to $T_{<Q,R>}$

in nondecreasing order of weights under S until a spanning tree is obtained. The spanning tree $T_{<Q<R>}$ is a candidate solution from set $\Phi_{<Q,R>}$.

6.4.4 Computational Results

We characterize now the performance of the branch and bound algorithm by describing some computational experiments that were presented in literature. Table 6.3 demonstrates the average computational times for graphs with the number of nodes varying from 10 to 40. All tests was performed for complete graphs with weights intervals specified as in Section 6.2.1. In the column MIP the average time required to solve the MIP formulation (using CPLEX 6.0) and in the column BB the average time required for the branch and bound algorithm are shown. We can see that the branch and bound algorithm, which exploits the particular structure of the problem, is significantly faster than the MIP approach and allows us to solve the problem for larger graphs. However, the required CPU time grows very quickly while increasing the number of nodes and it is quite large for graphs with 40 nodes. The performance of the branch and bound algorithm may be poor for larger graphs, having hundreds of nodes. The results shown in Table 6.3 also confirm the rather poor performance of the MIP formulation. Therefore, an alternative algorithm for large graphs should be designed.

Table 6.3. The average computational times in seconds required to solve the problem. The times marked with $*$ were obtained in [104] using Pentium II PC 400 MHz. The times marked with $**$ were obtained in [12] using Sun Sparc 440 MHz.

$\|V\|$	MIP	BB
10	0.85	0.04^{*}
15	286.47	1.84^{*}
20	>2016.28	56.46^{*}
25	>3600	484.24^{*}
30	-	442^{**}
35	-	2171^{**}
40	-	10324^{**}

6.5 Local Search Algorithms

One of the most successful methods of attacking hard combinatorial optimization problems is *local search*. The local search is an iterative procedure which moves from one solution to another as long as is necessary. Every local search technique is based on the concept of a *neighborhood function*, that is a mapping N, which for every $X \in \Phi$ defines the subset of solutions $N(X) \subseteq \Phi$, which can be reached in one step by moving from X. The set $N(X)$ is called a *neighborhood* of X. A local search algorithm starts from a certain feasible solution $X_0 \in \Phi$ and performs a sequence of moves, which consist in choosing a solution $X_{i+1} \in N(X_i)$. By applying different methods of choosing a solution and stopping criteria we

obtain some various types of local search algorithms like iterative improvement, simulated annealing, threshold acceptance or tabu search. In this section our goal is to construct a tabu search algorithm, which can be used to obtain good solutions for large problems.

6.5.1 Neighborhood Function

We now use the fact that for MINMAX REGRET MINIMUM SPANNING TREE there is a very natural definition of the neighborhood function. We define

$$N(T) = \{T_1 \in \Phi : \ |T \setminus T_1| = 1\}.$$

Hence $N(T)$ is a *1-exchange-neighborhood* and it consists of all spanning trees differing from T in exactly a single edge. The neighborhood of a spanning tree T can be listed by means of the algorithm presented in Figure 6.2. This algorithm can be implemented efficiently by applying some techniques and data structures described for instance in [2].

The correctness of algorithm `Neighborhood` follows form the matroidal structure of the problem. For every edge $\{i, j\} \in E \setminus T$ set $T \cup \{i, j\}$ contains an unique circuit (a simple cycle in G), which is formed by edge $\{i, j\}$ and the unique path

`Neighborhood`
Require: Graph $G = (V, E)$, a spanning tree T of G.
Ensure: Neighborhood $N(T)$.
1: $N(T) \leftarrow \emptyset$
2: **for all** $\{i, j\} \in E \setminus T$ **do**
3: Determine the set of edges $\{f_1, \ldots f_k\}$ that are on the path from i to j in T
4: **for all** $f \in \{f_1, \ldots, f_k\}$ **do**
5: Add $T \cup \{i, j\} \setminus f$ to $N(T)$
6: **end for**
7: **end for**
8: **return** $N(T)$

Fig. 6.2. Listing the neighborhood of a given spanning tree T

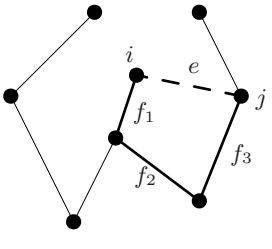


Fig. 6.3. A sample move. We add edge $e = \{i, j\}$ and remove an edge that is on path $\{f_1, f_2, f_3\}$ from i to j.

from i to j in T. A new spanning tree can be obtained by removing any edge on this path (see Figure 6.3).

6.5.2 Iterative Improvement

The simplest local search algorithm, called *iterative improvement*, starts from a given spanning tree T_0 and always chooses a solution from the neighborhood of the smallest value of the maximal regret. It terminates at a *local minimum* T, that is when it meets a solution T such that $Z(T_1) \geq Z(T)$ for all $T_1 \in N(T)$. Clearly, the local minimum may be not an optimal robust solution. Further in this section we provide the worst case analysis of the local minimum with respect to the *1-exchange-neighborhood*. The iterative improvement algorithm applied to the problem is shown in Figure 6.4.

```
Iterative Improvement
Require: Graph G = (V, E) with interval weights of edges.
Ensure: A local minimum T.
 1: Compute an initial spanning tree T using Algorithm AM
 2: STOP← false
 3: while STOP=false do
 4:    Find a spanning tree T1 ∈ N(T) of the smallest value of Z(T1)
 5:    if Z(T1) < Z(T) then
 6:       T ← T1 {Perform the move}
 7:    else
 8:       STOP← true {T is a local minimum}
 9:    end if
10: end while
11: return T
```

Fig. 6.4. The iterative improvement algorithm for Minmax Regret Minimum Spanning Tree

The algorithm `Iterative Improvement` scans in line 4 the whole neighborhood of a current spanning tree T and computes the maximal regret for every $T_1 \in N(T)$. A direct computation of the maximal regret requires solving the deterministic Minimum Spanning Tree problem. Therefore the calculations in line 4 may be time consuming. We now show that the computations may be significantly speeded up.

Let T be a current spanning tree and let T^* be a worst case alternative for T. Recall that T^* is the optimal solution under scenario S_T^+. Suppose that we explore a solution that has been obtained by adding edge $e \in E \setminus T$ and removing edge $f \in T$, so that $T_1 = T \cup \{e\} \setminus \{f\}$. It holds

$$F(T_1, S_{T_1}^+) = F(T, S_T^+) + \overline{w}_e - \overline{w}_f.$$

We now focus on computing the value of $F^*(S_{T_1}^+)$. Our goal is to quickly compute spanning tree T_1^*, that is the minimum spanning tree under scenario $S_{T_1}^+$, having spanning tree T^*.

Let σ be the sequence of edges sorted in nondecreasing order of weights under scenario S_T^+. We get spanning tree T^* by applying the greedy algorithm to σ. Scenario $S_{T_1}^+$ is obtained from S_T^+ by increasing the weight of e to $\overline{w}_e$ and decreasing the weight of f to $\underline{w}_f$. Therefore, in order to keep the order of elements under $S_{T_1}^+$, we move element e a number of positions to the right and we move element f a number of positions to the left in σ. Denote the resulting new sequence of elements by σ' and T_1^* can be obtained by applying the greedy algorithm to σ'. Assume that $e \notin T^*$. Since $pred(\sigma, e) \subseteq pred(\sigma', e)$, Proposition 2.14 implies $e \notin T_1^*$. Therefore, we must only check whether edge f belongs to T_1^*. This can be done as follows: if $f \in T^*$, then $T_1^* = T^*$; otherwise $T^* \cup \{f\}$ contains a unique simple cycle composed of edges $\{f, g_1, \ldots, g_k\}$; if there is g_i whose weight under $S_{T_1}^+$ is greater than $\underline{w}_f$, then $T_1^* = T^* \cup \{f\} \setminus \{g_i\}$; otherwise $T_1^* = T^*$. Thus if $e \notin T^*$, then the maximal regret $Z(T_1)$ can be computed in $\mathcal{O}(|V|)$ time, which is required to detect a cycle in $T^* \cup \{f\}$. If $e \in T^*$, then we solve the deterministic minimum spanning tree problem under scenario $S_{T_1}^+$ to obtain T_1^*. However, this must be only done $|V| - 1$ times since $|T^*| = |V| - 1$ and one may use sequence σ to speed up calculations (it is not necessary to sort the edges every time T_1^* is computed).

It is clear that the quality of the solution returned by `Iterative Improvement` strongly depends on the initial spanning tree computed in line 1. We can now use the results obtained in Chapter 4 to assure that the initial spanning tree is of good quality. Namely, we can choose a spanning tree T computed by `Algorithm AM`, which assures that $Z(T) \leq 2OPT$, where OPT is the maximal regret of the optimal robust spanning tree. In the next section we explore the quality of the local minimum returned by `Iterative Improvement`.

6.5.3 Local Minimum

We now discuss the worst case performance of `Iterative Improvement` by exploring the quality of the local minimum returned by this algorithm. Since we start from the solution computed by `Algorithm AM`, it is obvious that `Iterative Improvement` returns a solution T such that $Z(T) \leq 2OPT$. Unfortunately, it is possible to construct an instance of Minmax Regret Minimum Spanning Tree, for which the bound of 2 is asymptotically attained.

Consider an instance of the problem shown in Figure 6.5. The input graph $G = (V, E)$ is formed by a complete graph composed of nodes $1, 2, \ldots, m$; two nodes a, b with edges $\{a, 1\}$, $\{a, 2\}$, $\{b, 3\}$ and $\{b, 4\}$; nodes $5', \ldots, m'$ with edges $\{5', 5\}, \ldots, \{m', m\}$ and $\{b, 5'\}, \{5', 6'\}, \ldots, \{m', a\}$. The interval weights of all edges are $[0, 1]$ (these weights are not shown in Figure 6.5). Consider the spanning tree T shown in Figure 6.5a. This tree is composed of $2m-3$ edges, thus $F(T, S_T^+) = 2m - 3$. If we remove all edges of tree T from G, then the resulting graph G' (see Figure 6.5b) is still connected. Hence the worst case alternative T^* for T is such that $F(T^*, S_T^+) = 0$. So, $Z(T) = 2m - 3$. Consider now a spanning tree T_1, which is the result of a move from T. This move consists of adding an edge $e \in E \setminus T$ to T and removing an edge $f \in T$ from T. However, observe that it is not possible to disconnect graph G' by removing a single edge from

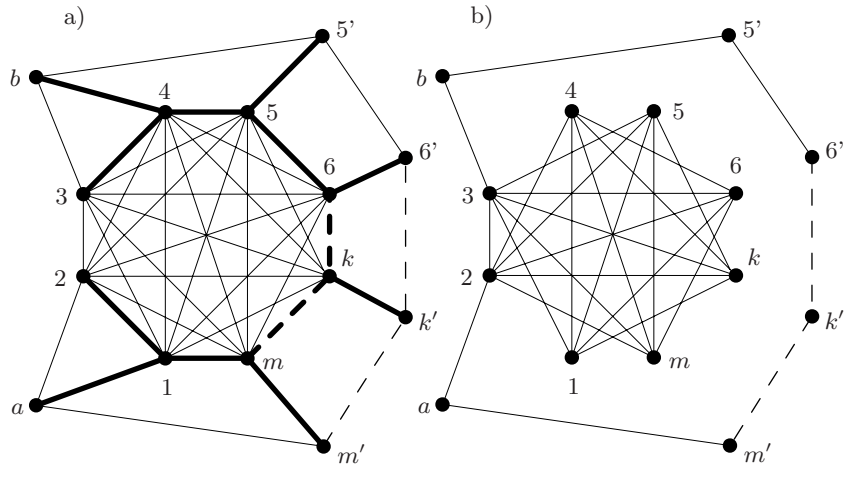


Fig. 6.5. A local robust spanning tree (in bold). All edges have interval weights equal to $[0, 1]$.

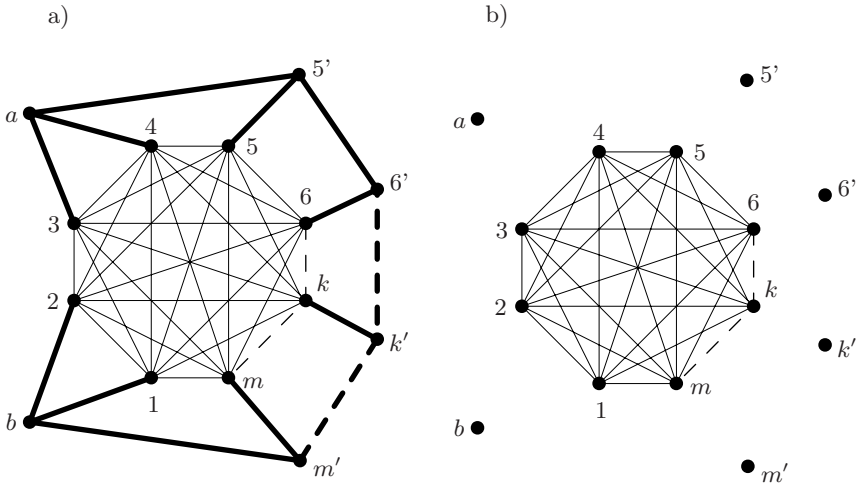


Fig. 6.6. The optimal robust spanning tree (in bold). All edges have interval weights equal to $[0, 1]$.

G'. Hence, removing T_1 from G also forms a connected graph and consequently $F^*(S^+_{T_1}) = 0$. Now it is clear that $Z(T_1) = Z(T) = 2m - 3$ and T is a local minimum.

Consider now the spanning tree R shown in Figure 6.6a. This tree is the optimal robust spanning tree in the sample graph. By removing R from G we obtain a graph G' (see Figure 6.6b), that is composed of $m - 1$ connected components. Hence the worst case alternative R^* for R is such that $F(R^*, S^+_R) = m - 2$. So,

$Z(R) = 2m - 3 - m + 2 = m - 1$. It holds $Z(T)/Z(R) = (2m-3)/m - 1$, which is asymptotically equal to 2 for large m. We have thus established the following result:

Theorem 6.4. *For the 1-exchange-neighborhood algorithm* ITERATIVE IMPROVEMENT *returns a spanning tree T such that $Z(T) \leq 2OPT$ and the bound of 2 is asymptotically attained even if all weight intervals are* $[0, 1]$.

6.5.4 Tabu Search Algorithm

Algorithm `Iterative Improvement`, constructed in the previous section, always terminates at a local minimum with respect to the *1-exchange-neighborhood*. As we have shown in the previous section, this local minimum may be in the worst case a factor of 2 away from a global one. In this section we construct an algorithm that avoids being trapped in a local minimum and allows a search of a greater part of the solution space. This algorithm is based on the idea of *tabu search*, which has been successfully applied to many hard, combinatorial optimization problems. The tabu search algorithm for the problem is shown in Figure 6.7. We now describe all its details.

Tabu list

In the tabu search algorithm we also start from a spanning tree constructed by 2-approximation `Algorithm AM`. We perform a move from the current solution T to solution $T_1 \in N(T)$, that has the smallest value of the maximal regret. We do not terminate at local minimum and we allow a move to a solution T_1 such that $Z(T_1) \geq Z(T)$. However, it is possible now to get back to solutions already visited. Therefore, an oscillation around a local minimum is possible. A simple way to avoid such a problem is to store the information about the performed moves in a so called *tabu list*. The tabu list contains a description of the moves that are forbidden for a certain number of iterations.

Let T be a given spanning tree of graph $G = (V, E)$. Suppose that we have performed a move by adding edge $e \in E \setminus T$ to T and removing edge f from T. Hence we have obtained a spanning tree $T_1 = T \setminus \{f\} \cup \{e\}$ from $N(T)$. In order to avoid getting back to T we add to the tabu list two elements: $(add(f), iter(f))$ and $(drop(e), iter(e))$. This means, that we forbid adding edge f to the current solution for $iter(f)$ iterations and dropping edge e from the current solution for $iter(e)$ iterations. In consequence, every move in which we add edge f or remove edge e is forbidden for a certain number of iterations.

Aspiration criterion

An important element of tabu search is the incorporation of an *aspiration criterion*. Suppose we want to perform a move from T to $T_1 \in N(T)$, which is forbidden by the tabu list. However, solution T_1 may be better than the current

```
Tabu Search
Require: Graph G = (V, E) with interval weights of edges.
Ensure: A spanning tree T_best of G.
 1: Compute an initial spanning tree T of G using Algorithm AM
 2: T_best ← T, Z_best ← Z(T)
 3: TABU ← ∅, E* ← T*, k ← 0
 4: while stop criterion=false do
 5:    N̄(T) ← {T_1 ∈ N(T) : the move to T_1 is not forbidden or Z(T_1) < Z_best}
 6:    Find a spanning tree T_1 ∈ N̄(T) of the smallest value of Z(T_1)
 7:    k ← k + 1
 8:    if Z(T_1) < Z_best then
 9:       T_best ← T_1, Z_best ← Z(T_1) {A better solution has been found}
10:       E* ← E* ∪ T_1*
11:       k ← 0
12:    end if
13:    if k > k_max then
14:       Compute a spanning tree T of G = (V, E*) using Algorithm AM
15:       if Z(T) < Z_best then
16:          T_best ← T, Z_best ← Z(T) {A better solution has been found}
17:       end if
18:       Go to line 3 {Restart algorithm}
19:    else
20:       T ← T_1 {Perform the move}
21:       Update TABU
22:    end if
23: end while
24: return T_best
```

Fig. 6.7. The tabu search algorithm for Minmax Regret Minimum Spanning Tree

best solution and in this case the move should be performed. The aspiration criterion is a condition that allows us to perform a move even though it is forbidden by the tabu list. In our algorithm we use the following simple aspiration criterion: we always allow a move that leads to a solution better than the current best solution.

Long-term memory function

The tabu list together with the aspiration criterion are so called short-term memory functions of the tabu search algorithm. This means that the information about search process, stored in a tabu list, is lost after a small number of iterations. Because the search strongly tends to focus on locally good solutions, only a small region of the solution space may be explored.

In order to achieve a global diversification of the search some long-term memory functions can be employed. In a long-term memory function we construct a solution using information stored during the whole search process. In our algorithm we incorporate the following method. We introduce a subset of edges $E^* \subseteq E$.

For the initial solution T and every time T improves the current best solution we add to E^* all edges that belong to T^* (that is to the worst case alternative of T). After a fixed number of iterations, during which the current best solution has not been improved, we restart the algorithm. This restart consists in generating a new solution by applying `Algorithm AM` to subgraph $G^* = (V, E^*)$ induced by E^*.

Observe that the distance between T and T^* (that is $|T \backslash T^*|$) is typically large. Therefore, we may expect that a new spanning tree, deriven from subgraph $G^* = (V, E^*)$, allows us to move the search process to another region of the solutions space. Moreover, a spanning tree composed of the edges of the worst case alternatives of the currently best solutions may be itself a good solution.

6.5.5 Computational Results

We now demonstrate the results of the computational tests. We used the same family of graphs as in Section 6.2.1. We carried out the experiments only for $|V|$ equal to 10, 15 and 20 since only for these graphs were we able to compute the optimal solutions using MIP formulation in reasonable time. We tested three approximation algorithms: `Algorithm AM`, `Algorithm AMU` and `Tabu Search`. In `Tabu Search` we used the following parameters: the algorithm terminated after performing 10 000 moves; a new solution was generated (the algorithm was restarted) when the current best solution was not improved within 500 moves ($k_{max} = 500$); every performed move (that is adding a just removed edge or removing a just added edge) was forbidden for $|V|/2 + 1$ iterations. The results of the tests are shown in Table 6.4. For every combination of $|V|$ and δ the worst and the average reported deviations from optimum are presented.

Algorithm `Tabu Search` returned the optimal robust spanning tree for almost all generated instances. This suggests that `Tabu Search` should be regarded as a good method of solving the problem. The tested instances were rather small but the algorithm can be applied to large graphs as well. Observe that two simple approximation algorithms: `Algorithm AM` and `Algorithm AMU` also returned solutions that are mostly close to optimum. It is worth pointing out that for

Table 6.4. The computational results

		`Algorithm AM`		`Algorithm AMU`		`Tabu Search`	
$\|V\|$	δ	Worst %	Aver. %	Worst %	Aver. %	Worst %	Aver. %
10	1	9.68	2.62	8.11	1.17	0.00	0.00
15	1	15.54	5.29	7.69	1.25	0.00	0.00
20	1	8.75	5.55	5.77	3.45	1.63	0.11
10	0.8	19.23	3.33	7.69	1.32	0.00	0.00
15	0.8	9.09	2.93	1.39	5.26	2.90	0.19
20	0.8	11.49	3.7	5.45	1.41	1.70	0.05
10	0.5	24.00	2.43	24.00	2.04	0.00	0.00
15	0.5	12.87	3.00	6.25	1.80	0.00	0.00
20	0.5	9.58	2.98	5.56	1.48	0.00	0.00

the tested instances `Algorithm AMU` behaved much better than `Algorithm AM`. Its average worst case performance for the tested instances was between 1 to 6 percent off the optimum. Since the running time of the heuristics is reasonable even if the input graphs are very large, they should be regarded as alternative methods for the exact algorithms.

6.6 Notes and References

The deterministic Minimum Spanning Tree is one of the oldest problems in combinatorial optimization. The first algorithm for solving Minimum Spanning Tree was developed by Boruvka [27] in 1926 and it is among the earliest algorithms for combinatorial optimization problems. Now, two simple algorithms for that problem are commonly used: Prim's [114] and Kruskal's [89] algorithms. Both are versions of a greedy algorithm that at each step add an edge of the minimum cost from a candidate list. The fastest minimum spanning tree algorithm up to now was developed by Chazelle [35]. Its running time is $\mathcal{O}(|E|\alpha(|E|,|V|))$, where α is an inverse-Ackermann function. This running time is very close to $\mathcal{O}(|E|)$. However, developing the fastest possible algorithm for Minimum Spanning Tree is one of the oldest open problems in computer science.

The Minmax Regret Minimum Spanning Tree problem was first studied by Yaman *et al.* [124]. When the paper [124] was published, the complexity status of the problem was unknown. Yaman *et al.* [124] proposed the MIP formulation, which is presented in Section 6.3. They also introduced the concepts of weak (possibly optimal), strong (necessarily optimal) edges and established the preprocessing rules for the problem. These rules were applied before solving the constructed MIP model and they allow a significant speed up in the calculations of the optimal solution. It follows from the fact that the number of nonweak (nonpossibly optimal) edges in a graph may be quite large (see Section 6.2.1). The proof of NP-hardness of Minmax Regret Minimum Spanning Tree was given independently by Aron and van Hentenryck [13] and Averbakh and Lebedev [17]. Aron and van Hentenryck [12] estabilished the connections between Minmax Regret Minimum Spanning Tree and Central Spanning Tree. The proof of Averbakh and Lebedev [17] was by a reduction from the Exact Cover By 3-Sets.

The MIP model constructed by Yaman *et al.* [124] was the first exact method of solving the problem. However, this method turned out to be efficient for rather small graphs (having up to 25 nodes). The branch and bound algorithms constructed by Aron and van Hentenryck [12] and Montemanni and Gambardella [104] are more efficient and allow us to solve the problems for graphs having up to 40 nodes. For larger problems the 2-approximation algorithms constructed by Kasperski and Zieliński [80] can be applied (see Section 4.1). Finally, for large graphs the algorithm `Tabu Search` designed in Section 6.5 can also be used. The framework of the tabu search technique can be found in a book by Glover and Laguna [59] and in the papers by Glover [57, 58]. A review of different large scale local search techniques can also be found in a paper by Ahuja *et al.* [3].

There are still some open questions connected with the problem. We know that it is strongly NP-hard and remains strongly NP-hard for complete graphs (Aron and van Hentenryck [13]). However, there are some special classes of graphs, for instance planar graphs, series-parallel graphs or graphs with a bounded node degree, for which the complexity status of the problem is unknown. The local search algorithms, in particular the `Tabu Search` presented in Section 6.5, seem to be a good tool to solve the problem. However, more exhaustive computational tests are required to evaluate the performance of these algorithms. There are also some open questions about the approximation. The problem is approximable within 2. But no algorithms with better worst case behavior are known. It would be interesting to design such algorithms even for Central Spanning Tree, which seems to be a core problem in which the structure of set Φ plays a central role.

7 Minmax Regret Shortest Path

This chapter is devoted to MINMAX REGRET SHORTEST PATH. In this problem we wish to find a path between two nodes s and t of a given graph, which minimizes the maximal regret. We will distinguish two cases. In the first case E is the set of arcs of a given directed graph (*digraph*) $G = (V, A)$, that is $E = A$, and a path P is a subset of arcs of the form $\{(s, v_1), (v_1, v_2), (v_2, v_3), \ldots, (v_k, t)\}$. In the second case E is a set of edges of a given undirected graph $G = (V, E)$ and a path P is a subset o edges of the form $\{\{s, v_1\}, \{v_1, v_2\}, \{v_2, v_3\}, \ldots, \{v_k, t\}\}$. In both cases set Φ consists of all paths from s to t in G. If we allow more than one arc (edge) joining two nodes u and v, then G is called a *multidigraph* (resp. a *multigraph*).

Proposition 1.3 implies that it is enough to consider only *simple paths*, that is the paths that do not contain a cycle. This follows from the fact that if path P contains a cycle, then there is another path $P_1 \subset P$ and, by Proposition 1.3, $Z(P_1) \leq Z(P)$.

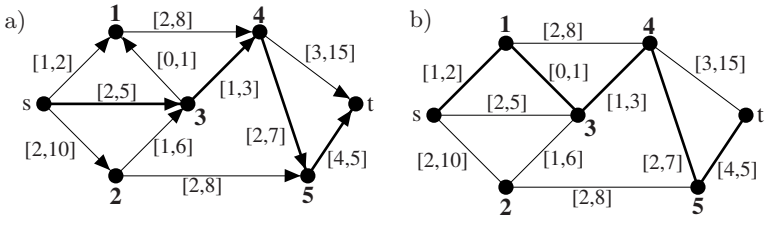


Fig. 7.1. Two examples of MINMAX REGRET SHORTEST PATH in a) directed and b) undirected graphs. The paths in bold are the optimal robust paths.

Figure 7.1 illustrates the problem. In Figure 7.1a there is a digraph $G = (V, A)$ given and path $P = \{(s, 3), (3, 4), (4, 5), (5, t)\}$ is the optimal robust path in G with the maximal regret equal to 14. In Figure 7.1b graph $G = (V, E)$ is undirected and the optimal robust path in G is $P = \{\{s, 1\}, \{1, 3\}, \{3, 4\}, \{4, 5\}, \{5, t\}\}$ with the maximal regret equal to 11.

A. Kasperski: Discrete Optimization with Interval Data, STUDFUZZ 228, pp. 81–112, 2008.
springerlink.com

The deterministic SHORTEST PATH problem is one of the most important and most extensively studied problems in combinatorial optimization, for which a lot of efficient algorithms have been developed. These algorithms are often specialized to take into account a particular structure of the input graph. In Section 7.1 we recall the definitions of some classes of graphs, which are considered in this chapter.

Contrary to the deterministic case, MINMAX REGRET SHORTEST PATH turned out to be computationally intractable. In Section 7.3 we explore in detail its computational complexity. We show that the general problem is strongly NP-hard both in directed and undirected graphs. We also identify a very restrictive class of digraphs for which the problem remains NP-hard.

In Section 7.4 we consider the problem o evaluating the optimality of the elements, that is arcs or edges. The deterministic SHORTEST PATH problem is not a matroidal one. Therefore, we cannot use the results from Section 2.2.3 to detect the possibly and necessarily optimal elements. In fact, the problem of asserting whether a given element is possibly optimal is strongly NP-complete and the nonpossibly optimal elements can be efficiently detected only in some special cases. On the other hand, the problem of asserting whether a given element is necessarily optimal is polynomially solvable if the input graph is assumed to be acyclic and directed. The results concerning the evaluation of optimality are presented in Section 7.4.

The exact methods of solving MINMAX REGRET SHORTEST PATH, that is a MIP formulation and a branch and bound algorithm, are described in Sections 7.5 and 7.7. The MIP formulation is provided for both directed and undirected graphs and it is based on the idea presented in Section 3.1.

In Section 7.7 we discuss the problem for a particular class of graphs, namely edge series-parallel multidigraphs. The problem remains NP-hard for this class of graphs. However, it is not strongly NP-hard and a pseudopolynomial algorithm for it can be designed. Moreover, we show that for this class of graphs the problem admits an FPTAS.

7.1 Some Special Classes of Graphs

The complexity of the deterministic SHORTEST PATH problem as well as its minmax regret version may depend on a particular structure of an input graph. In this section we briefly recall some basic graph theoretical definitions, which will be used in the next part of this chapter. A directed graph $G = (V, A)$, shortly called a *digraph*, is *acyclic* if it does not contain a *cycle*, that is a path that starts and finishes at the same node $v \in V$. In a digraph G we may assume that no arc enters s and no arcs leaves t. In this case s is called *source* and t is called *sink*.

An important class of directed graphs is formed by *layered digraphs*. In a layered digraph $G = (V, A)$, set V can be partitioned into disjoint subsets $V_1, V_2, \ldots, V_k$ called *layers* and arcs exist only between nodes from V_i and V_{i+1} for $i = 1, \ldots, k-1$. The maximal value of $|V_i|$ for $i = 1, \ldots, k$ is called a *width* of G. In every layered digraph all paths between two specified nodes s and t have

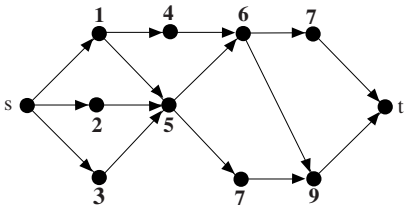


Fig. 7.2. An example of a layered digraph of width 3

the same number of arcs. Figure 7.2 demonstrates a sample layered digraph of width 3.

Another important class of acyclic digraphs is the class of so called *edge series-parallel multidigraphs*. An edge series-parallel multidigraph (shortly ESP) is recursively defined as follows. A digraph consisting of two nodes joined by a single arc is ESP. If G_1 and G_2 are ESP, so are the multidigraphs constructed by each of the operations:

- *parallel composition* $\mathtt{p}(G_1, G_2)$: identify the source of G_1 with the source of G_2 and the sink of G_1 with the sink of G_2.
- *series composition* $\mathtt{s}(G_1, G_2)$: identify the sink of G_1 with the source of G_2.

It results from this definition that each ESP is acyclic and has exactly one source and exactly one sink. An edge series-parallel multidigraph G is naturally associated with a rooted binary tree $\mathcal{T}$ called a *binary decomposition tree of* G. Each leaf of the tree represents an arc in G. Each internal node of $\mathcal{T}$ is marked with `s` or `p` and represents a series or a parallel composition of G_1 and G_2 represented by subtrees rooted at the children of the node. An example of ESP together with its binary decomposition tree is shown in Figure 7.3.

Valdes *et al.* [120] constructed an $\mathcal{O}(|E|)$ algorithm which recognizes whether a given digraph G is edge series-parallel. If the answer is positive, then this algorithm returns the binary decomposition tree of G as byproduct. The edge

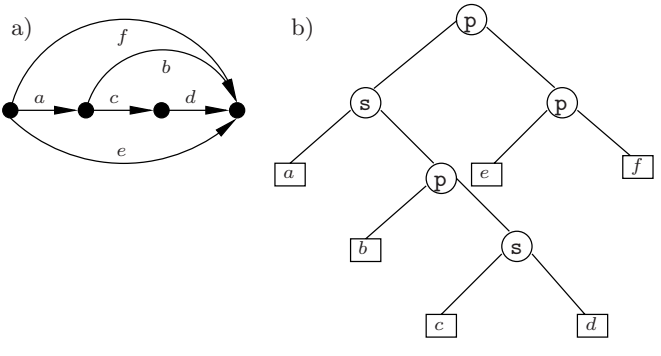


Fig. 7.3. a) An ESP and b) its binary decomposition tree

series-parallel multidigraphs have some applications, for instance in scheduling and analysis of electrical circuits. Due to the special structure of these graphs, many problems that are NP-hard for general graphs become polynomially solvable for ESP. As we will see in the next sections, the MINMAX REGRET SHORTEST PATH problem remains NP-hard for this class of graphs. However, a pseudopolynomial algorithm and an FPTAS in this case can be designed.

An important class of directed and undirected graphs consists of *planar* graphs. A graph is planar if it can be embedded in a plane without crossing arcs (edges). Both graphs shown in Figure 7.2 and Figure 7.3 are planar. In fact, it can be shown that every ESP is a planar digraph.

7.2 Minmax Regret Longest Path in Acyclic Graphs

In this section we consider the MINMAX REGRET LONGEST PATH problem. This is the minmax regret version of LONGEST PATH, in which we seek a path between two distinguished nodes s and t of a given graph of the maximal total weight (length). This problem is a special case of MINMAX REGRET $\mathcal{Q}$ considered in Section 1.3. We have seen, that if all solutions from set Φ in that problem have the same cardinality, then MINMAX REGRET $\mathcal{Q}$ can be transformed into MINMAX REGRET $\mathcal{P}$ and vice versa in $\mathcal{O}(n)$ time. However, this transformation cannot be immediately applied to MINMAX REGRET LONGEST PATH since some paths may exist in G having unequal number of arcs.

Assume that the input digraph $G = (V, A)$ in MINMAX REGRET LONGEST PATH is acyclic. We show that in this case the MINMAX REGRET LONGEST PATH problem can be transformed into MINMAX REGRET SHORTEST PATH by applying a procedure called *layering*. We start by partitioning the node set V into disjoint subsets $V_1, V_2, \ldots, V_k$. Set V_1 contains all nodes of G that have no incoming arcs (if digraph G is acyclic, then at least one such node must exist). We now remove all nodes of V_1 together with incident arcs from G obtaining a smaller digraph G_1. Set V_2 contains all nodes of G_1 that have no incoming arcs. Repeating this procedure until an empty graph is obtained, we compute all remaining sets $V_3, \ldots V_k$. Now we transform digraph G into G' in the following way: for all arcs $(u, v) \in A$, $u \in V_i$ and $v \in V_j$, if $j > i + 1$, then we replace arc (u, v) with path $\{(u, v^1), (v^1, v^2), \ldots, (v^{j-i-1}, v)\}$; arc (v^{j-i-1}, v) has interval weight $\tilde{w}_{(u,v)}$ and all the remaining arcs on this path have interval weight $[0, 0]$. It is clear that digraph G' is layered and the layering procedure can be performed in $\mathcal{O}(|E||V|)$ time.

A sample transformation is shown in Figure 7.4. An acyclic digraph with interval weights is shown in Figure 7.4a. First we compute the partition: $V_1 = \{s\}$, $V_2 = \{1, 2\}$, $V_3 = \{3\}$, $V_4 = \{4\}$, $V_5 = \{5\}$, $V_6 = \{t\}$. Consider arc $(2, 5)$. Since $2 \in V_2$ and $5 \in V_5$ we replace arc $(2, 5)$ with a path composed of three arcs as show in Figure 7.4b. The resulting layered digraph G' is shown in Figure 7.4b.

It is easy to see that there is one to one mapping between paths in G and G'. Every path P' in G' can be transformed into the corresponding path P in G by removing dummy arcs from P'. Since the dummy arcs have interval weights

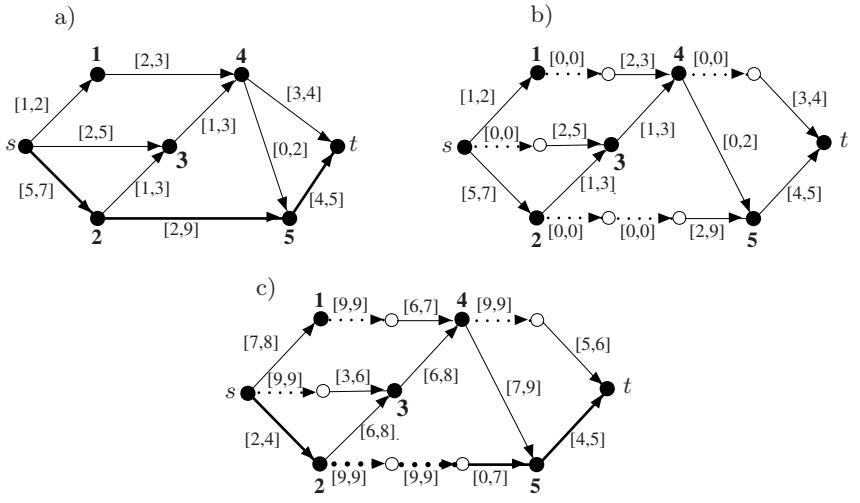


Fig. 7.4. A sample transformation of MINMAX REGRET LONGEST PATH into MINMAX REGRET SHORTEST PATH. Figure a) shows a sample acyclic digraph. In Figure b) there is an equivalent layered digraph G'. In Figure c) there is digraph G' with transformed weight intervals.

$[0,0]$, the maximal regret of path P in G equals the maximal regret of path P' in G'. In consequence MINMAX REGRET LONGEST PATH in G' is equivalent to MINMAX REGRET LONGEST PATH in G. Since digraph G' is layered, all paths from s to t in G' have the same number of arcs. Hence we can apply Theorem 1.5 to transform this problem into MINMAX REGRET SHORTEST PATH. This transformation consists in changing the interval weights of G'. In Figure 7.4 it is shown graph G' with transformed weight intervals. The optimal robust path in this graph (in bold) corresponds to the optimal robust path in G. Now solving MINMAX REGRET SHORTEST PATH for G' we obtain the optimal solution to the original problem, that is MINMAX REGRET LONGEST PATH.

Observe, that the same method can be applied to transform the MINMAX REGRET SHORTEST PATH problem for acyclic digraphs to MINMAX REGRET LONGEST PATH. We first perform layering and then change the weight intervals according to Theorem 1.5. Thus both problems are equivalent if the input graph is directed and acyclic.

7.3 Computational Complexity

We now explore the computational complexity of MINMAX REGRET SHORTEST PATH. We will show that this problem is strongly NP-hard and remains NP-hard even in some very restrictive cases. We start by defining a decision version of the problem in directed graphs.

Decision Minmax Regret Shortest Path:
Instance: A directed graph $G = (V, A)$ with interval arc weights, two distinguished nodes $s, t \in V$ and integer $K \geq 0$.
Question: Is there a path P from s to t in G such that $Z(P) \leq K$?

We show the proof of the following result, which is a slight modification of the proof given by Averbakh and Lebedev [17]:

Theorem 7.1. *The* Decision Minmax Regret Shortest Path *problem is strongly NP-complete even if the bounds of weight intervals are 0 or 1.*

Proof ([17]). We will show a polynomial time reduction from the Hamiltonian Path problem, which is known to strongly NP-complete [56, 72]. The Hamiltonian Path problem is defined as follows:

Hamiltonian Path:
Instance: An undirected graph $H = (V, E)$.
Question: Does H contain a *Hamiltonian path*, that is a path that visits every node of H exactly once?

Let $H = (V, E)$ be an instance of Hamiltonian Path, where $V = \{v_1, \ldots, v_q\}$, $q > 1$. Consider $2q$ copies of the set V indexed as $V^1, \ldots, V^{2q}$. The copy V^i contains nodes $\{v_1^i, \ldots, v_q^i\}$. We construct digraph $G' = (V', A)$ in the following way:

- $V' = \bigcup_{i=1}^{2q} V^i \cup \{s, t\}$.
- We add arcs $(v_i^1, v_i^2), (v_i^2, v_i^3), \ldots (v_i^{2q-1}, v_i^{2q})$ for all $i = 1, \ldots q$. All these arcs have interval weights equal to $[0, 1]$. Observe that we have created exactly q disjoint paths $P_1, \ldots, P_q$. These paths will be called *vertical paths*.
- For all edges $\{v_i, v_j\} \in E$ we add arcs $(v_i^2, v_j^3), (v_i^4, v_j^5), \ldots, (v_i^{2q-2}, v_j^{2q-1})$ and $(v_j^2, v_i^3), (v_j^4, v_i^5), \ldots, (v_j^{2q-2}, v_i^{2q-1})$. All these arcs have interval weights equal to $[1, 1]$.
- We add arcs (s, v_i^1) and (v_i^{2q}, t) for all $i = 1, \ldots q$, with interval weights $[1, 1]$.

A sample reduction is shown in Figure 7.5.

Let us complete the construction by setting $K = 2q - 2$. We will show that graph H has a Hamiltonian path if and only if there is a path P from s to t in G' such that $Z(P) \leq 2q - 2$. Observe that for every path P in G' it holds $F(P, S_P^+) = 2q + 1$. It follows from the fact that digraph G' is layered and all upper bounds of the weight intervals are equal to 1. Hence it is enough to show that H has a Hamiltonian Path if and only if there is a path P from s to t in G' such that $F^*(S_P^+) \geq 3$, since in this case $Z(P) = F(P, S_P^+) - F^*(S_P^+) \leq 2q - 2$.

We start by proving the following claim: *path P is such that $F^*(S_P^+) \geq 3$ if and only if $|P \cap P_i| \geq 1$ for all $i = 1, \ldots q$*, in other words P must use at least one arc from every vertical path. Indeed, if $|P \cap P_i| = 0$ for some vertical path P_i, then P_i together with arcs incident to s and t forms a path of the total weight equal to 2 under S_P^+, thus $F^*(S_P^+) \leq 2$. Conversely, if $|P \cap P_i| \geq 1$ for all $i = 1, \ldots, q$, then in scenario S_P^+ at least one arc on every vertical path has

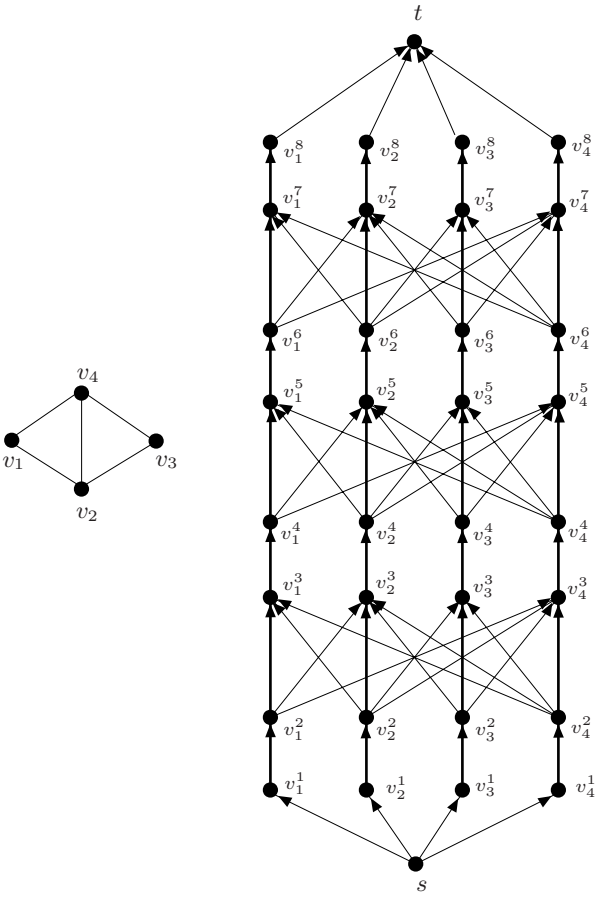


Fig. 7.5. A sample reduction. The arcs in bold have interval weights $[0, 1]$ and they form exactly q vertical paths. The remaining arcs have interval weights $[1, 1]$.

weight equal to 1. Hence all paths from s to t have the total weight not less than 3 under S_P^+.

Suppose that graph H has a Hamiltonian path that visits nodes of H in order $v_{i_1}, v_{i_2}, \ldots, v_{i_q}$. From the construction of G', there is path in G' of the form $P = \{(s, v_{i_1}^1), (v_{i_1}^1, v_{i_1}^2), (v_{i_1}^2, v_{i_2}^3), (v_{i_2}^3, v_{i_2}^4), \ldots, (v_{i_q}^{2q-1}, v_{i_q}^{2q}), (v_{i_q}^{2q}, t)\}$. The arcs $(v_{i_1}^1, v_{i_1}^2)$, $(v_{i_2}^3, v_{i_2}^4), \ldots, (v_{i_q}^{2q-1}, v_{i_q}^{2q})$ lie on vertical paths $P_{i_1}, P_{i_2}, \ldots, P_{i_q}$. Hence $|P \cap P_i| \geq 1$ for all $i = 1, \ldots, q$ and $F^*(S_P^+) \geq 3$.

Suppose there is a path P in G' from s to t such that $F^*(S_P^+) \geq 3$. The path P must traverse at least one arc on every vertical path P_i, that is $|P \cap P_i| \geq 1$ for $i = 1, \ldots q$. Observe that path P must also use at least $q - 1$ arcs that connect the arcs on vertical paths and two arcs incident to s and t. If there are more than 2 arcs of P that lie on the same vertical path, then we would have $|P| \geq 2+(q-1)+(q-1)+2 = 2q+2$. This is not possible, since every path from s to t in

G' has exactly $2q+1$ arcs. Hence $|P \cap P_i| = 1$ for all $i = 1, \ldots q$. Now, it is easily seen that path P can only use arcs $(v_i^1, v_i^2), (v_i^3, v_i^4), \ldots, (v_i^{2q-1}, v_i^{2q})$ on every vertical path P_i. There are exactly q such arcs on every vertical path P_i. Thus path P must use arcs $(v_{i_1}^1, v_{i_1}^2), (v_{i_2}^3, v_{i_2}^4), \ldots, (v_{i_q}^{2q-1}, v_{i_q}^{2q})$ for some permutation $(i_1, i_2, \ldots, i_q)$ of set $\{1, 2, \ldots, q\}$. Therefore path P must be of the following form:

$$P = \{(s, v_{i_1}^1), (v_{i_1}^1, v_{i_1}^2), (v_{i_1}^2, v_{i_2}^3), (v_{i_2}^3, v_{i_2}^4), \ldots, (v_{i_q}^{2q-1}, v_{i_q}^{2q}), (v_{i_q}^{2q}, t)\},$$

where $V = \{v_{i_1}, \ldots v_{i_q}\}$. Now visiting nodes of graph H in order $v_{i_1}, v_{i_2}, \ldots, v_{i_q}$ we obtain a Hamiltonian Path in H.

The Decision Minmax Regret Shortest Path problem is in NP. A nondeterministic Turing machine first guesses path P. It can be then verified in polynomial time whether $Z(P) \leq K$. □

Observe that the digraph G' constructed in the proof of Theorem 7.1 is acyclic layered and all the bounds of the weight intervals are 0 or 1. We have thus obtained the following result (see also [17]):

Theorem 7.2. *[[17]] The* Minmax Regret Shortest Path *problem is strongly NP-hard for acyclic directed layered graphs, even if the bounds of weight intervals are 0 or 1.*

In Decision Minmax Regret Shortest Path we have assumed that the input graph is directed. If we allow a graph to be undirected, then the proof of Theorem 7.1 is very similar. We only remove arcs directions while constructing graph G'. The proof requires then a slight modification (see [17] for details). Hence the following theorem is also true:

Theorem 7.3 ([17]). *Problem* Minmax Regret Shortest Path *is strongly NP-hard for undirected graphs, even if the bounds of weight intervals are 0 or 1.*

The general Minmax Regret Shortest Path problem is computationally intractable. It is important now to find a restrictive class of graphs for which it remains NP-hard. It turns out that such a restrictive class of graphs is formed by edge series-parallel digraphs in which every node has a degree at most 3. Recall that a *degree* of node v is the number of arcs incident to v. We prove the following result:

Theorem 7.4. *The* Decision Minmax Regret Shortest Path *problem remains NP-complete if the input graph is restricted to be edge series-parallel with the maximal node degree at most 3.*

Proof. Our goal will be to construct a polynomial time reduction from Partition to Decision Minmax Regret Shortest Path. The Partition problem, which is known to be NP-complete [56], is defined as follows:

Partition
Instance: A collection $\mathfrak{C} = (c_1, \ldots, c_n)$ of positive integers such that $\sum_{i=1}^{n} c_i = 2b$, where $b > 0$.
Question: Is there a subset $I \subset \{1, \ldots, n\}$ such that $\sum_{i \in I} c_i = b$?

Let $\mathfrak{C} = (c_1, \ldots, c_n)$ be a given instance of PARTITION. We construct the corresponding ESP $G = (V, A)$ as follows:

- $V = \{1, \ldots, n+1\}$, where $s = 1$ and $t = n + 1$;
- for every element c_i of collection $\mathfrak{C}$ we create two parallel arcs: $q_i = (i, i+1)$ with interval weight $[c_i, c_i]$ and $r_i = (i, i+1)$ with interval weight $[0, \frac{3}{2}c_i]$;
- we also create an additional arc $z = (1, n+1)$ with interval weight $[b, 3b]$.

We complete the construction by setting $K = \frac{3}{2}b$. The construction is shown in Figure 7.6.

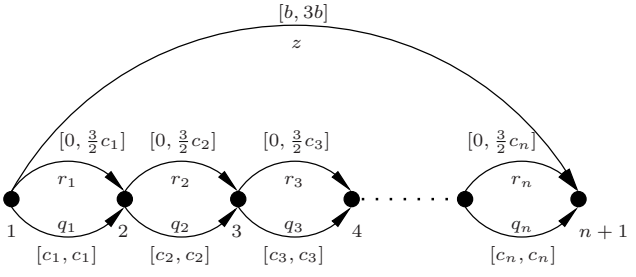


Fig. 7.6. The series-parallel multidigraph G for a given collection $\mathfrak{C}$

It is clear that G is an edge series-parallel multidigraph and it can be constructed in time bounded by a polynomial in the size of PARTITION. Notice that G contains multiarcs and the maximal node degree in G is 4. However, we will show later that G can be additionally transformed so that the multiarcs are removed and the maximal node degree is reduced to 3.

It is easy to observe that an optimal robust path in G cannot use arc z. The path consisting of the single arc z has the maximal regret equal to $3b$, while path $\{q_1, q_2, \ldots, q_n\}$ has the maximal regret equal to $2b$. Consequently, the path consisting of the single arc z cannot be optimal if $b > 0$.

Let P be any path from 1 to $n+1$ in G that does not use arc z. Path P must traverse either arc q_i or arc r_i for every $i = 1, \ldots, n$. Let $I_1 = \{i : \ q_i \in P\}$ and $I_2 = \{i : \ r_i \in P\}$. The worst case scenario S_P^+ for P can be obtained by fixing the weights of q_i, $i \in I_1$, and r_i, $i \in I_2$, to their upper bounds and the weights of the remaining arcs to their lower bounds. Thus it holds

$$F(P, S_P^+) = \sum_{i \in I_1} \overline{w}_{q_i} + \sum_{i \in I_2} \overline{w}_{r_i}. \tag{7.1}$$

The worst case alternative P^* for P (that is the shortest path under S_P^+) consists of the single arc z or it is composed of arcs r_i, $i \in I_1$, and q_i, $i \in I_2$. Hence

$$F^*(S_P^+) = \min \left\{ \sum_{i \in I_1} \underline{w}_{r_i} + \sum_{i \in I_2} \underline{w}_{q_i}, \underline{w}_z \right\}. \tag{7.2}$$

Using (7.1) and (7.2) we can express the maximal regret of P as follows:

$$Z(P)=\max\left\{\sum_{i\in I_1}\overline{w}_{q_i}+\sum_{i\in I_2}\overline{w}_{r_i}-\sum_{i\in I_1}\underline{w}_{r_i}-\sum_{i\in I_2}\underline{w}_{q_i},\ \sum_{i\in I_1}\overline{w}_{q_i}+\sum_{i\in I_2}\overline{w}_{r_i}-\underline{w}_z\right\},$$

which is equivalent to

$$Z(P)=\max\left\{\sum_{i\in I_1}c_i+\frac{3}{2}\sum_{i\in I_2}c_i-\sum_{i\in I_2}c_i,\ \sum_{i\in I_1}c_i+\frac{3}{2}\sum_{i\in I_2}c_i-b\right\}. \tag{7.3}$$

From the construction of G it follows that

$$\sum_{i\in I_1}c_i+\sum_{i\in I_2}c_i=2b. \tag{7.4}$$

Let us denote by b_1 the sum $\sum_{i\in I_1}c_i$. Thus from (7.4) we obtain $\sum_{i\in I_2}c_i=2b-b_1$. Then (7.3) can be rewritten as follows:

$$Z(P)=\max\left\{b+\frac{1}{2}b_1,2b-\frac{1}{2}b_1\right\}. \tag{7.5}$$

Notice that $Z(P)\geq\frac{3}{2}b$. It is easy to verify that $Z(P)=\frac{3}{2}b$ if and only if $b_1=b$. Since $b_1=\sum_{i\in I_1}c_i$, it follows that $Z(P)\leq\frac{3}{2}b$ if and only if set $I_1\subset\{1,\ldots,n\}$ properly determines a partition of collection $\mathfrak{C}=(c_1,\ldots,c_n)$.

The constructed ESP G contains multiarcs and some nodes have degree equal to 4. However, we can additionally transform graph G into $G^{'}=(V^{'},A^{'})$ using the replacement shown in Figure 7.7.

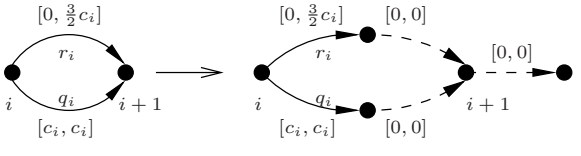


Fig. 7.7. An additional transformation of graph G

After this transformation we get ESP $G^{'}$ with no multiarcs and with node degrees at most 3. It is easy to verify that the maximal regrets of the optimal robust paths in G and $G^{'}$ are equal. Thus the answer to partition is Yes if and only if there is a path in G' such that $Z(P)\leq\frac{3}{2}b$.

The problem is in NP and the proof of this fact is exactly the same as in Theorem 7.1. □

We have thus established the following result:

Theorem 7.5. *The* Minmax Regret Shortest Path *problem is NP-hard for edge series-parallel digraphs with a maximal node degree at most 3.*

Observe that we have perhaps identified the minimal problem, which is NP-hard. If we remove arc z from ESP G (see Figure 7.6) then the optimal robust path in the resulting digraph can be found in polynomial time. Hence adding a single arc makes the problem intractable. Observe also that digraph G', constructed in the proof of Theorem 7.4 (see Figures 7.6 and 7.7), can be easily transformed to a layered digraph by applying layering, which in this case consists of splitting arc $z = (1, n+1)$. All the additional dummy arcs have interval weights equal to $[0, 0]$. Therefore the optimal robust path in the resulting layered digraph has the same maximal regret as the optimal robust path in the original graph G'. It is easy to see that the resulting layered digraph has a width equal to 3. If we allow the graph to have multiarcs, then we omit the replacement shown in Figure 7.7 and the resulting layered multidigraph has width 2. This leads to the following result:

Theorem 7.6. *The* Minmax Regret Shortest Path *problem is NP-hard for layered digraphs of width 3 and for layered multidigraphs of width 2.*

The complexity of the problem for layered digraphs of width 2 remains open. Applying the results from Section 7.2 it is easily seen that Minmax Regret Longest Path is strongly NP-hard in layered graphs in which all the bounds of weights intervals are 0 or 1. Moreover this problem remains NP-hard in series parallel digraphs with a maximal node degree at most 3 and in layered digraphs of width 3 (layered multidigraphs of width 2).

7.4 Evaluation of Optimality

In this section we discuss the problem of deciding whether a given element (arc or edge) is possibly or necessarily optimal. Recall that all elements that are nonpossibly optimal can be removed from graph G, which allows reduction of the size of the input graph. Moreover, if all weight intervals are nondegenerate, then all necessarily optimal arcs (edges) can be automatically added to the constructed optimal robust path.

In the previous chapter we have shown that the number of nonpossibly optimal edges in Minmax Regret Minimum Spanning Tree may be quite large and we may conjecture that the same holds for Minmax Regret Shortest Path. Unfortunately, for this problem the situation is more complicated, since the problem of checking whether an arc is possibly optimal is strongly NP-complete, even if the input digraph is acyclic. On the other hand, deciding whether a given arc is necessarily optimal in an acyclic digraph can be done in polynomial time and for general graphs this problem remains open.

In this section we first show that the Pos Shortest Path problem is strongly NP-complete. We show next some special cases for which the nonpossibly optimal arcs can be efficiently detected. Finally, we present a polynomial time algorithm for asserting whether an arc is necessarily optimal in an acyclic digraph.

7.4.1 Possibly Optimal Arcs

We first prove that Pos Shortest Path is strongly NP-complete and we describe then some polynomially solvable cases.

Computational complexity

Consider the following problem :

Pos Criticality
Instance: An acyclic digraph $G = (V, A)$ with interval weights of arcs, two distinguished nodes $s \in V$ and $t \in V$, arc $f \in A$.
Question: Is f *possibly critical* in G, in order words, does f belong to a longest path from s to t under some scenario $S \in \Gamma$?

The Pos Criticality problem arises in the analysis of criticality of activities in the interval scheduling. Digraph G models then a partially ordered set of activities whose processing times are not precisely known. The notion of possible criticality characterizes the criticality of an activity in the problem. An activity, modeled by arc f, is possibly critical if it belongs to a critical (longest) path under some scenario. The following result describes the complexity of Pos Criticality:

Theorem 7.7 ([32, 33]). *The* Pos Criticality *problem is strongly NP-complete for general acyclic digraphs and remains NP-complete for acyclic planar digraphs.*

We now use Theorem 7.7 to prove that Pos Shortest Path is NP-complete.

Theorem 7.8. *The* Pos Shortest Path *problem is strongly NP-complete for general acyclic digraphs and remains NP-complete for acyclic planar digraphs.*

Proof. Consider a digraph $G = (V, A)$, $s, t \in V$, with interval arc weights $[\underline{w}_a, \overline{w}_a]$, $a \in A$, and a distinguished arc $f = (k, l) \in A$. We ask whether f is possibly critical in G. We reduce this NP-complete problem to Pos Shortest Path in the following way. We apply first the layering to G (see Section 7.2) obtaining a layered digraph $G' = (V', A')$. Let f' be the last arc on the path from k to l in G'. It is evident, that arc f is possibly critical in G if and only if f' is possibly critical in G'. We replace now the interval weights of arcs in $G' = (V', A')$ with $[M - \overline{w}_a, M - \underline{w}_a]$ for all $a \in A'$, where $M = \max_{a \in A} \overline{w}_a$. Denote the resulting digraph by G''. A sample transformation is shown in Figure 7.8.

Both digraphs G' and G'' have the same set of paths from s to t. Every scenario $S' = (w_a^{S'})_{a \in A'}$ in G' has the corresponding scenario $S'' = (w_a^{S''})_{a \in A''}$ in G'' such that $w_a^{S''} = M - w_a^{S'}$ for all $a \in A$. Furthermore, every path P that has weight $F(P, S')$ in G' under S' has weight $kM - F(P, S')$ in G'' under S'', where k stands for the number of arcs in every path P. Obviously, kM is a constant that does not depend on path P, which follows from the layered structure of graphs G' and G''. Therefore, P is the longest path in G' under S' if and only if P is

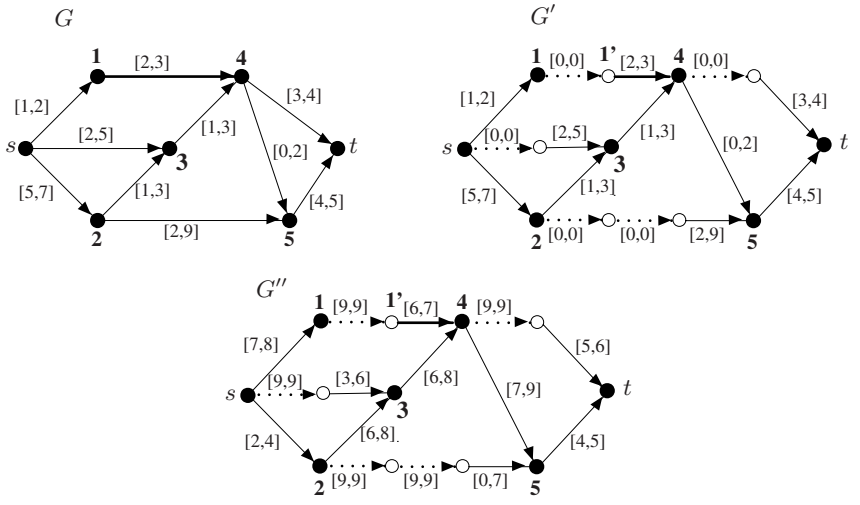


Fig. 7.8. A sample transformation in which $f = (1, 4)$ and $f' = (1', 4)$

the shortest path in G'' under S''. We conclude that f' is possibly critical in G' if and only if f' is possibly optimal in G''. The presented reduction can be performed in polynomial time. Observe that if the input graph G is planar so is the graph G''.

It remains to be shown that Pos Shortest Path belongs to NP. To see this, recall that arc f is possibly optimal if and only if it belongs to a possibly optimal path. Therefore, a nondeterministic Turing machine first guesses path P. It can be then verified in polynomial time whether $f \in P$ and P is possibly optimal. In order to do this it is enough to check whether P is the shortest path under scenario S_P^-. □

Contrary to the matroidal problems, the general problem of deciding whether an arc is possibly optimal in the Shortest Path problem with interval weights is NP-complete. In the next section we will identify some special cases of the problem for which the nonpossibly optimal arcs can be efficiently detected.

Polynomially solvable cases

We now present the results, which allow efficient identification of possibly and nonpossibly optimal arcs in some particular cases. From now on we make the assumption that the input graph is an acyclic digraph $G = (V, A)$ and we use the following, additional notations:

- The nodes of G are numbered from 1 to $|V|$ such that $s = 1$, $t = |V|$ and $i < j$ for all arcs $(i, j) \in A$;
- $PRED(i)$ and $SUCC(i)$, $i \in V$, are defined as follows: $j \in PRED(i)$ if node j precedes i on a path from s to i, similarly $j \in SUCC(i)$ if node j succeeds i on a path from i to t. For instance, in graph G shown in Figure 7.8 we have $PRED(4) = \{s, 1, 2, 3\}$ and $SUCC(2) = \{3, 4, 5, t\}$;

- $PRED(i,j)$ and $SUCC(i,j)$, $(i,j) \in A$, are defined as follows: $(k,l) \in PRED(i,j)$ if arc (k,l) precedes (i,j) on a path from s to j, similarly $(k,l) \in SUCC(i,j)$ if arc (k,l) succeeds (i,j) on a path from i to t. For instance, in graph G shown in Figure 7.8 we have $PRED(3,4) = \{(s,2),(s,3),(2,3)\}$ and $SUCC(3,4) = \{(4,5),(4,t),(5,t)\}$;
- $Succ(i) = \{j \in V : (i,j) \in A\}$ is the subset of direct successors of node i and $Pred(i) = \{j \in V : (j,i) \in A\}$ is the subset of direct predecessors of node i;
- $G(i,j)$ is the subgraph of G composed of all paths from i to j in G with source i and sink j;
- Arc $(k,l) \in A$ is optimal in graph $G(i,j)$ under scenario S if it belongs to a shortest path from i to j in $G(i,j)$ under S. Notice that if all arcs in $G(i,j)$ have precise weights, then asserting whether an arc (k,l) is optimal can be done in polynomial time.

Assume that we want to decide whether an arc (k,l) is possibly optimal and all arcs in $PRED(k,l)$ have deterministic weights. We will show that in this case the possible optimality of arc (k,l) can be efficiently verified. Consider the algorithm shown in Figure 7.9. This algorithm is very similar to that constructed for problem Pos Criticality by Fortin *et al.* [52].

`Pos SP`

Require: An acyclic digraph $G = (V,A)$ with interval arc weights $[\underline{w}_{ij}\overline{w}_{ij}]$ for $(i,j) \in A$, arc $(k,l) \in A$ and all arcs in $PRED(k,l)$ have precise weights.
Ensure: **true** if (k,l) is possibly optimal and **false** otherwise.
1: $w_{kl} \leftarrow \underline{w}_{kl}$
2: **forall** $(i,j) \notin SUCC(k,l) \cup \{(k,l)\}$ **do** $w_{ij} \leftarrow \overline{w}_{ij}$
3: **for** $i \leftarrow l$ **to** $|V|-1$ **such that** $i \in SUCC(l) \cup \{l\}$ **do**
4: **if** (k,l) is optimal in $G(1,i)$ **then forall** $j \in Succ(i)$ **do** $w_{ij} \leftarrow \underline{w}_{ij}$
5: **else forall** $j \in Succ(i)$ **do** $w_{ij} \leftarrow \overline{w}_{ij}$
6: **end for**
7: **if** (k,l) is optimal in $G(1,|V|)$ **then return true else return false**

Fig. 7.9. Algorithm that asserts whether arc (k,l) is possibly optimal under the assumption that all arcs in $PRED(k,l)$ have precise weights. This algorithm is a slight modification of the one constructed in [52, 134].

Proposition 7.9. *If all arcs in $PRED(k,l)$ have deterministic weights, then algorithm* `Pos SP` *returns* ***true*** *if and only if arc (k,l) is possibly optimal.*

Proof. Let us notice that the algorithm constructs a certain extreme scenario S' by successively assigning precise weights to the arcs of G. In line 4 all arcs in subgraph $G(1,i)$ and in line 7 all arcs in $G \equiv G(1,|V|)$ have deterministic weights, set according to scenario $S' \in \Gamma$. Hence if the algorithm returns **true**, then arc (k,l) belongs to a shortest path in G under S' and it is possibly optimal.

Suppose that arc (k,l) is possibly optimal or equivalently (k,l) belongs to a possibly optimal path P^* in G. Suppose, by contradiction, that the algorithm

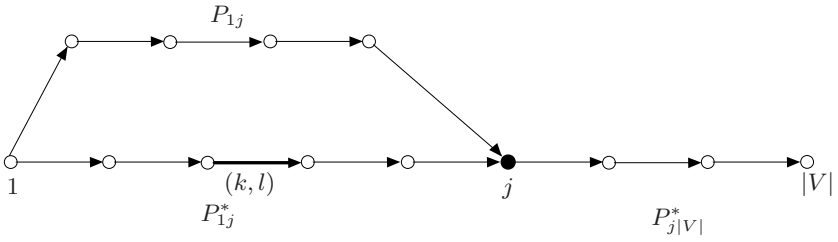


Fig. 7.10. The illustration of the proof. All arcs on path P_{1j} have weights set to their upper bounds and all arcs on path P^*_{1j} have weights set to their lower bounds under S'.

returns **false**. Let $j \in SUCC(l) \cup \{l\}$ be the first node on P^* such that (k,l) is not optimal in $G(1,j)$ under S'. Denote by P^*_{1j} the subpath of P^* from 1 to j and by $P^*_{j|V|}$ the subpath of P^* from j to $|V|$. Hence there is a shortest path P_{1j} from 1 to j that does not use (k,l), which is shorter than P^*_{1j} under scenario S'. It is easy to observe that all arcs in P^*_{1j} have weights set to their lower bounds under S'. It follows from the definition of node j and the fact that all arcs in $PRED(k,l)$ have precise weights (the corresponding intervals are degenerate). Now the crucial observation is that all arcs in P_{1j} have weights set to their upper bounds under S'. Indeed, the algorithm assigns the lower bound of a weight interval to an arc $(u,v) \neq (k,l)$ if and only if (k,l) belongs to the shortest path Q from 1 to u under S' (see line 4). In consequence, we could construct the shortest path from 1 to j under S' that uses (k,l) by concatenating Q with the subpath of P_{1j} from u to j. This contradicts the fact that (k,l) is not optimal in $G(1,j)$ under S'.

Consider path P which is created by concatenating P_{1j} with $P^*_{j|V|}$. It holds $F(P,S') < F(P^*,S')$ because P_{1j} is shorter than P^*_{1j} under S'. For every scenario S it holds

$$F(P,S) - F(P^*,S) = \sum_{a \in P_{1j}} w_a^S - \sum_{a \in P^*_{1j}} w_a^S \leq$$
$$\leq \sum_{a \in P_{1j}} \overline{w}_a - \sum_{a \in P^*_{1j}} \underline{w}_a = F(P,S') - F(P^*,S') < 0.$$

Hence path P is shorter that P^* under all scenarios S, which contradicts the assumption that P^* is possibly optimal. □

The algorithm `Pos SP` can be implemented so that it runs in $\mathcal{O}(|E|)$ time. The details of the implementation are similar to that shown by Zieliński [134]. Using this algorithm we can also assert whether arc (k,l) is possibly optimal under the assumption that all arcs in $SUCC(k,l)$ have precise weights. It is enough to reverse the arcs in graph G and carry out the computations from node $|V|$ to 1.

Observe now that using algorithm `Pos SP` we can evaluate efficiently the possible optimality of any arc incident to s or t, that is (s,l) or (k,t). This is the consequence of the fact that, in this case, either $PRED(s,l) = \emptyset$ or

$SUCC(k,t) = \emptyset$. For an intermediate arc (k,l) the following proposition (first observed by Karasan *et al.* [70]) is true:

Proposition 7.10. *If arc (k,l) is possibly optimal in G, then it is possibly optimal in $G(s,l)$ and it is possibly optimal in $G(k,t)$.*

Proof. If arc (k,l) is possibly optimal, then there is a scenario S such that (k,l) belongs to a shortest path P in G from s to t under S. Path P is of the form $P_1 \cup P_2$, where P_1 is the shortest path from s to l and P_2 is the shortest path from k to t in G under S. In consequence (k,l) is possibly optimal in $G(s,l)$ and $G(k,t)$. □

Proposition 7.10 gives a necessary condition for an arc (k,l) to be possibly optimal, which can be verified in $\mathcal{O}(|E|)$ time. By contraposition, if arc (k,l) is not possibly optimal in $G(s,l)$ or it is not possibly optimal in $G(k,t)$, then it is not possibly optimal in G. Otherwise, this test is not conclusive since the condition in Proposition 7.10 is not sufficient. Hence applying this test we can detect efficiently a subset of nonpossibly optimal arcs.

Now an interesting question is how many nonpossibly optimal arcs may be detected in a given acyclic digraph, by applying Proposition 7.10. To give an answer to this question some computational experiments were performed by Karasan *et al.* [70]. The experiments were carried out on layered digraphs of width varying from 2 to 5. It turns out that the number of nonpossibly optimal arcs, detected by means of Proposition 7.10, may be quite large and for the tested instances it averaged between 20 - 50 percent of all arcs of a graph (for details we refer the reader to [70]). Hence removing all nonpossibly optimal arcs may significantly reduce the size of the input digraph and,consequently, speed up the calculation of the optimal robust path.

It turns out that all possibly and nonpossibly optimal arcs can be efficiently detected if the input graph has edge series-parallel topology. This is the consequence of the following result:

Proposition 7.11. *Let $G = (V,A)$ be an edge series-parallel multidigraph. Then arc $(k,l) \in A$ is possibly optimal in G if and only if it is optimal under scenario $S^-_{PRED(k,l)\cup\{(k,l)\}\cup SUCC(k,l)}$.*

Proof. The proof is by induction on the number of arcs and it is similar to that presented in [47]. □

Proposition 7.11 can be applied as follows. We first set the weights in G according to scenario S given in Proposition 7.11. We compute then the shortest path P_1 from s to k and the shortest path P_2 from l to t under S. If path $P_1 \cup \{(k,l)\} \cup P_2$ is the shortest path from s to t under S, then arc (k,l) is possibly optimal. Otherwise it is nonpossibly optimal. This procedure can be performed in $\mathcal{O}(|E|)$ time. Therefore all nonpossibly optimal arcs in a series-parallel multidigraph can be detected in $\mathcal{O}(|E|^2)$ time.

7.4.2 Necessarily Optimal Arcs

Let us start by observing that detecting a necessarily optimal arc in an acyclic digraph G may be useful. Suppose we have detected a necessarily optimal arc

$(k,l) \in A$. Consider an optimal robust path P in G that uses (k,l). By Theorem 2.10, such a path exists. Path P is of the form $P_1 \cup \{k,l\} \cup P_2$, where P_1 is a subpath from s to k and P_2 is a subpath from l to t in G. Since (k,l) is necessarily optimal there must be a worst case alternative P^* for P (that is the shortest path under S_P^+) that also uses (k,l). Hence P^* is also of the form $P_1^* \cup \{(k,l)\} \cup P_2^*$, where P_1^* is a subpath from s to k and P_2^* is a subpath from l to t. Now P_1 is a path in $G(s,k)$ with the maximal regret $Z_1(P_1)$ and P_2 is a path in $G(l,t)$ with the maximal regret $Z_2(P_2)$. Moreover, P_1^* is the worst case alternative for P_1 and P_2^* is the worst case alternative for P_2. Now it is easy to see that $Z(P) = Z_1(P_1) + Z_2(P_2)$. Therefore, P is the optimal robust path in G if and only if P_1 and P_2 are the optimal robust paths in G_1 and G_2. Using this fact we can decompose graph G into two graphs $G(s,k)$ and $G(l,t)$ and solve the two smaller problems.

Surprisingly, contrary to the possible optimality, the problem of deciding whether a specified arc is necessarily optimal in an acyclic digraph G is polynomially solvable. Fortin *et al.* [52] designed an algorithm with running time $\mathcal{O}(|E||V|)$, which assets whether a given arc in G is *necessarily critical*. Recall that an arc is necessarily critical if it belongs to a longest path in G for all scenarios. This algorithm can also be applied to the Nec Shortest Path problem. We must only transform a given digraph G by applying the layering and changing the interval weights. The transformation and the reasoning are exactly the same as the ones in the proof of Theorem 7.8. Hence we can assert in $\mathcal{O}(|E||V|)$ time whether an arc is necessarily optimal in a given acyclic digraph. In consequence, all necessarily optimal arcs can be detected in $\mathcal{O}(|E|^2|V|)$ time.

We do not present here the algorithm for evaluating the necessary criticality, since its details are quite involving. This algorithm can be found in the paper by Fortin *et al.* [52]. Let us also point out that the Nec Shortest Path problem remains open for general directed and undirected graphs and the algorithm deigned in [52] can be applied only to acyclic digraphs.

7.5 Mixed Integer Programming Formulation

In this section we construct a MIP model for the problem in both directed and undirected graphs. We start with the simpler case, when the input graph is directed. We then extend the model to undirected graphs.

7.5.1 MIP Formulation for Directed Graphs

Consider a directed graph $G = (V,A)$ with two distinguished nodes s and t. Assume that arc $(i,j) \in A$ has interval weight $[\underline{w}_{ij}, \overline{w}_{ij}]$. Let us associate binary variable x_{ij} with every arc (i,j) of G. This variable will express whether arc (i,j) is a part of the constructed path. Consider the following set of constraints:

$$\sum_{\{i:\,(j,i)\in A\}} x_{ji} - \sum_{\{k:\,(k,j)\in A\}} x_{kj} = \begin{cases} 1 & \text{for } j = s \\ 0 & \text{for } j \in V \setminus \{s,t\} \\ -1 & \text{for } j = t \end{cases} \tag{7.6}$$

$$x_{ij} \in \{0,1\} \quad \text{for } (i,j) \in A$$

Constraints (7.6) describe a network flow problem in which we wish to send one unit of flow from s to t. This unit is sent on any directed path from s to t in G. Therefore every characteristic vector of $ch(\Phi)$ is a feasible solution to (7.6). On the other hand, if $(x_{ij})_{(i,j)\in A}$ is a feasible solution to (7.6), then the subset of variables whose values are equal to 1 must define a simple path from s to t plus possibly some cycles in G (observe, that this is only the case if the input graph G is not acyclic). Therefore, according to the conditions given in Section 3.1, we can use constraints (7.6) to describe set $ch(\Phi)$. Furthermore, it is well known that the constraints matrix $\boldsymbol{A}$ of (7.6) is totally unimodular. Hence we can construct the relaxed subproblem $\min_{\boldsymbol{y}\in ch(\Phi)} \phi(\boldsymbol{x},\boldsymbol{y})$ in the following way:

$$\begin{aligned}
&\min \sum_{(i,j)\in A} (\overline{w}_{ij}x_{ij} + \underline{w}_{ij}(1-x_{ij}))y_{ij} \\
&\sum_{\{i:\,(j,i)\in A\}} y_{ji} - \sum_{\{k:\,(k,j)\in A\}} y_{kj} = \begin{cases} 1 & \text{for } j = s \\ 0 & \text{for } j \in V\setminus\{s,t\} \\ -1 & \text{for } j = t \end{cases} \\
&y_{ij} \geq 0
\end{aligned} \tag{7.7}$$

We have replaced constraints $y_{ij} \in \{0,1\}$ with $y_{ij} \geq 0$ since $y_{ij} \leq 1$ in every optimal solution to (7.7). Now, the dual to (7.7), that is the problem $\max_{\boldsymbol{\lambda}} \phi^*(\boldsymbol{x},\boldsymbol{\lambda})$, has the following form:

$$\begin{aligned}
&\max \lambda_s - \lambda_t \\
&\lambda_i \leq \lambda_j + \overline{w}_{ij}x_{ij} + \underline{w}_{ij}(1-x_{ij}) \quad \text{for } (i,j)\in A
\end{aligned} \tag{7.8}$$

Now, from (3.6), (7.6) and (7.8) we get the following MIP formulation for Minmax Regret Shortest Path for a directed graph $G=(V,A)$:

$$\begin{aligned}
&\min \sum_{(i,j)\in A} \overline{w}_{ij}x_{ij} - \lambda_s + \lambda_t \\
&\lambda_i \leq \lambda_j + \overline{w}_{ij}x_{ij} + \underline{w}_{ij}(1-x_{ij}) \quad \text{for } (i,j)\in A \\
&\sum_{\{i:\,(j,i)\in A\}} x_{ji} - \sum_{\{k:\,(k,j)\in A\}} x_{kj} = \begin{cases} 1 & \text{for } j = s \\ 0 & \text{for } j \in V\setminus\{s,t\} \\ -1 & \text{for } j = t \end{cases} \\
&x_{ij} \in \{0,1\} \quad \text{for } (i,j)\in A
\end{aligned} \tag{7.9}$$

In the next section we construct a similar model for undirected graphs.

7.5.2 MIP Formulation for Undirected Graphs

Consider an undirected graph $G=(V,E)$ with two distinguished nodes s and t. Define variable $x_{\{i,j\}}$, which expresses whether edge $\{i,j\}$ is in the constructed path. Every path from s to t in G may traverse edge $\{i,j\}\in E$ in direction (i,j)

or (j,i). Let us construct digraph $G'=(V,A')$ by replacing every edge $\{i,j\}$ of G with two arcs (i,j) and (j,i). Let us associate variable $x'_{ij}\in\{0,1\}$ with every arc of G' and consider the following set of constraints:

$$\begin{aligned}
&\sum_{\{i:(j,i)\in A'\}} x'_{ji} - \sum_{\{k:(k,j)\in A'\}} x'_{kj} = \begin{cases} 1 & \text{for } j=s \\ 0 & \text{for } j\in V\setminus\{s,t\} \\ -1 & \text{for } j=t \end{cases} \\
&x_{\{i,j\}} = x'_{ij} + x'_{ji} \quad \text{for } \{i,j\}\in E \\
&x'_{ij}\in\{0,1\} \quad \text{for } (i,j)\in A' \\
&x_{\{i,j\}}\in\{0,1\} \quad \text{for } \{i,j\}\in E
\end{aligned} \tag{7.10}$$

It is clear that constraints (7.10) describe set $ch(\Phi)$ in the problem. Now the relaxed subproblem $\min_{\boldsymbol{y}\in ch(\Phi)}\phi(\boldsymbol{x},\boldsymbol{y})$ is as follows:

$$\begin{aligned}
&\min \textstyle\sum_{\{i,j\}\in E}(\overline{w}_{\{i,j\}}x_{\{i,j\}} + \underline{w}_{\{i,j\}}(1-x_{\{i,j\}}))(y'_{ij}+y'_{ji}) \\
&\sum_{\{i:(j,i)\in A'\}} y'_{ji} - \sum_{\{k:(k,j)\in A'\}} y'_{kj} = \begin{cases} 1 & \text{for } j=s \\ 0 & \text{for } j\in V\setminus\{s,t\} \\ -1 & \text{for } j=t \end{cases} \\
&y'_{ij}\geq 0 \quad \text{for } (i,j)\in A'
\end{aligned} \tag{7.11}$$

We have substituted $y_{\{i,j\}}=y'_{ij}+y'_{ji}$ in (7.11) and we have made use of the fact that $y'_{ij}\leq 1$ and $y'_{ij}+y'_{ji}\leq 1$ in an optimal solution to (7.11). Taking the dual to (7.11) we get the following problem $\max_{\boldsymbol{\lambda}}\phi^*(\boldsymbol{x},\boldsymbol{\lambda})$:

$$\begin{aligned}
&\max \lambda_s - \lambda_t \\
&\lambda_i \leq \lambda_j + \overline{w}_{\{i,j\}}x_{\{i,j\}} + \underline{w}_{\{i,j\}}(1-x_{\{i,j\}}) \quad \text{for } \{i,j\}\in E \\
&\lambda_j \leq \lambda_i + \overline{w}_{\{i,j\}}x_{\{i,j\}} + \underline{w}_{\{i,j\}}(1-x_{\{i,j\}}) \quad \text{for } \{i,j\}\in E
\end{aligned} \tag{7.12}$$

Now, from (3.6), (7.10) and (7.12) we get the following MIP formulation for Minmax Regret Shortest Path for an undirected graph $G=(V,E)$:

$$\begin{aligned}
&\min \sum_{\{i,j\}\in E}\overline{w}_{\{i,j\}}x_{\{i,j\}} - \lambda_t + \lambda_t \\
&\lambda_i \leq \lambda_j + \overline{w}_{\{i,j\}}x_{\{i,j\}} + \underline{w}_{\{i,j\}}(1-x_{\{i,j\}}) \quad \text{for } \{i,j\}\in E \\
&\lambda_j \leq \lambda_i + \overline{w}_{\{i,j\}}x_{\{i,j\}} + \underline{w}_{\{i,j\}}(1-x_{\{i,j\}}) \quad \text{for } \{i,j\}\in E \\
&\sum_{\{i:(j,i)\in A'\}} x'_{ji} - \sum_{\{k:(k,j)\in A'\}} x'_{kj} = \begin{cases} 1 & \text{for } j=s \\ 0 & \text{for } j\in V\setminus\{s,t\} \\ -1 & \text{for } j=t \end{cases} \\
&x_{\{i,j\}} = x'_{ij} + x'_{ji} \quad \text{for } \{i,j\}\in E \\
&x'_{ij}\in\{0,1\} \quad \text{for } (i,j)\in A' \\
&x_{\{i,j\}}\in\{0,1\} \quad \text{for } \{i,j\}\in E
\end{aligned} \tag{7.13}$$

7.5.3 Computational Results

Let us now present some computational results. We explore the behavior of the MIP approach as well as the approximation algorithms `Algorithm AM` and `Algorithm AMU`, constructed in Chapter 4. In our tests we used the following families of graphs:

- *Layered directed graphs.* A graph of type $L(|V|, w, c)$ is a layered digraph of width w, which has $|V|$ nodes. The interval weights are generated randomly in such way that $0 \leq \overline{w}_{ij} \leq c$ and $0 \leq \underline{w}_{ij} \leq \overline{w}_{ij}$ for all $(i, j) \in A$
- *Random directed graphs.* A graph of type $D(|V|, c, \delta)$ has $|V|$ nodes and an approximate arc density of δ, that is an arc between two nodes exists with

Table 7.1. Computational results for layered digraphs of width 2

		`Algorithm AM`		`Algorithm AMU`	
Graph	CPU Time	worst %	aver. %	worst %	aver. %
$L(80, 2, 20)$	0.146	5.56	1.99	5.56	1.99
$L(80, 2, 50)$	0.223	6.67	2.75	4.65	1.31
$L(80, 2, 100)$	0.244	11.58	2.78	11.58	2.52
$L(160, 2, 20)$	1.191	8.76	3.72	8.76	3.11
$L(160, 2, 50)$	1.559	7.97	5.17	7.97	4.38
$L(160, 2, 100)$	1.180	9.25	5.12	9.14	4.76
$L(240, 2, 20)$	4.408	8.17	2.75	3.11	2.11
$L(240, 2, 50)$	5.038	5.80	4.00	4.85	3.32
$L(240, 2, 100)$	12.486	4.32	2.78	4.32	2.60
$L(320, 2, 20)$	13.094	7.22	3.89	7.22	3.63
$L(320, 2, 50)$	20.128	5.92	3.44	5.92	3.26
$L(320, 2, 100)$	25.232	5.50	3.39	5.50	3.26

Table 7.2. Computational results for layered digraphs of width 3

		`Algorithm AM`		`Algorithm AMU`	
Graph	CPU Time	worst %	aver. %	worst %	aver. %
$L(120, 3, 20)$	0.671	11.24	5.53	6.25	2.76
$L(120, 3, 50)$	0.646	9.38	2.16	1.37	0.45
$L(120, 3, 100)$	0.836	12.57	4.32	2.81	1.07
$L(240, 3, 20)$	2.730	8.59	4.12	2.58	0.86
$L(240, 3, 50)$	3.805	7.65	4.28	4.22	1.69
$L(240, 3, 100)$	4.273	7.11	4.95	3.01	1.22
$L(360, 3, 20)$	8.880	10.34	4.46	2.96	1.48
$L(360, 3, 50)$	14.128	6.64	4.75	4.14	1.61
$L(360, 3, 100)$	18.623	8.61	4.19	4.07	2.19
$L(480, 3, 20)$	67.394	7.40	4.38	2.26	1.95
$L(480, 3, 50)$	45.791	7.52	4.57	3.43	2.19
$L(480, 3, 100)$	50.121	7.11	4.93	4.30	2.66

Table 7.3. Computational results for layered digraphs of width 4

Graph	CPU Time	Algorithm AM worst %	Algorithm AM aver. %	Algorithm AMU worst %	Algorithm AMU aver. %
$L(160, 4, 20)$	5.063	17.44	5.20	3.86	0.95
$L(160, 4, 50)$	1.127	7.37	3.88	2.13	1.17
$L(160, 4, 100)$	1.299	20.87	7.67	5.80	1.76
$L(320, 4, 20)$	5.132	11.59	5.60	6.58	1.98
$L(320, 4, 50)$	5.989	8.45	4.82	2.58	1.15
$L(320, 4, 100)$	4.813	10.89	5.94	3.33	1.25
$L(480, 4, 20)$	13.115	9.73	5.30	3.62	1.58
$L(480, 4, 50)$	13.163	9.85	5.25	2.73	1.08
$L(480, 4, 100)$	17.864	6.83	3.82	2.46	0.73
$L(640, 4, 20)$	30.352	9.12	6.19	3.61	1.66
$L(640, 4, 50)$	51.138	8.73	5.63	2.62	1.09
$L(640, 4, 100)$	68.617	7.99	4.96	2.23	1.20

Table 7.4. Computational results for random directed and undirected graphs

Graph	CPU Time	Algorithm AM worst %	Algorithm AM aver. %	Algorithm AMU worst %	Algorithm AMU aver. %
$D(100, 10, 0.1)$	0.175	16.67	1.67	0.00	0.00
$D(100, 100, 0.1)$	0.238	12.70	4.26	4.26	0.43
$D(100, 1000, 0.1)$	0.321	26.84	4.13	0.00	0.00
$D(500, 10, 0.01)$	2.37	60.00	6.56	14.29	0.57
$D(500, 100, 0.01)$	3.8	48.78	3.15	5.00	0.14
$D(500, 1000, 0.01)$	3.1	19.61	1.03	1.03	0.02
$U(100, 10, 0.1)$	0.31	33.33	2.5	0.00	0.00
$U(100, 100, 0.1)$	0.46	46.67	6.9	5.71	0.20
$U(100, 1000, 0.1)$	0.51	35.79	2.82	0.00	0.00
$U(500, 10, 0.01)$	3.04	20.00	3.48	0.00	0.00
$U(500, 100, 0.01)$	4.66	41.98	3.82	9.84	0.33
$U(500, 1000, 0.01)$	4.1	13.36	2.38	1.9	0.07

probability δ. The interval weights are generated randomly in such way that $0 \leq \overline{w}_{ij} \leq c$ and $0 \leq \underline{w}_{ij} \leq \overline{w}_{ij}$ for all $(i, j) \in A$. Nodes s and t are chosen randomly.

- *Random undirected graphs.* A graph of type $U(|V|, c, \delta)$ has $|V|$ nodes and an approximate edge density of δ, that is an edge between two nodes exists with probability δ. The interval weights are generated randomly in such way that $0 \leq \overline{w}_{ij} \leq c$ and $0 \leq \underline{w}_{ij} \leq \overline{w}_{ij}$ for all $(i, j) \in A$. Nodes s and t are chosen randomly.

For every particular class of graphs 10 instances were generated and solved. In order to obtain the optimal solutions CPLEX 8.0 and a Pentium III 866 MHz computer were used for solving the MIP model. The optimal solutions were

compared to the solutions returned by `Algorithm AM` and `Algorithm AMU`. The obtained results are presented in Tables 7.1-7.4. In the second column of each table the average CPU times in seconds required for solving the MIP model are shown. In the next columns the worst and the average percentage deviations from optimum reported for the approximation algorithms `Algorithm AM` and `Algorithm AMU` are presented.

As can be seen from the experimental results, for the tested classes of graphs the MIP approach appears to be quite efficient. For random directed and undirected graphs the optimal robust path can be obtained in a few seconds. For layered digraphs the MIP approach appears to be less efficient. Both approximation algorithms also behave reasonably. Notice that `Algorithm AMU` performs better than `Algorithm AM` and it is mostly within a few percentage points off the optimum (the worst reported deviation was 14.29%). The average deviation from optimum for `Algorithm AMU` is less than 5% for every class of graphs. Some results of computational tests using the MIP formulation can also be found in the papers by Karasan *et al.* [70] and Montemanni *et al.* [103]. In general the layered digraphs seem to be the hardest instances for the MIP approach.

7.6 Branch and Bound Algorithm

In this section we apply the framework of the branch and bound algorithm constructed in Section 3.2 to Minmax Regret Shortest Path. In order to do this we must specify a branching method, reduction rules and a method of deriving a feasible solution from $\Phi_{<Q,R>}$. Recall that $\Phi_{<Q,R>}$ is a subset of Φ containing all paths P from s to t in G such that $Q \subseteq P$ and no arc (edge) from R is in P. Structure $< Q, R >$ describes a node d of the search tree. We will assume that the input graph is directed, however, all the presented elements of the algorithm can be applied to undirected graphs as well. The resulting branch and bound algorithm is the one constructed by Montemanni *et al.* [103].

7.6.1 Branching Rule

Let d be a node of the search tree to be branched. This node is described by structure $< Q, R >$, which defines the subset of paths $\Phi_{<Q,R>}$. Let P_d be the shortest path in $\Phi_{<Q,R>}$ under scenario S_E^+, that is under the one in which the weights of all arcs are at their upper bounds. Initially, the root d_0 of the search tree is described by $< \emptyset, \emptyset >$ and P_{d_0} is the shortest path in G under S_E^+. If node d is to be branched, then the first arc a on the path P_d (starting from node s), which is not in Q is selected. Two new nodes $d' =< Q \cup \{a\}, R >$ and $d'' =< Q, R \cup \{a\} >$ are created. It is easy to prove by induction that the branching method assures that in every node $< Q, R >$ the nonempty set Q is always a path from s to a certain node of G. Moreover, Q is a subpath of a certain path from s to t in G. It can also be proved (see [103]) that the branching rule leads to an exhaustive search of the solution space.

The proposed branching rule has some additional important properties. It is easy to compute path P_d in node $d =< Q, R >$. If $Q = \emptyset$, then P_d is the shortest

path from s to t in subgraph $G' = (V, A \setminus R)$ under scenario S_E^+. If Q is a path from s to v in G. Then P_d is formed by Q and by the shortest path from v to t under scenario S_E^+ in $G' = (V, A \setminus R)$. Having P_d it is also easy to compute the value of the lower bound in node d. From Theorem 3.3 we have

$$LB(d) = F^*_{<Q,R>}(S_E^+) - F^*(S^+_{E\setminus R}).$$

But $F^*_{<Q,R>}(S_E^+) = F(P_d, S_E^+)$ since P_d is the shortest path in $\Phi_{<Q,R>}$ under scenario S_E^+. Therefore the lower bound $LB(d)$ can also be efficiently computed by solving two deterministic shortest path problems.

7.6.2 Reduction Rules

The first reduction rule can be applied if the input graph is acyclic. In this case we can detect in polynomial time a necessarily optimal arc. If such an arc exists, then we can split the problem into two smaller ones as shown in Section 7.4.2. We can again seek the necessarily optimal arcs in both subproblems. As a result we may obtain a set of smaller problems to which the branch and bound algorithm can be next applied separately. If the input digraph is acyclic, then it may also be advantageous to apply Proposition 7.10 and detect some nonpossibly optimal arcs. Such arcs can be then removed from the input graph, which clearly reduce the size of the search tree.

The second reduction rule is closely connected with the assumed branching method. Suppose we are at node $d =< Q, R >$ of the search tree. Let (u, v) be an arc in Q, that is (u, v) is on every path in $\Phi_{<Q,R>}$. It is evident that arc (u, i), $i \neq v$, cannot be on any simple path in $\Phi_{<Q,R>}$ and it can be automatically added to R. The same reasoning applies to arc (i, v), $i \neq u$. Hence every time we add to Q arc (u, v) we automatically add to R all outcoming arcs of u and all incoming arcs of v other than (u, v). This reduces the size of the search tree and gives a tighter lower bound $LB(d)$.

7.6.3 Deriving a Feasible Solution

We can assume that P_d is the feasible solution deriven from $\Phi_{<Q,R>}$. Obviously $Z(P_d)$ is the upper bound on the value of the maximal regret of the optimal robust solution.

7.6.4 Computational Results

We now present the computational results in order to evaluate the performance of the branch and bound algorithm. These results are according to Montemanni *et al.* [103]. The algorithm was tested on the families of directed random graphs described in Section 7.5.3. For every family 10 problems were created and solved. The computational results are shown in Table 7.5.

In the column MIP the average CPU time required to solve the MIP formulation and in column BB the average CPU time required for branch and bound

Table 7.5. The average computational times required for solving the MIP formulation and for the branch and bound algorithm [103]

Graph	MIP	BB
$D(500, 100, 0.01)$	4.663	1.481
$D(500, 100, 0.001)$	0.578	0.599
$D(500, 100, 0.1)$	14.365	2.388
$D(100, 100, 0.01)$	0.079	0.046
$D(900, 100, 0.01)$	25.161	5.244
$D(500, 10, 0.01)$	5.033	0.553
$D(500, 1000, 0.01)$	5.863	3.069

algorithm are shown. All the tests in [103] were performed using a Pentium II 400 MHz computer. In order to solve the MIP model a CPLEX 6.0 was used.

One can observe, that the branch and bound algorithm performs better than the MIP approach for random directed graphs. However, for directed layered graphs, which were also considered in [103], the CPLEX appeared to be faster than the branch and bound algorithm. Montemanni *et al.* [103] also tested the branch and bound algorithm for two graphs that model real road networks. The first graph had 387 nodes and 1038 arcs and the second graph had 2490 nodes and 16 153 arcs. The branch and bound algorithm required respectively 0.115 and 6.285 seconds to solve the problems, which is significantly faster than solving the MIP formulation by means of CPLEX. The computational results suggest that the branch and bound algorithm is a good choice for large realistic problems.

7.7 Series-Parallel Multidigraphs

In Section 7.1 we recalled the definition of a special class of directed graphs, namely edges series-parallel multidigraphs (shortly ESP). Every ESP $G = (V, A)$ has a special structure defined recursively by series and parallel compositions, which can be represented by a binary decomposition tree $\mathcal{T}(G)$. This tree can be constructed in $\mathcal{O}(|A|)$ time by applying an algorithm constructed by Valdes *et al.* [120].

An example of ESP together with its binary decomposition tree is shown in Figure 7.3. The special structure of ESP makes some NP-hard problems to be polynomially solvable for this class of graphs. Unfortunately, this is not the case for Minmax Regret Shortest Path. In Section 7.2 we proved that Minmax Regret Shortest Path becomes NP-hard even for some restrictive ESP's. We did not prove, however, that the problem is strongly NP-hard and there is a hope that a pseudopolynomial algorithm for this class of graphs exists. In this section we construct such an algorithm. Furthermore, we show that using this algorithm it is possible to apply the results obtained in Section 4.2 and construct an FPTAS for the problem.

7.7.1 Pseudopolynomial Algorithm

Let $G = (V, A)$ be an edge series-parallel multidigraph with source $s \in V$ and sink $t \in V$. From now on we will assume that all bounds of the uncertainty intervals are integer. Let us start by introducing some additional notations and assumptions.

- The arcs in path $P = \{a_1, a_2, \ldots, a_k\}$ are indexed so that the end node of a_i equals the start node of a_{i+1}. We will also use an alternative notation $P = a_1 a_2 \ldots a_k$ for path P;
- Φ_G is the set of all paths from s to t in a specified ESP G;
- $Z_G(P)$ is the maximal regret of path $P \in \Phi_G$ in the specified graph G;
- $\overline{L}_G(P)$ is the weight (length) of path $P \in \Phi_G$ under the scenario in which the weights of all arcs are at their upper bounds, that is $\overline{L}_G(P) = \sum_{a \in P} \overline{w}_a$;
- $\underline{L}^*_G$ is the weight (length) of the shortest path from s to t in G under the scenario in which the weights of all arcs are at their lower bounds, that is $\underline{L}^*_G = \min_{P \in \Phi_G} \sum_{a \in P} \underline{w}_a$;
- Notation $P_1 \circ P_2$ denotes a path that is constructed by identifying the end node of path P_1 with the start node of path P_2, i.e. if $P_1 = a_1 a_2 \ldots a_k$ and $P_2 = b_1 b_2 \ldots b_l$, then $P_1 \circ P_2 = a_1 a_2 \ldots a_k b_1 b_2 \ldots b_l$;
- notation $P_1 \preceq_G P_2$ means that $\overline{L}_G(P_1) \leq \overline{L}_G(P_2)$ and $Z_G(P_1) \leq Z_G(P_2)$. It is easy to check that relation $\preceq_G$ is *transitive*, that is $P_1 \preceq_G P_2$ and $P_2 \preceq_G P_3$ implies $P_1 \preceq_G P_3$.

The following proposition describes set Φ_G in a given ESP G:

Proposition 7.12. *Let $G = (V, A)$ be a given ESP with source s and sink t.*

- *If G consists of single arc (s, t), then $\Phi_G = \{(s, t)\}$.*
- *If $G = \mathtt{p}(G_1, G_2)$ then $\Phi_G = \Phi_{G_1} \cup \Phi_{G_2}$.*
- *If $G = \mathtt{s}(G_1, G_2)$ then $\Phi_G = \{P_1 \cup P_2 :\ P_1 \in \Phi_{G_1},\ P_2 \in \Phi_{G_2}\}$ (or equivalently $\Phi_G = \{P_1 \circ P_2 :\ P_1 \in \Phi_{G_1},\ P_2 \in \Phi_{G_2}\}$).*

Proof. It is a direct consequence of the recursive definition of ESP. □

The next two propositions show how to compute the maximal regret of a given path, making use of the particular structure of ESP.

Proposition 7.13. *Let $G = \mathtt{s}(G_1, G_2)$ be a series composition of G_1 and G_2, $P_1 \in \Phi_{G_1}$ and $P_2 \in \Phi_{G_2}$. Then it holds:*

$$Z_G(P_1 \circ P_2) = Z_{G_1}(P_1) + Z_{G_2}(P_2). \tag{7.14}$$

Proof. It is a consequence of the definition of the series composition. □

Proposition 7.14. *Let $G = \mathtt{p}(G_1, G_2)$ be a parallel composition of G_1 and G_2, $P_1 \in \Phi_{G_1}$, $P_2 \in \Phi_{G_2}$. Then it holds:*

$$\begin{aligned} Z_G(P_1) &= \max\{Z_{G_1}(P_1), \overline{L}_{G_1}(P_1) - \underline{L}^*_{G_2}\}, \\ Z_G(P_2) &= \max\{Z_{G_2}(P_2), \overline{L}_{G_2}(P_2) - \underline{L}^*_{G_1}\}. \end{aligned} \tag{7.15}$$

Proof. Consider the case when $P_1 \in \Phi_{G_1}$ (the same reasoning applies to the case $P_2 \in \Phi_{G_2}$). From the definition of the parallel composition, it follows that $\Phi_G = \Phi_{G_1} \cup \Phi_{G_2}$ and $\Phi_{G_1} \cap \Phi_{G_2} = \emptyset$. In consequence, $P_1 \in \Phi_G$. Let P_1^* be the worst case alternative for P_1 in G. Since $\Phi_{G_1} \cap \Phi_{G_2} = \emptyset$, either $P_1^* \in \Phi_{G_1}$ or $P_1^* \in \Phi_{G_2}$. If $P_1^* \in \Phi_{G_1}$, then equation $Z_G(P_1) = Z_{G_1}(P_1)$ holds, because P_1^* is also the worst case alternative for P_1 in G_1. On the other hand, if $P_1^* \in \Phi_{G_2}$, then it is clear that P_1^* is the shortest path in G_2 under scenario in which the weights of all arcs are at their lower bounds, that is $Z_G(P_1) = \overline{L}_{G_1}(P_1) - \underline{L}^*_{G_2}$. Hence the value of the maximal regret of P_1 in G equals (7.15). □

`Series`$(\Phi^*_{G_1}, \Phi^*_{G_2})$
Require: $\Phi^*_{G_1}$, $\Phi^*_{G_2}$
Ensure: $\Phi^*_{\mathtt{s}(G_1,G_2)}$
1: $\Phi^*_G \leftarrow \emptyset$
2: **for all** $P_1 \in \Phi^*_{G_1}$ **do**
3: **for all** $P_2 \in \Phi^*_{G_2}$ **do**
4: $\Phi^*_G \leftarrow \Phi^*_G \cup \{(P_1 \circ P_2, \overline{L}_{G_1}(P_1) + \overline{L}_{G_2}(P_2), Z_{G_1}(P_1) + Z_{G_2}(P_2))\}$
5: **end for**
6: **end for**
7: **while** there exist two paths $P_1, P_2 \in \Phi^*_G$ such that $P_1 \neq P_2$ and $P_1 \preceq_G P_2$ **do**
8: Remove P_2 from Φ^*_G
9: **end while**
10: **return** Φ^*_G

Fig. 7.11. Algorithm that computes $\Phi^*_{s(G_1,G_2)}$

The idea of the algorithm to solve the problem consists of performing the series and parallel compositions in G. During each composition a certain amount of information is stored, which allows us to retrieve the optimal robust path when the algorithm terminates. Let us associate with a given ESP G set Φ^*_G, which consists of triples $(P, \overline{L}_G(P), Z_G(P))$, where $P \in \Phi_G$. For simplicity, we will treat Φ^*_G as a subset of Φ_G with additional information stored for each path. Set Φ^*_G is defined recursively as follows:

- If G consists of a single arc a, then $\Phi^*_G = \{(\{a\}, \overline{c}_a, 0)\}$.
- If $G = \mathtt{s}(G_1, G_2)$, then Φ^*_G is constructed by algorithm `Series`$(\Phi^*_{G_1}, \Phi^*_{G_2})$ shown in Figure 7.11.
- If $G = \mathtt{p}(G_1, G_2)$, then P_G is constructed by algorithm `Parallel`$(\Phi^*_{G_1}, \Phi^*_{G_2})$ shown in Figure 7.12.

Notice that, from Propositions 7.13 and 7.14, it follows that algorithms `Series` and `Parallel` correctly calculate the value of the maximal regret of every $P \in \Phi^*_G$. Observe that during the execution of both algorithms some paths are removed. The crucial fact is that Φ^*_G always contains an optimal robust path in G, thus the optimal robust path is not lost. This is the consequence of the following proposition:

`Parallel`$(\Phi^*_{G_1}, \Phi^*_{G_2})$
Require: $\Phi^*_{G_1}$, $\Phi^*_{G_2}$
Ensure: $\Phi^*_{\mathtt{p}(G_1,G_2)}$
1: $\Phi^*_G \leftarrow \emptyset$
2: **for all** $P \in \Phi^*_{G_1}$ **do**
3: $\quad \Phi^*_G \leftarrow \Phi^*_G \cup \{(P, \overline{L}_{G_1}(P), \max\{Z_{G_1}(P), \overline{L}_{G_1}(P) - \underline{L}^*_{G_2}\})\}$
4: **end for**
5: **for all** $P \in \Phi^*_{G_2}$ **do**
6: $\quad \Phi^*_G \leftarrow \Phi^*_G \cup \{(P, \overline{L}_{G_2}(P), \max\{Z_{G_2}(P), \overline{L}_{G_2}(P) - \underline{L}^*_{G_1}\})\}$
7: **end for**
8: **while** there exist two paths $P_1, P_2 \in \Phi^*_G$ such that $P_1 \neq P_2$ and $P_1 \preceq_G P_2$ **do**
9: $\quad$ Remove P_2 from Φ^*_G
10: **end while**
11: **return** Φ^*_G

Fig. 7.12. Algorithm that computes $\Phi^*_{p(G_1,G_2)}$

Proposition 7.15. *For every path P in Φ_G, there exists a path $Q \in \Phi^*_G$ such that $Q \preceq_G P$.*

Proof. We will prove the proposition by induction on the number of arcs in G. The proposition is trivial if G consists of a single arc, since $\Phi^*_G = \Phi_G$ in that case. Now assume that it is true for all ESP in which the number of arcs is less than or equal to m, where $m \geq 1$. Consider an ESP $G = (V, A)$ with $m+1$ arcs. Since $m + 1 \geq 2$, it follows that G is a result of series or parallel composition of two ESP G_1 and G_2. Moreover, the number of arcs in G_1 and G_2 is less than $m + 1$ and by the induction hypothesis, the proposition is true for G_1 and G_2.

Suppose that $G = \mathtt{s}(G_1, G_2)$. Consider a path $P \in \Phi_G$. Since G is the series composition of G_1 and G_2, we have $P = P_1 \circ P_2$, where $P_1 \in \Phi_{G_1}$ and $P_2 \in \Phi_{G_2}$. By the induction hypothesis we obtain that there exists $Q_1 \in \Phi^*_{G_1}$ and $Q_2 \in \Phi^*_{G_2}$ such that $Q_1 \preceq_{G_1} P_1$ and $Q_2 \preceq_{G_2} P_2$. It is easy to verify, using Proposition 7.13, that $Q = Q_1 \circ Q_2 \preceq_G P_1 \circ P_2 = P$. Moreover, path Q is added to Φ^*_G by algorithm `Series` because $Q_1 \in \Phi^*_{G_1}$ and $Q_2 \in \Phi^*_{G_2}$. If $Q \in \Phi^*_G$ then we are done. Otherwise Q is removed from Φ^*_G by algorithm `Series`. This means that there is another path $Q_1 \in \Phi^*_G$ such that $Q_1 \preceq_G Q \preceq_G P$, since relation $\prec_G$ is transitive.

Suppose on the other hand that $G = \mathtt{p}(G_1, G_2)$. Consider a path $P \in \Phi_G$. Since G is the parallel composition of G_1 and G_2, it follows that either $P \in \Phi_{G_1}$ or $P \in \Phi_{G_2}$. Assume that $P \in \Phi_{G_1}$ (the case $P \in \Phi_{G_2}$ is similar). By the induction hypothesis, we obtain that there exists a path $Q \in \Phi^*_{G_1}$ such that $Q \preceq_{G_1} P$. From the definition of the parallel composition, it results that $Q \in \Phi_G$. Making use of Proposition 7.14 we get:

$$Z_G(Q) = \max\{Z_{G_1}(Q), \overline{L}_{G_1}(Q) - \underline{L}^*_{G_2}\},$$

$$Z_G(P) = \max\{Z_{G_1}(P), \overline{L}_{G_1}(P) - \underline{L}^*_{G_2}\}.$$

Since $\overline{L}_{G_1}(Q) \leq \overline{L}_{G_1}(P)$ and $Z_{G_1}(Q) \leq Z_{G_1}(P)$ (because $Q \preceq_{G_1} P$) we have $Z_G(Q) \leq Z_G(P)$. Moreover, since $\overline{L}_G(Q) = \overline{L}_{G_1}(Q) \leq \overline{L}_{G_1}(P) = \overline{L}_G(P)$, we

conclude that $Q \preceq_G P$. Furthermore, path Q is added to Φ^*_G by Algorithm `Parallel` because $Q \in \Phi^*_{G_1}$. If $Q \in \Phi^*_G$ then we are done. Otherwise Q is removed from Φ^*_G by algorithm `Parallel`. This means that there is another path $Q_1 \in \Phi^*_G$ such that $Q_1 \preceq_G Q \preceq_G P$, since relation $\prec_G$ is transitive.

We have thus proved that the proposition is true for G. □

A direct consequence of Proposition 7.15 and the definition of relation $\preceq_G$ is the following proposition:

Proposition 7.16. *Set Φ^*_G contains an optimal robust path in G.*

The next proposition gives a bound on the number of paths in Φ^*_G.

Proposition 7.17. *$|\Phi^*_G| \leq L_{max}$, where L_{max} is the length of the longest path from s to t in G under the scenario in which the weights of all arcs are at their upper bounds.*

Proof. Assume that Φ^*_G consists of paths $P_1, \ldots, P_k$ such that

$$\overline{L}_G(P_1) \leq \overline{L}_G(P_2) \leq \cdots \leq \overline{L}_G(P_k).$$

Consider two paths P_i and P_{i+1}. If $\overline{L}_G(P_i) = \overline{L}_G(P_{i+1})$, then either $P_i \preceq_G P_{i+1}$ or $P_{i+1} \preceq_G P_i$ and one of these paths is removed from P_G by algorithms `Series` or `Parallel`. Therefore it must hold

$$\overline{L}_G(P_1) < \overline{L}_G(P_2) < \cdots < \overline{L}_G(P_k).$$

All the lengths $\overline{L}_G(P_i)$, $i = 1, \ldots, k$, are integers and $\overline{L}_G(P_k) \leq L_{max}$, so the number of paths in Φ^*_G cannot be greater than L_{max}. □

`Algorithm REG-ESP`
Require: An ESP $G = (V, A)$ with interval arc weights.
Ensure: The optimal robust path in G.
1: Determine binary decomposition tree $\mathcal{T}(G)$ of G
2: **for all** leaves δ of $\mathcal{T}(G)$ **do**
3: $\quad\Phi^*_{G_\delta} \leftarrow \{(\{a\}, \overline{w}_a, 0)\}$ {δ corresponds to arc a in G}
4: $\quad\underline{L}^*_{G_i} \leftarrow \underline{w}_a$
5: **end for**
6: **while** there is a node δ in $\mathcal{T}(G)$ with two leaf nodes `Left`(δ) and `Right`(δ) **do**
7: $\quad$**if** δ is marked with s **then**
8: $\quad\quad\Phi^*_{G_\delta} \leftarrow$`Series`$(\Phi^*_{\texttt{Left}(\delta)}, \Phi^*_{\texttt{Right}(\delta)})$
9: $\quad\quad\underline{L}^*_{G_\delta} \leftarrow \underline{L}^*_{\texttt{Left}(\delta)} + \underline{L}^*_{\texttt{Right}(\delta)}$
10: $\quad$**else**
11: $\quad\quad\Phi^*_{G_\delta} \leftarrow$`Parallel`$(\Phi^*_{\texttt{Left}(\delta)}, \Phi^*_{\texttt{Right}(\delta)})$
12: $\quad\quad\underline{L}^*_{G_\delta} \leftarrow \min\{\underline{L}^*_{\texttt{Left}(\delta)}, \underline{L}^*_{\texttt{Right}(\delta)}\}$
13: $\quad$**end if**
14: $\quad$Delete `Right`(δ) and `Left`(δ) from $\mathcal{T}(G)$
15: **end while**
16: **return** $\arg\min_{P \in \Phi^*_G} Z_G(\pi)$ {Φ^*_G corresponds to the root of $\mathcal{T}(G)$}

Fig. 7.13. An algorithm for solving Minmax Regret Shortest Path for ESP

We are now ready to give an algorithm that computes the optimal robust path for a given ESP G. This algorithm is shown in Figure 7.13. First, the binary decomposition tree $\mathcal{T}(G)$ is determined. Every leaf δ of this tree corresponds to an arc a in G. In lines 2-5 we initialize sets $\Phi^*_{G_\delta}$ for all leaves $\delta \in \mathcal{T}(G)$. The algorithm computes then recursively sets $\Phi^*_{G_\delta}$ for all internal nodes $\delta \in \mathcal{T}(G)$, which correspond to subgraphs G_δ of G. Finally, we obtain set Φ^*_G that corresponds to the root of $\mathcal{T}(G)$. By Proposition 7.16, this set must contain an optimal robust path and this is precisely the path $P \in \Phi^*_G$ that has the minimal value of $Z_G(P)$.

Let us now estimate the running time of the constructed algorithm. The binary decomposition tree in line 1 can be constructed in $\mathcal{O}(|A|)$ time by the algorithm designed by Valdes *et al.* [120]. The initialization in lines 2 - 5 requires $\mathcal{O}(|A|)$ time. The number of series and parallel compositions in lines 6-15 is $\mathcal{O}(|A|)$. Algorithms `Series` and `Parallel` run in $\mathcal{O}(|\Phi^*_{G_\delta}|^2)$ time, which is bounded by $\mathcal{O}(L^2_{max})$ due to Proposition 7.17. Finding the optimal robust path in Φ^*_G in line 16 requires at most L_{max} steps. Hence the running time of the algorithm is $\mathcal{O}(|A|L^2_{max})$. Since $L_{max} \leq |A|w_{max}$, where $w_{max} = \max_{a \in A} \overline{w}_a$, the running time of the algorithm can also be expressed as $\mathcal{O}(|A|^2 w_{max})$. This running time is bounded by a polynomial in the input size and the maximal value of a number in the problem, so the algorithm runs in pseudopolynomial time.

Example 7.18. Let us illustrate the algorithm by an example. An ESP G with interval arc weights is shown in Figure 7.14a. Figure 7.14b demonstrates the binary

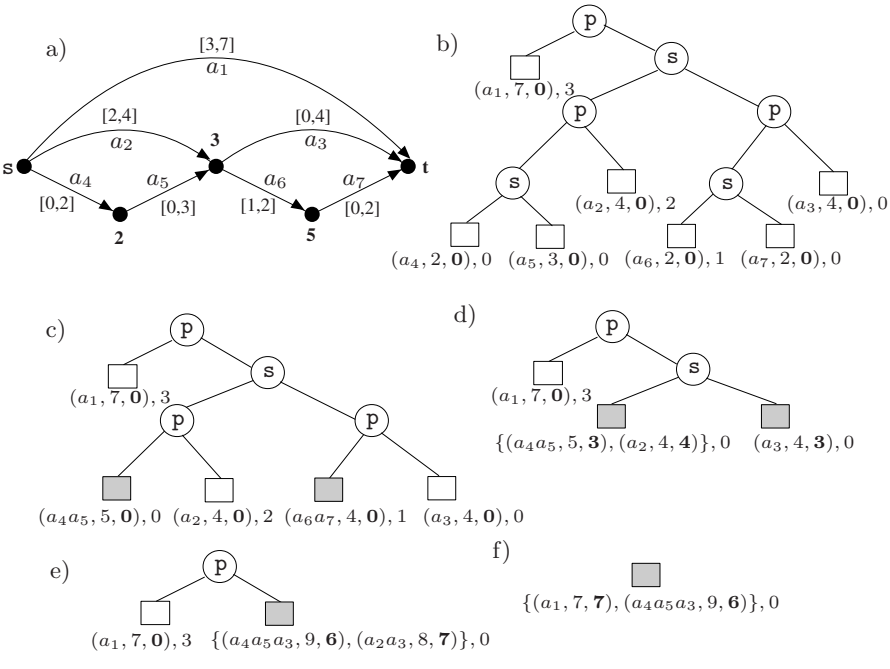


Fig. 7.14. The sample calculations

decomposition tree $\mathcal{T}(G)$. All leaves of this tree, that correspond to arcs in G, are initialized with $(a, \overline{w}_a, 0), \underline{w}_a$, $a \in A$. Figure 7.14c shows the binary decomposition tree after performing two series compositions. The decomposition tree in Figure 7.14d shows the tree after two next parallel compositions. Observe that in one of the leaves of this tree two subpaths must be stored. Finally, in Figure 7.14f the binary decomposition tree is reduced to the root and the obtained set Φ^*_G contains two paths in G, that is $P_1 = a_4a_5a_3$ with the maximal regret equal to 6 and $P_2 = a_1$ with the maximal regret equal to 7. Therefore the optimal robust path in G is $P_1 = a_4a_5a_3$ with the maximal regret equal to 6. □

7.7.2 Fully Polynomial Time Approximation Scheme

In Section 4.2 we proved that if there is an algorithm for the problem whose running time is bounded by a polynomial in the input size and the value of upper bound UB on OPT, then the problem admits an FPTAS. In the previous section we constructed a pseudopolynomial algorithm for the problem. However, its running time is not bounded by a polynomial in $|A|$ and UB. In this section w slightly modify this algorithm so that this required condition is satisfied.

Let UB be the upper bound on the value of the maximal regret of the optimal robust path in a given ESP $G = (V, A)$. This bound is computed by 2-approximation `Algorithm AM`, that is we compute the shortest path P for the midpoint scenario S, such that $w^S_a = \frac{1}{2}(\underline{w}_a + \overline{w}_a)$ for all $a \in A$, and set $UB = Z_G(P)$. We prove the following proposition:

Proposition 7.19. *Let G_δ be a subgraph of G induced by subtree of $\mathcal{T}(G)$ rooted at δ. If $P \in \Phi^*_{G_\delta}$ is a path such that $Z_{G_\delta}(P) > UB$, then P cannot be a part of the optimal robust path in G.*

Proof. Suppose that P is a subpath of an optimal robust path P_1. Then there must be a sequence of series and parallel compositions leading from δ to the root of $\mathcal{T}(G)$ that construct path P_1 from P. However, from Propositions 7.13 and 7.14 it follows that the maximal regrets of the subsequent subpaths cannot decrease after performing a series or a parallel composition. Therefore, we would have $Z(P_1) > UB$, which is a contradiction. □

From Proposition 7.19 it follows that we can remove from $\Phi^*_{G_\delta}$ all paths P such that $Z_{G_\delta}(P) > UB$ without violating the optimal robust path in G. We can thus modify algorithms `Series` and `Parallel` by adding a line in which all paths such that $Z_{G_\delta}(P) > UB$ are removed from Φ^*_G. Consider now set $\Phi^*_{G_\delta}$ computed at some node δ of $\mathcal{T}(G)$. Suppose that this set contains paths $P_1, P_2, \ldots P_k$, such that

$$Z_{G_\delta}(P_1) \leq Z_{G_\delta}(P_2) \leq \cdots \leq Z_{G_\delta}(P_k) \leq UB.$$

If $Z_{G_\delta}(P_i) = Z_{G_\delta}(P_{i+1})$ for some $i = 1, \ldots, k-1$, then $P_i \preceq_{G_\delta} P_{i+1}$ or $P_{i+1} \preceq_{G_\delta} P_i$, so one of these paths is removed from $\Phi^*_{G_\delta}$. Hence we have

$$Z_{G_\delta}(P_1) < Z_{G_\delta}(P_2) < \cdots < Z_{G_\delta}(P_k) \leq UB.$$

Since the maximal regrets of all paths are integers we conclude that $|\Phi^*_{G_\delta}| \leq UB$. Now, using the same reasoning as in the previous section, we can show that the running time of the algorithm that computes an optimal robust path (see Figure 7.13) is $\mathcal{O}(|A|UB^2)$. Observe that this running time is bounded by a bivariate polynomial in the input size $|A|$ and the upper bound UB. Moreover, the upper bound UB can be computed in polynomial time and it is such that $UB/LB = 2$, where LB is a lower bound on the maximal regret of the optimal robust path. In consequence, all the assumptions of Theorem 4.8 are fulfilled and we get the following result:

Theorem 7.20. *The* Minmax Regret Shortest Path *problem for ESP admits an FPTAS, that is an algorithm that for a given ESP $G = (V, A)$ computes path P such that $Z_G(P) \leq (1 + \epsilon)OPT$ in time $\mathcal{O}(|A|^3/\epsilon^2)$.*

The FPTAS for the problem is the algorithm shown in Figure 4.7. Algorithm `Exact` is the one shown in Figure 7.13 with the slight modifications of algorithms `Series` and `Parallel` presented in this section.

Recall that Minmax Regret Longest Path in acyclic digraphs can be transformed to Minmax Regret Shortest Path (see Section 7.2). Every ESP G is acyclic. Applying the transformation from Section 7.2 to G we obtain a digraph G' which also has a series-parallel topology. It follows form the fact that adding dummy arcs can be seen as performing additional series compositions in G. This leads to the following result:

Theorem 7.21. *Problem* Minmax Regret Longest Path *for ESP admits an FPTAS, that is an algorithm that for a given ESP $G = (V, A)$ computes path P such that $Z_G(P) \leq (1 + \epsilon)OPT$ in time $\mathcal{O}(|A|^3/\epsilon^2)$.*

7.8 Notes and References

An excellent review of different algorithms for solving the deterministic Shortest Path problem can be found in a book by Ahuja *et al.* [2]. For general graphs (directed or undirected), assuming that all weights in the problem are nonnegative, the shortest path can be computed in $\mathcal{O}(|V|^2)$ time be means of Dijkstra's algorithm [42]. If the input graph is acyclic, then a simple labeling algorithm outputs the shortest path in $\mathcal{O}(|E|)$ time.

The Minmax Regret Shortest Path problem was first discussed by Karasan *et al.* [70]. In [70] a MIP model for directed graphs was constructed, which is equivalent to the one presented in Section 7.5. Karasan *et al.* [70] also introduced the concept of weak (possibly optimal) arc and showed that one can set $x_{ij} = 0$ for all nonpossibly optimal arcs $(i, j) \in A$ before solving the MIP model. However, they were not able to detect efficiently all nonpossibly arcs. Only some of them were detected by means of the necessary condition presented in Section 7.4. Nevertheless, it was observed that the detected subset of nonpossibly optimal arcs had been quite large and removing them from a digraph significantly decreased the time required to solve the MIP model. It turns out

that all nonpossibly optimal arcs cannot be efficiently detected if P≠NP. This is a consequence of the result proved in Section 7.4, which states that asserting whether an arc is possibly optimal is strongly NP-complete.

We have seen in Section 7.4 that the problem of evaluating the possible and necessary optimality of a given arc is closely related to the problem of evaluating the possible and necessary criticality. The negative results concerning the possible criticality were obtained by Chanas an Zieliński [31, 32] and they are the basis for proving the hardness of evaluating the possible optimality. On the other hand, the necessary optimality of an arc in an acyclic digraph G can be verified in polynomial time by transforming G and applying the algorithm designed by Fortin *et al.* [52].

When the first paper concerning MINMAX REGRET SHORTEST PATH was published, the complexity status of the problem was unknow. The problem was proved to be computationally intractable by Zieliński [133], Averbakh and Lebedev [17] and Kasperski and Zieliński [79]. Zieliński [133] proved that the problem is NP-hard for planar graphs. At the same time Averbakh and Lebedev [17] proved that the problem is strongly NP-hard for undirected graphs and for directed layered graphs even if the bounds of all uncertainty intervals are 0 or 1. The reduction from HAMILTONIAN PATH used in the proof of Theorem 7.1 is according to Averbakh and Lebedev [17]. Finally, the fact that the problem remains NP-hard for edge series-parallel multidigraphs was proved by Kasperski and Zieliński [79].

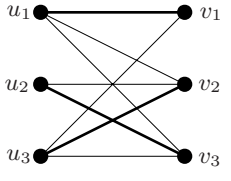
The MINMAX REGRET SHORTEST PATH problem can be solved by means of the branch and bound algorithm designed by Montemanni *et al.* [103]. This algorithm uses the framework (in particular the lower bound) shown in Section 3.2 and its details are described in Section 7.6. The pseudopolynomial algorithm for ESP was designed by Kasperski and Zieliński in [79] and the FPTAS was constructed by Kasperski and Zieliński in [83].

There is a number of open questions connected with the problem. Perhaps, the most important one is whether there exists an approximation algorithm for the problem with lower than 2 worst case performance ratio. There is also a lack of algorithms that exploit a particular structure of the input graph. The interesting cases are when the input graph is planar or layered of width 2.

In practical applications, for instance in scheduling or analysis the electrical circuits, the input graphs are either edge series-parallel or very close to be edge series-parallel digraphs. Bein *et al.* [22] introduced a concept of a *complexity* of a two-terminal acyclic digraph, which measures how nearly series-parallel topology this digraph is. If G is series-parallel, then it is possible to reduce G to a single arc by performing the series and parallel reductions, which are reverse to the series and parallel compositions. If an acyclic digraph G is not series-parallel, then a number of additional reductions, called node reductions, are required to reduce G to a single arc. The complexity of G is the minimal number of such node reductions and in practical applications this complexity is often a small number. We believe that this fact may be used to design an algorithm that is pseudopolynomial for a fixed graph complexity. Designing such an algorithm is an interesting subject of further research.

8 Minmax Regret Minimum Assignment

This chapter is devoted to Minmax Regret Minimum Assignment. In this problem we are given a *bipartite* graph $G = (V, E)$ with interval weight $[\underline{w}_e, \overline{w}_e]$ specified for every edge $e \in E$. We assume that the set of nodes V can be partitioned into two disjoint sets V_1 and V_2 such that $|V_1| = |V_2|$ and if $\{i, j\} \in E$, then $i \in V_1$ and $j \in V_2$. An *assignment* is a *perfect matching* in G that is a subset of edges $B \subseteq E$ in which no two edges share a common node and every node of G is incident to exactly one edge of B. An assignment can also be viewed as a one to one mapping from V_1 to V_2. The solution set Φ consists of all assignments in G and we seek an assignment $B \in \Phi$, which minimizes the maximal regret. A sample problem is shown in Figure 8.1. The optimal robust assignment, shown in bold, has the maximal regret equal to 8.



	v_1	v_2	v_3
u_1	$[1, 3]$	$[2, 4]$	$[3, 5]$
u_2	$-$	$[1, 6]$	$[3, 4]$
u_3	$[2, 8]$	$[1, 7]$	$[6, 8]$

Fig. 8.1. An example of Minmax Regret Minimum Assignment

We will distinguish the case in which G is a *complete* bipartite graph. A bipartite graph $G = (V_1 \cup V_2, E)$ is complete if there is an edge between every pair of nodes $i \in V_1$ and $j \in V_2$. The graph in Figure 8.1 is not complete because there is no edge between u_2 and v_1.

The deterministic Minimum Assignment, is one of the most important problems in combinatorial optimization. If all the weights are precisely known, then the assignment of the minimal total weight can be computed in $\mathcal{O}(|V|^3)$ time by means of a well-known Hungarain algorithm. However, the minmax regret version of this problem is NP-hard. It is a consequence of the fact that Minmax Regret Shortest Path is polynomially transformable to Minmax Regret Minimum Assignment (we show this transformation in Section 8.1). Using the results obtained for Minmax Regret Shortest Path we show in Section 8.2 that the problem of asserting whether a given edge is possibly optimal is strongly

A. Kasperski: Discrete Optimization with Interval Data, STUDFUZZ 228, pp. 113–120, 2008.
springerlink.com

NP-complete. On the other hand, deciding whether an edge is necessarily optimal is the problem whose complexity remains open. In Section 8.3 we construct a MIP model for solving the problem and we show the results of some computational experiments.

8.1 Computational Complexity

In this section we show that Minmax Regret Minimum Assignment is strongly NP-hard. We start by recalling a transformation constructed by Hoffman and Markowitz [61] (its description can also be found in the book by Ahuja *et al.* [2]). This transformation establishes one-to-one correspondence between paths in a given acyclic digraph G and assignments in a bipartite graph G'. Following Aissi *et al.* [4], we will then use this transformation to prove the NP-hardness of Minmax Regret Minimum Assignment.

Consider an acyclic digraph $G = (V, A)$ with two distinguished nodes s and t. Let us construct bipartite graph $G' = (V_1' \cup V_2', E')$ associated to G in the following way:

- For each node $i \in V \setminus \{s, t\}$ we create two nodes $i \in V_1'$ and $i' \in V_2'$. We also add node s to V_1' and t' to V_2'. It holds $|V_1'| = |V_2'| = |V| - 1$.
- For each arc $(i, j) \in A$ we add edge $\{i, j'\}$ to E' and for each node $i \in V \setminus \{s, t\}$ we add edge $\{i, i'\}$ to E'.

A sample transformation is shown in Figure 8.2.

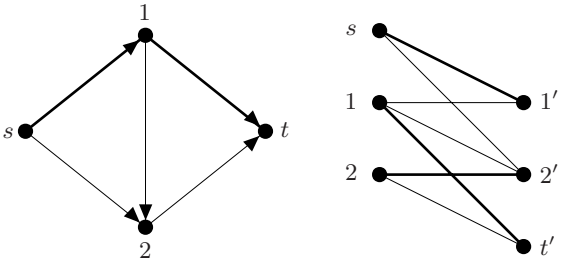


Fig. 8.2. A sample transformation. Path $P = \{(s, 1), (1, t)\}$ in G corresponds to assignment $B = \{\{s, 1'\}, \{1, t'\}, \{2, 2'\}\}$ in G'.

There is one-to-one correspondence between paths from s to t in G and assignments in G'. Let $P = \{(s, i_1), (i_1, i_2), \ldots, (i_k, t)\}$ be a subset of arcs that form a path from s to t in G. The corresponding unique assignment B_P in G' is formed by edges $\{\{s, i_1'\}, \{i_1, i_2'\}, \ldots, \{i_k, t'\}\}$ and by edges $\{i, i'\}$ if node i is not on path P. Conversely, let B be an assignment in G'. Assignment B must contain a subset of edges $\{\{s, i_1'\}, \{i_1, i_2'\}, \ldots, \{i_k, t'\}\}$ such that $P_B = \{(s, i_1'), (i_1, i_2'), \ldots, (i_k, t)\}$ is a unique path from s to t in G. The remaining edges of B are of the form $\{i, i'\}$. We are ready to prove the following result (see also [4]):

Theorem 8.1 ([4]). *The* Minmax Regret Minimum Assignment *problem is strongly NP-hard even if all bounds of the weight intervals are 0 or 1.*

Proof. We show a polynomial transformation from MINMAX REGRET SHORTEST PATH for acyclic digraphs, which is known to by strongly NP-hard. Let an acyclic digraph $G = (V, A)$ with two distinguished nodes s and t, interval weights of arcs $[\underline{w}_{ij}, \overline{w}_{ij}]$,where $\underline{w}_{ij}, \overline{w}_{ij} \in \{0, 1\}$ for all $(i, j) \in A$, be an instance of MINMAX REGRET SHORTEST PATH. This problem is strongly NP-hard (see Theorem 7.2). We construct the corresponding instance of MINMAX REGRET MINIMUM ASSIGNMENT in the following way:

- We construct a bipartite graph $G' = (V_1' \cup V_2', E')$ using the transformation of Hoffman and Markowitz, shown at the beginning of this section.
- Every edge $\{i, j'\} \in E$ that corresponds to arc $(i, j) \in A$ has interval weight $[\underline{w}_{ij}, \overline{w}_{ij}]$. All the remaining edges of G', that is the ones of the form $\{i, i'\}$, have interval weights equal to $[0, 0]$.

It is clear that the reduction can be performed in polynomial time.

It is easy to see that assignment B in G' has the same maximal regret as the corresponding path P_B in G and, conversely, path P in G has the maximal regret equal to the maximal regret of assignment B_P in G'. This follows from the one-to-one correspondence between paths and assignments in G and G' and the fact that all additional edges of the form $\{i, i\}$ in G' have weights equal to 0 under all scenarios. Hence the optimal robust assignment in G' corresponds to the optimal robust path in G and vice versa. Both problems are equivalent and, consequently, MINMAX REGRET MINIMUM ASSIGNMENT is strongly NP-hard. The theorem follows, since all bounds of the uncertainty intervals in the problem are 0 or 1. □

We now show that the problem remains strongly NP-hard for complete bipartite graphs.

Theorem 8.2. *The* MINMAX REGRET MINIMUM ASSIGNMENT *problem is strongly NP-hard for complete bipartite graphs.*

Proof. Suppose we have an instance of MINMAX REGRET MINIMUM ASSIGNMENT in which the input graph $G = (V_1 \cup V_2, E)$, $|V_1| = |V_2| = m$, is not complete and all bounds of the weight intervals are 0 or 1. By Theorem 8.1, computing the optimal robust assignment in this problem is strongly NP-hard. We can transform G into a complete bipartite graph G' in the following way: if $\{i, j\} \notin E$, $i \in V_1$, $j \in V_2$, then we add to G dummy edge $\{i, j\}$ with interval weight $[m + 1, m + 1]$.

It is easily seen that every dummy edge is nonpossibly optimal. Indeed, for every scenario S there is an assignment B in G (and in G') such that $F(B, S) \leq m$. It follows from the fact that $\overline{w}_e \leq 1$ for all $e \in E$ and $|B| = m$. On the other hand, for every assignment B' in G' that contains a dummy edge it holds $F(B', S) \geq m + 1$. In consequence every dummy edge is nonpossibly optimal. Every dummy edge is neither a part of an optimal robust assignment nor the worst case alternative of the optimal robust assignment in G'. We have thus established, that the optimal robust assignments in G and G' are the same. Having a polynomial time algorithm for complete bipartite graphs, we would be able to solve in polynomial time the NP-hard problem for general bipartite graphs. Hence the problem remains NP-hard for complete bipartite graphs. □

From the fact that all assignments in graph $G = (V_1 \cup V_2, E)$ have the same cardinality it follows that MINMAX REGRET MAXIMUM ASSIGNMENT is strongly NP-hard even if all bounds of the weight intervals are 0 or 1 and remains strongly NP-hard for complete bipartite graphs. In order to prove this we can use the transformation shown in the proof of Theorem 1.5.

The deterministic MAXIMUM ASSIGNMENT problem in complete bipartite graphs is a special case of more general MAXIMUM MATCHING, which is stated as follows. Given is an undirected graph $G = (V, E)$ (not necessarily bipartite) with a nonnegative weight w_e specified for every edge $e \in E$. A *matching* in G is a subset of edges such that no two of which share a common node. Set Φ consists of all matchings in G and the problem is to find a matching of the maximal total weight. The deterministic MAXIMUM MATCHING problem is polynomially solvable, for instance by means of an $\mathcal{O}(|V|^3)$ algorithm given by Gabow [54]. The MAXIMUM ASSIGNMENT problem is a special case of MAXIMUM MATCHING in which the input graph is complete bipartite and $|V_1| = |V_2|$. Therefore MINMAX REGRET MAXIMUM ASSIGNMENT is also a special case of MINMAX REGRET MAXIMUM MATCHING and MINMAX REGRET MAXIMUM MATCHING is strongly NP-hard as well.

8.2 Evaluation of Optimality

Consider the POS MINIMUM ASSIGNMENT problem that is the one in which we are given a bipartite graph $G = (V_1 \cup V_2, E)$, $|V_1| = |V_2|$, and a specified edge $f \in E$. We wish to decide whether edge f is possibly optimal, that is whether f belongs to a minimum assignment under some scenario. Using the same technique as in the previous section we will show that POS MINIMUM ASSIGNMENT is strongly NP-complete.

Theorem 8.3. *The* POS MINIMUM ASSIGNMENT *problem is strongly NP-complete.*

Proof. We will construct a polynomial time reduction from the POS SHORTEST PATH problem for acyclic directed graphs, which is known to be strongly NP-complete (see Section 7.3). Let acyclic digraph $G = (V, A)$ with interval arc weights $[\underline{w}_{ij}, \overline{w}_j]$ for $(i, j) \in A$ and a specified arc $f = (p, q) \in A$ be an instance of POS SHORTEST PATH. Recall that we ask whether arc f is possibly optimal. We construct the corresponding instance of POS MINIMUM ASSIGNMENT in the following way:

- We construct graph $G' = (V_1' \cup V_2', E')$ using the transformation shown at the beginning of Section 8.1.
- Every edge $\{i, j'\} \in E$ that corresponds to arc $(i, j) \in A$ has interval weight $[\underline{w}_{ij}, \overline{w}_{ij}]$. All the remaining edges of G' (that is the ones of the form $\{i, i'\}$) have interval weight $[0, 0]$.
- We set $f' = \{p, q'\}$.

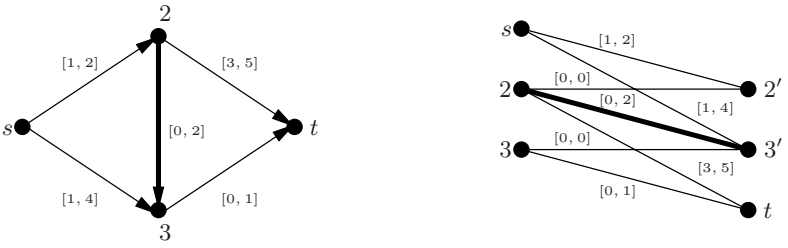


Fig. 8.3. An instance of Poss Shortest Path and the corresponding instance of Poss Minimum Assignment

A sample reduction is shown in Figure 8.3.

It is easy to see that there is one-to-one correspondence between scenarios in G and G'. Every scenario S in G has the corresponding scenario S' in G' such that $w^{S'}_{\{i,j\}} = w^S_{ij}$ for all arcs $(i,j) \in A$ and $w^{S'}_{\{i,j\}} = 0$ for all $(i,j) \notin A$ and vice versa. Moreover $F(P,S) = F(B_P,S')$ and $F(B,S') = F(P_B,S)$. Therefore, if arc (p,q) belongs to the shortest path P under some scenario S, then edge $\{p,q'\}$ belongs to the minimum assignment B_P under scenario S'. Conversely, if edge $\{p,q'\}$ is a part of the minimum assignment B under S', then arc (p,q) belongs the the shortest path P_B under S. In other words, arc (p,q) is possibly optimal if and only if edge $\{p,q'\}$ is possibly optimal.

The Pos Assignment problem is in NP and the proof of this fact goes the same as for Pos Shortest Path (see Theorem 7.8). □

We have thus established that detecting the possibly (and nonpossibly) optimal edges in the problem is computationally intractable. However, the Nec Minimum Assignment problem, in which we wish to decide whether an edge is necessarily optimal is open. As in Minmax Regret Shortest Path, having detected a necessarily optimal edge we would be able to reduce the size of the problem. Therefore, exploring the complexity of Nec Minimum Assignment is an interesting subject of further research.

8.3 Mixed Integer Programming Formulation

In this section we apply the method presented in Section 3.1 to construct a mixed integer programming model for the problem. We will use the fact, that the deterministic Minimum Assignment is a special case of a network flow problem. Let $G = (V_1 \cup V_2, E)$ be a given bipartite graph. Let us convert graph G into directed graph $G' = (V_1 \cup V_2, A)$ by replacing every edge $\{i,j\} \in E$, $i \in V_1$, $j \in V_2$, with arc (i,j). Suppose that V_1 is a set of sources and V_2 is a set of sinks. Every source $i \in V_1$ supplies one unit of flow and every sink $j \in V_2$ requires one unit of flow. The arcs of G' represent available shipping links and the problem is to send the flow from the sources to the sinks. The amount of flow that is sent on every arc is assumed to be an integer or equivalently 0 or 1. A sample problem is shown in Figure 8.4.

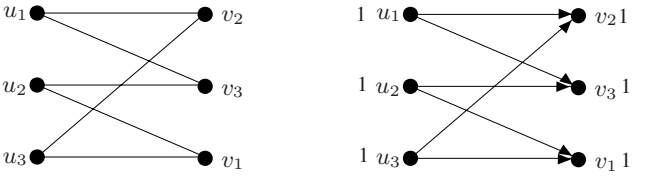


Fig. 8.4. A bipartite graph G and the corresponding network flow problem

Let us define binary variable $x_{ij} \in \{0,1\}$ for every arc $(i,j) \in A$. Consider the following set of constraints:

$$\begin{aligned} &\sum_{\{j:(i,j)\in A\}} x_{ij} = 1 \text{ for } i \in V_1 \\ &\sum_{\{i:(i,j)\in A\}} x_{ij} = 1 \text{ for } j \in V_2 \\ &x_{ij} \in \{0,1\} \qquad \text{for } (i,j) \in A \end{aligned} \tag{8.1}$$

It is evident that every solution to (8.1) represents a solution to the network flow problem. That is, if $x_{ij} = 1$, then one unit of flow is sent from source i to sink j. Having a solution to (8.1) we can construct an assignment B in the original graph G as follows: $\{i,j\} \in B$ if and only if $x_{ij} = 1$. Conversely, if B is an assignment in G, then solution $x_{ij} = 1$ if $\{i,j\} \in B$ and $x_{ij} = 0$ otherwise is a feasible solution to (8.1) and it represents a solution to the network flow problem. Hence we can use constraints (8.1) do describe set $ch(\Phi)$ in Minmax Regret Minimum Assignment.

It is well known that the constraints matrix $\boldsymbol{A}$ of (8.1) is totally unimodular. Therefore we can construct the relaxed subproblem $\min_{\boldsymbol{y}\in ch(\Phi)} \phi(\boldsymbol{x},\boldsymbol{y})$ in the following way:

$$\begin{aligned} &\min \sum_{(i,j)\in A} (\overline{w}_{\{i,j\}} x_{ij} + \underline{w}_{\{i,j\}}(1-x_{ij}))y_{ij} \\ &\sum_{\{j:(i,j)\in A\}} y_{ij} = 1 && \text{for } i \in V_1 \\ &\sum_{\{i:(i,j)\in A\}} y_{ij} = 1 && \text{for } j \in V_2 \\ &y_{ij} \geq 0 && \text{for } (i,j) \in A \end{aligned} \tag{8.2}$$

We have replaced constraints $0 \leq y_{ij} \leq 1$ with $y_{ij} \geq 0$ because $y_{ij} \leq 1$ in every optimal solution to (8.2). Now, the dual to (8.2), that is the problem $\max_{\boldsymbol{\lambda}} \phi^*(\boldsymbol{x},\boldsymbol{\lambda})$, has the following form:

$$\begin{aligned} &\max \sum_{i\in V_1} \alpha_i + \sum_{i\in V_2} \beta_i \\ &\alpha_i + \beta_j \leq \overline{w}_{\{i,j\}} x_{ij} + \underline{w}_{\{i,j\}}(1-x_{ij}) \text{ for } (i,j) \in A \end{aligned} \tag{8.3}$$

From (3.6), (8.1) and (8.3) we get the following MIP formulation for MINMAX REGRET MINIMUM ASSIGNMENT:

$$\begin{aligned}
&\min \sum_{(i,j)\in A} \overline{w}_{\{i,j\}} x_{ij} - \sum_{i\in V_1} \alpha_i - \sum_{i\in V_2} \beta_i \\
&\sum_{\{j:(i,j)\in A\}} x_{ij} = 1 && \text{for } i \in V_1 \\
&\sum_{\{i:(i,j)\in A\}} x_{ij} = 1 && \text{for } j \in V_2 \\
&\alpha_i + \beta_j \le \overline{w}_{\{i,j\}} x_{ij} + \underline{w}_{\{i,j\}}(1 - x_{ij}) && \text{for } (i,j) \in A \\
&x_{ij} \in \{0,1\} && \text{for } (i,j) \in A
\end{aligned} \tag{8.4}$$

8.3.1 Computational Results

In this section we present some computational results using the MIP formulation constructed in the previous section. We compare the MIP formulation to the simple 2-approximation algorithms designed in Section 4.1. In order to solve the MIP model we used a CPLEX 8.0 and a Pentium III 866 MHz computer. We used the family of graphs denoted as $G(|V|, \delta, c)$, where:

- $|V|$ is the number of nodes in set V_1 (and V_2);
- δ is the probability of the existence of an edge between $i \in V_1$ and $j \in V_2$. If $\delta = 1$, then G is a complete bipartite graph;
- for every edge $e \in E$ the value of $\overline{w}_e$ is a randomly selected integer from interval $[0, c]$ and the value of $\underline{w}_e$ is a randomly selected integer from interval $[0, \overline{w}_e]$.

Table 8.1. Computational results

			Algorithm AM		Algorithm AMU	
$\|V\|$	δ	CPU Time	worst %	aver. %	worst %	aver. %
10	1	0.08	6.31	1.35	0.00	0.00
30	1	1.60	20.00	9.36	2.04	0.24
50	1	8.28	30.00	7.31	1.68	0.20
70	1	15.92	42.86	7.45	1.79	0.18
120	1	31.97	22.58	6.80	1.69	0.14
10	0.8	0.08	14.81	14.14	14.20	2.38
30	0.8	1.35	10.71	4.03	3.95	0.36
50	0.8	5.56	14.29	6.05	2.99	0.39
70	0.8	16.80	40.00	11.73	1.56	0.32
120	0.8	21.28	11.67	5.24	0.00	0.00
10	0.5	0.04	9.26	0.94	2.67	0.19
30	0.5	1.40	18.75	6.29	2.17	0.35
50	0.5	9.45	14.29	7.27	2.91	0.83
70	0.5	20.15	16.81	7.34	2.50	0.42
120	0.5	58.32	15.38	8.31	0.85	0.16

We used the following values of parameters: $|V| \in \{10, 30, 50, 70\}$, $\delta \in \{1, 0.8, 0.5\}$ and $c \in \{20, 50, 100, 120\}$. For every combination of the parameters we generated and solved 5 instances. Table 8.1 shows the results for every combination of $|V|$ and δ. In this table the average CPU time required for solving the model and the worst and average percentage deviations from optimum reported for `Algorithm AM` and `Algorithm AMU` are shown. One can observe that the MIP formulation appears to be quite efficient for the tested classes of graphs. The 2-approximation `Algorithm AMU` returns mostly good solutions, which seem to be significantly better than the solutions returned by `Algorithm AM`. The worst reported deviation for `Algorithm AMU` over 225 instances is 14.2% while the average deviation for this algorithm is less than 1%.

8.4 Notes and References

The deterministic Minimum Assignment is one of the basic problems in combinatorial optimization. The first polynomial algorithm for this problem was developed by Kuhn [90]. This algorithm, called the *Hungarian method*, runs in $\mathcal{O}(|V|^3)$ time. Up to now researchers have developed several different algorithms for the problem and the comprehensive review of them can be found in book by Ahuja *et al.* [2] or in a detailed survey by Burkard [30].

It turns out that the Shortest Path problem in directed acyclic graphs can be reduced in polynomial time to Minimum Assignment. This reduction can be found in a paper by Hoffman and Markowitz [61] or in the book by Ahuja *et. al* [2]. Hence, Minimum Assignment has no simpler combinatorial structure than Shortest Path. This fact was used by Aissi *et al.* [4] to prove that Minmax Regret Minimum Assignment is strongly NP-hard. The proof is shown in Section 8.1 and it exploits the fact that Minmax Regret Shortest Path is strongly NP-hard.

The Minmax Regret Minimum Assignment problem was first discussed by Kasperski and Zieliński [74]. In [74] it was proven that the problem is NP-hard. However, a stronger complexity result was obtained by Aissi *et al.* [4], who proved that the problem is strongly NP-hard. The proof in [4] is also simpler than the one in [74]. The MIP formulation for the problem (presented in Section 8.3) was also given by Kasperski and Zieliński [74]. In papers by Kasperski and Zieliński [74] and Kasperski *et al.* [77] one can also find some additional computational tests exploiting the performance of the MIP formulation.

Perhaps the most important question on Minmax Regret Minimum Assignment is about the approximation. We know that the problem is approximable within 2. However, no additional results in this field are known. In particular we do not know whether 2 is the best possible bound. This problem is an important subject of further research. Another interesting and open question is whether it is possible to detect in polynomial time all necessarily optimal edges.

9 Minmax Regret Minimum $s-t$ Cut

This chapter is devoted to the MINMAX REGRET MINIMUM CUT problem. Let $G=(V,E)$ be a given undirected graph with two distinguished nodes s and t called respectively a *source* and a *sink*. Consider a partition of set V into two disjoined subsets V_1 and V_2 such that $s \in V_1$ and $t \in V_2$. An $s-t$ *cut* (*cut* for short) is a subset of edges $C \subseteq E$ that have one endpoint in V_1 and the second in V_2, that is $C=\{\{i,j\} \in E : i \in V_1, j \in V_2\}$. If graph $G=(V,A)$ is directed, then cut C is formed by arcs that start in V_1 and end in V_2, that is $C=\{(i,j) \in A : i \in V_1, j \in V_2\}$. In this case cut C is called a *directed cut*. In other words a cut (a directed cut) is a subset of edges (arcs) whose removal disconnects s and t. Set Φ consists of all cuts (directed cuts) of a given undirected (directed) graph. The weights of edges (arcs) are specified as intervals and we seek a cut that minimizes the maximal regret. We will assume that in the input graph G there is at least one path from node s to node t, otherwise there is no cut in G.

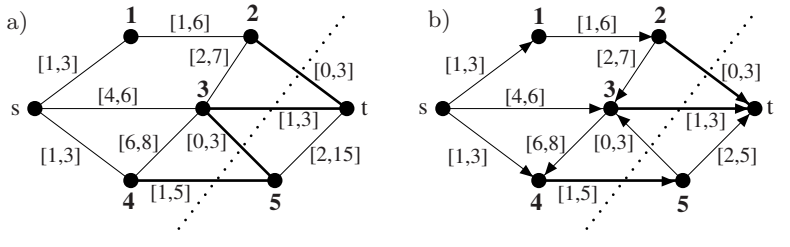


Fig. 9.1. Two examples of MINMAX REGRET MINIMUM CUT. The cuts in bold are the optimal robust cuts.

Two sample problems are shown in Figure 9.1. In Figure 9.1a we are given an undirected graph and cut $C=\{\{2,t\},\{3,t\},\{3,5\},\{4,5\}\}$ is the optimal robust cut with the maximal regret equal to 8. In Figure 9.1b there is a directed graph given and directed cut $C=\{(2,t),(3,t),(4,5)\}$ is the optimal robust cut in this graph with the maximal regret equal to 5.

Using Proposition 1.3 we can see that it is enough to consider only *simple cuts*, that is the cuts which do not contain any other cut. It follows from the fact

A. Kasperski: Discrete Optimization with Interval Data, STUDFUZZ 228, pp. 121–135, 2008.
springerlink.com

that if there are two cuts C and C_1 such that $C_1 \subset C$, then $Z(C_1) \leq Z(C)$ (see Proposition 1.3).

The deterministic Minimum Cut problem is polynomially solvable. The optimal solution, that is the minimum cut, is typically obtained by computing the maximum flow in graph G. However, the minmax regret version of this problem both in directed and undirected graphs is NP-hard. We discuss the complexity of Minmax Regret Minimum Cut in Section 9.1. In Section 9.2 we prove that Pos Minimum Cut, that is the problem of asserting whether an arc is possibly optimal, is also NP-hard. In Section 9.3 we construct a MIP formulation both for directed and undirected graphs and we present the results of some computational experiments. In Section 9.4 we show that the problem can be solved in pseudopolynomial time if the input graph is restricted to be an edge series-parallel digraph. Furthermore, in this case it also admits an FPTAS. Finally, in Section 9.5 we discuss a version of the problem in undirected graphs, which is polynomially solvable.

9.1 Computational Complexity

The first result concerning the complexity of the problem was obtained by Aissi *et al.* 5.

Theorem 9.1 (5). *The* Minmax Regret Minimum Cut *problem is strongly NP-hard for undirected graphs.*

In this section we show that the problem is NP-hard for directed and undirected planar graphs and remains NP-hard even for a very restrictive class of directed graphs. Consider the decision version of the problem for undirected graphs:

Decision Minmax Regret Minimum Cut
Instance: An undirected graph $G = (V, E)$, $s \in V$, $t \in V$, with interval weights of edges $[\underline{w}_e, \overline{w}_e]$, $e \in E$, integer $K \geq 0$.
Question: Is there a cut C in G such that $Z(C) \leq K$?

Theorem 9.2. *The* Decision Minmax Regret Minimum Cut *problem is NP-complete even if the input graph is planar.*

Proof. We construct a polynomial reduction from the Partition problem. Recall that this problem is known to be NP-complete [56] and is defined as follows:

Partition
Instance: A collection $\mathfrak{C} = (c_1, \ldots, c_n)$ of positive integers such that $\sum_{i=1}^{n} c_i = 2b$, where $b > 0$.
Question: Is there a subset $I \subset \{1, \ldots, n\}$ such that $\sum_{i \in I} c_i = b$?

Let $\mathfrak{C} = (c_1, \ldots, c_n)$ be a given instance of Partition. We construct graph $G = (V, E)$ in the following way:

- the set of nodes of G is $V = \{s, t, w, u_1, \ldots, u_n\}$;

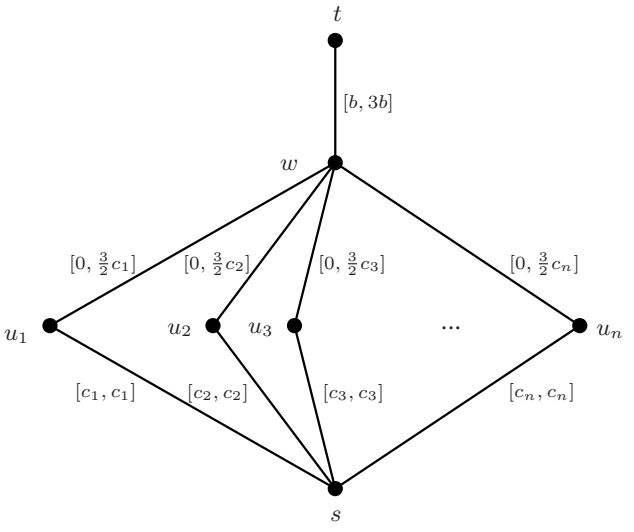


Fig. 9.2. Graph G for a given collection $\mathfrak{C}$

- for every element c_i of collection $\mathfrak{C}$ we create two edges: $\{s, u_i\}$ with interval weight $[c_i, c_i]$ and $\{u_i, w\}$ with interval weight $[0, \frac{3}{2}c_i]$;
- we also add edge $\{w, t\}$ with interval weight $[b, 3b]$.

We complete the construction by setting $K = \frac{3}{2}b$. The construction of G is shown in Figure 9.2.

We will show that the answer to PARTITION is Yes if and only if there is a cut C in G such that $Z(C) \leq \frac{3}{2}b$. Observe, that no robust optimal cut in G uses edge $\{w, t\}$. Indeed, $C_1 = \{\{w, t\}\}$ is a cut in G with the maximal regret $Z(C_1) = 3b$ while cut $C_2 = \{\{s, u_1\}, \{s, u_2\}, \ldots, \{s, u_n\}\}$ has the maximal regret $Z(C_2) = 2b$, where $b > 0$. Since C_1 is a cut in G, every cut that contains C_1 has the maximal regret not less than C_1 (see Proposition 1.3). Hence edge $\{w, t\}$ does not belong to any optimal robust cut in G.

Let V_1 and V_2 be a partition of the nodes of G that corresponds to a cut C that does not contain edge $\{w, t\}$. It is clear that $s \in V_1$ and $t, w \in V_2$ (if $w \in V_1$, then edge $\{w, t\}$ belongs to C). If node $u_i \in V_1$ then $\{u_i, w\} \in C$ and $\{s, u_i\} \notin C$, otherwise if $u_i \in V_2$, then $\{s, u_i\} \in C$ and $\{u_i, w\} \notin C$. Denote by I_1 the subset of indices $i \in \{1, \ldots, n\}$ such that $\{s, u_i\} \in C$ and $I_2 = \{1, \ldots, n\} \setminus I_1$. Clearly, subsets I_1 and I_2 define a partition of collection $\mathfrak{C}$. It holds

$$F(C, S_C^+) = \sum_{i \in I_1} \overline{w}_{\{s,u_i\}} + \sum_{i \in I_2} \overline{w}_{\{u_i,w\}} = \sum_{i \in I_1} c_i + \frac{3}{2} \sum_{i \in I_2} c_i.$$

From the fact that $\sum_{i \in I_1} c_i + \sum_{i \in I_2} c_i = 2b$ we obtain

$$F(C, S_C^+) = 2b + \frac{1}{2} \sum_{i \in I_2} c_i. \tag{9.1}$$

It is easy to see that the worst case alternative C^* for C, that is the minimum cut under scenario S_C^+, is either formed by the single edge $\{w,t\}$ or it is formed in the following way: if $\{s,u_i\} \in C$ then $\{u_i,w\} \in C^*$, if $\{u_i,w\} \in C$ then $\{s,u_i\} \in C^*$ for $i=1,\ldots,n$. Thus, we obtain

$$F(C^*,S_C^+) = \min\left\{b, \sum_{i\in I_1} \underline{w}_{\{u_i,w\}} + \sum_{i\in I_2} \underline{w}_{\{s,u_i\}}\right\} = \min\left\{b, \sum_{i\in I_2} c_i\right\}. \quad (9.2)$$

Since $Z(C) = F(C,S_C^+) - F(C^*,S_C^+)$ and from (9.1) and (9.2) we get

$$Z(C) = \max\left\{b + \frac{1}{2}\sum_{i\in I_2} c_i, 2b - \frac{1}{2}\sum_{i\in I_2} c_i\right\}. \quad (9.3)$$

Equation (9.3) implies that $Z(C) \leq \frac{3}{2}b$ if and only if $\sum_{i\in I_2} c_i = b$. In other words $Z(C) \leq \frac{3}{2}b$ if and only if the answer to Partition is Yes.

It is easily seen that Decision Minmax Regret Minimum Cut is in NP. A nondeterministic Turing machine first guesses cut C. It can be then verified in polynomial time whether $Z(C) \leq K$.

From Theorem 9.2 we immediately get the following result:

Theorem 9.3. *The* Minmax Regret Minimum Cut *problem is NP-hard for undirected planar graphs.*

Consider now the case in which the input graph is directed and we seek a directed cut, which minimizes the maximal regret. The decision version of this problem is exactly the same as for the undirected case (we only allow the input graph to be directed). The proof that this problem is NP-complete is very similar to the proof of Theorem 9.2. In the construction of graph G we replace edge $\{s,u_i\}$ with arc (s,u_i), edge $\{u_i,w\}$ with arc (u_i,w) for all $i=1,\ldots,n$ and edge $\{w,t\}$ with arc (w,t). The proof goes then without any modifications. Observe that the resulting digraph G is planar and acyclic. Moreover, it has a series parallel topology. Hence we have obtained the following result:

Theorem 9.4. *The* Minmax Regret Minimum Cut *problem is NP-hard for edge series-parallel digraphs.*

Observe also that after the modifications specified above the resulting digraph is layered. We thus get the following result:

Theorem 9.5. *The* Minmax Regret Minimum Cut *problem is NP-hard for layered digraphs.*

9.2 Evaluation of Optimality

In this section we discuss the problem of evaluating the optimality of elements in the problem. We show that Pos Minimum Cut, that is the problem of asserting

whether an arc is possibly optimal is NP-complete, even when the input graph is acyclic and planar. On the other hand, the NEC MINIMUM CUT problem, that is the one in which we wish to assert the necessary optimality of an arc or edge, remains open. It is an interesting and important subject of further research.

In order to prove the NP-completeness of POS MINIMUM CUT we use the fact that problem POS SHORTEST PATH is NP-complete for planar digraphs. The proof of this fact has been shown in Section 7.4. Recall that a digraph $G = (V, A)$ is planar if it can be embedded in a plane without crossing arcs. An embedding of a planar digraph partitions the plane into separate regions called *faces*. Exactly one such face is unbounded and it is called an *outer face*. A *right face* of an arc $a = (i, j) \in A$ is the face, which is on the right hand side of (i, j) when traversing this arc from i to j. Similarly we define the *left face* of a.

Suppose that $G = (V, A)$ is a directed planar graph with two distinguished nodes s and t. We assume, without loss of generality, that no arc enters s and no arc leaves t. Such arcs can be removed from G because they cannot be a part of any cut in G. We also assume that G is given as a plane representation in which s and t touch the outer face. The digraph G is associated with a digraph $G^* = (V^*, A^*)$ called a *dual digraph*. The dual digraph is constructed as follows:

- we first modify G by introducing an artificial arc (t, s) in the outer face of G, so that this arc is directed clockwise when looking from the inside of G;
- the nodes of G^* are the faces of the modified digraph G;
- for every arc $a \in A$, except for the artificial one, we add to A^* arc a^* that intersects a and joins the nodes in the faces on either side of it, arc $a^* = (i^*, j^*)$ is oriented in such a way that node i^* is in the right face of $a = (i, j)$;
- finally $s^* \in V^*$ is the node corresponding to the face limited by the artificial arc (t, s) in G and $t^* \in V^*$ is the node that corresponds to the outer face of G.

It is easy to verify that G^* is also a planar digraph and the dual of G^* is G. An example of a planar digraph and its dual are shown in Figure 9.3.

Let $V_1 \cup V_2$ be a partition of V such that $s \in V_1$ and $t \in V_2$. A directed cut C that corresponds to this partition is said to be *uniformly directed $s - t$ cut* if no arcs in G lead from V_2 to V_1. The following theorem characterizes a planar digraph and its dual:

Theorem 9.6 ([1], [92]). *Let G be a planar acyclic digraph and let G^* be its dual. Then $P = \{a_1, a_2, \ldots, a_k\}$ is a simple path from s to t in G if and only if $C = \{a_1^*, a_2^*, \ldots, a_k^*\}$ is an uniformly directed $s^* - t^*$ cut in G^**

Hence there is one-to-one correspondence between simple paths in a directed planar graph and uniformly directed cuts in its dual. We will use this fact to prove the following result:

Theorem 9.7. *The* POS MINIMUM CUT *problem is NP-complete for planar digraphs.*

Proof. We will construct a polynomial time reduction from POS SHORTEST PATH for acyclic planar digraphs, which is known to be NP-complete (see Theorem 7.8).

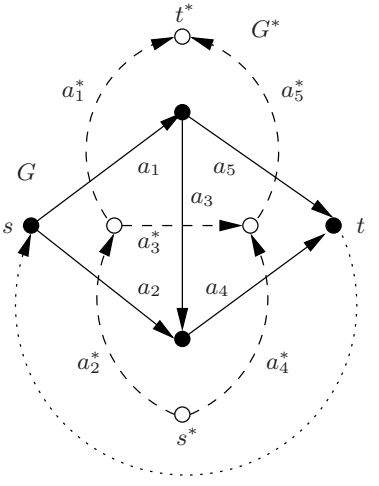


Fig. 9.3. Construction of a dual graph G^* (dashed arcs)

Let an acyclic planar digraph $G = (V, A)$, $s \in V$, $t \in V$, with interval arc weights $[\underline{w}_a, \overline{w}_a]$, $a \in A$, and a specified arc $f = (k, l) \in A$ be an instance of Pos Shortest Path. Recall that in this problem we ask whether arc f belongs to a shortest path from s to t under some scenario S. We assume that there is at least one path from s to t in G that uses arc f, otherwise we see at once that f is nonpossibly optimal.

We construct the corresponding instance of Pos Minimum Cut as follows. We first construct a dual digraph $G^* = (V^*, A^*)$ for G. The interval weight of arc $a^* \in A^*$ is the same as the interval weight of the corresponding arc $a \in A$. Then for every arc $a^* = (i^*, j^*) \in A^*$ we create the reverse arc $b^* = (j^*, i^*)$ with interval weight $[M, M]$. This arc is called a *dummy arc*. We fix $M = |A|w_{max} + 1$, where $w_{max} = \max_{a \in A} \overline{w}_a$. We denote the resulting digraph with dummy arcs as G^{**}. Finally, we distinguish arc f^* in G^{**}. A sample reduction is shown in Figure 9.4.

We claim that arc f^* belongs to a minimum cut in G^{**} under some scenario (that is f^* is possibly optimal) if and only if f belongs to a shortest path in G under some scenario. Observe first that every uniformly directed cut C in G^* is also a cut in G^{**}. It follows from the fact that adding dummy arcs to G^* only backward arcs leading from V_2 to V_1 are created, where $V_1 \cup V_2$ is a partition of nodes that corresponds to cut C.

Consider a scenario S in G^{**}. A minimum cut under S cannot use any dummy arc. Indeed, there is at least one cut in G^{**} that does not use any dummy arc. To see this consider a path from s to t in G. This path corresponds to a uniformly directed cut C in G^* and obviously C is also the cut in G^{**}. Every cut C_1 in G^{**} that uses a dummy arc has the weight of at least $|A|w_{max} + 1$, which is strictly greater than the weight of C under S. Consequently, C_1 cannot be a minimum cut under S. Now from Theorem 9.6 we get that path $P = \{a_1, a_2, \ldots, a_k\}$ is the shortest path in G under scenario S if and only if cut $C = \{a_1^*, a_2^*, \ldots, a_k^*\}$ is the minimum cut under scenario S', such that $w_a^S = w_{a^*}^{S'}$ for all arcs $a \in A$.

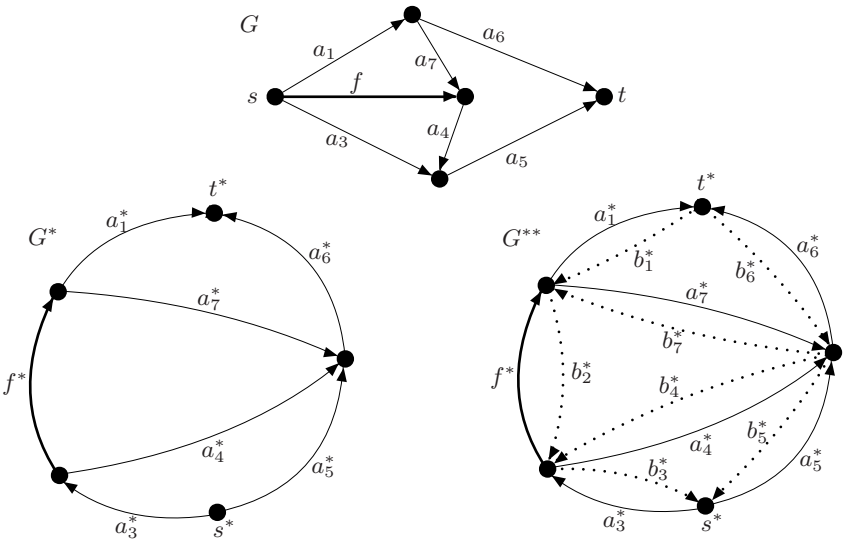


Fig. 9.4. A sample reduction. Graph G, its dual G^* and the modified dual G^{**}. Arcs a_i^* have the same weight intervals as a_i and arcs b_i^* have interval weights $[M, M]$.

Moreover $f \in P$ if and only if $f^* \in C$. Hence f^* is possibly optimal if and only if f belongs to a shortest path under some scenario S in G. It is easily seen that digraph G^{**} is planar and it can be constructed from G in polynomial time.

The Pos Minimum Cut problem is in NP and the reasoning is the same as for Pos Shortest Path in Theorem 7.8. □

We have shown that Pos Minimum Cut is computationally intractable if the input graph is directed. The complexity of the problem remains open if the input graph is undirected. Similarly to the problems considered in the previous chapters, also the Nec Minimum Cut problem is open.

9.3 Mixed Integer Programming Formulation

In this section we propose the mixed integer programming formulations for Min-max Regret Minimum Cut in both directed and undirected graphs.

9.3.1 MIP Formulation for Directed Graphs

Consider a directed graph $G = (V, A)$ with two distinguished nodes s and t. Define a binary variable $x_{ij} \in \{0, 1\}$ for every arc $(i, j) \in A$ and consider the following set of constraints:

$$\begin{aligned} &\pi_i - \pi_j + x_{ij} \geq 0 \text{ for } (i,j) \in A \\ &\pi_t - \pi_s \geq 1 \\ &x_{ij} \in \{0,1\} \qquad \text{for } (i,j) \in A \end{aligned} \tag{9.4}$$

where π_i, $i \in V$, are unrestricted auxiliary variables. Let $V = V_1 \cup V_2$ be a partition of nodes such that $s \in V_1$ and $t \in V_2$. This partition corresponds to a directed cut C. Consider solution $x_{ij} = 1$ if $(i,j) \in C$ and $x_{ij} = 0$ if $(i,j) \notin C$; $\pi_i = 0$ if $i \in V_1$ and $\pi_i = 1$ if $i \in V_2$. It is easy to see that this solution fulfills constraints (9.4). Hence if $\boldsymbol{x}$ is a characteristic vector of a cut, then $\boldsymbol{x}$ fulfills (9.4).

Consider a feasible solution $\boldsymbol{x}$ to (9.4), which is the characteristic vector of a subset of arcs $Y \in A$. We will show that the subset Y must contain a directed cut. Let P be a path from s to t in G. For at least one of the arcs of P, say (k,l), it must hold $x_{kl} = 1$ (or equivalently $(k,l) \in Y$). Indeed if $x_{ij} = 0$ for all arcs $(i,j) \in P$, then $\pi_i \geq \pi_j$ for all $(i,j) \in P$ and $\pi_s \geq \pi_t$, which contradicts the fact that $\boldsymbol{x}$ is feasible to (9.4). We add arc (k,l) to cut C and remove (k,l) from G. We repeat this procedure until there is no path from s to t in G. Let us remove from G all arcs that belong to C and denote the resulting digraph by G'. Let V_1 be the subset of nodes that are reachable from node s in G' and let $V_2 = V \setminus V_1$. It is evident that $s \in V_1$ and $t \in V_2$. Moreover, there is no arc (i,j) in G' that starts in V_1 and ends in V_2 (otherwise node j would be reachable from s and $j \in V_1$). Let us now choose $C' \subseteq C$ composed of all arcs that start in V_1 and end in V_2. At least one such arc must exist since we have assumed that there is a path from s to t in G. The subset $C' \subseteq C \subseteq Y$ forms a directed cut in G.

We have thus established that constraints (9.4) can be used to define set $ch(\Phi)$ in the problem (see Section 3.1). The constraints matrix $\boldsymbol{A}$ of (9.4) is totally unimodular (see [112]). We can construct the relaxed subproblem $\min_{\boldsymbol{y} \in ch(\Phi)} \phi(\boldsymbol{x}, \boldsymbol{y})$ in the following way:

$$\begin{array}{ll} \min \sum_{(i,j)\in A} (\overline{w}_{ij} x_{ij} + \underline{w}_{ij}(1 - x_{ij}))y_{ij} & \\ \pi_i - \pi_j + y_{ij} \geq 0 & \text{for } (i,j) \in A \\ \pi_t - \pi_s \geq 1 & \\ y_{ij} \geq 0 & \text{for } (i,j) \in A \end{array} \tag{9.5}$$

We have replaced $y_{ij} \in \{0,1\}$ with $y_{ij} \geq 0$ because $y_{ij} \leq 1$ in an optimal solution to (9.5). Taking the dual to (9.5) we get the problem $\max_{\boldsymbol{\lambda}} \phi^*(\boldsymbol{x}, \boldsymbol{\lambda})$:

$$\begin{array}{l} \max f \\ \sum\limits_{\{j:(s,j)\in A\}} f_{sj} - \sum\limits_{\{i:(i,s)\in A\}} f_{is} = f \\ \sum\limits_{\{j:(t,j)\in A\}} f_{tj} - \sum\limits_{\{i:(i,t)\in A\}} f_{it} = -f \\ \sum\limits_{\{j:(k,j)\in A\}} f_{kj} - \sum\limits_{\{i:(i,k)\in A\}} f_{ik} = 0 \quad \text{for } k \in V \setminus \{s,t\} \\ 0 \leq f_{ij} \leq \overline{w}_{ij} x_{ij} + \underline{w}_{ij}(1 - x_{ij}) \quad \text{for } (i,j) \in A \end{array} \tag{9.6}$$

Of course, for fixed x_{ij}, formulation (9.6) is a network flow model in which we wish to find the *maximal flow* f from s to t in G and the arc capacities are

given as $\overline{w}_{ij}x_{ij} + \underline{w}_{ij}(1 - x_{ij})$ for all $(i,j) \in A$. The variable f_{ij} denotes the flow on arc $(i,j) \in A$. Now, from (3.6), (9.4) and (9.6) we get the following MIP formulation for Minmax Regret Minimum Cut in a directed graph:

$$\begin{aligned}
&\min \sum_{(i,j)\in A} \overline{w}_{ij}x_{ij} - f \\
&\pi_i - \pi_j + x_{ij} \geq 0 && \text{for } (i,j) \in A \\
&\pi_t - \pi_s \geq 1 \\
&\sum_{\{j:(s,j)\in A\}} f_{sj} - \sum_{\{i:(i,s)\in A\}} f_{is} = f \\
&\sum_{\{j:(t,j)\in A\}} f_{tj} - \sum_{\{i:(i,t)\in A\}} f_{it} = -f \\
&\sum_{\{j:(k,j)\in A\}} f_{kj} - \sum_{\{i:(i,k)\in A\}} f_{ik} = 0 && \text{for } k \in V \setminus \{s,t\} \\
&0 \leq f_{ij} \leq \overline{w}_{ij}x_{ij} + \underline{w}_{ij}(1 - x_{ij}) && \text{for } (i,j) \in A \\
&x_{ij} \in \{0,1\} && \text{for } (i,j) \in A
\end{aligned} \tag{9.7}$$

In the next section we extend the MIP formulation for the case when the input graph is undirected.

9.3.2 MIP Formulation for Undirected Graphs

Let us now construct a MIP model for Minmax Regret Minimum Cut if the input graph $G = (V,E)$ is undirected. Let us define the binary variable $x_{\{i,j\}} \in \{0,1\}$ for every edge $\{i,j\} \in E$. Let us transform graph G into digraph $G' = (V', A')$ by replacing every edge $\{i,j\} \in E$ with two arcs (i,j) and (j,i). Associate binary variable x'_{ij} with every arc $(i,j) \in A'$. Let C be a cut in G. The partition of nodes of G that corresponds to C is $V_1 \cup V_2$, where $s \in V_1$ and $t \in V_2$. Then $C' = \{(i,j) \in A' : \{i,j\} \in C, i \in V_1, j \in V_2\}$ is a directed cut in G'. Conversely, if C' is a directed cut in G', then $C = \{\{i,j\} : (i,j) \in A'\}$ is a cut in G. Hence, there is one-to-one correspondence between cuts in G and G'. Moreover, if arc (i,j) belongs to cut C' in G', then the reverse arc (j,i) does not belong to C'. Thus we can set $x_{\{i,j\}} = x'_{ij} + x'_{ji}$. Set $ch(\Phi)$ can be then described by the following constraints:

$$\begin{aligned}
&\pi_i - \pi_j + x'_{ij} \geq 0 \text{ for } (i,j) \in A' \\
&\pi_t - \pi_s \geq 1 \\
&x_{\{i,j\}} = x'_{ij} + x'_{ji} \text{ for } \{i,j\} \in E \\
&x'_{ij} \in \{0,1\} \qquad \text{for } (i,j) \in A' \\
&x_{\{i,j\}} \in \{0,1\} \qquad \text{for } \{i,j\} \in E
\end{aligned} \tag{9.8}$$

We can now construct the relaxed subproblem $\min_{\boldsymbol{y}\in ch(\Phi)} \phi(\boldsymbol{x}, \boldsymbol{y})$ in the following way:

$$
\begin{aligned}
&\min \sum_{\{i,j\}\in E} (\overline{w}_{\{i,j\}} x_{\{i,j\}} + \underline{w}_{\{i,j\}}(1 - x_{\{i,j\}}))(y'_{ij} + y'_{ji}) \\
&\pi_i - \pi_j + y'_{ij} \geq 0 \text{ for } (i,j) \in A' \\
&\pi_t - \pi_s \geq 1 \\
&y'_{ij} \geq 0 \qquad\qquad \text{for } (i,j) \in A'
\end{aligned} \tag{9.9}
$$

W have substituted $y_{\{i,j\}} = y'_{ij} + y'_{ji}$ in the objective of (9.9). We have also replaced constraints $y_{ij} \in \{0,1\}$ with $y_{ij} \geq 0$ and the reasoning is the same as for the directed case. Now taking the dual to (9.9) we get the problem $\max_{\boldsymbol{\lambda}} \phi^*(\boldsymbol{x}, \boldsymbol{\lambda})$, which has the following form:

$$
\begin{aligned}
&\max f \\
&\sum_{\{j:(s,j)\in A'\}} f_{sj} - \sum_{\{i:(i,s)\in A'\}} f_{is} = f \\
&\sum_{\{j:(t,j)\in A'\}} f_{tj} - \sum_{\{i:(i,t)\in A'\}} f_{it} = -f \\
&\sum_{\{j:(k,j)\in A'\}} f_{kj} - \sum_{\{i:(i,k)\in A'\}} f_{ik} = 0 \qquad \text{for } k \in V \setminus \{s,t\} \\
&0 \leq f_{ij} \leq \overline{w}_{\{i,j\}} x_{\{i,j\}} + \underline{w}_{\{i,j\}}(1 - x_{\{i,j\}}) \text{ for } \{i,j\} \in E \\
&0 \leq f_{ji} \leq \overline{w}_{\{i,j\}} x_{\{i,j\}} + \underline{w}_{\{i,j\}}(1 - x_{\{i,j\}}) \text{ for } \{i,j\} \in E
\end{aligned} \tag{9.10}
$$

Now, from (3.6), (9.8) and (9.10) we get the following MIP formulation for Minmax Regret Minimum Cut for a graph $G = (V, E)$:

$$
\begin{aligned}
&\min \sum_{\{i,j\}\in E} \overline{w}_{\{i,j\}} x_{\{i,j\}} - f \\
&\pi_i - \pi_j + x'_{ij} \geq 0 && \text{for } (i,j) \in A' \\
&\pi_t - \pi_s \geq 1 \\
&x_{\{i,j\}} = x'_{ij} + x'_{ji} && \text{for } \{i,j\} \in E \\
&\sum_{\{j:(s,j)\in A'\}} f_{sj} - \sum_{\{i:(i,s)\in A'\}} f_{is} = f \\
&\sum_{\{j:(t,j)\in A'\}} f_{tj} - \sum_{\{i:(i,t)\in A'\}} f_{it} = -f \\
&\sum_{\{j:(k,j)\in A'\}} f_{kj} - \sum_{\{i:(i,k)\in A'\}} f_{ik} = 0 && \text{for } k \in V \setminus \{s,t\} \\
&0 \leq f_{ij} \leq \overline{w}_{\{i,j\}} x_{\{i,j\}} + \underline{w}_{\{i,j\}}(1 - x_{\{i,j\}}) && \text{for } \{i,j\} \in E \\
&0 \leq f_{ji} \leq \overline{w}_{\{i,j\}} x_{\{i,j\}} + \underline{w}_{\{i,j\}}(1 - x_{\{i,j\}}) && \text{for } \{i,j\} \in E \\
&x'_{ij} \in \{0,1\} && \text{for } (i,j) \in A' \\
&x_{\{i,j\}} \in \{0,1\} && \text{for } \{i,j\} \in E
\end{aligned} \tag{9.11}
$$

In the next section we show some computational tests using the constructed MIP formulations.

9.3.3 Computational Results

In order to evaluate the performance of the MIP formulation we constructed a family of graphs for which the number of cuts is large and all of them have similar cardinalities. The random graphs have appeared to be inappropiate for our purposes. For random graphs, like those used in Section 7.5.3, CPLEX returns an optimal robust cut within a few seconds. In order to perform the tests for hard instances, we used the family of directed layered graphs denoted as $L(|V|, w, c)$, where:

- the number of nodes is $|V| + 2$; $|V|$ nodes form layers of G and the two additional nodes are the source s and the sink t;
- w stands for the width of the graph;
- all arcs starting from s and all arcs terminating at t have interval weights $[M, M]$, where M is a sufficiently large number so that none of these arcs belongs to the optimal robust cut; for all remaining arcs $a \in A$ the value of $\overline{w}_a$ is a randomly selected integer from interval $[0, c]$ and the value of $\underline{w}_a$ is a randomly selected integer from interval $[0, \overline{w}_a]$.

A graph $L(|V|, w, c)$ is shown in Figure 9.5.

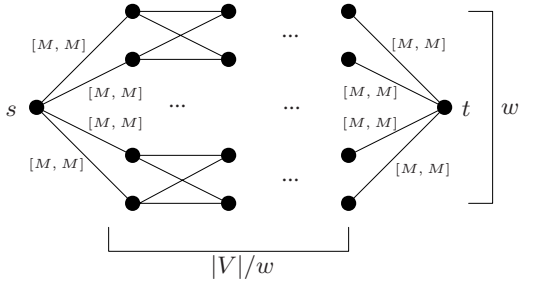


Fig. 9.5. A graph $L(|V|, w, c)$

We chose $|V| \in \{20, 40, 60, 80\}$, $w \in \{5, 10, 20\}$ and $c \in \{20, 50, 100\}$. For every assumed combination of $|V|$ and w we generated and solved five instances for every $c \in \{20, 50, 100\}$. The problems were solved by means of CPLEX 8.0 using a Pentium III 866 MHz computer. The results are presented in Table 9.1. In this table the average CPU times in seconds required to solve the model are shown. We also investigated the quality of the solutions returned by 2-approximation algorithms `Algorithm AM` and `Algorithm AMU`. The worst and average percentage deviations from optimum reported for these algorithms are also shown in Table 9.1. As for the problems discussed in the previous chapters, both approximation algorithms returned reasonably good solutions. The worst reported deviation from optimum was 18.16%. The average deviation from optimum of

Table 9.1. Computational results

			Algorithm AM		Algorithm AMU	
$\|V\|$	w	CPU Time	worst %	aver. %	worst %	aver. %
20	5	0.535	16.90	2.54	16.90	2.04
40	5	3.01	12.02	3.13	12.02	1.03
40	10	57.35	18.16	5.75	18.16	4.36
60	5	5.600	13.68	4.45	6.21	0.67
60	10	696.88	12.11	1.64	10.48	0.93
80	5	5.770	14.34	2.58	3.25	0.22
80	10	408.096	9.51	3.55	3.38	0.59
80	20	246.85	3.48	1.38	3.35	0.38

`Algorithm AMU`, which performed better than `Algorithm AM`, was about 1%. It is interesting, that the performance of both approximation algorithms was better for larger graphs.

9.4 Series-Parallel Multidigraphs

In Section 7.1 we introduced a special class of directed graphs, namely edge series-parallel multidigraphs (ESP). In Section 9.1 we have shown that Minmax Regret Minimum Cut remains NP-hard for this class of graphs (see Theorem 9.4). In this section we show that the Minmax Regret Minimum Cut for ESP can be solved in pseudpolynomial time and admits an FPTAS. In fact, we show that Minmax Regret Minimum Cut can be transformed to Minmax Regret Shortest Path and the algorithms used to solve this problem can be applied to Minmax Regret Minimum Cut as well.

Recall that in order to compute an optimal robust cut it is enough to consider only simple cuts in G. Denote by Φ_G the set of all simple cuts in a given ESP G. The following proposition characterizes the set Φ_G:

Proposition 9.8. *Let $G=(V,A)$ be a given ESP with source s and sink t.*

1. *If G consists of a single arc (s,t) then $\Phi_G=\{(s,t)\}$.*
2. *If $G=\mathtt{s}(G_1,G_2)$ then $\Phi_G=\Phi_{G_1}\cup\Phi_{G_2}$.*
3. *If $G=\mathtt{p}(G_1,G_2)$ then $\Phi_G=\{C_1\cup C_2:\ C_1\in\Phi_{G_1},\ C_2\in\Phi_{G_2}\}$.*

Proof. A direct consequence of the recursive definition of ESP. □

Recall that we can associate a binary decomposition tree $\mathcal{T}(G)$ with every ESP G. The internal nodes of this tree are marked with `s` or `p` and denote the series or parallel compositions in G. Let us interchange labels `s` and `p` of each node in $\mathcal{T}(G)$. As a result we obtain a new binary decomposition tree, which is associated with an ESP G' with source s' and sink t'. The graph G' is called a *dual* of G. It is easily seen that the dual graph G' is also ESP and the dual of G' is G. Moreover, graphs G and G' have the same set of arcs. An example of ESP G and its dual G' are shown in Figure 9.6.

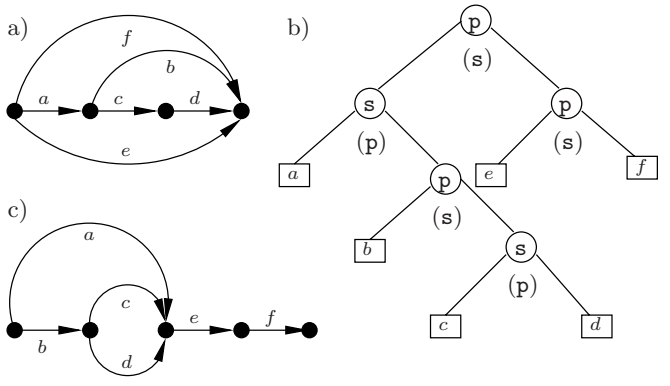


Fig. 9.6. a) An ESP G, b) the binary decomposition tree $\mathcal{T}(G)$ c) G' – the dual graph of G. The node labels of $\mathcal{T}(G')$ are shown in parentheses

Theorem 9.9. *Let $G = (V, A)$ be a given ESP with source s and sink t. A set of arcs $\{a_1, a_2, \ldots, a_k\}$ is a simple $s-t$ cut in G if and only if it is a path from s' to t' in the dual ESP G'.*

Proof. Let us denote by Φ_G the set of all simple $s-t$ cuts in G and by $\Phi'_{G'}$ the set of all paths from s' to t' in G'. We need to show that $\Phi_G = \Phi'_{G'}$.

We will prove the theorem by induction on the number of arcs in G. Theorem is trivial if G consists of a single arc. Now assume that it is true for all ESP in which the number of arcs is less than or equal to m, where $m \geq 1$. Consider an ESP $G = (V, A)$ with $m+1$ arcs. Since $m+1 \geq 2$, it follows that G is the result of a series or a parallel composition of two ESP G_1 and G_2. Moreover, the number of arcs in G_1 and G_2 is less than $m+1$ and thus, by the induction hypothesis, the theorem is true for G_1 and G_2.

Assume that $G = \mathtt{s}(G_1, G_2)$. Denote by G'_1 and G'_2 the dual graphs for G_1 and G_2 respectively. By definition of the dual graph $G' = \mathtt{p}(G'_1, G'_2)$ is the dual of G. From the assumption $\Phi_{G_1} = \Phi'_{G'_1}$ and $\Phi_{G_2} = \Phi'_{G'_2}$. Proposition 9.8 and Proposition 7.12 yield $\Phi_G = \Phi_{G_1} \cup \Phi_{G_2} = \Phi'_{G'_1} \cup \Phi'_{G'_2} = \Phi'_{G'}$.

Assume that $G = \mathtt{p}(G_1, G_2)$. Denote by G'_1 and G'_2 the dual graphs for G_1 and G_2 respectively. By definition of the dual graph $G' = \mathtt{s}(G'_1, G'_2)$ is the dual of G. From the assumption $\Phi_{G_1} = \Phi'_{G'_1}$ and $\Phi_{G_2} = \Phi'_{G'_2}$. Proposition 9.8 and Proposition 7.2 yield $\Phi_G = \{C_1 \cup C_2 : \ C_1 \in \Phi_{G_1}, C_2 \in \Phi_{G_2}\} = \{P_1 \cup P_2 : \ P_1 \in \Phi'_{G'_1}, P_2 \in \Phi'_{G'_2}\} = \Phi'_{G'}$. □

Theorem 9.9 immediately implies that we can apply all the results obtained for Minmax Regret Shortest Path for ESP to Minmax Regret Minimum Cut. It follows from the fact that the optimal robust cut in ESP G is the optimal robust path in its dual G'. The dual graph can be constructed in $\mathcal{O}(|A|)$ time. We have thus established the following result:

Theorem 9.10. *The* Minmax Regret Minimum Cut *problem for ESP admits an FPTAS, that is an algorithm that for a given ESP $G=(V,A)$ computes cut C such that $Z_G(C)\leq(1+\epsilon)OPT$ in time $\mathcal{O}(|A|^3/\epsilon^2)$.*

The FPTAS for the problem is precisely the same as for Minmax Regret Shortest Path. It must only be applied to the dual graph G'.

9.5 Polynomially Solvable Version of the Problem

Following Aissi *et al.* [5] we now present an interesting result on a certain modification of Minmax Regret Minimum Cut. We are again given an undirected graph $G=(V,E)$ with interval weights of edges. Contrary to the main problem considered in this chapter, we do not distinguish nodes s and t in graph G. We consider instead any subset of nodes W such that $W\neq\emptyset$ and $W\neq V$. The subset W defines a *global cut* in G composed of all edges that have one endpoint in W and the second in $V\setminus W$. Now a global cut can be seen as a subset of edges whose removal separates two subsets of nodes of G. Let Φ denote the set of all global cuts in G, that is Φ contains global cuts for all possible subsets W. In the Minmax Regret Minimum Global Cut problem we seek a global cut in Φ that minimizes the maximal regret.

We now show that Minmax Regret Minimum Global Cut is polynomially solvable. Consider first the deterministic Minimum Global Cut problem, in which all edges have precise weights w_e, $e\in E$. Define by $F(C)$ the weight of cut $C\in\Phi$, that is $F(C)=\sum_{e\in C}w_e$. Denote also by F^* the weight of the minimum global cut in G, that is $F^*=\min_{C\in\Phi}F(C)$. Karger [71] proved that the number of global cuts $C\in\Phi$ such that $F(C)\leq kF^*$ is $\mathcal{O}(n^{2k})$ for every fixed $k\geq 1$. Nagamochi *et al.* [109] designed an $\mathcal{O}(|E|^2|V|+|V|^{2k}|E|)$ algorithm for enumerating all global cuts in G of weights less than kF^*. Observe, that this algorithm is polynomial for any fixed k.

Consider now Minmax Regret Minimum Global Cut. Let us fix scenario S_E^+ in G and define

$$\Phi^*=\{C\in\Phi: F(C,S_E^+)\leq 2F^*(S_E^+)\}.$$

Set Φ^* contains all global cuts in G that have weights under S_E^+ not greater than $2F^*(S_E^+)$, where $F^*(S_E^+)$ is the weight of the minimum global cut under S_E^+. The following proposition is true (see also Aissi *et al.* [5]):

Proposition 9.11. *Set Φ^* contains an optimal robust global cut.*

Proof. Denote by C an optimal robust global cut and by C_1 the minimum global cut under scenario S_E^+. It holds:

$$F(C,S_E^+)=F(C,S_C^+)=Z(C)+F^*(S_C^+)\leq Z(C_1)+F^*(S_E^+)\leq$$

$$\leq F(C_1,S_{C_1}^+)+F^*(S_E^+)=F(C_1,S_E^+)+F^*(S_E^+)=2F^*(S_E^+).$$

Hence $C\in\Phi^*$ and the proof is completed. □

Applying the algorithm designed by Nagamochi *et al.* [109] we can compute in polynomial time set Φ^*. We then choose $C \in \Phi^*$ of the minimal value of the maximal regret. Since the maximal regret of a given global cut can be computed in polynomial time, the optimal robust global cut can be obtained in polynomial time as well. Observe that the additional requirement, that the global cut must separate two distinguished nodes s and t in G makes the problem strongly NP-hard.

9.6 Notes and References

The minimum cut in a given graph G with deterministic weights is typically obtained as a byproduct of computing the maximum flow in G. It follows from the well known *max flow - min cut* theorem, which states that Minimum Cut is a dual problem to Maximum Flow. This close relationship between these two problems was first established by Ford and Fulkerson [51] who also designed an augmenting path algorithm that computes both the maximum flow and the minimum cut in a given graph. A comprehensive review of different polynomial and pseudopolynomial algorithms for computing the maximum flow and minimum cut can be found in the book by Ahuja *et al.* [2].

The Minmax Regret Minimum Cut problem was first discussed by Aissi *et al.* [5]. They proved that computing the optimal robust cut in an undirected graph is strongly NP-hard. The polynomially solvable version of the problem, presented in Section 9.5, also follows from [5] and it is based on the results obtained by Karger [71] and Nagamochi *et al.* [109]. In Section 9.1 we have proved that the problem is NP-hard for directed graphs, even if they have a series-parallel topology. The close relationships between Minmax Regret Minimum Cut and Minmax Regret Shortest Path in series-parallel multidigraphs were established by Kasperski and Zieliński [83]. The FPTAS for this special version of the problem was also designed in [83].

10 Fuzzy Combinatorial Optimization Problem

Modeling the weights by means of closed intervals is perhaps the simplest form of uncertainty representation. For every weight we must only specify a range of possible values. In this chapter we discuss a more sophisticated uncertainty evaluation. The key idea is to extend the notion of the classical closed interval to the fuzzy one. A *fuzzy interval* can be seen as a family of closed intervals parametrized by the value of $\lambda \in [0, 1]$. It is reacher in information than a classical one and allows a representation that is at once both pessimistic and optimistic. A fuzzy interval has an interpretation in the setting of *possibility theory*, which is described for instance in a book by Dubois and Prade [44]. In Section 10.1 we describe the concept of a fuzzy interval and its possibilistic interpretation. We will consider then a combinatorial optimization problem in which the uncertain weights are modeled by means of fuzzy intervals.

In Chapter 2 we have introduced the concepts of possibly and necessarily optimal solutions and elements. The optimality evaluation has a clear interpretation in the setting of possibility theory. Intuitively, a solution X is possibly optimal if the event that X will be optimal may happen and it is necessarily optimal if the event that it will be optimal is sure. The same holds for elements. In the fuzzy case, the Boolean notions of possible and necessary optimality are generalized to *degrees of possible and necessary optimality*, which are numbers from interval [0,1]. Both degrees can also be deriven from a *fuzzy deviation*, which generalizes the interval deviation discussed in Chapter 2 and provides a full possibilistic characterization of optimality. The optimality evaluation and some methods of computing the fuzzy deviation are discussed in Section 10.3.

In Section 10.4 we consider the problem of choosing a solution under fuzzy weights. We discuss two solution concepts that are closely related to the minmax regret approach. The first concept consists in computing a solution that has the greatest necessary optimality degree. This solution can be obtained in polynomial time if the underlying deterministic problem is polynomially solvable. The second concept consists in determining so called *necessarily soft optimal* solution, which can be determined in polynomial time if the underlying minmax regret problem is polynomially solvable. Similar solution concepts were applied to the linear programming problem with a fuzzy objective function [67, 68].

A. Kasperski: Discrete Optimization with Interval Data, STUDFUZZ 228, pp. 137–153, 2008.
springerlink.com

10.1 Fuzzy Interval and Its Possibilistic Interpretation

The concept of a *fuzzy set* was introduced by Zadeh in 1965 [130]. A fuzzy set $\tilde{A}$ is a reference set Ω and a mapping $\mu_{\tilde{A}}$ from Ω into $[0,1]$, the unit interval. The value of $\mu_{\tilde{A}}(\omega)$, for $\omega \in \Omega$, is interpreted as the degree of membership of ω in the fuzzy set $\tilde{A}$. The *membership function* $\mu_{\tilde{A}}$ has also a possibilistic interpretation and it will be described later in this section. A *fuzzy quantity* $\tilde{a}$ is a fuzzy set in which Ω is the set of reals R and $\mu_{\tilde{a}}$ is a mapping from R into $[0,1]$. If additionally

1. $\mu_{\tilde{a}}$ is *normal*, i.e. there exists $x \in R$ such that $\mu_{\tilde{a}}(x) = 1$,
2. $\mu_{\tilde{a}}$ is *quasiconcave*, i.e. for all $x, y, z \in R$ such that $x \leq y \leq z$ it holds $\mu_{\tilde{a}}(y) \geq \min\{\mu_{\tilde{a}}(x), \mu_{\tilde{a}}(z)\}$,
3. $\mu_{\tilde{a}}$ is upper semicontinuous,
4. the set $cl\{x : \mu_{\tilde{a}}(x) > 0\}$, called a *suppport* of $\tilde{a}$, is bounded,

then $\tilde{a}$ is called a *fuzzy interval* [44]. A fuzzy interval $\tilde{a}$ is nonnegative if $\mu_{\tilde{a}}(x) = 0$ for all $x < 0$. Recall that every closed interval $\tilde{a} = [\underline{a}, \overline{a}]$ can be represented by its *characteristic function*, that is a mapping $\mu_{\tilde{a}}$ from R into set $\{0,1\}$, where $\mu_{\tilde{a}}(x) = 1$ if and only if $x \in \tilde{a}$. It is easy to verify that this characteristic function fulfills conditions 1-4. Hence every closed interval can be viewed a special case of a fuzzy one.

A λ-*cut* of a fuzzy interval $\tilde{a}$ is a subset of R defined as follows:

$$\tilde{a}^{\lambda} = \{x : \mu_{\tilde{a}}(x) \geq \lambda\}, \ \lambda \in (0,1].$$

We will additionally assume that $\tilde{a}^0$ is the support of $\tilde{a}$. It can be shown, that for every $\lambda \in [0,1]$, $\tilde{a}^{\lambda}$ is a closed interval. Thus a fuzzy interval can be seen as a family of closed intervals $\tilde{a}^{\lambda} = [\underline{a}(\lambda), \overline{a}(\lambda)]$, parametrized by the value of $\lambda \in [0,1]$. This family is monotone, that is if $\lambda_1 \geq \lambda_2$, then $\tilde{a}^{\lambda_1} \subseteq \tilde{a}^{\lambda_2}$. Clearly, if $\tilde{a}$ is a closed interval, then $\tilde{a}^{\lambda} = [\underline{a}, \overline{a}]$ for all $\lambda \in [0,1]$. We can obtain the membership function $\mu_{\tilde{a}}$ from the family of λ-cuts of $\tilde{a}$ in the following way:

$$\mu_{\tilde{a}}(x) = \sup\{\lambda \in [0,1] : x \in \tilde{a}^{\lambda}\} \tag{10.1}$$

and $\mu_{\tilde{a}}(x) = 0$ is $x \notin \tilde{a}^0$.

In practical applications *trapezoidal fuzzy intervals* are often used. A trapezoidal fuzzy interval $\tilde{a}$ is shown in Figure 10.1a,

Every trapezoidal fuzzy interval $\tilde{a}$ can be described by a quadruple $(\underline{a}, \overline{a}, \alpha, \beta)$, where $\underline{a} \leq \overline{a}$ and $\alpha, \beta \geq 0$. It can also be represented as the following family of λ-cuts:

$$\tilde{a}^{\lambda} = [\underline{a} - (1-\lambda)\alpha, \overline{a} + (1-\lambda)\beta], \ \lambda \in [0,1]. \tag{10.2}$$

If $\underline{a} = \overline{a}$, then $\tilde{a}$ is called a *triangular fuzzy interval* and it is shortly described by triple (a, α, β). Note that a closed interval $\tilde{a}$ can be described as $(\underline{a}, \overline{a}, 0, 0)$ and a real number a (degenerate interval) as $(a, a, 0, 0)$ or $(a, 0, 0)$.

We now give an interpretation of a fuzzy interval in the setting of *possibility theory*. This theory provides an alternative method of dealing with imprecision

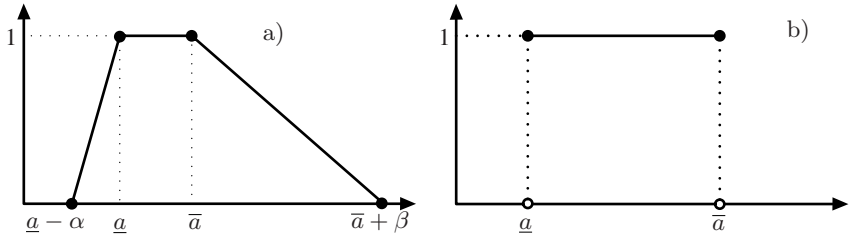


Fig. 10.1. a) A trapezoidal fuzzy interval $\tilde{a} = (\underline{a}, \overline{a}, \alpha, \beta)$ and b) a classical closed interval $\tilde{a} = [\underline{a}, \overline{a}]$, which can be viewed as a special case of a fuzzy interval

and, similarly to probability theory, it is based on several axioms described for instance in [44]. Let $\mathfrak{u}$ be a real-valued variable whose value is not precisely known. Let $\tilde{u}$ be a fuzzy interval associated with variable $\mathfrak{u}$. The membership function $\mu_{\tilde{u}}$ of $\tilde{u}$ is a *possibility distribution* for the values of $\mathfrak{u}$. The interpretation of the possibility distribution and some methods of obtaining it from a possessed knowledge (for instance from statistical data) are described in a book by Dubois and Prade [44], which is entirely devoted to possibility theory. In general, the support of $\tilde{u}$ should be chosen so as to be sure that the value of $\mathfrak{u}$ will not fall outside it. On the other hand, set $\tilde{u}^1$ should contain the most plausible values of $\mathfrak{u}$ (see [44]).

In the possibilistic interpretation the value of $\mu_{\tilde{u}}(x)$ denotes the *possibility* of the event that $\mathfrak{u}$ will take the value of x, that is

$$\Pi(\mathfrak{u} = x) = \mu_{\tilde{u}}(x). \tag{10.3}$$

Let A be a subset of R. The possibility of the event "$\mathfrak{u} \in A$" is defined in the following way:

$$\Pi(\mathfrak{u} \in A) = \sup_{x \in A} \mu_{\tilde{u}}(x).$$

The necessity of the event "$\mathfrak{u} \in A$" is defined as follows:

$$N(\mathfrak{u} \in A) = 1 - \Pi(\mathfrak{u} \notin A) = \inf_{x \notin A} (1 - \mu_{\tilde{u}}(x)).$$

Furthermore, observe that if $\tilde{u}$ is a closed interval, then $\Pi(\mathfrak{u} \in A) = 1$ if $A \cap \tilde{u} \neq \emptyset$ and $\Pi(\mathfrak{u} \in A) = 0$ otherwise. In other words, $\Pi(\mathfrak{u} \in A) = 1$ if and only if the event that $\mathfrak{u} \in A$ is possible. Similarly, $N(\mathfrak{u} \in A) = 1$ if $\tilde{u} \subseteq A$ and $N(\mathfrak{u} \in A) = 0$ otherwise. Hence $N(\mathfrak{u} \in A) = 1$ if and only if the event that $\mathfrak{u} \in A$ is sure. In general case both possibility and necessity can take their values from the whole interval $[0, 1]$.

10.2 Combinatorial Optimization Problem with Fuzzy Weights

Let us denote by $\mathfrak{w}_e$ the uncertain weight of element $e \in E$ in a combinatorial optimization problem $\mathcal{P}$. We can treat $\mathfrak{w}_e$ as a real variable whose value is not

precisely known. We will assume that the weights in the problem are unrelated, that is the value of $\mathfrak{w}_e$ does not depend on the values of the remaining weights. For every weight $\mathfrak{w}_e$, a nonnegative fuzzy interval $\tilde{w}_e$ is given whose membership function $\mu_{\tilde{w}_e}$ is a possibility distribution for the values of $\mathfrak{w}_e$. The interpretation of the possibility distribution has been described in the previous section. Recall that the value of $\mu_{\tilde{w}_e}(x)$ denotes the possibility of the event that $\mathfrak{w}_e$ will take the value of x, that is

$$\Pi(\mathfrak{w}_e = x) = \mu_{\tilde{w}_e}(x). \tag{10.4}$$

Let $S = (w_e^S)_{e\in E}$ be a scenario, that is a configuration of the weights in $\mathcal{P}$ that represents a certain state of the world, where $\mathfrak{w}_e = w_e^S$ for all $e \in E$. The *joint possibility distribution* on scenarios is as follows:

$$\pi(S) = \Pi(\wedge_{e\in E}(\mathfrak{w}_e = w_e^S)) = \min_{e\in E} \Pi(\mathfrak{w}_e = w_e^S) = \min_{e\in E} \mu_{\tilde{w}_e}(w_e^S). \tag{10.5}$$

Observe that if all $\tilde{w}_e$ are closed intervals, then $\pi(S) = 1$ if $S \in \Gamma$ and $\pi(S) = 0$ otherwise, where Γ is the Cartesian product of all $\tilde{w}_e$. Hence in the interval-valued case $\pi(S)$ is a characteristic function of scenario set Γ. In the fuzzy-valued case $\pi(S)$ can be seen as a membership function of the fuzzy scenario set $\tilde{\Gamma}$ and $\pi(S)$ is the possibility of the event that scenario S will occur. In the next two sections we will discuss the optimality evaluation and the problem of choosing a solution under fuzzy weights.

10.3 Fuzzy Evaluation of Optimality

According to possibility theory, the *degrees of possible and necessary optimality* of a solution $X \in \Phi$ are defined as follows:

$$\Pi(X \text{ is optimal}) = \sup_{\{S:X \text{ is optimal under } S\}} \pi(S), \tag{10.6}$$

$$N(X \text{ is optimal}) = 1 - \Pi(X \text{ is not optimal}) = \tag{10.7}$$

$$= \inf_{\{S:X \text{ is not optimal under } S\}} (1 - \pi(S)). \tag{10.8}$$

Similarly, the degrees of possibile and necessary optimality of an element $e \in E$ are defined as follows:

$$\Pi(e \text{ is optimal}) = \sup_{\{S:e \text{ is optimal under } S\}} \pi(S), \tag{10.9}$$

$$N(e \text{ is optimal}) = 1 - \Pi(e \text{ is not optimal}) =$$

$$= \inf_{\{S:e \text{ is not optimal under } S\}} (1 - \pi(S)). \tag{10.10}$$

Hence the optimality of a solution or an element is now characterized by two numbers from interval $[0, 1]$. It is easy to see that the degrees of optimality generalize the notions of possible and necessary optimality. If all the weights are specified as closed intervals, then $\Pi(X \text{ is optimal}) \in \{0, 1\}$ and $\Pi(X \text{ is optimal}) = 1$

if and only if X is possibly optimal. The same property is true for the degree of necessary optimality of X and for the degrees of possible and necessary optimality of elements. Due to the possibility-necessity relations [44], we have:

$$\begin{aligned} \Pi(X \text{ is optimal}) < 1 &\Rightarrow N(X \text{ is optimal}) = 0, \\ N(X \text{ is optimal}) > 0 &\Rightarrow \Pi(X \text{ is optimal}) = 1, \\ N(X \text{ is optimal}) &\leq \Pi(X \text{ is optimal}). \end{aligned}$$

The same relations are true for elements. The following relations hold between the optimality degrees of solutions and elements:

$$\Pi(X \text{ is optimal}) \leq \min_{e \in X} \Pi(e \text{ is optimal}), \tag{10.11}$$

$$N(X \text{ is optimal}) \leq \min_{e \in X} N(e \text{ is optimal}). \tag{10.12}$$

It is not difficult to show examples in which the strict inequalities in (10.11) and in (10.12) hold. To see this, consider the Minimum Spanning Tree problem shown in Figure 2.1. For the graph in Figure 2.1a it holds $\Pi(c \text{ is optimal}) = \Pi(d \text{ is optimal}) = \Pi(e \text{ is optimal}) = 1$ but $\Pi(\{c, d, e\} \text{ is optimal}) = 0$. Similarly, in Figure 2.1b we have $N(a \text{ is optimal}) = N(c \text{ is optimal}) = 1$ but $N(\{a, c\} \text{ is optimal}) = 0$. This is another way of expressing the fact, that there may exist a nonpossibly (nonnecessarily) optimal solution entirely composed of possibly (necessarily) optimal elements.

Recall that in the interval case the possible and necessary optimality can be expressed by means of the concept of a deviation. In Chapter 2 we have introduced intervals Δ_X and Δ_e that contain all possible values of deviations of solution X and element e. If the weights in the problem are specified as fuzzy intervals, then the deviations become possibilistic variables $\mathfrak{d}_X$ and $\mathfrak{d}_e$, whose possibility distributions are the membership functions of the fuzzy intervals $\tilde{\Delta}_X$ and $\tilde{\Delta}_e$. These membership functions are defined in the following way:

$$\mu_{\tilde{\Delta}_X}(x) = \Pi(\mathfrak{d}_X = x) = \sup_{\{S:\, \delta_X(S) = x\}} \pi(S), \tag{10.13}$$

$$\mu_{\tilde{\Delta}_e}(x) = \Pi(\mathfrak{d}_e = x) = \sup_{\{S:\, \delta_e(S) = x\}} \pi(S), \tag{10.14}$$

where $\delta_X(S)$ and $\delta_e(S)$ are deviations of X and e under scenario S (see formulae (2.1) and (2.4)). Using the fact that X is optimal under S if and only if $\delta_X(S) = 0$, we can express the possible and necessary optimality of a solution X in the following way:

$$\begin{aligned} \Pi(X \text{ is optimal}) &= \sup_{\{S:\, X \text{ is optimal under } S\}} \pi(S) \\ &= \sup_{\{S:\, \delta_X(S) = 0\}} \pi(S) = \Pi(\mathfrak{d}_X = 0) = \mu_{\tilde{\Delta}_X}(0), \end{aligned} \tag{10.15}$$

$$\begin{aligned} N(X \text{ is optimal}) &= 1 - \sup_{\{S:\, X \text{ is not optimal under } S\}} \pi(S) \\ &= 1 - \sup_{\{S:\, \delta_X(S) > 0\}} \pi(S) = N(\mathfrak{d}_X = 0) = 1 - \sup_{x > 0} \mu_{\tilde{\Delta}_X}(x). \end{aligned}$$

Similary, for a given element the following formulae are true:

$$\begin{aligned} \Pi(e \text{ is optimal}) &= \Pi(\mathfrak{d}_e = 0) = \mu_{\tilde{\Delta}_e}(0), \\ N(e \text{ is optimal}) &= N(\mathfrak{d}_e = 0) = 1 - \sup_{x>0} \mu_{\tilde{\Delta}_e}(x). \end{aligned}$$

Hence we can derive the degrees of optimality of X and e from their fuzzy deviations. In the next sections we focus on the problem of computing the optimality degrees and fuzzy deviations.

10.3.1 Computing the Degrees of Optimality

We now show a general method of computing the degrees of possible and necessary optimality. Let us denote by $\mathcal{P}^\lambda$, $\lambda \in [0,1]$, the combinatorial optimization problem in which the weights are given as closed intervals being the λ-cuts of the corresponding fuzzy intervals. Hence the interval weight of element $e \in E$ in $\mathcal{P}^\lambda$ is $\tilde{w}_e^\lambda$. Let us denote by $\tilde{\Delta}_X^\lambda = [\underline{\delta}_X(\lambda), \overline{\delta}_X(\lambda)]$ the interval of deviations of solution X in problem $\mathcal{P}^\lambda$. Clearly, interval $\tilde{\Delta}_X^\lambda$ is the λ-cut of the fuzzy interval $\tilde{\Delta}_X$. From (10.1) and (10.15) we get:

$$\begin{aligned} \Pi(X \text{ is optimal}) = \mu_{\tilde{\Delta}_X}(0) &= \sup\{\lambda \in [0,1] :\ 0 \in \tilde{\Delta}_X^\lambda\} = \\ &= \sup\{\lambda \in [0,1] : \underline{\delta}_X(\lambda) = 0\}. \end{aligned} \tag{10.16}$$

In a similar way one can prove the following equality:

$$N(X \text{ is optimal}) = 1 - \inf\{\lambda \in [0,1] : \overline{\delta}_X(\lambda) = 0\}. \tag{10.17}$$

It can also be verified that $\Pi(X \text{ is optimal}) = 0$ if $\underline{\delta}_X(0) > 0$ and, similarly, $N(X \text{ is optimal}) = 0$ if $\overline{\delta}_X(1) > 0$. Exactly the same reasoning can be applied to an element $e \in E$ for which the formulae equivalent to (10.16) and (10.17) exist. It is enough to replace X with e in (10.16) and (10.17).

Function $\underline{\delta}_X(\lambda)$ is nondecreasing and function $\overline{\delta}_X(\lambda)$ is nonincreasing. Therefore formulae (10.16) and (10.17) suggest standard binary search algorithms for determining the optimality degrees with a given accuracy ϵ. These algorithms, for a solution, are shown in Figures 10.2 and 10.3. Of course, similar algorithms can be constructed to calculate the degrees of optimality of a given element $e \in E$. It is enough to replace X with e in both algorithms.

The running time of `Degree Pos` and `Degree Nec` is $\mathcal{O}(f(n) \log \epsilon^{-1})$, where $\epsilon > 0$ is a given accuracy and $f(n)$ is the time required to evaluate the possible and necessary optimality in problem $\mathcal{P}^\lambda$ with interval weights (that is to decide whether $\underline{\delta}_X(\lambda) = 0$ or $\overline{\delta}_X(\lambda) = 0$ for a given value of λ). We now can see that the complexity of computing the optimality degrees relies on the classical interval case. If some efficient algorithms for evaluating the optimality in the interval case exist, then generalization to the fuzzy case is straightforward. On the other hand, if evaluating the optimality in the interval case is computationally intractable, then computing the degrees of optimality in the fuzzy case is computationally intractable as well.

`Degree Pos`

Require: Problem $\mathcal{P}$ with fuzzy weights, solution $X \in \Phi$, accuracy ϵ.
Ensure: $\Pi(X \text{ is optimal})$ with accuracy ϵ.
1: **if** $\underline{\delta}_X(0) > 0$ **then return** 0 $\{\Pi(X \text{ is optimal}) = 0\}$
2: $\lambda_1 \leftarrow 0.5$; $\lambda_2 \leftarrow 0$; $k \leftarrow 1$
3: **while** $|\lambda_1 - \lambda_2| \geq \epsilon$ **do**
4: $\quad \lambda_2 \leftarrow \lambda_1$
5: $\quad$ **if** $\underline{\delta}_X(\lambda_1) = 0$ **then** $\lambda_1 \leftarrow \lambda_1 + \frac{1}{2^{k+1}}$
6: $\quad$ **else** $\lambda_1 \leftarrow \lambda_1 - \frac{1}{2^{k+1}}$
7: $\quad k \leftarrow k + 1$
8: **end while**
9: **return** λ_1 $\{\Pi(X \text{ is optimal}) = \lambda_1\}$

Fig. 10.2. The calculation of $\Pi(X \text{ is optimal})$. After replacing X with element e we obtain the algorithm for computing $\Pi(e \text{ is optimal})$.

`Degree Nec`

Require: Problem $\mathcal{P}$ with fuzzy weights, solution $X \in \Phi$, accuracy ϵ.
Ensure: $N(X \text{ is optimal})$ with accuracy ϵ.
1: **if** $\overline{\delta}_X(1) > 0$ **then return** 0 $\{N(X \text{ is optimal}) = 0\}$
2: $\lambda_1 \leftarrow 0.5$; $\lambda_2 \leftarrow 0$; $k \leftarrow 1$
3: **while** $|\lambda_1 - \lambda_2| \geq \epsilon$ **do**
4: $\quad \lambda_2 \leftarrow \lambda_1$
5: $\quad$ **if** $\overline{\delta}_X(\lambda_1) = 0$ **then** $\lambda_1 \leftarrow \lambda_1 - \frac{1}{2^{k+1}}$
6: $\quad$ **else** $\lambda_1 \leftarrow \lambda_1 + \frac{1}{2^{k+1}}$
7: $\quad k \leftarrow k + 1$
8: **end while**
9: **return** $1 - \lambda_1$ $\{N(X \text{ is optimal}) = 1 - \lambda_1\}$

Fig. 10.3. The calculation of $N(X \text{ is optimal})$. After replacing X with element e we obtain the algorithm for computing $N(e \text{ is optimal})$.

Observe that if problem $\mathcal{P}$ is polynomially solvable, then the degrees of optimality of a solution can always be efficiently computed. It follows from the fact, that in this case we can efficiently solve the corresponding interval problems by applying the results presented in Section 2.1. On the other hand, the evaluation of optimality of an element may be more complex and it can be efficiently done only for some particular cases, for instance if $\mathcal{P}$ is a matroidal problem.

Example 10.1. Let us illustrate the degrees of possible and necessary optimality of elements by an example. Consider the Minimum Spanning Tree problem shown in Figure 10.4a. The uncertain weights in this problem are given as triangular fuzzy intervals.

The degrees of possible and necessary optimality of all edges in the sample problem can be computed by means of algorithms `Degree Pos` and `Degree Nec`. The corresponding interval problems in $\mathcal{P}^\lambda$ can be efficiently solved by exploiting the fact that Minimum Spanning Tree is a matroidal problem. Thus we can

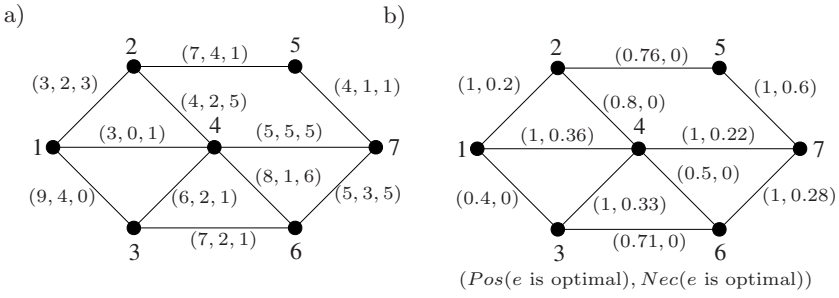


Fig. 10.4. a) A sample MINIMUM SPANNING TREE problem with fuzzy weights and b) degrees of possible and necessary optimality computed for every edge

use the results obtained in Section 2.2.3 to decide efficiently whether a given element is possibly (necessarily) optimal in $\mathcal{P}^\lambda$ for a fixed value of $\lambda \in [0,1]$. This is equivalent to deciding whether $\underline{\delta}_X(\lambda) = 0$ ($\overline{\delta}_X(\lambda) = 0$). The computed degrees of optimality have a possibilisitc interpretation and they provide an information about the optimality of the edges in the problem. For instance, edge $\{1,3\}$ has a low possible optimality degree equal to 0.4, and it can be viewed as less important than edge $\{5,7\}$, whose necessary optimality degree is equal to 0.6. Therefore, while constructing a solution one should prefer edge $\{5,7\}$ over $\{1,3\}$. □

10.3.2 Computing Fuzzy Deviations

In this section we discuss the problem of computing the fuzzy deviation of a given solution or element. As in Section 10.3.1 we first decompose the problem with fuzzy weights into a family of problems $\mathcal{P}^\lambda$, $\lambda \in [0,1]$, with interval weights. Consider a solution X. Recall that $\tilde{\Delta}_X^\lambda$, $\lambda \in [0,1]$, is a family of λ-cuts of the fuzzy interval $\tilde{\Delta}_X$, whose membership function expresses the possibility distribution for deviation of X. Therefore, by (10.1), the possibility distribution for the deviation $\mathfrak{d}_X$ is the following:

$$\Pi(\mathfrak{d}_X = x) = \mu_{\tilde{\Delta}_X}(x) = \sup\{\lambda \in [0,1] : x \in \tilde{\Delta}_X^\lambda\} \tag{10.18}$$

and $\mu_{\tilde{\Delta}_X}(x) = 0$ if $x \notin \tilde{\Delta}_X^0$. Exactly the same reasoning can be applied to the elements. It holds

$$\Pi(\mathfrak{d}_e = x) = \mu_{\tilde{\Delta}_e}(x) = \sup\{\lambda \in [0,1] : x \in \tilde{\Delta}_e^\lambda\},$$

where $\tilde{\Delta}_e^\lambda$, $\lambda \in [0,1]$, is the family of λ-cuts of $\tilde{\Delta}_e$.

Let us focus first on computing $\mu_{\tilde{\Delta}_X}$ for a given solution $X \in \Phi$. If we fix $\lambda \in [0,1]$, then we get problem $\mathcal{P}^\lambda$ with interval weights. We can now compute the bounds $\underline{\delta}_X(\lambda)$ and $\overline{\delta}_X(\lambda)$ using formulae (2.2) and (2.3). The extreme scenario $S_X^-(\lambda)$ in $\mathcal{P}^\lambda$ is obtained by fixing the weights of elements $e \in X$ to $\underline{w}_e(\lambda)$ and

the weights of the remaining elements to $\overline{w}_e(\lambda)$. Now the weight of X under $S_X^-(\lambda)$ is $\sum_{e\in X} \underline{w}_e(\lambda)$ and it is a function of $\lambda \in [0,1]$. It holds

$$\underline{\delta}_X(\lambda) = \sum_{e\in X} \underline{w}_e(\lambda) - F^*(S_X^-(\lambda)),\ \lambda \in [0,1].$$

In the same way we can obtain the extreme scenario $S_X^+(\lambda)$ in $\mathcal{P}^\lambda$ and

$$\overline{\delta}_X(\lambda) = \sum_{e\in X} \overline{w}_e(\lambda) - F^*(S_X^+(\lambda)),\ \lambda \in [0,1].$$

In order to compute functions $F^*(S_X^-(\lambda))$ and $F^*(S_X^+(\lambda))$ for $\lambda \in [0,1]$ we can apply a known technique called a *parametric approach*. In the parametric approach the weight of every element is specified as a function of a single parameter $\lambda \in [0,1]$. In our problem, in order to compute $F^*(S_X^-(\lambda))$ we set the weights of the elements according to scenario $S_X^-(\lambda)$ and we treat them as functions of the single parameter $\lambda \in [0,1]$. We wish to compute sequences $0 = \lambda_0 < \lambda_1 < ... < \lambda_k = 1$ and $X_0, \ldots, X_{k-1}$ such that X_i is the optimal solution for $\lambda \in [\lambda_i, \lambda_{i+1}]$. Having these sequences it is easy to describe function $F^*(S_X^-(\lambda))$ for $\lambda \in [0,1]$. In the same way we can compute the function $F^*(S_X^+(\lambda))$ for $\lambda \in [0,1]$. The corresponding parametric problem is obtained by setting the weights according to scenario $S_X^+(\lambda)$.

It turns out that for some particular problems like Shortest Path or Minimum Spanning Tree the corresponding parametric problem can be efficiently solved (see notes and references at the end of this chapter). In consequence the possibility distribution of the deviation of a given solution for these problems can be efficiently determined. Let us illustrate the computation of the fuzzy deviation by an example.

Example 10.2. Consider the Shortest Path problem shown in Figure 10.5a.

In this problem, for every imprecise arc weight $\mathfrak{w}_a$, $a \in A$, a triangular fuzzy interval $\tilde{w}_a = (w_a, \alpha_a, \beta_a)$ is given. This fuzzy interval expresses a possibility distribution for the values of $\mathfrak{w}_a$. Consider path $X = \{a_1, a_3\}$. In Figures 10.5a and 10.5b scenarios $S_X^-(\lambda)$ and $S_X^+(\lambda)$ are shown. Note that these scenarios are functions of λ and they are computed by means of formula (10.2).

It holds $F(X, S_X^-(\lambda)) = 8 - 4(1-\lambda)$ and $F(X, S_X^+(\lambda)) = 8 + 3(1-\lambda)$ for $\lambda \in [0,1]$. Applying the parametric approach to the problem shown in Figure 10.5b we get the sequence of λ's $0 < \frac{4}{7} < \frac{4}{5} < 1$ that corresponds to the sequence of optimal solutions $\{a_1, a_3\}, \{a_1, a_5, a_4\}, \{a_2, a_4\}$. That is solution $\{a_1, a_3\}$ is optimal for $\lambda \in [0, \frac{4}{7}]$, solution $\{a_1, a_5, a_4\}$ is optimal for $\lambda \in [\frac{4}{7}, \frac{4}{5}]$ and solution $\{a_2, a_4\}$ is optimal for $\lambda \in [\frac{4}{5}, 1]$. Hence $F^*(S_X^-(\lambda))$ is a piecewise linear function, whose value is $8 - 4(1-\lambda)$ for $\lambda \in [0, \frac{4}{7}]$, $5 + 3(1-\lambda)$ for $\lambda \in [\frac{4}{7}, \frac{4}{5}]$ and $4 + 8(1-\lambda)$ for $\lambda \in [\frac{4}{5}, 1]$. Subtracting $F^*(S_X^-(\lambda))$ from $F(X, S_X^-(\lambda))$ yields $\underline{\delta}_X(\lambda)$. Similarly, the function $\overline{\delta}_X(\lambda)$ is obtained by applying the parametric approach to the problem shown in Figure 10.5c. The obtained functions $\underline{\delta}_X(\lambda)$ and $\overline{\delta}_X(\lambda)$ are shown in Figure 10.6.

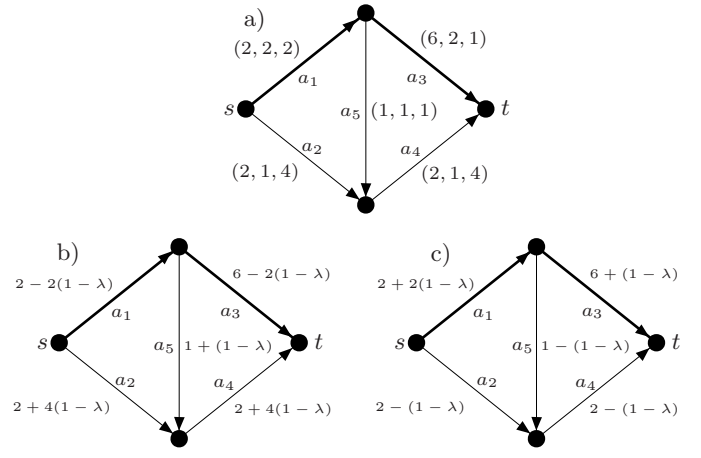


Fig. 10.5. a) The SHORTEST PATH problem with fuzzy weights. b) The extreme scenario $S^-_{\{a_1,a_3\}}(\lambda)$. c) The extreme scenario $S^+_{\{a_1,a_3\}}(\lambda)$.

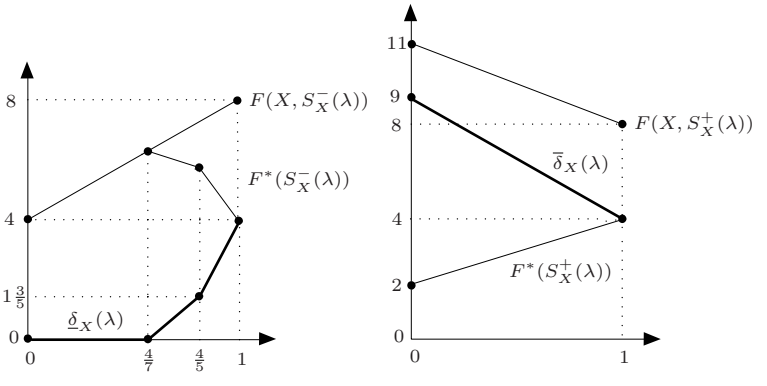


Fig. 10.6. The computation of $\underline{\delta}_X(\lambda)$ and $\overline{\delta}_X(\lambda)$. Both functions are shown in bold.

Having bounds $\underline{\delta}_X(\lambda)$ and $\overline{\delta}_X(\lambda)$ for $\lambda \in [0,1]$ we can construct the possibility distribution $\mu_{\tilde{\Delta}_X}$ for the deviation of X by applying formula (10.18). This possibility distribution is shown in Figure 10.7.

We can now compute $\Pi(X \text{ is optimal}) = \mu_{\tilde{\Delta}_X}(0) = \frac{4}{7}$. Applying possibility theory we can obtain more information about X, for instance $\Pi(\mathfrak{d}_X \geq 8) = \sup_{x \geq 8} \mu_{\tilde{\Delta}_X}(x) = 0.2$ and $N(\mathfrak{d}_X \leq 8) = 1 - \Pi(\mathfrak{d}_X > 8) = 1 - \sup_{x>8} \mu_{\tilde{\Delta}_X}(x) = 0.8$. In other words, possibility that the deviation of X will be greater than or equal to 8 is 0.2 and necessity that this deviation will be less than or equal to 8 is 0.8. Moreover, all possible values of $\mathfrak{d}_X$ are contained in the support of $\tilde{\Delta}_X$, that is in the interval $[0, 9]$. Therefore we are sure that $\mathfrak{d}_X \leq 9$ or equivalently $N(\mathfrak{d}_X \leq 9) = 1$. □

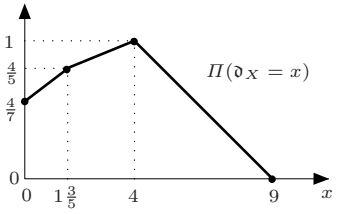


Fig. 10.7. The possibility distribution for deviation $\mathfrak{d}_X$ of solution X

Contrary to the solutions, computing $\mu_{\tilde{\Delta}_e}$ for a given element e may be much more complex. There are two difficulties that must be overcome. We must first identify the extreme scenarios that minimize and maximize the deviation of e in problem $\mathcal{P}^\lambda$. Having these scenarios, say $S_1(\lambda)$ and $S_2(\lambda)$, we must then compute the values of $F_e^*(S_1(\lambda))$ and $F_e^*(S_2(\lambda))$. This requires a modification of the parametric approach since we now seek an optimal solution in Φ_e, that is in the set of all feasible solutions that contain the element e. We now show how the computations can be performed for the MINIMUM SELECTING ITEMS problem, which has the simplest structure among the problems discussed in this monograph.

Example 10.3. Consider the MINIMUM SELECTING ITEMS problem, in which $E = \{e_1, e_2, e_3, e_4, e_5\}$, $p = 3$, and the item weights are specified by the following triangular fuzzy intervals: $\tilde{w}_{e_1} = (2, 2, 6)$, $\tilde{w}_{e_2} = (4, 3, 6)$, $\tilde{w}_{e_3} = (6, 3, 2)$, $\tilde{w}_{e_4} = (7, 2, 2)$, $\tilde{w}_{e_5} = (9, 7, 1)$. Let us choose item e_1 and consider the problem of computing $\overline{\delta}_{e_1}(\lambda)$ for $\lambda \in [0, 1]$ (the computation of $\underline{\delta}_{e_1}(\lambda)$ will be similar). We know that MINIMUM SELECTING ITEMS is a matroidal problem. Hence we can apply Theorem 2.16 that allows us to identify the extreme scenario $S = S^+_{\{e_1\}}(\lambda)$ that maximizes $\delta_{e_1}(S)$ in $\mathcal{P}^\lambda$. The family of scenarios $S^+_{\{e_1\}}(\lambda)$ for $\lambda \in [0, 1]$ is shown in Figure 10.8.

Before we proceed, let us make the following observation. If item e_1 is among the three least weighted items under S, then $\delta_{e_1}(S) = 0$. It follows from the fact that e_1 is then a part of an optimal solution under S and its deviation is 0. Otherwise,

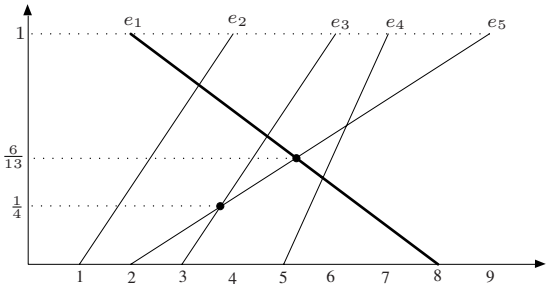


Fig. 10.8. The family of scenarios $S^+_{\{e_1\}}(\lambda)$ for $\lambda \in [0, 1]$

let Y be the optimal solution under S and let $f \in Y$ be the largest weighted item under S. Now it is easy to see that $\delta_{e_1}(S) = w^S_{e_1} - w^S_f$, which results from the fact that $Y \cup \{e_1\} \setminus \{f\}$ is the minimum weighted solution that contains e_1 under scenario S (we choose e_1 and two smallest weighted items under S).

We can now proceed as follows. For $\lambda \in [\frac{6}{13}, 1]$ element e_1 is among the three smallest weighted items under $S^+_{\{e_1\}}(\lambda)$ (see Figure 10.8), thus $\overline{\delta}_{e_1}(\lambda) = 0$ for $\lambda \in [\frac{6}{13}, 1]$. For $\lambda \in [0, \frac{6}{13})$ there exist three items of the weights strictly less than the weight of e_1 under $S^+_{\{e_1\}}(\lambda)$ and in consequence deviation $\overline{\delta}_{e_1}(\lambda)$ becomes positive. In order to compute the deviation it is enough to subtract the weight of the largest weighted item among the tree optimal ones from the weight of e_1. For $\lambda \in [\frac{1}{4}, \frac{6}{13}]$ the largest weighted item among the three optimal ones is e_5. In consequence $\overline{\delta}_{e_1}(\lambda) = 2+6(1-\lambda)-(9-7(1-\lambda)) = -7+13(1-\lambda)$ for $\lambda \in [\frac{1}{4}, \frac{6}{13}]$. For $\lambda \in [0, \frac{1}{4}]$ the largest weighted item among the three optimal ones is e_3 and $\overline{\delta}_{e_1}(\lambda) = 2 + 6(1-\lambda) - (6 - 3(1-\lambda)) = -4 + 9(1-\lambda)$ for $\lambda \in [0, \frac{1}{4}]$. We have thus obtained function $\overline{\delta}_{e_1}(\lambda)$ for $\lambda \in [0, 1]$, which is a piecewise linear one.

In the same way one can compute $\underline{\delta}_{e_1}(\lambda)$ by considering the family of scenarios $S^-_{\{e_1\}}(\lambda)$, $\lambda \in [0, 1]$. However, in this particular case it is easily seen that $\underline{\delta}_{e_1}(\lambda) = 0$ for all $\lambda \in [0, 1]$. Observe that e_1 is a part of an optimal solution under scenario $S^-_{\{e_1\}}(1)$ and consequently $\underline{\delta}_{e_1}(1) = 0$. This immediately gives $\underline{\delta}_{e_1}(\lambda) = 0$ for all $\lambda \in [0, 1]$ since $\underline{\delta}_{e_1}(\lambda)$ is a nonnegative and nondecreasing function. Having functions $\underline{\delta}_{e_1}(\lambda)$ and $\overline{\delta}_{e_1}(\lambda)$ we can construct the possibility distribution of deviation of e_1 (see Figure 10.9).

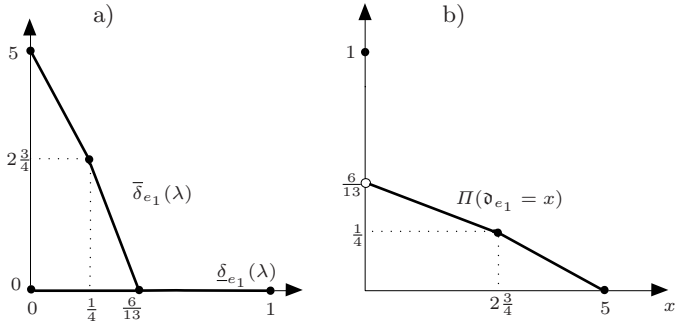


Fig. 10.9. a) The computation of $\underline{\delta}_{e_1}(\lambda)$ and $\overline{\delta}_{e_1}(\lambda)$. b) The possibility distribution for deviation $\mathfrak{d}_{e_1}$ of element e_1.

We can now compute $\Pi(e_1 \text{ is optimal}) = \mu_{\tilde{\Delta}_{e_1}}(0) = 1$, $N(e_1 \text{ is optimal}) = 1 - \sup_{x>0} \mu_{\tilde{\Delta}_{e_1}}(x) = 1 - \frac{6}{13} = \frac{7}{13}$. The support of $\tilde{\Delta}_{e_1}$, that is the set $[0, 5]$ contains all possible values of the deviation of e_1. Hence, we are sure that the deviation of e_1 will be not greater than 5. In other words $N(\mathfrak{d}_{e_1} \leq 5) = 1$. □

The method used in the above example can also be applied to the MINIMUM SELECTING ITEMS problem of a larger size. W have seen that in order to compute

the possibility distribution of an element deviation it is enough to explore carefully the intersection points of the weights under scenarios $S_e^+(\lambda)$ and $S_e^-(\lambda)$. We believe that a similar technique can be applied to all matroidal problems.

10.4 Choosing a Solution under Fuzzy Weights

The important question concerned with a problem with fuzzy weights is how to choose a solution. A natural approach is to choose a solution in Φ that has the greatest necessary optimality degree. Consider the following problem:

$$\text{MOST NEC } \mathcal{P}: \max_{X \in \Phi} N(X \text{ is optimal}). \tag{10.19}$$

Recall that the value of $N(X \text{ is optimal})$ can be computed by means of formula (10.17). Thus the MOST NEC $\mathcal{P}$ problem is equivalent to the following optimization problem:

$$\begin{aligned} &\min \lambda \\ &\overline{\delta}_X(\lambda) = 0 \\ &X \in \Phi \\ &0 \leq \lambda \leq 1 \end{aligned} \tag{10.20}$$

Hence we must find the smallest value of $\lambda \in [0,1]$, say λ^*, for which there exists a solution X such that $\overline{\delta}_X(\lambda) = 0$. The solution X is then the optimal one in MOST NEC $\mathcal{P}$ and $N(X \text{ is optimal}) = 1 - \lambda^*$. If problem (10.20) is infeasible, then $N(X \text{ is optimal}) = 0$ for all solutions $X \in \Phi$. Recall that the function $\overline{\delta}_X(\lambda)$ is nonincreasing. So, the value of λ^* together with solution X can be found by a applying binary search on $\lambda \in [0,1]$, provided that we can detect a necessarily optimal solution, that is the one such that $\overline{\delta}_X(\lambda) = 0$, in problem $\mathcal{P}^\lambda$. Using the results obtained in Section 2.1 we can see that solution X is necessarily optimal in $\mathcal{P}^\lambda$ if and only if the maximal regret of X in $\mathcal{P}^\lambda$ is equal to 0. Furthermore, by Theorem 4.5, a solution with the maximal regret equal to 0 can be detected by applying `Algorithm AM`. The MOST NEC $\mathcal{P}$ problem can by solved by the algorithm shown in Figure 10.10.

The running time of algorithm `Most Nec` is $\mathcal{O}(f(n) \log \epsilon^{-1})$, where $\epsilon > 0$ is a given precision and $f(n)$ is the running time of `Algorithm AM`. If the deterministic problem $\mathcal{P}$ is polynomially solvable, then `Algorithm AM` runs in polynomial time and the MOST NEC $\mathcal{P}$ problem can be solved in polynomial time as well.

The necessary optimality is a very strong criterion for choosing a solution. That is because we require an ideal solution, that is the one, which is optimal regardless of the weight realization, in some problem $\mathcal{P}^\lambda$. In many cases, however, algorithm `Most Nec` will return solution X such that $N(X \text{ is optimal}) = 0$. This is the case if there is not a necessarily optimal solution in problem $\mathcal{P}^1$. We now present a weaker optimization criterion by applying a concept of Inuiguchi and Sakawa [67, 68]. The idea consists in replacing the optimality requirement in (10.19) with a suboptimality one.

`Most Nec`
Require: Problem $\mathcal{P}$ with fuzzy weights, accuracy ϵ.
Ensure: The most necessarily optimal solution $X \in \Phi$ with accuracy ϵ.
Compute Y by applying `Algorithm AM` to $\mathcal{P}^1$.
if $Z(Y) > 0$ in $\mathcal{P}^1$ **then return** any $X \in \Phi$ $\{N(X \text{ is optimal}) = 0 \text{ for all } X \in \Phi\}$
$\lambda_1 \leftarrow 0.5$; $\lambda_2 \leftarrow 0$; $k \leftarrow 1$
while $|\lambda_1 - \lambda_2| \geq \epsilon$ **do**
  $\lambda_2 \leftarrow \lambda_1$
  Compute Y by applying `Algorithm AM` to $\mathcal{P}^{\lambda_1}$.
  if $Z(Y) = 0$ in $\mathcal{P}^{\lambda_1}$ **then** $X \leftarrow Y$, $\lambda_1 \leftarrow \lambda_1 - \frac{1}{2^{k+1}}$
  else $\lambda_1 \leftarrow \lambda_1 + \frac{1}{2^{k+1}}$
  $k \leftarrow k + 1$
end while
return X $\{N(X \text{ is optimal}) = 1 - \lambda_1\}$

Fig. 10.10. The calculation of the most necessarily optimal solution

Let μ_Z be a nonincreasing mapping from $[0, \infty)$ to interval $[0, 1]$ such that $\mu_Z(0) = 1$. This mapping is given by decision maker in advance and it expresses a *fuzzy goal* on deviation δ_X. In other words, the value of $\mu_Z(\delta_X)$ denotes the degree to which deviation δ_X satisfies a decision maker. Observe that $\delta_X = 0$ is the most preferred deviation and the fuzzy goal maps the values of deviation into interval $[0, 1]$. We can now define the degree of *necessary soft optimality* of a given solution X as follows:

$$N(X \text{ is soft optimal}) = \inf_S \max\{1 - \pi(S), \mu_Z(\delta_X(S))\}.$$

Consider the following optimization problem:

$$\text{Most Nec Soft } \mathcal{P}: \max_{X \in \Phi} N(X \text{ is soft optimal}). \tag{10.21}$$

Observe that Most Nec Soft $\mathcal{P}$ is a generalization of the Most Nec $\mathcal{P}$ problem. Indeed, if we define $\mu_Z(x) = 1$ if $x = 0$ and $\mu_Z(x) = 0$ for all $x \in (0, \infty)$, then (10.21) becomes equivalent to (10.19), because in this case $N(X \text{ is optimal}) = N(X \text{ is soft optimal})$. We prove the following theorem:

Theorem 10.4. *The* Most Nec Soft $\mathcal{P}$ *is equivalent to the following optimization problem:*

$$\begin{aligned} &\min \lambda \\ &\overline{\delta}_X(\lambda) \leq \mu_Z^{-1}(1 - \lambda) \\ &X \in \Phi \\ &0 \leq \lambda \leq 1 \end{aligned} \tag{10.22}$$

where μ_Z^{-1} *is a pseudo-inverse function of* μ_Z *defined as* $\mu_Z^{-1}(y) = \sup\{x : \mu_Z(x) \geq y\}$ *for* $y \in [0, 1]$.

Proof. The MOST NEC SOFT $\mathcal{P}$ can be expressed as follows:

$$\begin{aligned} &\max t \\ &\inf_S \max\{1-\pi(S), \mu_Z(\delta_X(S))\} \geq t \\ &X \in \Phi \\ &0 \leq t \leq 1 \end{aligned} \tag{10.23}$$

Let us focus on condition:

$$\inf_S \max\{1-\pi(S), \mu_Z(\delta_X(S))\} \geq t \tag{10.24}$$

Condition (10.24) is true if and only if for all scenarios S such that $1-\pi(S) < t$ it holds $\mu_Z(\delta_X(S)) \geq t$. From the definition of $\pi(S)$ it follows that all scenarios S such that $1-\pi(S) < t$ (or equivalently $\pi(S) > 1-t$) must belong to Cartesian product $\Gamma^{1-t} = \times_{e\in E}\tilde{w}_e^{1-t}$. We know that $\overline{\delta}_X(1-t)$ is the maximal value of $\delta_X(S)$ over all $S \in \Gamma^{1-t}$. Note that $\delta_X(S)$ is a continuous function of S and set Γ^{1-t} is compact. Since the function μ_Z is nonincreasing, condition (10.24) is equivalent to condition $\mu_Z(\overline{\delta}_X(1-t)) \geq t$, which is equivalent to

$$\overline{\delta}_X(1-t) \leq \mu_Z^{-1}(t). \tag{10.25}$$

Replacing (10.24) with (10.25) and substituting $t = 1-\lambda$ yield (10.22). □

It easy to check that μ_Z^{-1} is a nonincreasing function from $[0,1]$ to $R \cup \{+\infty\}$, where $\mu_Z^{-1}(0) = +\infty$. Therefore $\mu_Z^{-1}(1-\lambda)$ is nondecreasing function of $\lambda \in [0,1]$. Observe now that problem (10.22) can also be solved by binary search on $\lambda \in [0,1]$, which is similar to algorithm `Most Nec`. Now, however, we must compute the solution that minimizes the value of $\overline{\delta}_X(\lambda)$ for a given value of $\lambda \in [0,1]$. Since $\overline{\delta}_X(\lambda)$ is the maximal regret of X in problem $\mathcal{P}^\lambda$ (see Proposition 2.2), we must be able to solve the MINMAX REGRET $\mathcal{P}^\lambda$ problem. The algorithm to solve the MOST NEC SOFT $\mathcal{P}$ problem is shown in Figure 10.11.

The running time of algorithm `Most Nec Soft` is $\mathcal{O}(g(n)\log\epsilon^{-1})$, where $\epsilon > 0$ is a given precision and $g(n)$ is the time required to solve the minmax regret version of problem $\mathcal{P}$.

Example 10.5. Consider the SHORTEST PATH problem with fuzzy weights shown in Figure 10.12. The weights in this problem are given as trapezoidal fuzzy intervals.

Let us find first the most necessarily optimal solution in the problem. We start by computing the optimal robust solution in $\mathcal{P}^1$, that is in the problem in which the interval weights are 1-cuts of the fuzzy weights. We get path $P = \{(s,1),(1,4),(4,t)\}$ with the maximal regret in $\mathcal{P}^1$ equal to 5. We get at once that there is no necessarily optimal solution in problem $\mathcal{P}^1$, therefore $N(P \text{ is optimal }) = 0$ for all paths $P \in \Phi$. We can thus see that in this problem the necessary optimality is too strong criterion for choosing a solution.

`Most Nec Soft`
Require: Problem $\mathcal{P}$ with fuzzy weights, accuracy ϵ.
Ensure: The necessarily soft optimal solution $X \in \Phi$ with accuracy ϵ.
1: $\lambda_1 \leftarrow 0.5$; $\lambda_2 \leftarrow 0$; $k \leftarrow 1$
2: **while** $|\lambda_1 - \lambda_2| \geq \epsilon$ **do**
3: $\lambda_2 \leftarrow \lambda_1$
4: Compute Y by solving MINMAX REGRET $\mathcal{P}^{\lambda_1}$.
5: **if** $Z(Y) \leq \mu_Z^{-1}(1-\lambda_1)$ in $\mathcal{P}^{\lambda_1}$ **then** $X \leftarrow Y$, $\lambda_1 \leftarrow \lambda_1 - \frac{1}{2^{k+1}}$
6: **else** $\lambda_1 \leftarrow \lambda_1 + \frac{1}{2^{k+1}}$
7: $k \leftarrow k+1$
8: **end while**
9: **return** X $\{N(X \text{ is soft optimal}) = 1 - \lambda_1\}$

Fig. 10.11. The calculation of the most necessarily soft optimal solution

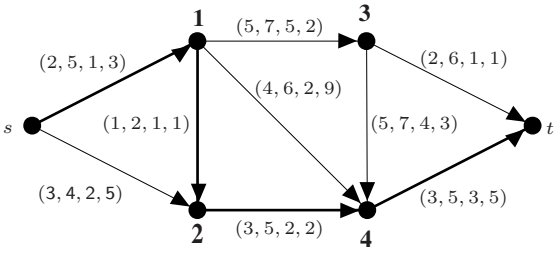


Fig. 10.12. An example of SHORTEST PATH with fuzzy weights. The path in bold is the most necessarily soft optimal path.

We now apply the second, weaker criterion. Let us introduce the following fuzzy goal μ_Z:

$$\mu_Z(x) = \begin{cases} 1 & \text{for } x \in [0,4] \\ -\frac{1}{11}x + \frac{15}{11} & \text{for } x \in [4,15] \\ 0 & \text{for } x \in [15,\infty) \end{cases}$$

Let us apply algorithm `Most Nec Soft` to the problem. As the result we get path $P = \{(s,1),(1,2),(2,4),(4,t)\}$ such that $N(P \text{ is soft optimal}) = 0.438$ with accuracy 0.01. This is the most necessarily soft optimal path and it can be regraded as the best choice in the sample problem. □

10.5 Notes and References

The possibilistic characterization of optimality of solutions and elements has some tradition in optimization. It was applied to analyze the criticality of paths and activities in fuzzy CPM by Chanas and Zieliński [31, 32, 33], Zieliński [134] and Dubois *et al.* [45, 46]. Thus it was applied to the LONGEST PATH problem

in a directed and acyclic network. In [45] and [46] a possibilistic interpretation of the criticality evaluation was provided, which is similar to the interpretation of the optimality evaluation discussed in this chapter. In particular, the concept of a deviation, introduced by Kasperski and Zieliński in [84], is similar to the concept of a float in the interval and fuzzy scheduling [45, 47, 134]. In [34] Chanas and Kasperski characterized the optimality of sequences of jobs in a single machine scheduling problem with imprecise parameters. The possibly and necessarily optimal solutions of linear programming problem with imprecise objective coefficients were described by Inuiguchi and Sakawa [64, 65, 66].

The concepts of degrees of possible and necessary optimality in a combinatorial optimization problem with imprecise weights were described by Kasperski and Zieliński in [78, 81, 82]. In [53] some efficient methods of computing the optimality degrees in matroidal problems with fuzzy weights were discussed. In [84] the concept of a deviation was introduced. A description of the parametric approach, which is a tool for computing the possibility distribution of fuzzy deviation, can be found for instance in [3, 50, 126]. The parametric techniques can be applied if the weights are specified as linear functions of a parameter λ. Hence they can be applied to the fuzzy problem if the weights are modeled as trapezoidal fuzzy intervals.

A comprehensive description of possibility theory and the possibilistic interpretation of fuzzy intervals can be found in [43, 44]. This theory allows us to describe the notion of optimality and choose a robust solution in a fuzzy problem. The possibilistic analysis appears to be much easier than a probabilistic modeling. In particular, the computation of the possibility distribution of a solution or element deviation in a possbilisitic framework is easier than with a probabilistic model. The solution concepts in a fuzzy problem, described in Section 10.4, were first applied to the linear programming problem with a fuzzy objective in [67, 68].

11 Conclusions and Open Problems

In this part of the monograph we have discussed a particular class of discrete optimization problems in which every feasible solution can be expressed as a subset of a given generic set E and the criterion, in the deterministic case, is the sum of the weights. The first important property of the minmax regret version of this problem is that the maximal regret of every feasible solution can be computed in polynomial time if the underlying problem $\mathcal{P}$ is polynomially solvable. It results from a simple characterization of the worst case scenario of a given solution. The second important property of this class of problems is the preprocessing based on the concepts of possibly and necessary optimal elements. It turns out that the number of nonpossibly optimal elements may be quite large and removing them may significantly speed up calculations. Unfortunately, detecting all nonpossibly optimal elements may be itself NP-hard. For some particular problems, however, like Minmax Regret Minimum Spanning Tree or Minmax Regret Minimum Selecting Items, all nonpossibly and necessarily optimal elements can be efficiently detected.

As we have seen, all particular problems except for Minmax Regret Minimum Selecting Items, which has a very simple combinatorial structure, are NP-hard or strongly NP-hard. Hence the minmax regret problems are, in general, more difficult to solve than their deterministic counterparts.

In order to compute the optimal robust solution we can choose between the exact methods, such as a MIP formulation or a branch and bound algorithm, and approximation. The existence of a general 2-approximation algorithm is another important property of Minmax Regret $\mathcal{P}$. The approximation algorithms proposed in this monograph behave quite well for the tested instances. They can be applied to large problems for which the MIP formulation and the branch and bound algorithm fail to compute the optimal solution in a reasonable time. The solutions returned by the approximation algorithms are also good starting points for more sophisticated metaheuristics based on local search. In particular, in this monograph we have shown that the algorithm tabu search is a very promising tool to solve Minmax Regret Minimum Spanning Tree.

There are a number of open problems connected with Minmax Regret $\mathcal{P}$. We list below some of them which seem to be of particular importance.

A. Kasperski: Discrete Optimization with Interval Data, STUDFUZZ 228, pp. 155–157, 2008.
springerlink.com

1. The MINMAX REGRET $\mathcal{P}$ problem is approximable within 2 if $\mathcal{P}$ is polynomially solvable. But we do not know whether 2 is the best possible worst case ratio for any particular problem. In the proof of the worst case performance of `Algorithm AM` we have not taken into account any particular structure of problem $\mathcal{P}$. The proof is essentially based on the properties of the maximal regret criterion. Therefore, one should investigate the particular structure of some problems $\mathcal{P}$. There are three general research directions for every particular MINMAX REGRET $\mathcal{P}$ problem:
   a) design an approximation algorithm with lower than 2 worst case performance ratio;
   b) prove that a particular problem is not approximable within a fixed constant lower than 2 (no results in this field are known up to now);
   c) construct an PTAS (polynomial time approximation scheme) or an FPTAS (fully polynomial time approximation scheme). In this monograph we have presented a framework to construct an FPTAS. The most involving task is to design a pseudopolynomial algorithm whose running time is bounded by a polynomial in n and UB.
2. Investigate more deeply the CENTRAL $\mathcal{P}$ problem for polynomially solvable problems $\mathcal{P}$. CENTRAL $\mathcal{P}$ can be viewed as a special version of MINMAX REGRET $\mathcal{P}$ in which all elements have interval weights [0,1]. We know that the general problem CENTRAL $\mathcal{P}$ is strongly NP-hard (because CENTRAL SPANNING TREE is strongly NP-hard). The CENTRAL $\mathcal{P}$ problem seems to be a core of MINMAX REGRET $\mathcal{P}$ in which the particular structure of set Φ plays a crucial role.
3. Explore the computational complexity of NEC $\mathcal{P}$ if $\mathcal{P}$ is polynomially solvable. We know some problems $\mathcal{P}$ for which NEC $\mathcal{P}$ is polynomially solvable, for instance the matroidal ones and SHORTEST PATH for acyclic digraphs. On the other hand, we do not know any such problem which is NP-hard. Therefore it is possible that, contrary to POS $\mathcal{P}$, there is a general polynomial algorithm for NEC $\mathcal{P}$ if $\mathcal{P}$ is polynomially solvable. It would be interesting to design such an algorithm even for some particular problems, for instance for MINMAX REGRET SHORTEST PATH in general graphs.
4. Design algorithms that would be able to detect quickly a subset of nonpossibly optimal elements if problem POS $\mathcal{P}$ is NP-complete.
5. We know that MINMAX REGRET MINIMUM SPANNING TREE is strongly NP-hard. However, the complexity of this problem for planar graphs or for the graphs with bounded node degrees is unknown. Designing fast and exact algorithms for this problem is particularly important, since the MIP formulation and the branch and bound algorithm allow us to solve rather small problems. The local search techniques are quite promising and they must be explored more deeply by carrying out more exhaustive computational tests.

6. The complexity status of Minmax Regret Shortest Path seems to be well known. The next research direction is to design algorithms that take into account a particular structure of the input graph.
7. The branch and bound algorithms for Minmax Regret Minimum Assignment and Minmax Regret Minimum Cut should be implemented and compared to the MIP approach.

12 Problem Formulation

In this chapter we discuss another class of discrete optimization problems, namely *sequencing problems*, which belong to a wide class of scheduling problems. Theory of scheduling is an important part of discrete optimization. It deals with a huge number of problems involving different types of resources. In this chapter we discuss only the simplest problems in which the set of resources consists of one machine and a schedule is a sequence (permutation) of a given set of jobs. We seek a feasible sequence to achieve a given goal. Every job has a nonnegative processing time and, typically, we wish to find an order of jobs in which the jobs are completed as quickly as possible (however, in some applications we should also avoid jobs completion being too early). Even this restricted class includes a large number of different problems. Unfortunately, most of them turned out to be NP-hard and their minmax regret versions are NP-hard as well.

In this chapter we adopt the maximal regret criterion to solve a sequencing problem under uncertainty. As in the previous part of this monograph, we will model the imprecise parameters by closed intervals. The first difficulty that arises is that there may be now several parameters associated with a job instead of a single weight as for the minmax regret combinatorial optimization problems considered in the previous part. Hence the characterization of the worst case scenario and the computation of the value of the maximal regret of a given solution become more difficult. In consequence, solving the minmax regret problem is also more challenging.

We start this chapter by recalling the formulation of the deterministic sequencing problem together with some examples. We introduce then the minmax regret sequencing problem, that is the one in which the imprecise parameters are specified as closed intervals and we seek a solution that minimizes the maximal regret. Later, we consider three particular minmax regret sequencing problems whose deterministic counterparts are polynomially solvable.

12.1 Deterministic Sequencing Problem

Let $J = \{J_1, J_2, \ldots, J_n\}$ be a given set of *jobs*. For simplicity of notations we will identify every job $J_i \in J$ with its index $i \in \{1, \ldots, n\}$. Associated with

A. Kasperski: Discrete Optimization with Interval Data, STUDFUZZ 228, pp. 161–165, 2008.
springerlink.com

every job $i \in J$ is a *processing time* p_i, which is a nonnegative real number. For every job $i \in J$ there may also be given some other parameters like a *due date* d_i expressing a desired completion time of i or a *weight* w_i expressing the importance of job i. There may also be given some *precedence constraints* between jobs. These constraints are represented by an acyclic directed graph $G = (V, A)$, where $V = \{1, \ldots, n\}$ corresponds to the jobs, and $(i, k) \in A$ if i must be completed before k starts. In this case we write $i \to k$ and we say that k is a *successor* of i. A *schedule* is a sequence of jobs

$$\pi = (\pi(1), \pi(2), \ldots, \pi(n))$$

and it expresses an order in which the jobs are processed. In particular, job $\pi(1)$ is processed first and job $\pi(n)$ is processed last. We consider only the case in which the jobs are *nonpreemptive*, that is processing of every job cannot be interrupted, and there are no *idle times* in a schedule. So, when processing of a job is finished, processing of the next job in a schedule is immediately started. Furthermore, we also assume that all jobs are available for processing at time 0.

We will denote by $P(\pi, i)$ the subset of jobs containing job i and all jobs that are processed before i in schedule π. A schedule π is *feasible* if the precedence constraints in π are preserved. Hence π is feasible if $i \to k$ implies $i \in P(\pi, k)$. We will denote by Π the set of all feasible schedules. For every job $i \in J$ we define its *completion time* in schedule π in the following way:

$$C_i(\pi) = \sum_{j \in P(\pi, i)} p_j. \tag{12.1}$$

There is a *cost function* $f_i(t)$ which measures the cost of completing i at time t. Some other parameters like due dates and weights may be used in defining function $f_i(t)$. Finally, $F(\pi)$ denotes the cost of schedule π. There are essentially two types of the cost function F. The first one, called a *bottleneck objective*, is of the following form:

$$F(\pi) = \max_{i=1}^{n} f_i(C_i(\pi)).$$

The second, called a *sum objective*, is as follows:

$$F(\pi) = \sum_{i=1}^{n} f_i(C_i(\pi)).$$

A *sequencing problem* $\mathcal{S}$ is the following:

$$\mathcal{S}: \min_{\pi \in \Pi} F(\pi). \tag{12.2}$$

Hence we seek a feasible schedule of the minimal total cost. The deterministic scheduling problems are denoted by means of a convenient Graham's notation. Using this notation, every sequencing problem is denoted by $\alpha|\beta|\gamma$, where α is the machine environment ($\alpha = 1$ for most sequencing problems), β specifies the job characteristics and γ describes the cost function F. The following examples demonstrate well-known sequencing problems:

- $1|prec|L_{max}$. For every job $i \in J$ a nonnegative due date d_i is given. The expression $L_i(C_i(\pi)) = C_i(\pi) - d_i$ is called a *lateness* of job i in schedule π. We seek a feasible schedule for which the maximal lateness, that is the value of
$$F(\pi) = \max_{i=1}^{n} L_i(C_i(\pi))$$
is minimal. This is the type of problem with the bottleneck objective function. We can also introduce a nonnegative weight w_i for every job $\in J$ and the maximal weighted lateness is then minimized.
- $1||\sum C_i$. In this problem there are no precedence constraints between jobs (any sequence of jobs is feasible) and we wish to find a sequence π in which the *total flow time*, that is the value of
$$F(\pi) = \sum_{i=1}^{n} C_i(\pi)$$
is minimal.
- $1|p_i = 1|\sum w_i U_i$. In this problem for every job $i \in J$ there are given a nonnegative due date d_i and a nonnegative weight w_i. It is assumed that there are no precedence constraints between jobs and every job has the unit processing time ($p_i = 1$ for all $i \in J$). A job is called *late* in schedule π if $C_i(\pi) > d_i$. Define $U_i(\pi) = 1$ if $C_i(\pi) > d_i$ and $U_i(\pi) = 0$ otherwise. We seek a schedule that minimizes the weighted number of *late jobs*, that is the value of
$$F(\pi) = \sum_{i=1}^{n} w_i U_i(\pi).$$

All the particular sequencing problems presented above are polynomially solvable and in this part of the monograph we discuss their minmax regret versions.

12.2 Minmax Regret Sequencing Problem with Interval Data

In a deterministic sequencing problem there is given a set of parameters associated with every job $i \in J$. This set always contains a processing time of job i and it may also contain some other parameters like a due date or a weight of i. Each of these parameters may be imprecise and it can be specified as a range of possible values given by a closed interval. If the value of a parameter is precise, then the corresponding interval is degenerate. A *scenario* is a particular realization of the parameters, such that the value of a parameter in scenario S belongs to its uncertainty interval. The set of all scenarios Γ is the cartesian product of all uncertainty intervals. Among scenarios we will distinguish the extreme ones in which all the parameters take the lower or upper bounds of their uncertainty intervals. Now, the completion time of job $i \in J$ in schedule π depends on scenario S and it is expressed as follows:

$$C_i(\pi, S) = \sum_{j \in P(\pi,i)} p_j^S, \tag{12.3}$$

where p_j^S is a processing time of job j under scenario S. The cost of a given schedule under scenario S is denoted by $F(\pi, S)$ and it is a function of jobs completion times under S. The value of $F^*(S)$ is the cost of an optimal schedule under S, that is

$$F^*(S) = \min_{\pi \in \Pi} F(\pi, S). \tag{12.4}$$

In order to obtain the value of $F^*(S)$ we must solve the deterministic sequencing problem under a fixed scenario $S \in \Gamma$. The *maximal regret* of a given schedule π is defined as follows:

$$Z(\pi) = \max_{S \in \Gamma} \{F(\pi, S) - F^*(S)\}. \tag{12.5}$$

Scenario S_π that maximizes the right hand side of (12.5) is called the *worst case scenario* for schedule π. Hence the maximal regret of π can also be expressed as follows:

$$Z(\pi) = \max_{\sigma \in \Pi} \{F(\pi, S_\pi) - F(\sigma, S_\pi)\} = F(\pi, S_\pi) - F(\pi^*, S_\pi), \tag{12.6}$$

where π^* is the optimal schedule under S_π and it is called the *worst case alternative* for π.

Contrary to the minmax regret combinatorial optimization problems (see Section 1.2), there is not a general characterization of the worst case scenario S_π. A method of computing S_π depends on the structure of the deterministic sequencing problem $\mathcal{S}$. In particular, it depends on the cost function F and the type of precedence constraints between jobs. The *minmax regret sequencing problem* $\mathcal{S}$ is defined as follows:

$$\text{Minmax Regret } \mathcal{S}: \ \min_{\pi \in \Pi} Z(\pi), \tag{12.7}$$

hence we seek a feasible schedule (sequence of jobs) for which the maximal regret is minimal. The optimal solution for Minmax Regret $\mathcal{S}$ is called the *optimal robust schedule*.

It is clear that Minmax Regret $\mathcal{S}$ is a generalization of problem $\mathcal{S}$. If all parameters are precise, then there is exactly one scenario and the problem boils down to computing the optimal schedule under this scenario. We now see that the minmax regret version of problem $\mathcal{S}$ is computationally not easier, than problem $\mathcal{S}$. If $\mathcal{S}$ is NP-hard, then the Minmax Regret $\mathcal{S}$ problem is NP-hard as well.

12.3 Notes and References

An excellent introduction to the deterministic scheduling and sequencing problems can be found in a book by Brucker [29]. In this book a lot of sequencing

models, together with complexity results, are described. Also described there is Graham's notation, introduced first in [60], which is commonly used for classifying scheduling problems. Unfortunately, most of the general sequencing problems are computationally intractable (see [29] for a comprehensive review). The particular problems described in Section 12.1 are among the problems that are polynomially solvable. Hence there is a hope that their minmax regret versions will be polynomially solvable as well.

Contrary to the minmax regret combinatorial optimization problems discussed in the first part of this monograph, there is a rather small number of papers devoted to minmax regret sequencing problems. The first problems of this type were discussed by Daniels and Kouvelis [39], Yu [127] and Kuovelis and Yu [87]. Some recent results in this field can be found in papers by Kasperski [75], Lebedev and Averbakh [94] and Montemanni [106].

13 Sequencing Problem with Maximum Lateness Criterion

This chapter is devoted to MINMAX REGRET $1|prec|L_{max}$. In this problem, for every job $i \in J$ we are given an interval processing time $\tilde{p}_i = [\underline{p}_i, \overline{p}_i]$ and an interval due date $\tilde{d}_i = [\underline{d}_i, \overline{d}_i]$. A scenario S is a particular realization of the processing times and the due dates in the problem. The set of jobs is partially ordered by some arbitrary precedence constraints. A *lateness* of job i under scenario S is defined as follows:

$$L_i(\pi, S) = C_i(\pi, S) - d_i^S, \tag{13.1}$$

where d_i^S is a due date of job i under scenario S. The cost of a given schedule under scenario S is as follows:

$$F(\pi, S) = \max_{i=1}^{n} L_i(\pi, S), \tag{13.2}$$

thus the cost is the maximal lateness in π under S. In the MINMAX REGRET $1|prec|L_{max}$ problem we wish to find a feasible schedule $\pi \in \Pi$ that minimizes the maximal regret. It turns out that this problem is polynomially solvable and we construct an algorithm whose running time is $\mathcal{O}(n^4)$ in Section 13.2. This algorithm is a generalization of the well known algorithm designed by Lawler, which is used to solve the deterministic problem $1|prec|L_{max}$. We recall Lawler's algorithm in Section 13.1.

13.1 Deterministic Problem and Lawler's Algorithm

If scenario S is fixed, then an optimal schedule under S can be obtained by solving the deterministic $1|prec|L_{max}$ problem. This can be done in $\mathcal{O}(n^2)$ time by means of the algorithm designed by Lawler [91]. In fact, Lawler's algorithm can be applied to more general problem $1|prec|f_{max}$, in which all functions $f_i(t)$ that measure the completion costs of jobs $i \in J$ are nondecreasing and the objective can be expressed in the following way:

$$F(\pi) = \max_{i \in J} f_j(C_j(\pi)).$$

A. Kasperski: Discrete Optimization with Interval Data, STUDFUZZ 228, pp. 167–173, 2008.
springerlink.com

`Lawler's Algorithm`
Require: $n, (p_i)_{i=1}^n, (d_i)_{i=1}^n, prec.$
Ensure: The optimal schedule π.
1: $D \leftarrow \{1, \ldots, n\}$
2: **for** $r \leftarrow n$ **downto** 1 **do**
3: Find $i \in D$, which has no successor in D and has a maximal value of d_i
4: $\pi(r) \leftarrow i$
5: $D \leftarrow D \setminus \{i\}$
6: **end for**
7: **return** π

Fig. 13.1. Lawler's algorithm for solving problem $1|prec|L_{max}$

A version of Lawler's algorithm for solving $1|prec|L_{max}$ is shown in Figure 13.1.

`Lawler's Algorithm` is based on the following rule: if D is a subset of jobs, then we choose a job that has no successor in D and has a maximal value of a due date among all jobs in D. The selected job is processed as the last one among the jobs in D. At the beginning set D consists of all jobs. At each iteration the cardinality of D is decreased by one and the procedure is repeated until set D is empty. Finally, an optimal schedule is constructed. The time required for computing the optimal schedule is $\mathcal{O}(n^2)$. If there are no precedence constraints between jobs, then the algorithm can be simplified. It is then enough to apply the following *earliest due dates* rule (EDD): order the jobs with respect to nondecreasing due dates. Obviously, this can be done in $\mathcal{O}(n \log n)$ time.

Observe that the processing times of jobs are not involved in `Lawler's Algorithm` and, consequently, the optimal schedule does not depend on the values of the processing times. As we will see in the next section, the same holds for the minmax regret version of the problem if all due dates are precise. The processing times become important if there are some uncertain due dates in the problem. In the next section we will show that the MINMAX REGRET $1|prec|L_{max}$ problem with imprecise processing times and imprecise due dates can be solved by an algorithm being a generalization of `Lawler's Algorithm` shown in Figure 13.1.

13.2 Polynomial Algorithm for the Problem

We focus first on the problem of determining the worst case scenario S_π for a given schedule $\pi \in \Pi$. Having the worst case scenario we will be able to compute the value of $Z(\pi)$ in polynomial time, which follows from the fact that the value of $F^*(S_\pi)$ can be computed by means of `Lawler's Algortithm` shown in Figure 13.1. A job $c \in J$ is said to be *critical* in schedule π under scenario S if c has the greatest lateness in π under S, that is

$$L_c(\pi, S) = \max_{i=1}^{n} L_i(\pi, S) = F(\pi, S).$$

The following proposition characterizes the worst case scenario for a given, feasible schedule π:

Proposition 13.1. *Let π be a feasible schedule and let S_π be a worst case scenario for π, in which some job c is critical. Then there exists a worst case scenario S_c for π such that under this scenario:*

1. *Job c is critical in π.*
2. *The processing times of all the jobs $i \in P(\pi, c)$ are equal to $\overline{p}_i$.*
3. *The processing times of all the jobs $i \notin P(\pi, c)$ are equal to $\underline{p}_i$.*
4. *The due date of the job c is equal to $\underline{d}_c$.*
5. *The due dates of all the jobs $i \in J \setminus \{c\}$ are equal to $\overline{d}_i$.*

Proof. Let S_π be a worst case scenario for a given schedule π and let c be a critical job in π under S_π. We will show that the worst case scenario S_π can be transformed into a worst case scenario S_c that fulfills all the conditions 1-5. Let us first transform scenario S_π into S_1 by replacing

- $p_i^{S_\pi}$ with $\underline{p}_i$ for all $i \notin P(\pi, c)$,
- $d_i^{S_\pi}$ with $\overline{d}_i$ for all $i \neq c$.

It is clear that $F(\pi, S_\pi) = L_c(\pi, S_\pi) = F(\pi, S_1) = L_c(\pi, S_1)$, so c is critical in π under S_1. Moreover $F^*(S_\pi) \geq F^*(S_1)$, since the maximal lateness cannot increase in any schedule if we increase due dates and decrease processing times. Consequently, we get

$$F(\pi, S_\pi) - F^*(S_\pi) \leq F(\pi, S_1) - F^*(S_1). \tag{13.3}$$

Inequality (13.3) implies that S_1 is also the worst case scenario for π. Notice that S_1 satisfies conditions 1, 3 and 5. Let us now transform S_1 into S_0 by replacing

- $p_i^{S_1}$ with $\overline{p}_i$ for all $i \in P(\pi, c)$,
- $d_c^{S_1}$ with $\underline{d}_c$.

Let $\Delta^p = \sum_{i \in P(\pi,c)} (\overline{p}_i - p_i^{S_1})$ and $\Delta^d = d_c^{S_1} - \underline{d}_c$, so $\Delta^p \geq 0$ denotes the total increase of the processing times in S_0 and $\Delta^d \geq 0$ denotes the decrease of the due date of c in S_0 in comparison with S_1. It is easy to verify that:

$$L_i(\pi, S_0) \leq L_i(\pi, S_1) + \Delta^p \text{ for all } i \neq c, \tag{13.4}$$

$$L_c(\pi, S_0) = L_c(\pi, S_1) + \Delta^p + \Delta^d. \tag{13.5}$$

Conditions (13.4) and (13.5) imply that c is critical in π under S_0 (since c is critical in π under S_1). It means that scenario S_0 satisfies all the conditions 1 - 5. It remains to be shown that S_0 is the worst case scenario for π. From (13.5) we get

$$F(\pi, S_0) - F(\pi, S_1) = L_c(\pi, S_0) - L_c(\pi, S_1) = \Delta^p + \Delta^d. \tag{13.6}$$

Let σ be a feasible schedule such that $F^*(S_1) = F(\sigma, S_1)$, that is σ is an optimal schedule in scenario S_1. It holds

$$F^*(S_0) - F^*(S_1) \leq F(\sigma, S_0) - F(\sigma, S_1) \leq \Delta^p + \Delta^d. \tag{13.7}$$

Condition (13.7) results from the fact that the maximal lateness in σ cannot increase by more than $\Delta^p + \Delta^d$ in S_0 in comparison with S_1. From (13.6) and (13.7) we obtain

$$F(\pi, S_1) - F^*(S_1) \leq F(\pi, S_0) - F^*(S_0),$$

hence S_0 is also the worst case scenario for π. The proof is completed since $S_0 = S_c$ is the worst case scenario that satisfies all the conditions 1-5. □

We show now how to apply Proposition 13.1 to compute the maximal regret of a given schedule. Let $D \subseteq J$ be a subset of jobs and let S_j^D, $j \in J$, be a scenario such that

1. the processing times of all the jobs $i \in D$ are equal to $\overline{p}_i$;
2. the processing times of all the jobs $i \notin D$ are equal to $\underline{p}_i$;
3. the due date of the job j is equal to $\underline{d}_j$;
4. the due dates of all the jobs $i \in J \setminus \{j\}$ are equal to $\overline{d}_i$;

Let us define $\overline{C}_j(\pi) = \sum_{i \in P(\pi,j)} \overline{p}_i$, $j \in J$. The following proposition shows how to compute the value of the maximal regret of a given schedule π:

Proposition 13.2. *For any feasible schedule π it holds*

$$Z(\pi) = \max_{j \in J} \{\overline{C}_j(\pi) - (\underline{d}_j + F^*(S_j^{P(\pi,j)}))\}. \tag{13.8}$$

Proof. Consider a feasible schedule π with a worst case scenario S_π and let c be a critical job in π under S_π. Then there exists a worst case scenario S_c that satisfies all conditions 1-5 of Proposition 13.1. Note that $S_c = S_c^{P(\pi,c)}$ and $\overline{C}_c(\pi)$ is the completion time of job c in π under scenario S_c. Since c is critical in π under S_c, we obtain

$$F(\pi, S_c) = L_c(\pi, S_c) = \overline{C}_c(\pi) - \underline{d}_c$$

and, from the fact that $S_c = S_c^{P(\pi,c)}$ is a worst case scenario for π, we get

$$Z(\pi) = \overline{C}_c(\pi) - (\underline{d}_c + F^*(S_c^{P(\pi,c)})). \tag{13.9}$$

From (13.9) it follows that

$$Z(\pi) \leq \max_{j \in J} \{\overline{C}_j(\pi) - (\underline{d}_j + F^*(S_j^{P(\pi,j)}))\}. \tag{13.10}$$

Suppose, by contradiction, that there exists a job $k \in J$ such that

$$Z(\pi) < \overline{C}_k(\pi) - \underline{d}_k - F^*(S_k^{P(\pi,k)}). \tag{13.11}$$

Since $F(S_k^{P(\pi,k)}, \pi) \geq \overline{C}_k(\pi) - \underline{d}_k$, it results from (13.11) that

$$Z(\pi) < F(S_k^{P(\pi,k)}, \pi) - F^*(S_k^{P(\pi,k)}),$$

which contradicts the definition of $Z(\pi)$. Therefore, the equality in (13.10) must hold and the proof is completed. □

`Algorithm REG-LMAX`
Require: $n, (\tilde{p}_i)_{i=1}^n, (\tilde{d}_i)_{i=1}^n, prec.$
Ensure: The optimal robust schedule π.
1: $D \leftarrow \{1, \ldots, n\}$
2: **for** $r \leftarrow n$ **downto** 1 **do**
3: Find $i \in D$, which has no successor in D and has a maximal value of $\underline{d}_i + F^*(S_i^D)$
4: $\pi(r) \leftarrow i$
5: $D \leftarrow D \setminus \{i\}$
6: **end for**
7: **return** π

Fig. 13.2. The algorithm for solving MINMAX REGRET $1|prec|L_{max}$

Using formula (13.8) it is possible to calculate the value of the maximal regret $Z(\pi)$ for a given feasible schedule π in $\mathcal{O}(n^3)$ time. The values of $F^*(S_j^{P(\pi,j)})$ for all $j \in J$ can be computed by means of `Lawler's Algorithm`. Let us denote

$$f_j(\pi) = \overline{C}_j(\pi) - (\underline{d}_j + F^*(S_j^{P(\pi,j)}),\ j \in J. \tag{13.12}$$

Thus $Z(\pi) = \max_{j \in J} f_j(\pi)$.

Consider now an algorithm presented in Figure 13.2. We will show that this algorithm returns an optimal robust schedule for MINMAX REGRET $1|prec|L_{max}$.

Theorem 13.3. `Algorithm REG-LMAX` *returns an optimal robust schedule in* $\mathcal{O}(n^4)$ *time.*

Proof. It is easy to check that the schedule constructed by the algorithm is feasible. Let π be the schedule constructed by the algorithm and let σ be an optimal schedule with respect to the maximal regret criterion. Let r be the last position in which σ and π differ, i.e. $\pi(i) = \sigma(i)$ for $i = r+1, \ldots, n$ and $\pi(r) \neq \sigma(r)$. We chose σ such that number r is minimal. If $r = 0$ then $\sigma = \pi$ and the proof is completed since π is also the optimal robust schedule. Assume that $r > 0$. Let us create a new schedule σ' by moving job $\sigma(s) = \pi(r)$ just after job $\sigma(r)$ in σ. Relations between schedules π, σ and σ' are presented in Figure 13.3. Let us also introduce the subsets of jobs E, F, G, H shown in Figure 13.3. The schedule σ' is feasible since $\sigma(s) = \pi(r)$ has no successor in set $E = J \setminus F$.

Let $k \in J$ be a job such that $Z(\sigma') = f_k(\sigma') = \max_{j \in J} f_j(\sigma')$. Consider three cases:

1. Let $k \in F \cup G$. Then it is easy to check that
$$Z(\sigma') = f_k(\sigma') = f_k(\sigma) \leq Z(\sigma).$$
It results from the fact that $\overline{C}_k(\sigma) = \overline{C}_k(\sigma')$ and $P(\sigma, k) = P(\sigma', k)$ (see (13.12)). We conclude that σ' is also an optimal robust schedule.
2. Let $k = \sigma(s) = \pi(r)$. It holds $P(\pi, k) = P(\sigma, \sigma(r)) = P(\sigma', k) = E$. Since π is constructed by the algorithm (see line 3), we obtain
$$\underline{d}_k + F^*(S_k^E) \geq \underline{d}_{\sigma(r)} + F^*(S_{\sigma(r)}^E).$$

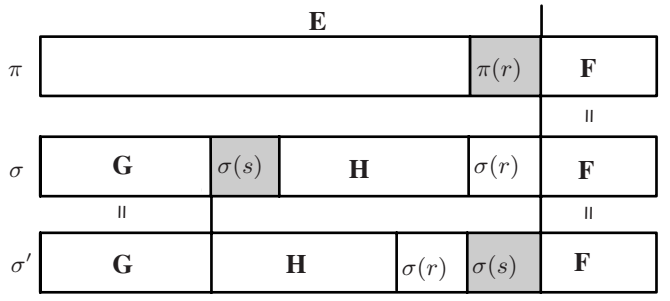


Fig. 13.3. Relations between schedules π, σ and σ'. It holds $\pi(r) = \sigma(s)$.

Moreover, $\overline{C}_{\sigma(r)}(\sigma) = \overline{C}_{\sigma(s)}(\sigma') = \overline{C}_k(\sigma')$. Thus we have

$$\overline{C}_k(\sigma') - (\underline{d}_k + F^*(S_k^E)) \leq \overline{C}_{\sigma(r)}(\sigma) - (\underline{d}_{\sigma(r)} + F^*(S_{\sigma(r)}^E)),$$

which means that $Z(\sigma') = f_k(\sigma') \leq f_{\sigma(r)}(\sigma) \leq Z(\sigma)$ and σ' is also an optimal robust schedule.

3. Let $k \in H \cup \{\sigma(r)\}$. It holds

$$\overline{C}_k(\sigma) - \overline{C}_k(\sigma') = \overline{p}_{\sigma(s)}. \tag{13.13}$$

Consider scenarios $S^1 = S_k^{P(\sigma,k)}$ and $S^2 = S_k^{P(\sigma',k)}$. Let us notice that $P(\sigma', k) = P(\sigma, k) \setminus \{\sigma(s)\}$. Thus S^1 can be transformed into S^2 by replacing $\overline{p}_{\sigma(s)}$ with $\underline{p}_{\sigma(s)}$. Let ρ be a sequence such that $F^*(S^2) = F(S^2, \rho)$. Then it holds

$$F^*(S^1) - F^*(S^2) \leq F(S^1, \rho) - F(S^2, \rho) \leq \overline{p}_{\sigma(s)} - \underline{p}_{\sigma(s)} \leq \overline{p}_{\sigma(s)}. \tag{13.14}$$

Using (13.13) and (13.14) we obtain

$$\overline{C}_k(\sigma) - \overline{C}_k(\sigma') \geq F^*(S^1) - F^*(S^2),$$

which implies

$$\overline{C}_k(\sigma) - (\underline{d}_k + F^*(S^1)) \geq \overline{C}_k(\sigma') - (\underline{d}_k + F^*(S^2))$$

thus, from definitions of S^1 and S^2 we get $Z(\sigma') = f_k(\sigma') \leq f_k(\sigma) \leq Z(\sigma)$, which means that σ' is also an optimal robust schedule.

From the cases 1 - 3 we obtain that σ' is an optimal robust schedule, which contradicts the minimality of r. It is easy to check that the running time of the algorithm is bounded by $\mathcal{O}(n^4)$. It follows from the fact that line 3 of the algorithm requires $\mathcal{O}(n^3)$ time and it is executed n times. □

`Algorithm REG-LMAX` is a generalization of `Lawler's Algorithm`. The only difference appears in line 4, in which the due date of every job $i \in D$ is modified

by the value of $F^*(S_i^D)$. This allows us to derive an additional conclusion. Suppose that all due dates are precisely known, that is $\underline{d}_i = \overline{d}_i = d_i$ for all $i \in J$. Observe that the value of $F^*(S_i^D)$ in line 4 of the algorithm is now the same for all $i \in D$. Therefore, the algorithm chooses a job in D of the largest value of d_i. Thus in this case the uncertain processing times are not involved and the optimal robust schedule can be obtained in $\mathcal{O}(n^2)$ time by means of `Lawler's Algorithm`. Moreover, if there are no precedence constraints between jobs, then we obtain the optimal robust schedule by applying the simple EDD rule.

13.3 Notes and References

The deterministic sequencing problem $1|prec|L_{max}$ is a special version of the more general $1|prec|f_{max}$ problem, in which $F(\pi) = \max_{i \in J} f_i(C_i(\pi))$ and all cost functions f_i are nondecreasing. Provided that all functions $f_i(t)$ are computable in constant time for a given t, the $1|prec|f_{max}$ problem can be solved in $\mathcal{O}(n^2)$ time by an algorithm developed by Lawler [91]. The proof of correctness of Lawler's algorithm can also be found in the book by Brucker [29]. The algorithm shown in Figure 13.1 is Lawler's algorithm applied to the particular deterministic $1|prec|L_{max}$ problem.

The polynomial algorithm for the minmax regret version of $1|prec|L_{max}$, shown in Figure 13.2, was designed by Kasperski [75].

Problem $1|prec|L_{max}$ is one of the simplest sequencing problems with a bottleneck objective function. There are some generalizations and modifications of this problem, which are also polynomially solvable in the deterministic case. For instance, we can introduce a nonnegative weight w_i for every job and minimize the maximum *weighted lateness*. However, the corresponding Minmax Regret $1|prec|\max w_i L_i$ problem is open even if all the weights are deterministic.

Another modification consists in replacing the lateness criterion with a *tardiness* one. This criterion, for a given scenario S, is defined as follows:

$$T_i(\pi, S) = \max\{0, L_i(\pi, S)\} = \max\{0, C_i(\pi, S) - d_i^S\}.$$

In the deterministic case, problem $1|prec|T_{max}$ can be solved by exactly the same algorithm as problem $1|prec|L_{max}$. However, it is not clear that `Algorithm REG-LMAX` can be applied to Minmax Regret $1|prec|T_{max}$. Finally, we can introduce a nonnegative weight w_i for every job and minimize the maximum *weighted tardiness*. The minmax regret version of this problem was discussed by Averbakh [15], who assumed that all processing times and all due dates are precisely known and only the weights are imprecise. In this case the Minmax Regret $1|prec|\max w_i T_i$ problem can be solved in polynomial time be applying a similar technique as to the combinatorial optimization problems with the bottleneck objective function (see Section 1.3.1). For details we refer the reader to [15]. However, it would be interesting to explore this problem for uncertain processing times and uncertain due dates.

14 Sequencing Problem with Weighted Number of Late Jobs

In this chapter we discuss MINMAX REGRET $1|p_i = 1|\sum w_i U_i$. In this problem all jobs have unit processing times, that is $p_i = 1$ for all $i \in J$, and all jobs have precise due dates d_i, $i \in J$. We also assume that there are no precedence constraints between jobs. For every job $i \in J$ there is given an interval weight $\tilde{w}_i = [\underline{w}_i, \overline{w}_i]$. Thus a scenario is a particular realization of the uncertain weights. A job $i \in J$ is said to be *late* in schedule π if $C_i(\pi) = |P(\pi, i)| > d_i$; otherwise job i is said to be *on-time* in π. Notice that the fact of being late or not does not depend on a particular scenario, since all processing times and all due dates in the problem are precise. Let us define

$$U_i(\pi) = \begin{cases} 1 \text{ if } C_i(\pi) > d_i \\ 0 \text{ if } C_i(\pi) \leq d_i \end{cases}$$

Hence $U_i(\pi) = 1$ if and only if job i is late in π. The cost of a given schedule π under scenario S is expressed as follows:

$$F(\pi, S) = \sum_{i=1}^{n} w_i^S U_i(\pi).$$

So, the cost is the weighted number of late jobs in π under S. In the MINMAX REGRET $1|p_i = 1|\sum w_i U_i$ problem we seek a schedule which minimizes the maximal regret.

If scenario S is fixed, then the optimal schedule under this scenario can be constructed in $\mathcal{O}(n^2)$ time by a version of the greedy algorithm. It follows from the matroidal structure of the problem, which is discussed in Section 14.1. It turns out that both the deterministic problem and its minmax regret version can be reduced to a certain matroidal combinatorial optimization problem. This matroidal problem is a generalization of the polynomially solvable MINMAX REGRET MINIMUM SELECTING ITEMS, which was discussed in Chapter 5. However, the complexity status of MINMAX REGRET $1|p_i = 1|\sum w_i U_i$ is unknown. In Section 14.2 we show an efficient method of preprocessing the problem. Finally, in

A. Kasperski: Discrete Optimization with Interval Data, STUDFUZZ 228, pp. 175–182, 2008.
springerlink.com

Section 14.3 we construct a MIP model for solving the problem and we present the results of some computational tests.

14.1 Matroidal Structure of the Problem

It turns out that in order to solve the problem it is enough to consider only some particular schedules, namely the *canonical* ones. A schedule π is said to be in a *canonical form* if all on-time jobs are processed before all late jobs in π and all on-time jobs are ordered with respect to nondecreasing due dates. In other words, in a canonical schedule we first process all on-time jobs in order of nondecreasing due dates, and we process then all the remaining (late) jobs in any order. The following proposition characterizes the canonical schedules:

Proposition 14.1. *For every schedule π there exists a canonical schedule π_1 such that $F(\pi_1, S) \leq F(\pi, S)$ for all scenarios $S \in \Gamma$.*

Proof. If π is a canonical schedule, then we are done. If not, denote by O the set of on-time jobs in π and transform π into π_1 in the following way: move all late jobs in π after the last on-time job in π and order the on-time jobs with respect to nondecreasing due dates. It is clear that the resulting schedule π_1 is canonical and no job in O becomes late in π_1. In consequence, $F(\pi_1, S) \leq F(\pi, S)$ for all scenarios $S \in \Gamma$. □

We now show that it is enough to consider only canonical schedules while solving the minmax regret version of the problem.

Proposition 14.2. *There exists an optimal canonical robust schedule.*

Proof. From Proposition 14.1 it follows that for every schedule π there is a canonical schedule π_1 such that $F(\pi_1, S) \leq F(\pi, S)$ for all $S \in \Gamma$. Hence

$$Z(\pi) = \max_{S \in \Gamma} \{F(S, \pi) - F^*(S)\} \geq \max_{S \in \Gamma} \{F(S, \pi_1) - F^*(S)\} = Z(\pi_1)$$

and there must be an optimal robust schedule which is canonical. □

Let us now define set $\mathcal{I}$ consisting of all subsets of jobs $A \subseteq J$, such that $A \in \mathcal{I}$ if and only if all jobs from A are on-time in a certain schedule π. It is not difficult to decide whether $A \in \mathcal{I}$. We can proceed as follows: schedule the jobs in A in order of nondecreasing due dates; if at least one job becomes late then $A \notin \mathcal{I}$, otherwise all the jobs in A are on-time and $A \in \mathcal{I}$. It turns out that system $(J, \mathcal{I})$ is a matroid (see e.g. [38]) and we will call it a *sequencing matroid*. A base of this matroid is a maximal subset of jobs that are on-time in a certain schedule π. We will call this base a *maximal subset of on-time jobs*. It is easily seen that every base $I \in \mathcal{I}$ corresponds to a canonical schedule in which all jobs from I are on-time and all the remaining jobs are late. Moreover, if I is a base with the total weight of $\sum_{i \in I} w_i^S$ under scenario S, then the corresponding canonical schedule π for I has the cost

$$\sum_{i \in J} w_i^S - \sum_{i \in I} w_i^S = \sum_{i \in J} w_i^S U_i(\pi).$$

`Greedy Algorithm for` $1|p_i = 1|\sum w_i U_i$
Require: $n, (d_i)_{i=1}^n, (w_i)_{i=1}^n$.
Ensure: The optimal schedule π.
1: Number jobs so that $w_1 \geq w_2 \geq \cdots \geq w_n$
2: $I \leftarrow \emptyset$
3: **for** $i \leftarrow 1$ **to** n **do**
4: **if** all jobs from $I \cup \{i\}$ can be scheduled on-time **then** $I \leftarrow I \cup \{i\}$
5: **end for**
6: **return** the canonical schedule π for I

Fig. 14.1. A greedy algorithm for solving the deterministic problem $1|p_1 = 1|\sum w_i U_i$

Hence if I is the base of the maximal total weight under S, then π is the optimal schedule under S. Since $(J, \mathcal{I})$ is a matroid, the maximal weighted base under a given scenario S can be obtained by applying a greedy algorithm shown in Figure 14.1. This algorithm greedily constructs a maximal subset of on time jobs I of the maximal total weight and returns the canonical schedule for I. Its direct implementation runs in $\mathcal{O}(n^2)$ time and it can be additionally refined (see [38] for details).

Consider now a minmax regret combinatorial optimization problem, denoted by Minmax Regret J, which is associated with Minmax Regret $1|p_i = 1|\sum w_i U_i$. This problem is defined as follows: the set of elements E is equal to J and the set of feasible solutions Φ consists of all bases of the sequencing matroid $(J, \mathcal{I})$. The interval weights of elements (jobs) in J are $[M - \overline{w}_i, M - \underline{w}_i]$ for all $i \in J$, where $M = \max_{i \in J} \overline{w}_i$. We seek a solution (base) that minimizes the maximal regret. The following proposition is true:

Proposition 14.3. *If I is the optimal robust solution in* Minmax Regret J*, then the canonical schedule π that corresponds to I is the optimal robust schedule.*

Proof. It is easy to see that there is an optimal robust schedule π in which the set of all on-time jobs I is maximal. Otherwise, there is schedule π_1 such that $F(\pi_1, S) \leq F(\pi, S)$ for all scenarios S and $Z(\pi_1) \leq Z(\pi)$. Thus I is a solution in Minmax Regret J. On the other hand, every solution I in Minmax Regret J corresponds to a schedule π in which all jobs in I are on time. Let S_π be a worst case scenario and let π^* be a worst case alternative for π. We denote by I^* the set of on-time jobs in π^* and we can also assume that set I^* is a maximal subset of on-time jobs. It holds:

$$Z(\pi) = F(\pi, S_\pi) - F(\pi^*, S_\pi) = (\sum_{i \in J} w_i^{S_\pi} - \sum_{i \in I} w_i^{S_\pi}) - (\sum_{i \in J} w_i^{S_\pi} - \sum_{i \in I^*} w_i^{S_\pi}) =$$

$$= \sum_{i \in I^* \setminus I} w_i^{S_\pi} - \sum_{i \in I \setminus I*} w_i^{S_\pi} = \max_{O \in \Phi} \left\{ \sum_{i \in O \setminus I} \overline{w}_i - \sum_{i \in I \setminus O} \underline{w}_i \right\}.$$

The last equation follows from the fact that S_π is the worst case scenario for π and Φ is the set of all maximal subsets of on-time jobs (recall that every such subset corresponds to a canonical schedule and vice versa). Now I is a feasible solution in Minmax Regret J. Applying formula (1.6) we obtain:

$$Z(I) = \max_{O \in \Phi} \left\{ \sum_{i \in I \setminus O} (M - \underline{w}_i) - \sum_{i \in O \setminus I} (M - \overline{w}_i) \right\}. \tag{14.1}$$

Since all solutions from Φ have the same cardinality (because they are bases of a matroid), we can rewrite (14.1) as follows:

$$Z(I) = \max_{O \in \Phi} \left\{ \sum_{i \in O \setminus I} \overline{w}_i - \sum_{i \in I \setminus O} \underline{w}_i \right\} = Z(\pi). \tag{14.2}$$

Therefore if I is the optimal robust solution for Minmax Regret J, then the corresponding canonical schedule π is the optimal robust schedule in the corresponding sequencing problem. □

In order to solve Minmax Regret $1|p_i = 1|\sum w_i U_i$ it is enough to solve the corresponding Minmax Regret J problem. Observe that if all jobs have the same integer due date, that is $d_1 = d_2 = \cdots = d_n$, then Minmax Regret J is equivalent to Minmax Regret Minimum Selecting Items. It follows from the fact, that in this case the sequencing matroid is a uniform one. In this particular case the problem can be solved in polynomial time by means of the algorithm designed in Section 5.2. However, the complexity status of the problem with arbitrary due dates is unknown and we conjecture that it is NP-hard.

Since Minmax Regret J is a combinatorial optimization problem whose deterministic counterpart is polynomially solvable, we can apply 2-approximation algorithms constructed in Chapter 4 to obtain an approximate solution for this problem. In consequence, we also obtain an approximate solution to the corresponding sequencing problem. In the next two sections we will show some additional consequences of the matroidal structure of the problem.

14.2 Possibly and Necessarily Optimal Jobs

From the fact that Minmax Regret J has a matroidal structure it follows that we can detect in polynomial time all possibly and necessarily optimal jobs in this problem. A nonpossibly optimal job cannot be a part of the optimal robust solution in Minmax Regret J and, consequently, it cannot be on-time in any optimal robust schedule. Similarly, a necessarily optimal job is always a part of an optimal robust solution for Minmax Regret J and it is on-time in an optimal robust schedule. Applying the results from Section 2.2.3 we can see that all possibly optimal jobs can be detected in $\mathcal{O}(n^2)$ time and all necessarily optimal ones can be detected in $\mathcal{O}(n^3)$ time. We can perform then the preprocessing described in Section 2.2 to reduce the problem size.

Table 14.1. The numer of nonpossibly and necessarily optimal jobs

	Nonposs.				Nec.			
n	Min	Max	Aver.	Perc.	Min	Max	Aver.	Perc.
20	0	7	3.3	16.66%	1	7	4.27	21.35%
50	3	11	7.3	14.6%	6	16	9.73	19.46%
100	8	21	14.56	14.56%	12	25	18.93	18.93%
150	10	34	20.78	13.85%	21	35	28.85	19.2%
200	15	31	24.86	12.43%	32	46	37	18.5%

In order to estimate the number of nonpossibly and necessarily optimal jobs that may appear in the problem, we performed some computational tests. We used the family of problems denoted as $S(n, c)$, where:

- n is the number of jobs;
- for every job $i \in J$ the value of $\overline{w}_i$ is a randomly selected integer from interval $[1, c]$ and the value of $\underline{w}_i$ is a randomly selected integer from interval $[0, \overline{w}_i)$;
- for every job $i \in J$ the due date d_i is a randomly selected integer from interval $[0, n/2]$.

We chose $n \in \{20, 50, 100, 150, 200\}$ and $c \in \{20, 50, 100\}$. For every combination of n and c we generated randomly 5 instances. The obtained results are shown in Table 14.1. For every value of n the minimal, maximal and average number of nonpossibly and necessarily optimal jobs are presented. The average number is also expressed as the percent of the number of jobs.

Observe that the average number of nonpossibly optimal jobs is 12% - 17% of the total number of jobs. The average number of necessarily optimal jobs is even larger and it is 18% - 22% of the total number of jobs. Hence after detecting the nonpossibly and necessarily optimal jobs we may obtain a lot of information about the structure of an optimal robust solution.

14.3 Mixed Integer Programming Formulation

In the previous section we have shown that after solving the Minmax Regret J problem we get the optimal robust schedule for the corresponding minmax regret sequencing problem. Since Minmax Regret J is a minmax regret combinatorial optimization problem, we can use the framework designed in Section 3.1 to construct a MIP model to solve it.

Let us introduce a binary variable $x_i \in \{0, 1\}$ for every job $i \in J$. This variable will express whether job i is a part of the constructed solution. Recall that this solution is a maximal subset of on-time jobs. Assume that the jobs in J are numbered so that $d_1 \leq d_2 \leq \cdots \leq d_n$. Denote by p the cardinality of a solution from Φ. The value of p is unique since all solutions from Φ have the same cardinality. The value of p can be obtained in $\mathcal{O}(n^2)$ time by applying `Greedy Algorithm for` $1|p_i = 1|\sum w_i U_i$ for any scenario S. Consider the following set of constraints:

$$\begin{aligned}
& x_1 \le d_1 \\
& x_1 + x_2 \le d_2 \\
& \cdots \\
& x_1 + x_2 + \cdots + x_n \le d_n \\
& x_1 + x_2 + \cdots + x_n = p \\
& x_i \in \{0,1\} \qquad\qquad \text{for } i \in J
\end{aligned} \tag{14.3}$$

It is not difficult to see that constraints (14.3) describe set $ch(\Phi)$, where Φ is the set of all feasible solutions in MINMAX REGRET J (see Section 3.1 for the definition of $ch(\Phi)$). Suppose that (x_i) is a characteristic vector of a feasible solution $I \in \Phi$. Since $|I| = p$, the equality $x_1 + \cdots + x_n = p$ is satisfied. Moreover, all jobs from I are on-time in a canonical schedule π that corresponds to I. In this schedule all jobs $\pi(1), \ldots, \pi(k)$ are on time and they are processed according to nondecreasing due dates. Hence $x_{\pi(1)} + \cdots + x_{\pi(j)} \le d_{\pi(j)}$ for $j = 1, \ldots k$. Since $x_{\pi(k+1)} = \cdots = x_{\pi(n)} = 0$, we can see that the characteristic vector (x_i) satisfies constraints (14.3).

Conversely, suppose that (x_i) satisfies (14.3). Then (x_i) contains precisely p variables which take the value of 1. These variables correspond to a subset of jobs I. Since all jobs have processing times equal to 1, it is obvious that all jobs from I can be scheduled on-time in a certain schedule. Since $|I| = p$ we get that I is a maximal subset of on-time jobs. Thus I is a base of the sequencing matroid and (x_i) is a characteristic vector of solution $I \in \Phi$.

One can easily verify that the constraints matrix $\boldsymbol{A}$ of (14.3) is totally unimodular. Hence we can construct the relaxed subproblem $\min_{\boldsymbol{y} \in ch(\Phi)} \phi(\boldsymbol{x}, \boldsymbol{y})$:

$$\begin{aligned}
& \min \sum_{i=1}^{n} (\overline{w}_i x_i + \underline{w}_i (1 - x_i)) y_i \\
& \sum_{i=1}^{j} -y_i \ge -d_i \qquad\qquad \text{for } j \in J \\
& y_1 + y_2 + \cdots + y_n = p \\
& -y_i \ge -1 \qquad\qquad \text{for } i \in J \\
& y_i \ge 0 \qquad\qquad \text{for } i \in J
\end{aligned} \tag{14.4}$$

We have multiplied some inequalities by -1 to obtain a canonical form of the linear program. The dual to (14.4), that is the problem $\max_{\boldsymbol{\lambda}} \phi^*(\boldsymbol{x}, \boldsymbol{\lambda})$, is as follows:

$$\begin{aligned}
& \max - \sum_{i=1}^{n} d_i \alpha_i - \sum_{i=1}^{n} \beta_i + p\gamma \\
& - \sum_{i=j}^{n} \alpha_i - \beta_j + \gamma \le \overline{w}_i x_i + \underline{w}_i (1 - x_i) \text{ for } j \in J \\
& \alpha_i, \beta_i \ge 0 \qquad\qquad \text{for } i \in J
\end{aligned} \tag{14.5}$$

From (3.6), (14.3) and (14.5) the final MIP model is:

$$\begin{aligned}
&\min \sum_{i=1}^{n} \overline{w}_i x_i + \sum_{i=1}^{n} d_i \alpha_i + \sum_{i=1}^{n} \beta_i - p\gamma \\
&\sum_{i=1}^{j} x_i \le d_i && \text{for } j \in J \\
&x_1 + x_2 + \cdots + x_n = p \\
&-\sum_{i=j}^{n} \alpha_i - \beta_j + \gamma \le \overline{w}_j x_j + \underline{w}_j (1 - x_j) && \text{for } j \in J \\
&\alpha_i, \beta_i \ge 0 && \text{for } i \in J \\
&x_i \in \{0, 1\} && \text{for } i \in J
\end{aligned} \tag{14.6}$$

Using the matroidal structure of the problem we can first detect all nonpossibly optimal jobs and remove all the variables that correspond to them from the model. If all weights are specified as nondegenerate intervals, that is $\underline{w}_i < \overline{w}_i$ for all $i \in J$, then we can set $x_i = 1$ for all necessarily optimal jobs $i \in J$. This preprocessing may significantly reduce the time required to solve the MIP model.

14.3.1 Computational Results

In this section we present the results of some computational tests. We used the MIP formulation constructed in the previous section. In order to obtain approximate solutions to MINMAX REGRET J we applied 2-approximation algorithms `Algorithm AM` and `Algorithm AMU` designed in Section 4.1. In our test we used the same family of problems as in Section 14.2. All instances were first preprocessed before solving. The obtained results are shown in Table 14.2. In order to solve the MIP model CPLEX 8.0 and a Pentium III 866MHz computer were used. In Table 14.2 the average CPU time in seconds required to solve the MIP model and the maximal and average percentage deviations from optimum reported for Algorithm `AM` and `Algorithm AMU` are presented.

Most of the tested instances were solved by CPLEX within a few seconds. However, some instances appeared with 200 jobs which required more CPU time. The results in Table 14.2 indicate that the behavior of both MIP approach

Table 14.2. The computational results

		`Algorithm AM`		`Algorithm AMU`	
n	CPU time	Worst %	Aver. %	Worst %	Aver. %
20	0.020	11.11	1.42	1.11	1.22
50	1.145	8.76	1.94	1.87	1.76
100	2.116	3.68	1.01	3.68	1.01
150	3.737	5.26	0.93	5.26	0.93
200	218.68	3.31	0.80	3.31	0.80

and the approximation algorithms is reasonably good. The error incurred in approximating the optimal solution averaged about 1%-2% and the maximal reported deviation was 11.11%.

14.4 Notes and References

The deterministic problem $1|p_i = 1|\sum w_i U_i$ and its matroidal structure were discussed by Lawler [92], Horowitz and Sahni [62] and Brassard and Bratley [28]. The greedy algorithm for this problem was also presented in the book by Cormen *et al.* [38]. The problem $1||\sum w_i U_i$, that is a generalization with arbitrary processing times of jobs, is NP-hard [72] but it is solvable in pseudopolynomial time. The problem with unit processing times becomes strongly NP-hard if some arbitrary precedence constraints between jobs may be specified [96].

It turns out that the Minmax Regret $1|p_i = 1|\sum w_i U_i$ problem can be viewed as a generalization of Minmax Regret Minimum Selecting Items, which is polynomially solvable [16, 37]. However, the complexity of the more general sequencing problem is open. We conjecture that it is NP-hard but it may be solvable in pseudopolynomial time and it may admit an FPTAS as well. It would also be interesting to explore the case in which the due dates are imprecise. In this case the problem becomes more complex since the fact of being on-time by a job now depends on a realization of due dates and the matroidal structure of the problem cannot be used immediately.

One of the most basic deterministic sequencing problems is $1||\sum U_i$ in which for every job $i \in J$ there is given a processing time p_i and a due date d_i. There are no precedence constraints between jobs. We seek a schedule that minimizes the number of late jobs, that is the value of $\sum_{i \in J} U_i(\pi)$. The problem $1||\sum U_i$ is polynomially solvable and an algorithm with running time $\mathcal{O}(n \log n)$ for this problem was designed by Moor [101]. Consider the minmax regret version of this problem in which the processing times of jobs are uncertain. This problem is interesting because it is possible to give an example in which no extreme scenario is the worst case scenario for a given schedule. To see this consider a problem with two jobs and the following parameters: $\tilde{p}_1 = [10, 10]$, $\tilde{p}_2 = [1, 20]$, $d_1 = 100$, $d_2 = 12$. There are precisely two extreme scenarios in this simple problem. In the first one we have $p_1^{S_1} = 10$ and $p_2^{S_1} = 1$ and in the second one $p_1^{S_2} = 10$ and $p_2^{S_2} = 20$. Consider schedule $\pi = (1, 2)$. It is easy to check that $F(\pi, S_1) - F^*(S_1) = 0$ and $F(\pi, S_2) - F^*(S_2) = 0$. However, under scenario S such that $p_1^S = 10$ and $p_2^S = 3$ it holds $F(\pi, S) - F^*(S) = 1$. Thus, no worst case scenario for π is an extreme scenario. No results concerning the Minmax Regret $1||\sum U_i$ problem are known. This problem is interesting even if all jobs have the same due date.

15 Sequencing Problem with the Total Flow Time Criterion

This chapter is devoted to MINMAX REGRET $1||\sum C_i$. In this problem for every job $i \in J$ an interval processing time $\tilde{p}_i = [\underline{p}_i, \overline{p}_i]$ is given. Thus every scenario is a particular realization of the processing times. It is also assumed that there are no precedence constraints between jobs. The cost of a given schedule π under scenario S is the following:

$$F(\pi, S) = \sum_{i \in J} C_i(\pi, S),$$

so it is the *total flow time* in π under S. For a fixed scenario S, the optimal schedule can be found in $\mathcal{O}(n \log n)$ time by applying the following *shortest processing times* (SPT) rule: schedule the jobs in order of nondecreasing processing times under scenario S.

It has recently been proved [94] that MINMAX REGRET $1||\sum C_i$ is NP-hard. Hence the minmax regret version of this very basic sequencing problem is computationally intractable. We present this result, together with some additional properties of the problem in Section 15.2. Earlier, in Section 15.1, we show how to compute the worst case scenario for a given schedule π. Following Kouvelis and Yu [87], we show that this can be done in polynomial time by solving an assignment problem. In Section 15.3 we construct a simple 2-approximation algorithm for the problem, which returns a schedule σ such that $Z(\sigma) \leq 2OPT$. In Section 15.4 we propose a local search algorithm which is based on a natural neighborhood structure. Finally, in Section 15.5 we present a mixed integer programming formulation, which is according to Montemanni [106]. We also present the results of some computational experiments, which compare the MIP formulation to the performance of the 2-approximation and the local search algorithms.

Before we proceed we make an important observation. It is well known that the deterministic $1||\sum C_i$ problem is equivalent to the following MINIMUM ASSIGNMENT:

A. Kasperski: Discrete Optimization with Interval Data, STUDFUZZ 228, pp. 183–195,2008.
springerlink.com

$$\begin{array}{ll} \sum_{i=1}^{n}\sum_{j=1}^{n}(n-j+1)p_i x_{ij} & \\ \sum_{i=1}^{n} x_{ij} = 1 & j = 1,\ldots,n \\ \sum_{j=1}^{n} x_{ij} = 1 & i = 1,\ldots,n \\ x_{ij} \in \{0,1\} & \end{array} \tag{15.1}$$

where $x_{ij} = 1$ if job i occupies position j. Observe that the cost of assigning job i to position j or, equivalently, the weight of edge $\{i,j\}$ is $w_{\{i,j\}} = (n-j+1)p_i$. Thus one could think that some results presented in the first part of this monograph, in particular the MIP formulation and the 2-approximation algorithm, can be immediately applied to Minmax Regret $1||\sum C_i$. It is, however, not true because the weights in the resulting Minmax Regret Minimum Assignment problem are not independent. In other words, the weights $w_{\{i,j\}} = (n-j+1)p_i$, for $j = 1,\ldots,n$, must have the same value of $p_i \in [\underline{p}_i, \overline{p}_i]$ under all scenarios because they represent the same job. In consequence, we cannot use a very simple characterization of the worst case scenario, which is true for combinatorial optimization problems discussed in the first part of this monograph (see Proposition 1.1). In fact, as we will see in Section 15.1, computing the worst case scenario for a given schedule seems to be more complex.

The Minmax Regret $1||\sum C_i$ problem has an additional interesting property, which makes it different from the robust combinatorial optimization problems considered in the first part of this monograph. Recall that every optimal robust solution X to a minmax regret combinatorial optimization problem is optimal under an extreme scenario (this is precisely scenario S_X^-, see Section 2.1). This general result, which holds for all combinatorial optimization problems, is not true for the considered sequencing problem. Following Kouvelis and Yu [87], consider a sample problem in which $J = \{1,2,3\}$ and the processing times of jobs are specified as follows: $\tilde{p}_1 = [10,20]$, $\tilde{p}_2 = [5,50]$ and $\tilde{p}_3 = [48,49]$. The optimal robust schedule in this problem is $\pi = (1,2,3)$. Observe that under every extreme scenario job 2 occupies either the first or the third position, according to SPT rule. Moreover, under every extreme scenario all processing times are distinct. In consequence schedule $(1,2,3)$ is not optimal under any extreme scenario. However, it is not difficult to show that every optimal robust schedule is optimal under some scenario but not necessarily under an extreme one (see [87]).

15.1 Characterization of the Worst Case Scenario

In this section e show (following [39, 87]) how to compute the worst case scenario and the maximal regret of a given schedule π. We first show that for every schedule π there exists an extreme worst case scenario, that is the one in which all processing times takes the lower or upper bounds in their intervals. Let S_π be a worst case scenario and let π^* be a worst case alternative for π, that is π^* is the optimal schedule under S_π. The maximal regret of π can be expressed as follows:

$$Z(\pi) = F(\pi, S_\pi) - F(\pi^*, S_\pi) = \sum_{i \in J} C_i(\pi, S_\pi) - \sum_{i \in J} C_i(\pi^*, S_\pi). \tag{15.2}$$

Denote by i^* the position occupying by job $\pi(i)$ in schedule π^*. It holds

$$\sum_{i \in J} C_i(\pi, S_\pi) = \sum_{i \in J} (n - i + 1) p^{S_\pi}_{\pi(i)}, \tag{15.3}$$

$$\sum_{i \in J} C_i(\pi^*, S_\pi) = \sum_{i \in J} (n - i^* + 1) p^{S_\pi}_{\pi(i)}. \tag{15.4}$$

Subtracting (15.4) from (15.3), together with (15.2) yields

$$Z(\pi) = \sum_{i \in J} (i^* - i) p^{S_\pi}_{\pi(i)}. \tag{15.5}$$

We now see that if $i^* < i$, then we can decrease the processing time of $\pi(i)$ to $\underline{p}_{\pi(i)}$ and if $i^* > i$, then we can increase the processing time of $\pi(i)$ to $\overline{p}_{\pi(i)}$ obtaining in this way another worst case scenario for π, which is an extreme one. Hence the maximal regret of π can be expressed as follows:

$$Z(\pi) = \sum_{\{i: i^* < i\}} (i^* - i) \underline{p}_{\pi(i)} + \sum_{\{i: i^* > i\}} (i^* - i) \overline{p}_{\pi(i)}. \tag{15.6}$$

Observe that the extreme worst case scenario can be immediately obtained if we know a worst case alternative π^* for π. In order to compute π^*, we must find positions i^*, $i \in J$, that maximize the right hand side of (15.6). We now show that it can be done by solving an assignment problem.

Define binary variables $z_{ij} \in \{0, 1\}$, $i, j = 1, \ldots, n$. The variable z_{ij} takes the value of 1 if and only if $i^* = j$, that is job $\pi(i)$ occupies position j in the worst case alternative π^*. Since every job occupies exactly one position and every position must be occupied by exactly one job, it is clear that variables z_{ij} must fulfill the following assignment constraints:

$$\begin{aligned} &\sum_{i=1}^{n} z_{ij} = 1 \ \text{ for } j = 1, \ldots, n \\ &\sum_{j=1}^{n} z_{ij} = 1 \ \text{ for } i = 1, \ldots, n \\ &z_{ij} \in \{0, 1\} \text{ for } i, j \in 1, \ldots, n \end{aligned} \tag{15.7}$$

Using (15.6) we can now express the maximal regret for π in the following way:

$$Z(\pi) = \max_{(z_{ij})} \sum_{i=1}^{n} \left(\sum_{j=1}^{i} z_{ij} (j - i) \underline{p}_{\pi(i)} + \sum_{j=i+1}^{n} z_{ij} (j - i) \overline{p}_{\pi(i)} \right), \tag{15.8}$$

where variables z_{ij} fulfill constraints (15.7). Observe that (15.8) can be rewritten as

$$Z(\pi) = \max_{(z_{ij})} \sum_{i=1}^{n} \sum_{j=1}^{n} c_{ij}^{\pi} z_{ij}, \tag{15.9}$$

where c_{ij}^{π} are fixed coefficients contained in the following matrix $[c_{ij}^{\pi}]_{n\times n}$:

$$[c_{ij}^{\pi}]_{n\times n} = \begin{bmatrix} 0 & \overline{p}_{\pi(1)} & 2\overline{p}_{\pi(1)} & 3\overline{p}_{\pi(1)} & \dots & (n-1)\overline{p}_{\pi(1)} \\ -\underline{p}_{\pi(2)} & 0 & \overline{p}_{\pi(2)} & 2\overline{p}_{\pi(2)} & \dots & (n-1)\overline{p}_{\pi(2)} \\ -2\underline{p}_{\pi(3)} & -\underline{p}_{\pi(3)} & 0 & \overline{p}_{\pi(3)} & \dots & (n-1)\overline{p}_{\pi(3)} \\ \dots & \dots & \dots & \dots & \dots & \dots \\ (1-n)\underline{p}_{\pi(n)} & (2-n)\underline{p}_{\pi(n)} & (3-n)\underline{p}_{\pi(n)} & (4-n)\underline{p}_{\pi(n)} & \dots & 0 \end{bmatrix}$$

Now combining the objective (15.9) with constraints (15.7) we can see that the worst case alternative π^* and the maximal regret for π can be obtained by solving an assignment problem. This assignment problem is polynomially solvable. For instance the well known Hungarian algorithm, that runs in $\mathcal{O}(n^3)$ time can be applied.

Observe that computing the value of $Z(\pi)$ is much more time consuming than computing the optimal schedule under a given scenario, which requires only $\mathcal{O}(n \log n)$ time. This makes the problem different from the minmax regret combinatorial optimization problems discussed in the first part of this monograph. Recall that for a minmax regret combinatorial optimization problem the time required for computing the value of the maximal regret is the same as the time required for computing the optimal solution under a fixed scenario.

15.2 Computational Complexity and Preprocessing Rules

The following theorem has recently been proved by Lebedev and Averbakh [94] (for the proof we refer the reader to [94]):

Theorem 15.1 ([94]). *The* MINMAX REGRET $1||\sum C_i$ *problem is NP-hard.*

The problem remains NP-hard even if all processing time intervals have equal midpoints, that is the values of $\frac{1}{2}(\underline{p}_i + \overline{p}_i)$, $i \in J$. However, in the proof of Theorem 15.1 a reduction from a variant of the PARTITION problem was applied. Therefore we do not know whether the problem is strongly NP-hard and it is possible that a pseudopolynomial algorithm for the problem exists.

We now recall an additional important result, which allows us to preprocess the problem before applying an algorithm for solving it. Let i and j be two jobs. We say that $i \preceq j$ if $\underline{p}_i \leq \underline{p}_j$ and $\overline{p}_i \leq \overline{p}_j$. It is easy to see that the relation $\preceq$ is transitive, that is $i \preceq j$ and $j \preceq k$ imply $i \preceq k$. The following proposition holds:

Proposition 15.2 ([87]). *Let π be a given schedule and let i and j be two jobs such that $i \preceq j$ and i is processed after j in π. Let us create a new schedule σ be interchanging jobs i and j in π. Then $Z(\sigma) \leq Z(\pi)$.*

From Proposition 15.2 we immediately get the following theorem:

Theorem 15.3 ([87]). *There exists an optimal robust schedule π, in which for every pair of jobs i and j, if $i \preceq j$, then i is processed before j in π.*

Using Theorem 15.3 we can immediately determine the relative positions in the optimal robust schedule for every pair of jobs i, j such that $i \preceq j$. In particular, if it is possible to order all jobs in J so that $\pi(1) \preceq \pi(2) \preceq \cdots \preceq \pi(n)$, then π is the optimal robust schedule. This is not possible if there are some nested processing time intervals, that is when for some two jobs i and j it holds $\tilde{p}_i \subset \tilde{p}_j$. It is then hard to establish which of these two jobs must be processed first. Hence the complexity of the problem increases if there are a lot of nested processing time intervals.

15.3 A 2-Approximation Algorithm

In this section we construct a 2-approximation algorithm for the problem, which is based on the same idea as the 2-approximation `Algorithm AM` for minmax regret combinatorial optimization problems (see Section 4.1). We simply compute an optimal schedule under the midpoint scenario. The algorithm, denoted by `Algorithm SEQ-AM`, is shown in Figure 15.1.

`Algorithm SEQ-AM`
Require: n, $(\tilde{p}_i)_{i=1}^n$.
Ensure: A schedule σ such that $Z(\sigma) \leq 2OPT$.
1: **for all** $i \in \{1, \ldots, n\}$ **do**
2: $\quad p_i^S \leftarrow \frac{1}{2}(\underline{p}_i + \overline{p}_i)$
3: **end for**
4: Compute an SPT schedule σ for scenario S
5: **return** σ

Fig. 15.1. A 2-approximation algorithm for Minmax Regret $1||\sum C_i$

We now explore the worst case performance of `Algorithm SEQ-AM`. Let us denote by i_π the position occupying by job i in schedule π. The cost of π under scenario S can be then expressed as follows:

$$F(\pi, S) = \sum_{i \in J} (n - i_\pi + 1) p_i^S. \tag{15.10}$$

The following propositions are true:

Proposition 15.4. *For any two schedules π and σ the following inequality holds:*

$$Z(\pi) \geq \sum_{\{i: i_\sigma > i_\pi\}} (i_\sigma - i_\pi)\overline{p}_i + \sum_{\{i: i_\sigma < i_\pi\}} (i_\sigma - i_\pi)\underline{p}_i. \tag{15.11}$$

Proof. Let S_π be the worst case scenario for π and let π^* be the worst case alternative for π. Applying formula (15.10) we obtain

$$Z(\pi) = F(\pi, S_\pi) - F(\pi^*, S_\pi) = \sum_{i \in J} (i_{\pi^*} - i_\pi) p_i^{S_\pi}.$$

Since S_π is the worst case scenario for π we can obtain another worst case scenario by increasing the processing times of all jobs i such that $i_{\pi^*} > i_\pi$ to $\overline{p}_i$ and decreasing the processing times of all jobs such that $i_{\pi^*} < i_\pi$ to $\underline{p}_i$. Therefore it holds

$$Z(\pi) = \sum_{\{i: i_{\pi^*} > i_\pi\}} (i_{\pi^*} - i_\pi)\overline{p}_i + \sum_{\{i: i_{\pi^*} < i_\pi\}} (i_{\pi^*} - i_\pi)\underline{p}_i.$$

Since π^* is the worst case alternative for π, for any schedule σ it must hold:

$$Z(\pi) \geq \sum_{\{i: i_\sigma > i_\pi\}} (i_\sigma - i_\pi)\overline{p}_i + \sum_{\{i: i_\sigma < i_\pi\}} (i_\sigma - i_\pi)\underline{p}_i$$

and the proposition follows. □

Proposition 15.5. *For any two schedules π and σ the following inequality holds*

$$Z(\sigma) \leq Z(\pi) + \sum_{\{i: i_\pi > i_\sigma\}} (i_\pi - i_\sigma)\overline{p}_i + \sum_{\{i: i_\pi < i_\sigma\}} (i_\pi - i_\sigma)\underline{p}_i. \tag{15.12}$$

Proof. For any scenario S it holds

$$F(\sigma, S) - F(\pi, S) = \sum_{i \in J} (i_\pi - i_\sigma) p_i^S \leq \sum_{\{i: i_\pi > i_\sigma\}} (i_\pi - i_\sigma)\overline{p}_i + \sum_{\{i: i_\pi < i_\sigma\}} (i_\pi - i_\sigma)\underline{p}_i,$$

hence

$$F(\sigma, S) \leq F(\pi, S) + \sum_{\{i: i_\pi > i_\sigma\}} (i_\pi - i_\sigma)\overline{p}_i + \sum_{\{i: i_\pi < i_\sigma\}} (i_\pi - i_\sigma)\underline{p}_i. \tag{15.13}$$

Subtracting $F^*(S)$ from both sides of (15.13) we get

$$F(\sigma, S) - F^*(S) \leq F(\pi, S) - F^*(S) + \sum_{\{i: i_\pi > i_\sigma\}} (i_\pi - i_\sigma)\overline{p}_i + \sum_{\{i: i_\pi < i_\sigma\}} (i_\pi - i_\sigma)\underline{p}_i.$$

Substituting $S = S_\sigma$ in the above inequality yields

$$\begin{aligned} Z(\sigma) = F(\sigma, S_\sigma) - F^*(S_\sigma) &\leq F(\pi, S_\sigma) - F^*(S_\sigma) + \\ &+ \sum_{\{i: i_\pi > i_\sigma\}} (i_\pi - i_\sigma)\overline{p}_i + \sum_{\{i: i_\pi < i_\sigma\}} (i_\pi - i_\sigma)\underline{p}_i \leq \\ &\leq Z(\pi) + \sum_{\{i: i_\pi > i_\sigma\}} (i_\pi - i_\sigma)\overline{p}_i + \sum_{\{i: i_\pi < i_\sigma\}} (i_\pi - i_\sigma)\underline{p}_i, \end{aligned}$$

which completes the proof. □

We are ready to prove the following theorem:

Theorem 15.6. *Let σ be an optimal schedule under the midpoint scenario, that is the one in which $p_i = \frac{1}{2}(\underline{p}_i + \overline{p}_i)$ for all $i \in J$. Then $Z(\sigma) \leq 2OPT$, where OPT is the maximal regret of an optimal robust schedule.*

Proof. Let π be an optimal robust schedule, that is $Z(\pi) = OPT$. Since σ is the optimal schedule under the midpoint scenario we have

$$\frac{1}{2}\sum_{i\in J}(n - i_\pi + 1)(\overline{p}_i + \underline{p}_i) \geq \frac{1}{2}\sum_{i\in J}(n - i_\sigma + 1)(\overline{p}_i + \underline{p}_i),$$

which is equivalent to

$$\sum_{i\in J}(i_\sigma - i_\pi)(\overline{p}_i + \underline{p}_i) \geq 0. \tag{15.14}$$

Inequality (15.14) can be rewritten as follows:

$$\sum_{\{i:i_\sigma > i_\pi\}}(i_\sigma - i_\pi)\overline{p}_i + \sum_{\{i:i_\sigma < i_\pi\}}(i_\sigma - i_\pi)\underline{p}_i \geq \sum_{\{i:i_\pi > i_\sigma\}}(i_\pi - i_\sigma)\overline{p}_i + \sum_{\{i:i_\pi < i_\sigma\}}(i_\pi - i_\sigma)\underline{p}_i. \tag{15.15}$$

Now applying formula (15.11) to (15.15) we obtain

$$Z(\pi) \geq \sum_{\{i:i_\pi > i_\sigma\}}(i_\pi - i_\sigma)\overline{p}_i + \sum_{\{i:i_\pi < i_\sigma\}}(i_\pi - i_\sigma)\underline{p}_i. \tag{15.16}$$

Now inequalities (15.12) and (15.16) yield $Z(\sigma) \leq Z(\pi) + Z(\pi) = 2OPT$, which completes the proof. □

The following corollary is an immediate consequence of Theorem 15.6:

Corollary 15.7. `Algorithm SEQ-AM` *is a 2-approximation algorithm for the* Minmax Regret $1||\sum C_i$ *problem.*

We now show that the bound of 2 is tight. Consider a problem with three jobs. The interval processing times of jobs are $p_1 = [0, 2]$, $p_2 = [1, 1]$ and $p_3 = [1, 1]$. The midpoint scenario assigns to all jobs the processing time equal to 1. In consequence, `Algorithm SEQ-AM` may return any schedule. Suppose that it returns schedule $\sigma = (1, 3, 2)$. Schedule σ has the maximal regret equal to 2 while the optimal robust schedule $\pi = (3, 1, 2)$ has the maximal regret equal to 1. In consequence $Z(\sigma)/Z(\pi) = 2$.

Using `Algorithm SEQ-AM` we can obtain in $\mathcal{O}(n \log n)$ time a solution σ that is at most two times as bad as optimum. Observe that the algorithm recognizes whether $i \preceq j$. Indeed, if $i \preceq j$, then $\frac{1}{2}(\underline{p}_i + \overline{p}_i) \leq \frac{1}{2}(\underline{p}_i + \overline{p}_i)$ and job i is placed before job j in the constructed schedule. In particular, the algorithm returns an optimal robust schedule if there are no nested processing time intervals in the problem. If there are some nested processing time intervals, then the obtained schedule may be refined by applying a local search, which will be described in the next section.

Observe that the whole argumentation presented in Proposition 15.4, Proposition 15.5 and Theorem 15.6 can be repeated without any modifications if the set of jobs is partially ordered by some precedence constraints. It is enough to assume that all schedules π and σ, which appear in the proofs, are feasible. This leads to the following theorem:

Theorem 15.8. *If the deterministic* $1|prec|\sum C_i$ *problem for some particular structure of the precedence constraints is polynomially solvable, then the* Minmax Regret $1|prec|\sum C_i$ *problem is approximable within 2.*

The general deterministic $1|prec|\sum C_i$ problem is strongly NP-hard. It is, however, polynomially solvable if a graph of the precedence constraints is a tree or has a series-parallel topology. But even in the most general case we can still obtain an approximate solution to Minmax Regret $1|prec|\sum C_i$ by solving to optimality the deterministic problem $1|prec|\sum C_i$ under the midpoint scenario. It may, however, require an exponential time under arbitrary precedence constraints.

15.4 Local Search Algorithm

In the previous section we have constructed a simple algorithm that returns a schedule whose maximal regret is at most two times as bad as optimum. This solution is a good starting point for a local search. Recall that a local search algorithm starts from an initial solution and iteratively improves the current solution by moving to a better solution from its neighborhood. We now use the fact that every sequencing problem has a natural neighborhood function, that is a mapping N, which for every schedule π defines a subset of schedules $N(\pi)$, which can be reached in one step from π. In our approach, the neighborhood $N(\pi)$ consists of all schedules π' obtained by interchanging the positions of two jobs in π. Observe that $N(\pi)$ consists of $\mathcal{O}(n^2)$ schedules and it can be easily listed in $\mathcal{O}(n^2)$ time.

We now apply the simplest local search algorithm, called an *iterative improvement*. The idea of this algorithm was described in Section 7.4 while considering the Minmax Regret Minimum Spanning Tree problem. Recall that the iterative improvement algorithm returns a local minimum with respect to the neighborhood N. Notice that in the considered problem it is not necessary to check all solutions from $N(\pi)$. By Proposition 15.2, if $\pi(i) \preceq \pi(j)$ and $i < j$, then schedule π' obtained from π by interchanging jobs i and j is such that $Z(\pi') \geq Z(\pi)$. So, in this case it is not necessary to compute $Z(\pi')$ and we may skip solution π'. This may significantly speed up calculations, since computing the maximal regret of a given solution requires solving an assignment problem. Algorithm `Iterative Improvement SEQ` for the problem is shown in Figure 15.2.

It is obvious that solution σ returned by `Iterative Improvement SEQ` is such that $Z(\sigma) \leq 2OPT$, where OPT is the maximal regret of an optimal robust schedule. We can expect, however, that its performance will be much better

```
Iterative Improvement SEQ
Require: n, (p̃_i)_{i=1}^n
Ensure: A local minimum σ.
Compute an initial schedule π using Algorithm SEQ-AM
σ ← π, best ← Z(π)
stop← false
while stop=false do
  stop ← true
  for i ← 1 to n do
    for j ← i + 1 to n do
      if not π(i) ⪯ π(j) then
        Create π′ by interchanging π(i) and π(j) in π
        if Z(π′) < best then σ ← π′, best ← Z(π′), stop ← false
      end if
    end for
  end for
  π ← σ
end while
return σ
```

Fig. 15.2. The iterative improvement algorithm for MINMAX REGRET $1||\sum C_i$

in the average case. In the next section we will present some computational results which justify this conjecture. The theoretical characterization of the local minimum returned by the algorithm is an important subject of further research.

15.5 Mixed Integer Programming Formulation

In this section we construct a mixed integer programming model to solve the problem, which is according to Montemanni [106]. In order to describe a schedule we introduce binary variables $x_{ij} \in \{0,1\}$ for $i,j = 1,\ldots,n$. The variable x_{ij} takes the value of 1 if and only if job $i \in J$ occupies position j in a constructed schedule. The variables (x_{ij}) must fulfill the following assignment constraints:

$$\begin{aligned} &\sum_{i=1}^{n} x_{ij} = 1 \ \text{ for } j = 1,\ldots,n \\ &\sum_{j=1}^{n} x_{ij} = 1 \ \text{ for } i = 1,\ldots,n \\ &x_{ij} \in \{0,1\} \text{ for } i,j \in 1,\ldots,n \end{aligned} \tag{15.17}$$

It is clear that every feasible solution to (15.17) represents a schedule and every schedule can be represented as a feasible solution to (15.17). Let variables (x_{ij}) represent a schedule π. Now formulae (15.8) and (15.9) imply that the optimal schedule can be obtained by solving the following problem:

$$\min_{(x_{ij})} \max_{(z_{ij})} \sum_{i=1}^{n} \sum_{j=1}^{n} c_{ij} z_{ij}, \tag{15.18}$$

where

$$c_{ij} = \overline{p}_i \sum_{k=1}^{j} (j-k) x_{ik} + \underline{p}_i \sum_{k=j+1}^{n} (j-k) x_{ik} \tag{15.19}$$

and both (x_{ij}) and (z_{ij}) fulfill the assignment constraints. The binary variable z_{ij} takes the value of 1 if and only if job i occupies position j in the worst case alternative π^*. Problem (15.18) is not linear but it can be transformed to a linear one. Let us fix variables (x_{ij}) in (15.18) and consider the following subproblem:

$$\begin{aligned}
&\max \sum_{i=1}^{n} \sum_{j=1}^{n} c_{ij} z_{ij} \\
&\sum_{i=1}^{n} z_{ij} = 1 && \text{for } j = 1, \ldots, n \\
&\sum_{j=1}^{n} z_{ij} = 1 && \text{for } i = 1, \ldots, n \\
&z_{ij} \geq 0 && \text{for } i, j \in 1, \ldots, n
\end{aligned} \tag{15.20}$$

Formulation (15.20) is an assignment problem with fixed coefficients c_{ij}. We have replaced constraints $z_{ij} \in \{0, 1\}$ with $z_{ij} \geq 0$, which does not change the optimal value of the objective function because the constraint matrix of the assignment problem is totally unimodular. Consider the dual to (15.20), which has the following form:

$$\begin{aligned}
&\min \sum_{i=1}^{n} \alpha_i + \sum_{i=1}^{n} \beta_i \\
&\alpha_i + \beta_j \geq \overline{p}_i \sum_{k=1}^{j} (j-k) x_{ik} + \underline{p}_i \sum_{k=j+1}^{n} (j-k) x_{ik} \text{ for } i, j = 1, \ldots, n
\end{aligned} \tag{15.21}$$

Observe that we have replaced costs c_{ij} with expression (15.19) in (15.21). It is well known that problems (15.20) and (15.21) have the same optimal values of the objective functions. We can now express (15.18) in the following way:

$$\begin{aligned}
&\min \sum_{i=1}^{n} \alpha_i + \sum_{i=1}^{n} \beta_i \\
&\alpha_i + \beta_j \geq \overline{p}_i \sum_{k=1}^{j} (j-k) x_{ik} + \underline{p}_i \sum_{k=j+1}^{n} (j-k) x_{ik} \text{ for } i, j = 1, \ldots, n \\
&\sum_{i=1}^{n} x_{ij} = 1 && \text{for } j = 1, \ldots, n \\
&\sum_{j=1}^{n} x_{ij} = 1 && \text{for } i = 1, \ldots, n \\
&x_{ij} \in \{0, 1\} && \text{for } i, j \in 1, \ldots, n
\end{aligned} \tag{15.22}$$

Formulation (15.22) is the MIP model for MINMAX REGRET $1||\sum C_i$. Following Montemanni [106], we now refine this model using Theorem 15.3. Recall that if for any two jobs i and j it holds $i \preceq j$, then we know that i is processed before j in an optimal robust schedule. Using this fact we can add some additional constraints to (15.22), which reduce the solution space. Let $a_i = |\{j : j \preceq i\}|$ and $b_i = |\{j : i \preceq j\}|$. Thus a_i is the number of jobs that are known to precede i and b_i is the number of jobs that are known to follow i in an optimal robust schedule. For every two jobs i and j such that $i \preceq j$ we add the following three constraints to the model (see also Montemanni [106]):

$$\sum_{k=1}^{n} k(x_{ik} - x_{jk}) \leq -1 \tag{15.23}$$

$$\sum_{j=1}^{a_i} x_{ij} = 0 \tag{15.24}$$

$$\sum_{j=n-b_i+1}^{n} x_{ij} = 0 \tag{15.25}$$

Constraint (15.23) expresses the fact that the position occupied by job i is less than the position occupied by job j. Indeed, if $x_{iu} = 1$ and $x_{jv} = 1$ for $u \geq v$, then $u - v \geq 0$ and constraint (15.23) is violated. Hence it must hold $u < v$. Constraints (15.24) and (15.25) express the fact that at least a_i jobs precede i (therefore first a_i positions cannot be occupied by job i) and at least b_i jobs follow i (therefore the last $n - b_i + 1$ positions cannot be occupied by job i). Constraints (15.23)-(15.25) may significantly reduce the solution space and consequently speed up the calculations.

15.5.1 Computational Results

In this section we present some results of computational tests. We used the MIP formulation constructed in the previous section and we compared the obtained optimal robust solutions to the solutions returned by `Algorithm SEQ-AM` and the local search algorithm `Iterative Improvement`. In our experiments we used the family of problems denoted as $S(n, c)$, where:

- n is the number of jobs;
- for every job $i \in J$ the value of $\overline{p}_i$ is a randomly selected integer from interval $[0, c]$ and the value of $\underline{p}_i$ is a randomly selected integer from interval $[0, \overline{p}_i]$.

We chose $n \in \{10, 20, 30, 40\}$ and $c \in \{20, 50, 100\}$. For every combination of n and c we generated and solved 5 instances. In order to solve the MIP model CPLEX 8.0 and a Pentium III 866MHz computer were used. The obtained results are shown in Table 15.1. In this table the average CPU times in seconds required to solve the MIP model is shown. Notice that this CPU time grows fast with the number of jobs and for n larger than 40 the computations may be very time

Table 15.1. The computational results

		Alg. SEQ-AM		Iter. Impr.	
n	CPU Time	Worst %	Aver. %	Worst %	Aver. %
10	0.116	27.03	5.40	10.81	1.12
20	2.317	16.72	8.46	1.00	0.13
30	104.154	17.34	10.47	2.06	0.63
40	861.55	13.53	8.36	1.02	0.26

consuming. Hence, the MIP approach allows us to solve in reasonable time rather small problems.

In order to solve the generated instances the 2-approximation `Algorithm SEQ-AM` and the local search algorithm `Iterative Improvement` were also applied. The reported average and maximal percentage deviations from optimum for every value of n are also shown in Table 15.1. One can observe that `Iterative Improvement` performs reasonably well and it seems to be a good choice for the problems with large number of jobs.

For more extensive computational tests of the MIP formulation we refer the reader to the Montemanni's paper [106].

15.6 Notes and References

The deterministic $1||\sum C_i$ is a very basic sequencing problem. The SPT rule, which allows us to obtain the optimal schedule for this problem, was first given by Smith [117] (it is sometimes called Smith's rule) and it was among the earliest results in scheduling theory. The minmax regret version of the problem was first discussed by Daniels and Kouvelis [39] and it was also described by Kouvelis and Yu in [87]. The method of computing the worst case scenario follows from [87] and the proof of Theorem 15.3 can also be found in [87]. However, when book [87] was published, the complexity status of the problem was unknown. The proof of NP-hardness of the problem is a very recent result obtained by Lebedev and Averbakh [94]. In paper [94] some additional interesting properties of the problem were presented. In particular, it is polynomially solvable if all processing time intervals have equal midpoints and the number of jobs is even. The problem becomes NP-hard if the number of jobs is odd.

Kouvelis and Yu [87] proposed several heuristics for solving the problem. Among them, there is the one in which the optimal schedule under the midpoint scenario is computed. However, no analysis of the worst case performance of this heuristic was provided, and the proof that its worst case performance ratio is 2 was given by Kasperski and Zieliński in [86]. This result immediately implies that the worst case performance ratio of the local search algorithm is also at most 2.

The MIP formulation presented in Section 15.5 is according to Montemanni [106]. There is also a branch and bound algorithm for the problem designed by Daniels and Kouvelis [39] (it is also presented in [87]). However, it was shown

in [106] that this branch and bound algorithm is not significantly faster than the MIP formulation solved by means of CPLEX. For the instances with a large number of jobs the local search algorithm constructed in Section 15.4 seems to be a good choice. More exhaustive computational tests concerning the MIP formulation and some other heuristics can be found in [87] and [106].

There is a number of open questions connected with the Minmax Regret $1||\sum C_i$ problem. It is not known whether this problem is strongly NP-hard. In particular, the problem may admit a pseudpolynomial algorithm or even an FPTAS. Having a pseudopolynomial algorithm one can try to apply the scaling and rounding technique (see Section 4.2) to construct an FPTAS. Observe that we have just constructed an algorithm that computes the upper and lower bounds UB and LB such that $UB = 2LB$. This is precisely `Algorithm SEQ-AM`. We can also ask about the quality of the local minimum returned by `Iterative Improvement`. In particular, it is interesting to check whether a local minimum σ is always such that $Z(\sigma) \leq kOPT$ for k as small as possible (we know that for $k = 2$ it follows).

It is well known that the deterministic problem $1|prec|\sum C_i$ is strongly NP-hard [93, 95]. However, if a graph of the precedence constrains is a tree or has a series-parallel topology, then problem $1|prec|\sum C_i$ is polynomially solvable. A description of the algorithms can be found in [29]. We can also define a nonnegative weight for every job and minimize the total weighted flow, that is the value of $\sum_{i\in J} w_i C_i(\pi)$. This problem is solvable in $\mathcal{O}(n \log n)$ time by the following modified Smith's rule: schedule jobs in order of nondecreasing ratios p_i/w_i. It is clear that the minmax regret versions of the generalizations of $1||\sum C_i$ are NP-hard. However, it would be important and interesting to design some exact and approximation algorithms for these problems.

16 Conclusions and Open Problems

In this part of the monograph we have discussed minmax regret problems in which the set of feasible solutions is composed of sequences of a given core set. There are several important differences between these problems and the minmax regret combinatorial optimization problems discussed in the first part of this monograph. For every element, called a job, there may be given more than one imprecise parameter, for instance a processing time and a due date. In consequence, it may be more difficult to describe the worst case scenario for a given solution. Sequencing problems have typically more complex objective functions, which also makes the computation of the maximal regret difficult. There is also a lack of general properties for this class of problems and every problem has its own specific features.

The state of the art in the field of the minmax regret sequencing is very far from being complete. In this monograph we have explored only a small part of the large variety of problems. The sequencing models itself form a small part of scheduling theory and minmax regret versions of some other scheduling problems should also be investigated.

There are a number of open questions connected with the sequencing problems considered in this monograph. We present below some of them, which seem to be of particular importance.

1. The polynomially solvable MINMAX REGRET $1|prec|L_{max}$ problem is one of the simplest problems with the bottleneck objective function. Its modifications like MINMAX REGRET $1|prec|\max w_iL_i$ and MINMAX REGRET $1|prec|\max w_iT_i$, in which the processing times and due dates are uncertain, remain open. Even characterization of the worst case scenario for these problems must be investigated.
2. The complexity of MINMAX REGRET $1|p_i = 1|\sum w_iU_i$ is open. We only know that if all jobs have the same due date, then the problem is polynomially solvable. An interesting generalization of this problem is to allow the due dates to be uncertain. No results concerning such a problem are known. It would also be interesting to investigate the MINMAX REGRET $1||\sum U_i$

A. Kasperski: Discrete Optimization with Interval Data, STUDFUZZ 228, pp. 197–198, 2008.
springerlink.com

problem with uncertain processing times. This problem is interesting and nontrivial even if all jobs have the same due date.

3. The most important question on MINMAX REGRET $1||\sum C_i$ is whether this problem can be solved in pseudopolynomial time. If so, then it may even admit an FPTAS, which may be constructed by applying the scaling and rounding technique. The existence of approximation algorithms with lower than 2 worst case performance ratio for this problem is also unknown. The generalizations of the problem like MINMAX REGRET $1||\sum w_i C_i$ or MINMAX REGRET $1|sp-graph|\sum w_i C_i$ are also interesting subjects of further research. The local search appears to be a promising technique for solving MINMAX REGRET $1||\sum C_i$. An important open question is about the worst case performance of `Iterative Improvement`. It is possible that the worst case ratio of this algorithm is lower than 2.

A Discrete Scenario Representation of Uncertainty

Consider a deterministic combinatorial optimization problem $\mathcal{P}$, which is of the form (1.1). An alternative method of defining scenario set Γ in this problem is applying a *discrete scenario* representation of uncertainty. In this approach we explicitly list all scenarios, that is we define

$$\Gamma = \{S_1, S_2, \ldots, S_K\},\ K \geq 1.$$

If $K = 1$, then there is only one scenario and the problem boils down to the deterministic problem $\mathcal{P}$. The maximal regret of a given solution $X \in \Phi$ is defined in the following way:

$$Z(X) = \max_{i=1,\ldots,K} \{F(X, S_i) - F^*(S_i)\}. \tag{A.1}$$

Let D-Minmax Regret $\mathcal{P}$ denote the problem in which we seek a solution that minimizes the maximal regret. The letter "D" indicates that we deal with the discrete scenario representation of uncertainty. We distinguish two important cases of this problem. In the first case the number of scenarios K is *bounded* by a constant, for instance we may consider only problems with 2 scenarios. In the second case the number of scenarios is *unbounded* and K is a part of the input.

A major part of book [87] is devoted to the D-Minmax Regret $\mathcal{P}$ problem. In particular, in this book some general approaches to solve the problem are described. The aim of this section is to present some recent developments in this area and compare them to the results known for the interval uncertainty representation. In Section A.1 we show how to represent the problem as a mixed integer linear programming one. In Section A.2 we describe the complexity results and in Section A.3 the approximation results concerning the problem.

A.1 MIP Formulation

It is not difficult to represent D-Minmax Regret $\mathcal{P}$ as a mixed integer linear programing problem. Suppose that the set of the feasible solutions Φ can be described by the following set of linear constraints:

A. Kasperski: Discrete Optimization with Interval Data, STUDFUZZ 228, pp. 199–207, 2008.
springerlink.com

$$\begin{aligned} &\boldsymbol{A}[\boldsymbol{x}, \boldsymbol{x'}]^T = \boldsymbol{b} \\ &x_i \in \{0,1\} \quad \text{for } i = 1, \ldots, n, \end{aligned} \tag{A.2}$$

where $\boldsymbol{x} = [x_1, \ldots, x_n]$ represents a characteristic vector of a feasible solution (or a characteristic vector of a subset of E that includes a feasible solution, see Section 3.1) and $\boldsymbol{x'}$ is a vector of auxiliary variables (if required). We also allow signs $\leq$ or $\geq$ instead of equalities in the formulation (A.2). Denote by $\boldsymbol{w}_j = [w_1^{S_j}, \ldots, w_n^{S_j}]$ the vector of weights under scenario S_j. Now D-Minmax Regret $\mathcal{P}$ is equivalent to the following optimization problem:

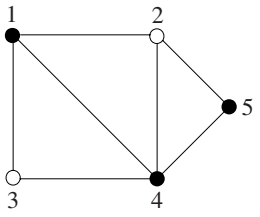
$$\begin{aligned} &\min t \\ &\boldsymbol{w}_j \boldsymbol{x}^T - F^*(S_j) \leq t \text{ for } j = 1, \ldots, K \\ &\boldsymbol{A}[\boldsymbol{x}, \boldsymbol{x'}]^T = \boldsymbol{b} \\ &x_i \in \{0,1\} \qquad\qquad \text{for } i = 1, \ldots, n \end{aligned} \tag{A.3}$$

If the underlying deterministic problem $\mathcal{P}$ is polynomially solvable, then all values $F^*(S_j)$ for $j = 1, \ldots, K$ can be computed in polynomial time. Problem (A.3) can be solved by means of a standard software.

A.2 Complexity Results

A major part of a book by Kouvelis and Yu [87] is devoted to the D-Minmax Regret $\mathcal{P}$ problem. Kouvelis and Yu [87] proved that D-Minmax Regret Minimum Spanning Tree, D-Minmax Regret Shortest Path and D-Minmax Regret Minimum Assignment are NP-hard even when the number of scenarios equals 2. Averbakh [16] proved that D-Minmax Regret Minimum Selecting Items is NP-hard for 2 scenarios. This latter result demonstrates that the discrete scenario representation of uncertainty may lead to the problems of a different complexity than the interval one. Recall that Minmax Regret Minimum Selecting Items, in which the interval representation of uncertainty is adopted, is solvable in polynomial time. Finally, Aissi *et al.* [5] showed that D-Minmax Regret Minimum Cut is NP-hard for 2 scenarios. We can thus see that all the combinatorial optimization problems considered in this monograph become NP-hard even if the number of scenarios is equal to 2. However, most of these problems are not strongly NP-hard. They are solvable in pseudopolynomial time [9] and admit and FPTAS.

The complexity of D-Minmax Regret $\mathcal{P}$ becomes worse if we move from 2 to unbounded number of scenarios. If the number of scenarios is a part of the input, then D-Minmax Regret Shortest Path, D-Minmax Regret Minimum Spanning Tree, D-Minmax Regret Minimum Assignment and D-Minmax Regret Minimum Cut become strongly NP-hard [4, 5, 87]. The next three theorems (first proved in [85]) demonstrate how the results obtained in [4, 5, 87] can be strengthen.



v	S_{12}	S_{13}	S_{14}	S_{24}	S_{25}	S_{34}	S_{45}
1	0	0	0	1	1	1	1
2	0	1	1	0	0	1	1
3	1	0	1	1	1	0	1
4	1	1	0	0	1	0	0
5	1	1	1	1	0	1	0

Fig. A.1. A sample transformation. Graph $G = (V, E')$ and the corresponding D-Minmax Regret Minimum Selecting Items problem with scenario set Γ.

Theorem A.1. *The* D-Minmax Regret Minimum Selecting Items *problem is strongly NP-hard for the unbounded case. Moreover, it is not* $(2-\epsilon)$*-approximable for any* $\epsilon > 0$*, unless P=NP.*

Proof. We first show that the problem is strongly NP-hard. We use a polynomial transformation from the following problem, which is known to be strongly NP-complete [56]:

Vertex Cover
Instance: Undirected graph $G = (V, E')$, a number T such that $1 \leq T < |V|$.
Question: Is there a *vertex cover* W in G such that $|W| \leq T$? (a vertex cover is a subset of nodes $W \subseteq V$ such that, for every edge $\{i, j\} \in E'$ of G, at least one of i and j belongs to W)

Let a graph G and a number T be an instance of Vertex Cover. We construct an instance of D-Minmax Regret Minimum Selecting Items in the following way: the item set E equals the set of nodes of G, that is $E = V$; for every edge $\{i, j\} \in E'$ of graph G we construct scenario S_{ij} such that $w_i^{S_{ij}} = w_j^{S_{ij}} = 0$ and $w_k^{S_{ij}} = 1$ for all $k \neq i$ and $k \neq j$. Hence the number of scenarios equals the number of edges of graph G. We finish the construction by fixing $p = T$. A sample transformation is shown in Figure A.1.

We now show that G has a vertex cover W such that $|W| \leq T$ if and only if there is a selection X of exactly p items such that

$$\max_{S \in \Gamma} F(X, S) \leq p - 1. \tag{A.4}$$

($\Rightarrow$) Suppose that G has a vertex cover $W \subseteq V$ such that $|W| \leq T$. Then G has also a vertex cover exactly of the cardinality T (if $|W| < T$, then we add some arbitrary nodes from $V \setminus W$ to W until $|W| = T$). Consider the selection $X = W$. Since W covers all edges of G, under each scenario $S \in \Gamma$ at least one item $e \in X$ has $w_e^S = 0$. Hence $F(X, S) \leq p - 1$ for every $S \in \Gamma$ and thus (A.4) holds.

($\Leftarrow$) Suppose that a solution X satisfies (A.4). Then for each scenario $S \in \Gamma$ there must be an element $e \in X$ such that $w_e^S = 0$. Furthermore, since $|X| = p = T$, $W = X$ is a vertex cover of cardinality T in G.

Observe now, that under every scenario $S \in \Gamma$ precisely 2 items have weights equal to 0 and all the remaining items have weights equal to 1. Hence

$F^*(S) = p - 2$ for all $S \in \Gamma$. From (A.4) we get that G has a vertex cover of size T if and only if there is a solution X such that

$$Z(X) = \max_{S \in \Gamma} \{F(X, S) - F^*(S)\} \leq p - 1 - (p - 2) = 1.$$

Thus G has a vertex cover of the size at most T if and only of there is a solution X such that $Z(X) \leq 1$. In consequence, the D-Minmax Regret Minimum Selecting Items problem is strongly NP-hard.

We now apply a well known gap technique to show that the problem is not approximable within $2-\epsilon$ for any $\epsilon > 0$. Suppose that we have a polynomial $(2-\epsilon)$-approximation algorithm for the problem, where $\epsilon > 0$. Let us apply this algorithm to the instance of D-Minmax Regret Minimum Selecting Items obtained from the instance of Vertex Cover. If G has a vertex cover of the size at most T, then there is an optimal robust solution X such that $Z(X) \leq 1$ and the approximation algorithm returns a solution Y such that $Z(Y) \leq 2-\epsilon$. On the other hand, if G does not have the proper vertex cover, then the approximation algorithm returns a solution Y such that $Z(Y) \geq 2$. Having $(2 - \epsilon)$-approximation algorithm for D-Minmax regret Minimum Selecting Items we would be able to solve NP-complete Vertex Cover in polynomial time. Therefore, there is no $(2 - \epsilon)$-approximation algorithm for the problem if $P \neq NP$. □

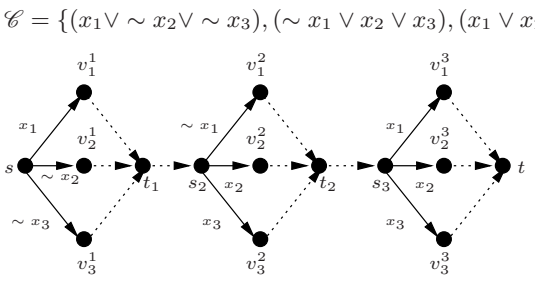
Let us now focus on the D-Minmax Regret Shortest Path problem. The following result demonstrates that the problem is strongly NP-hard for edge series-parallel digraphs. Recall that for the interval uncertainty representation this problem admits a pseudpopolynomial algorithm for this class of graphs.

Theorem A.2. *The* D-Minmax Regret Shortest Path *problem for the unbounded case is strongly NP-hard and not* $(2 - \epsilon)$*-approximable for any* $\epsilon > 0$*, unless P=NP, even for edge series-parallel digraphs, directed and undirected, with node degrees at most 4.*

Proof. We use a polynomial transformation from 3-CNF-SAT, which is known to be strongly NP-complete [56].

3-CNF-SAT
Instance: Set $\mathscr{U} = \{x_1, \ldots, x_n\}$ of Boolean variables, collection $\mathscr{C} = \{C_1, \ldots, C_m\}$ of clauses, where every clause in $\mathscr{C}$ has exactly three distinct literals.
Question: Is there an assignment to $\mathscr{U}$ that satisfies all clauses in $\mathscr{C}$?

For a given instance of 3-CNF-SAT we construct the corresponding instance of D-Minmax Regret Shortest Path as follows: we associate with each clause $C_i = (l_i^1 \vee l_i^2 \vee l_i^3)$ a digraph G_i composed of 5 nodes: $s_i, v_1^i, v_2^i, v_3^i, t_i$ and 6 arcs: the *literal arcs* (s_i, v_1^i), (s_i, v_2^i), (s_i, v_3^i) correspond to literals in C_i, the *dummy arcs* (v_1^i, t_i), (v_2^i, t_i), (v_3^i, t_i) have weights equal to 0 under every scenario; in order to construct digraph G we connect all digraphs $G_1, \ldots, G_m$ using additional dummy arcs (t_i, s_{i+1}) for $i = 1, \ldots m-1$; we finish the construction of G by setting $s = s_1$ and $t = t_m$. For every pair of arcs of G, (s_i, v_j^i) and (s_q, v_r^q),



	S_1	S_2	S_3	S_4	S_5	S_6
(s, v_1^1)	**1**	0	0	0	0	0
(s, v_2^1)	0	**1**	**1**	0	0	0
(s, v_3^1)	0	0	0	**1**	**1**	0
(s_2, v_1^2)	**1**	0	0	0	0	**1**
(s_2, v_2^2)	0	**1**	0	0	0	0
(s_2, v_3^2)	0	0	0	**1**	0	0
(s_3, v_1^3)	0	0	0	0	0	**1**
(s_3, v_2^3)	0	0	**1**	0	0	0
(s_3, v_3^3)	0	0	0	0	**1**	0

Fig. A.2. An example of the transformation. All the dummy arcs (the dotted arcs) have weights equal to 0 under all scenarios and they are not listed in the table.

that correspond to *contradictory* literals l_i^j and l_q^r, i.e. $l_i^j = \sim l_q^r$, we create scenario S such that under this scenario the weights of the arcs (s_i, v_j^i) and (s_q, v_r^q) are set to 1 and the weights of all the remaining arcs are set to 0. An example of the transformation is shown in Figure A.2.

We now show that the answer to 3-CNF-SAT is "Yes" if and only if there is an $s-t$ path P in G such that $F(P,S) \leq 1$ under every scenario $S \in \Gamma$, which yields $\max_{S\in\Gamma} F(P,S) \leq 1$.

($\Rightarrow$) Suppose that there is an assignment to the variables in $\mathscr{U}$ that satisfies all clauses. Thus every clause C_i contains at least one literal whose value is true and we choose the arcs that correspond to these literals in subgraphs G_i, $i = 1, \ldots, m$. To complete P, we add some dummy arcs. It is clear that path P does not contain arcs that correspond to contradictory literals and thus $F(P,S) \leq 1$ for every scenario $S \in \Gamma$.

($\Leftarrow$) Suppose that there is an $s-t$ path P in G whose weight is less than or equal to 1 under every scenario $S \in \Gamma$. Therefore, P cannot contain arcs that correspond to contradictory literals. Consider the set of literal arcs that belong to P and denote this set by $\mathscr{L}$. Since $\mathscr{L}$ does not contain contradictory literals, there is an assignment to the variables of $\mathscr{U}$ for which all literals in $\mathscr{L}$ become true. Since exactly one literal of each clause belongs to $\mathscr{L}$, the answer to 3-CNF-SAT is "Yes".

Observe now that the weight of the shortest path under every scenario $S \in \Gamma$ equals 0. Hence the answer to 3-CNF-SAT is "Yes" if and only if there is a path P such that $F(P,S) - F^*(S) \leq 1$ for every scenario $S \in \Gamma$, or equivalently

$Z(P) \leq 1$. Therefore, D-Minmax Regret Shortest Path problem is strongly NP-hard. Moreover, applying the gap technique we can see that the problem is not $(2-\epsilon)$-approximable for any $\epsilon > 0$, unless P=NP. Note also that G is an edge series-parallel digraph with node degrees at most 4.

The above results remain true if a graph is undirected edge series-parallel with node degrees at most 4. The proof is very similar. □

Theorem A.3. *The* D-Minmax Regret Minimum Spanning Tree *problem for the unbounded case is strongly NP-hard and not approximable within* $(2-\epsilon)$ *for any* $\epsilon > 0$*, unless P=NP, even for edge series-parallel graphs with node degrees at most 4.*

Proof. The reduction from 3-CNF-SAT goes in the same manner as in the proof of Theorem A.2. It is enough to remove arcs directions in the resulting digraph G obtaining the edge series-parallel graph G' with node degrees at most 4. It is easily seen that there is an $s-t$ path in G that does not contain arcs corresponding to contradictory literals if and only if there is a spanning tree in G' that does not contain edges corresponding to contradictory literals. Every such path can be transformed to a spanning tree by adding some dummy arcs and, vice versa, by removing some dummy arcs. The rest part of the proof is very similar to the one of Theorem A.2. □

Theorem A.4. *The* D-Minmax Regret Minimum Cut *problem for the unbounded case is strongly NP-hard and not* $(2-\epsilon)$*-approximable for any* $\epsilon > 0$*, unless P=NP, even for edge series-parallel directed or undirected graphs.*

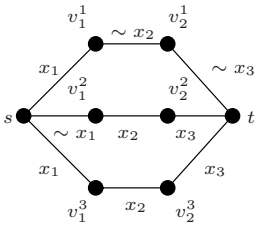
Proof. We use a transformation from 3-CNF-SAT (see the proof of Theorem A.2). For a given instance of 3-CNF-SAT, we construct the corresponding instance of D-Minmax Regret Minimum Cut in the following way: for each clause $C_i = (l_i^1 \vee l_i^2 \vee l_i^3)$ in $\mathscr{C}$ we create three edges of the form $\{s, v_1^i\}$, $\{v_1^i, v_2^i\}$ and $\{v_2^i, t\}$, which correspond to the literals in C_i. Observe that the resulting graph G is composed of exactly m disjoint $s-t$ paths and it has a series-parallel topology. For every pair of edges that correspond to contradictory literals l_i^j and l_q^r, we form scenario S such that under this scenario the weights of edges that correspond to l_i^j and l_q^r are set to 1 and the weights of all the remaining edges are set to 0. An example of the transformation is shown in Figure A.3.

We now show that the answer to 3-CNF-SAT is "Yes" if and only if there is a cut X in G such that $F(X,S) \leq 1$ for all scenarios $S \in \Gamma$.

($\Rightarrow$) Suppose that there is an assignment to the variables in $\mathscr{U}$ that satisfies all clauses. Thus every clause C_i contains at least one literal whose value is true. We choose exactly one such literal in C_i and add the edge that corresponds to this literal to set X, forming in this way a cut in G. Indeed, X is a cut since it contains exactly one edge from every $s-t$ path in G. It is clear that X does not contain edges that correspond to contradictory literals and, consequently, $F(X,S) \leq 1$ for every scenario $S \in \Gamma$.

($\Leftarrow$) Suppose that there is a cut X in G whose weight is less than or equal to 1 under every scenario. Therefore X cannot contain edges that correspond

$\mathscr{C} = \{(x_1 \vee \sim x_2 \vee \sim x_3), (\sim x_1 \vee x_2 \vee x_3), (x_1 \vee x_2 \vee x_3)\}$



	S_1	S_2	S_3	S_4	S_5	S_6
$\{s, v_1^1\}$	**1**	0	0	0	0	0
$\{v_1^1, v_2^1\}$	0	**1**	**1**	0	0	0
$\{v_2^1, t\}$	0	0	0	**1**	**1**	0
$\{s, v_1^2\}$	**1**	0	0	0	0	**1**
$\{v_1^2, v_2^2\}$	0	**1**	0	0	0	0
$\{v_2^2, t\}$	0	0	0	**1**	0	0
$\{s, v_1^3\}$	0	0	0	0	0	**1**
$\{v_1^3, v_2^3\}$	0	0	**1**	0	0	0
$\{v_2^3, t\}$	0	0	0	0	**1**	0

Fig. A.3. An example of the reduction

to contradictory literals. Since X is a cut, at least one edge on every $s-t$ path in G must belong to X. Let us choose exactly one such edge from every $s-t$ path and denote by $\mathscr{L}$ the set of literals that correspond to these selected edges. Since exactly one literal of each clause belongs to $\mathscr{L}$ and $\mathscr{L}$ does not contain contradictory literals, there is an assignment to the variables of $\mathscr{U}$, for which all literals in $\mathscr{L}$ become true. Thus the answer to 3-CNF-SAT is "Yes".

It is easily seen that under every scenario $S \in \Gamma$ there is a cut with the total weight of 0. Therefore the answer to 3-CNF-SAT is "Yes" if and only if there is a cut X in G such that $Z(X) \leq 1$ and consequently D-Minmax Regret Minimum Cut is strongly NP-hard. Applying the gap technique we can prove that it is not approximable within $(2-\epsilon)$ for any $\epsilon > 0$, unless P=NP.

Let us now replace edges $\{s, v_1^i\}$, $\{v_1^i, v_2^i\}$ and $\{v_2^i, t\}$ with arcs (s, v_1^i), (v_1^i, v_2^i) and (v_2^i, t) for all $i = 1, \ldots, m$, forming in this way a directed graph G'. Applying the same reasoning as for the undirected case, it is easy to prove that the answer to 3-CNF-SAT is "'Yes"' if and only if there is a directed $s-t$ cut in G' of the maximal regret value less than or equal to 1. In consequence, the theorem follows for directed graphs as well. □

The negative results proved in this section demonstrate that the discrete scenario representation of uncertainty leads to problems, which may be harder from the computational point of view than the problems with the interval uncertainty representation. In particular, the minmax regret version of the Minimum Selecting Items problem, which is polynomially solvable in the interval case, becomes strongly NP-hard in the discrete representation of uncertainty. Moreover, we also have some negative approximation results for the discrete representation, while no such results are known for the interval case.

A.3 Approximation

We now describe some results concerning the approximation of D-Minmax Regret $\mathcal{P}$. We start by showing a proof of a general result, which is a version of the result proved by Aissi *et al.* [9]:

Theorem A.5. *Consider scenario set* $\Gamma = \{S_1, \ldots, S_K\}$ *and assume that the weights under every scenario are nonnegative. Define* $w_e = \frac{1}{K}\sum_{i=1}^{K} w_e^{S_i}$ *for all* $e \in E$. *Let* Y *be an optimal solution to the deterministic problem* $\mathcal{P}$ *with weights* $(w_e)_{e\in E}$. *Then* $Z(Y) \leq K * OPT$, *where* OPT *is the maximal regret of the optimal robust solution.*

Proof. Let X be an optimal robust solution. It holds

$$Z(X) = \max_{i=1,\ldots,K}\{F(X,S_i) - F^*(S_i)\} \geq \frac{1}{K}\sum_{i=1}^{K}(F(X,S_i) - F^*(S_i)). \qquad (A.5)$$

Since $\frac{1}{K}\sum_{i=1}^{K} F(Y,S_i) \leq \frac{1}{K}\sum_{i=1}^{K} F(X,S_i)$ and there are no negative weights under all scenarios we obtain from (A.5)

$$Z(X) \geq \frac{1}{K}\sum_{i=1}^{K}(F(Y,S_i) - F^*(S_i)) \geq \frac{1}{K}\max_{i=1,\ldots,K}\{F(Y,S_i) - F^*(S_i)\} = \frac{1}{K}Z(Y)$$

and theorem follows since $Z(X) = OPT \geq \frac{1}{K}Z(Y)$. □

From Theorem A.5 we immediately get a K-approximation algorithm for every problem with K scenarios, under the assumption that the underlying deterministic problem $\mathcal{P}$ is polynomially solvable and the weights are nonnegative under every scenario. In particular, we get 2-approximation algorithm for the NP-hard problems with 2-scenarios. Observe, however, that the worst case ratio of the algorithm increases with the number of scenarios. We do not know whether there is an approximation algorithm with constant worst case ratio if the number of scenarios is unbounded. No such algorithm is known even for some particular problems. In contrast, for the interval uncertainty representation, in which the number of scenarios is also unbounded, we have an approximation algorithm with worst case ratio of 2 for the general problem. This is `Algorithm AM`, which we have designed in Section 4.1. In the previous section we have shown some negative results concerning the approximation of D-Minmax Regret $\mathcal{P}$ for unbounded number of scenarios. Notice that no such negative results are known for the problems with interval uncertainty representation.

We present now another known result on D-Minmax Regret $\mathcal{P}$. Denote by $|I|$ the size of the instance of the problem and by UB and LB the upper and lower bounds on the maximal regret of the optimal robust solution. The following theorem, which is similar to that obtained for the interval case (see Theorem 4.8), holds:

Theorem A.6 ([9]). *Given the* D-Minmax Regret $\mathcal{P}$ *problem. If*

1. *for any instance* I *a lower and upper bounds* LB *and* UB *can be computed in time* $p(|I|)$, *such that* $UB \leq q(|I|)LB$, *where* p *and* q *are two polynomials with* q *nondecreasing and* $p(|I|) \geq 1$,
2. *and there exists an algorithm that finds for any instance* I *an optimal robust solution in time* $r(|I|, UB)$, *where* r *is a nondecreasing polynomial,*

then D-Minmax Regret $\mathcal{P}$ *admits an FPTAS.*

If the problem fulfills the assumptions of Theorem A.6, then the scaling and rounding technique can be applied to obtain an FPTAS (see [9] for details).

Suppose that the number of scenarios is bounded by a constant $K > 1$ and the underlying deterministic problem $\mathcal{P}$ is polynomially solvable. Then the assumption 1 of Theorem A.6 can be fulfilled by applying Theorem A.5. That is, we can compute in polynomial time solution Y such that $Z(Y) \leq K * OPT$ and fix $UB = Z(Y)$ and $LB = Z(Y)/K$. Notice that in the assumption 2 we require a pseudopolynomial algorithm to solve the problem. Aissi *et al.* [9] constructed such pseudopolynomial algorithms for D-Minmax Regret Shortest Path and D-Minmax Regret Minimum Spanning Tree. Consequently, both problems admit a fully polynomial time approximation scheme if the number of scenarios is bounded by a constant K. Unfortunately, the running time of the approximation schemes is exponential in K. Therefore they can be applied to the problems in which the number of scenarios is not very large.

Index