Department of Fundamentals of Computer Science
Wrocław University of Science and Technology

Recoverable Robust Discrete Optimization Problems under Interval Uncertainty Representation

Mikita Hradovich

DOCTORAL THESIS

Supervisor
prof. dr. hab. Paweł Zieliński

Wrocław 2021

Dedicated to my beloved wife Anna and daughter Halinka

Acknowledgments

This work would not have been possible without the support of a number of people. First of all I would like to thank my supervisor prof. dr. hab. Paweł Zieliński not only for his example and inspiration but also for knowledge, experience, willingness to help and also for his patience. I am also thankful to prof. dr. hab. Adam Kasperski for his knowledge, example and openness.

I would like to thank all my colleagues at Wrocław University of Science and Technology. I would like to express my gratitude to dr. Maciej Gębala for his cheerful nature and a sense of humor, to dr. Zbigniew Gołębiewski for his demanding but also extremely rewarding approach to education and to dr. hab. Szymon Żeberski for being a great example of a teacher. Furthermore, I would like to thank prof. dr. hab. Jacek Cichoń, dr. Rafał Kapelko, dr. Marcin Kik, dr. Przemysław Kobylański, prof. dr. hab. Mirosław Kutyłowski, dr. Małgorzata Sulkowska, dr. Filip Zagórski and dr. Marcin Zawada.

Finally, I thank all of my friends and my family for their support and patience. In particular I would like to thank my wife Hanna who inspired and supported me on many occasions and my brother Ilya Hradovich for his support, deep insights and fruitful discussions.

Credits

Results presented in this thesis is an outcome of a joint work previously published as articles in different journals, i.e.:

- Chapter 2 is based on a joint work with Adam Kasperski and Paweł Zieliński, namely "The recoverable robust spanning tree problem with interval costs is polynomially solvable" appeared in *Optimization Letters* [34] and "Recoverable robust spanning tree problem under interval uncertainty representations" appeared in *Journal of Combinatorial Optimization* [33].

- Chapter 3 is also based on on a joint work with Adam Kasperski and Paweł Zieliński. A work titled "Robust recoverable 0-1 optimization problems under polyhedral uncertainty" appeared in *European Journal of Operational Research* [36].

- The research presented in this work was possible thanks to the support of the National Science Centre, Poland, grant 2017/25/B/ST6/00486 and Wrocław University of Science and Technology, grant S50129/K1102.

Contents

Introduction

Combinatorial optimization problems are a class of problems in which we are looking for the "best" configuration or a set of parameters to meet some objective. There is a number of applications of combinatorial optimization in real-world situations, including logistics, production planning, package delivery, designing networks, scheduling and more. Among widely known combinatorial optimization problems there are the knapsack problem, the minimum spanning tree problem, the minimum selection problem, the shortest path problem, etc. [1].

Such an optimization problem is characterized by the cost function and the constraints defining a set of feasible solutions. According to a classical approach the exact values of all problem parameters are known in advance [21, 61]. Such a model is called deterministic. However in many practical applications this is rarely the case. In many instances parameter values can not be measured precisely but can only be estimated. They may also depend on some future events or they may belong to an one-of-a-kind type of problem, where no historical data exists, e.g., a realization of a specific project or development of the stock market.

A widely adopted way to model such uncertainty is to introduce a scenario set [50]. A scenario set is a set of all possible realizations of parameters. Each individual realization of parameters is then called a scenario. Depending on the utilized model and available data it is possible but not necessary that a probability distribution for a scenario set is available. This probability distribution indicates a likelihood of a specific scenario to occur. There are several ways to define a scenario set, among the most common are discrete and interval uncertainty representations. Additionally, some modifications to the scenario sets may be introduced. For example, scenario sets with a budget constraint allow us to control the degree of uncertainty by a number of parameters which are allowed to deviate from their nominal values or by a maximum total deviation allowed from the nominal values of parameters.

There are two main approaches to deal with the uncertainty: *stochastic programming* and *robust optimization*. Stochastic programming requires probability distributions of uncertain parameters to be known and it attempts to find some solution feasible (i.e., eligible, "legal") under a number of scenarios and it minimizes some stochastic criterion, typically the expected cost [11]. There are several drawbacks and limitations to stochastic programming. For instance, this approach is applicable only if a probability distribution in the scenario set is known or can be estimated. Moreover, minimizing the expected cost

for a solution that is used only once may be questionable. For a more detailed description of stochastic programming the reader may also want to check Yu and Li [70] and bibliography compiled by van der Vlerk [65].

In contrast to stochastic programming, *robust optimization* tackles uncertainty by being extremely risk averse and taking into account only the worst case scenario while maintaining the feasibility under all considered scenarios. This approach can be used, for example, when dealing with high risk decisions, such as related to power supply, public health or transportation. Robust optimization is applied when no probability distribution in the scenario set is given. Solution is chosen using several proposed robust decision rules, also called criteria, (see, e.g., [62]), namely the min-max criterion, under which we choose a solution minimizing the largest cost over all scenarios or the min-max-regret criterion, under which we choose a solution minimizing the maximum difference from an optimal solution over all scenarios.

Researches were interested whether the robust optimization problems remain computationally easy, or tractable, under different scenario sets. Additionally, new types of scenario sets were introduced over time to address concerns over conservatism of the robust approach and in attempt to somewhat relax it, see, e.g., [7, 9, 10]. Bertsimas and Sim in [8] investigate tractability of robust optimization problems under scenario set with a budget constraint.

Traditional robust approach has one stage nature. This means that a complete solution must be decided before the true scenario is revealed. Thus, we have to pay a cost of the solution induced by the actual scenario. In practice, a limited action is often allowed after the true scenario is revealed. It is possible to construct a complete solution in a number of stages or to partially modify a solution after the true scenario is revealed. In a *two-stage robust* approach we first construct a partial first stage solution and then complement it after the true scenario is revealed. In a *recoverable robust* approach a complete solution is decided in the first stage. It is allowed to modify this solution after the true scenario is revealed with a *recourse action*. It is assumed that a modified solution resides in a neighborhood of the first stage solution defined by some distance function. In both two-stage approaches the goal is to find a solution which minimizes overall cost in a worst case. Multi-stage problems have gathered a considerable attention in the recent literature. As an example, two-stage problems were discussed in [45, 47, 49]. In turn a recoverable model was first proposed for a linear programming in [53]. The recoverable robustness was studied in [6, 53] and recently in [27] and in [32]. It was later applied to the selected discrete optimization problems in [13, 14, 15, 17, 33, 34]. This approach also has its drawbacks, namely it increases the complexity of a problem rendering most of them NP-hard. On the other hand, it comes in handy for solving many real world problems [16, 19, 20, 24, 53, 73].

It is important to note here that both two-stage and recoverable models are not required to be used in conjunction with the robust framework. In this case the recoverable model is reduced to a problem of finding the pair of solutions, the first stage solution and the second stage solution in its neighborhood, which minimizes overall cost. Similarly, in a two-stage model partial solutions are determined in a deterministic setting.

Multi-stage recoverable models also address another limitation of the robust optimiza-

tion, namely, extremely conservative, risk-averse approach to problem solving which may be excessive in some applications. Finding a robust solution which is feasible under all constraints and scenarios is demanding and may not be achievable at all. But even if such a solution exists, it may generate high costs in the rest of the scenarios. Some of the drawbacks of the robust optimization have been already discussed in the foundational work by Soyster in [64]. *Recoverable robust* models discussed in this thesis are designed to tackle the limitations of the pure robust approach.

Yager *et al.* in [68] proposes *Ordered Weighted Averaging* (OWA) aggregation operator. This operator is commonly used to aggregate criteria in multiobjective decision problems (see, e.g., [29, 59, 69]) and also to choose a solution under the discrete uncertainty representation. The OWA operator generalizes several criteria such as: the maximum, minimum, average or median criterion traditionally used in decision making under uncertainty.

Thesis outline

Chapter 1 contains a short introduction to the recoverable robust models discussed in the thesis providing the necessary definitions and some known complexity results. Chapter 2 and Chapter 3 constitute the substantial part of the thesis. These chapters are based on the published papers (see [34, 35, 36]). Appendix A dives into technical details related to implementation of software package used in Chapter 3. The results presented in this thesis relate to actively researched problems and are direct continuation of research results conducted by several research groups. For instance, Christina Büsing in [13] considers a recoverable robust spanning tree problem and provides some results for this problem. Our contribution has already been recognized by a scientific community. As an example, Lendl *et al.* in [52] have generalized and improved some results related to a recoverable robust matroid basis problem and Fischer *et al.* in [27] also refer to results related to the recoverable robust matroid basis problem. We provide a brief overview of each chapter below.

Chapter 1: Formal model, notation, definitions and complexity. In this chapter we successively introduce concepts of robustness and recoverability in the context of discrete optimization problems to the reader. We also provide definitions of different neighborhoods, uncertainty sets, and related problems. The chapter also has a brief summary of the complexity of recoverable robust problems under multiple uncertainty sets. We conclude the chapter with a variety of definitions necessary for understanding this thesis.

Chapter 2: The recoverable robust spanning tree problem. Here we consider a recoverable robust spanning tree problem with uncertain costs modeled as classical intervals. In this case an uncertainty set $\mathcal{U}$ is the Cartesian product of uncertain intervals. The complexity of the recoverable robust version of the spanning tree problem was not known at that moment. We first managed to prove that the problem has polynomial complexity and built a polynomial algorithm for it. It also turned out that the proposed algorithm can

be generalized to solve the recoverable robust matroid basis problem. We later construct a combinatorial algorithm to solve the recoverable robust spanning tree problem under the interval uncertainty representation in a strongly polynomial time. Moreover, with this algorithm we obtained several approximation results for the recoverable robust spanning tree problem under the Bertsimas and Sim [8, 9] interval uncertainty representation and the interval uncertainty representation with a continuous budget constraint.

Chapter 3: Robust recoverable 0-1 optimization problems under polyhedral uncertainty. In this chapter we discuss the robust recoverable approach to 0-1 programming problems. As before, it is assumed that the initial solution is built in the first stage and later can be modified to some extent in the second stage. This modification consists of choosing a solution in some defined neighborhood of the first stage solution. The second stage costs can be uncertain and this uncertainty is modeled using the interval budgeted uncertainty. The resulting robust recoverable problem is a min-max-min problem, which can be difficult to solve when it has a large number of variables. To tackle this limitation, several lower bounds of the optimal solution and also approximate solutions are proposed, which can be used for a wide class of 0-1 optimization problems. The results of computational tests for two problems, namely the minimum assignment problem and the minimum knapsack problem are presented.

Chapter 4: Summary and conclusions. This chapter briefly summarizes the thesis and highlights prospects of future research.

Chapter 1

Formal model, notation, definitions and complexity

1.1 Linear combinatorial optimization problems

In this chapter we are concerned with a class of *optimization problems*. In this class of problems we are looking for a solution which is the "best" under some decision rule, or criterion. In the most generic way the problem can be defined as follows:

$$\mathcal{P} : \min_{X \in \Phi} f(X) \tag{1.1}$$

where $f : \Phi \to \mathbb{R}$ is a cost function and Φ is a *set of feasible solutions*.

We call an optimization problem *continuous* if its variables are continuous, whereas an optimization problem with discrete variables is called a *combinatorial* (*discrete*) problem. In the combinatorial problems we are looking for an object from some finite set, e.g., a set, permutation or graph. On the other hand, in the continuous problems we are looking for a set of real numbers or a function.

Let $E = \{e_1, \ldots, e_n\}$ be a finite set of elements and $\Phi \subseteq 2^E$ be a set of feasible solutions, a set of subsets of E. A linear *cost function* $c : E \to \mathbb{R}_{\geq 0}$ assigns a non-negative cost for each element $e \in E$. We seek a feasible solution $X^* \in \Phi$ with the minimum cost $f(X^*) = \sum_{e \in X^*} c(e)$. Thus, (1.1) becomes a *linear combinatorial minimization problem*. As an example, E can be defined as a set of edges in a graph $G = (V, E)$ and Φ to contain objects like spanning trees or paths, so we get problems of finding a minimum spanning tree or a shortest path.

In most cases problem $\mathcal{P}$ can be reformulated as a *0-1 programming problem*. For this purpose we associate a vector of binary decision variables $\boldsymbol{x} = (x_1, \ldots, x_n) \in \{0, 1\}^n$ with an each element $e_i \in E$, $i \in \{1, \ldots, n\}$. With c_i we denote a non-negative cost of element $e_i \in E$. Thus, $\mathcal{P}$ has the following formulation:

$$\begin{aligned} \mathcal{P} : \quad & \min \textstyle\sum_{i=1}^{n} c_i x_i \\ & \text{s.t. } (x_1, \ldots, x_n) \in \Phi, \end{aligned} \tag{1.2}$$

where Φ is a set of characteristic vectors describing feasibility of a solution with a system of constraints involving $\boldsymbol{x} = (x_1, \ldots, x_n)^T$. We do not make any additional assumptions about the problem $\mathcal{P}$. In particular, $\mathcal{P}$ can be NP-hard (see, e.g., [21]) and not at all approximable (see Definition 1.3).

We will now provide a few examples of linear combinatorial optimization problems.

Minimum selection problem We are given a set of items E ($|E| = n$). A non-negative cost c_e is associated with each element $e \in E$. In a *minimum selection problem* we are looking for such subset X of size p of elements E so that the total cost of its constituent elements is minimum. For this problem a set of feasible solutions is defined as $\Phi = \{X \subseteq E : |X| = p\}$.

Minimum assignment problem We are given a bipartite graph $G = (V, E)$, where V is a set of vertices and E is a set of edges. Each edge has a non-negative cost c_e, $e \in E$ associated with it. Recall that an *independent set* is a set of vertices where none of them are adjacent. In this problem we are looking for a *perfect matching* of the minimum cost, where the perfect matching is a set of pairwise non-adjacent edges covering all vertices in the graph G. Thus, Φ consists of all perfect matchings in G. The problem may be seen as a problem of assigning executors to tasks so that the cost of completing the tasks is minimum.

Minimum spanning tree problem We are given a connected graph $G = (V, E)$, in which a non-negative cost c_e is associated with each edge $e \in E$. A set of feasible solutions Φ contains all spanning trees in G, where a spanning tree is an acyclic subgraph of G that connects all the nodes in V. We are looking for a spanning tree $X \in \Phi$ whose sum of edge costs is minimum.

Shortest path problem We are given a directed graph $G = (V, E)$ with distinguished vertices s and t, called a *source* node and a *terminal* node respectively. A non-negative cost c_e is associated with each arc $e \in E$. In this problem we are looking for a path from s to t such that the sum of the costs of its constituent arcs is minimal. Set of feasible solutions Φ consists of all paths from the source node s to the terminal node t in graph G.

Minimum matroid basis problem Let $M = (E, \mathcal{I})$ be a matroid (see Definition 1.6), where E is a non-empty ground set and $\mathcal{I}$ is a family of subsets of E. A *basis* of a matroid M is a maximal under inclusion element of $\mathcal{I}$. A non-negative cost c_e is associated with each element $e \in E$. In a *minimum matroid basis problem* we are looking for the basis of the minimum total cost. A set of feasible solutions Φ contains all bases in M.

Minimum s-t cut Let $G = (V, E)$ be a directed graph with distinguished source and sink nodes s and t. An *s-t cut* is partition of the set of vertices V of a graph into two disjoint subsets, so that nodes s and t belong to the different subsets. An $s - t$ cut only

includes edges going from the source set to the sink side. Each edge has a non-negative capacity assigned to it. We seek an $s-t$ cut with the minimum total capacity. A set of feasible solutions Φ consists of all s $s-t$-cuts in graph G.

Travelling salesman problem We are given a complete graph $G = (V, E)$, where $|V| = n$. Each edge in E has a cost (or distance) assigned to it. A *tour* is a closed path that visits each node exactly once. In this problem we want to find the tour of the minimum total cost (length). A set of feasible solutions Φ consists of all cyclic permutations π on n elements.

1.2 Robustness and recoverability in combinatorial optimization problems

1.2.1 Uncertainty representation

Let us now assume that costs assigned to elements $e \in E$ can be uncertain. We model this uncertainty by introduction of a scenario set $\mathcal{U}$ containing all possible realizations of the uncertain costs. Each particular realization of the element cost $S = (c_{e_1}^S, ..., c_{e_n}^S) \in \mathcal{U}$ is called a *scenario*, $S \in \mathcal{U}$.

Over the years researchers have proposed several ways to define a scenario set $\mathcal{U}$. Two commonly utilized approaches are *discrete uncertainty representation* and *interval uncertainty representation* (see, e.g., [2, 46, 50]). The discrete uncertainty scenario set is finite and consists of $K \geq 1$ explicitly enumerated scenarios, each of the scenarios defines some realization of costs values:

$$\mathcal{U}^D = \{S_1 = (c_e^{S_1})_{e\in E}, \ldots, S_K = (c_e^{S_K})_{e\in E}\}. \tag{1.3}$$

The scenarios of $\mathcal{U}^D$ can be seen as a result of a global events influencing every parameter.

Under the *interval uncertainty representation* parameter values for each element $e \in E$ is known to belong to the closed interval $[c_e, c_e + d_e]$ where c_e is a *nominal cost* of $e \in E$ and $d_e \geq 0$ is the maximum deviation of the cost of e from its nominal value. Thus, scenario set is defined as the Cartesian product of all these closed intervals [9, 50]:

$$\mathcal{U}^I = \{S = (c_e^S)_{e\in E} \,:\, c_e^S \in [c_e, c_e + d_e], e \in E\} = \prod_{e\in E}[c_e, c_e + d_e]. \tag{1.4}$$

Note that the scenario set defined in this way allows parameters to vary their values independently of each other.

Several variations of the interval uncertainty representation were proposed over the years. In [8, 9] a modification of the scenario set $\mathcal{U}^I$ has been introduced. This modified scenario set is called a *discrete interval budgeted uncertainty* and is denoted by $\mathcal{U}_1^I(\Gamma)$, is a subset of $\mathcal{U}^I$ such that under each scenario in $\mathcal{U}_1^I(\Gamma)$, the costs of at most Γ elements are

greater than their nominal values c_e. Here Γ, called a budget, is assumed to be a fixed integer in $[0, n]$ ($|E| = n$). Scenario set $\mathcal{U}_1^I(\Gamma)$ is formally defined as follows:

$$\mathcal{U}_1^I(\Gamma) = \{S = (c_e^S)_{e\in E} : c_e^S \in [c_e, c_e + \delta_e d_e], \delta_e \in \{0, 1\}, e \in E, \sum_{e\in E} \delta_e \leq \Gamma\}. \tag{1.5}$$

The parameter Γ allows us to model the degree of uncertainty. Namely, if $\Gamma = n$ then $U_1^I(\Gamma)$ is equivalent to U^I, on the other hand, if $\Gamma = 0$ then $U_1^I(\Gamma)$ contains only the nominal scenario, thus there is no uncertainty in this model.

There is another interesting variation of $\mathcal{U}^I$ which allows to control the amount of uncertainty. It is called *continuous budgeted interval uncertainty* (see, e.g., [58]), denoted by $\mathcal{U}_2^I(\Gamma)$ and defined as follows:

$$\mathcal{U}_2^I(\Gamma) = \{S = (c_e^S)_{e\in E} : c_e^S = c_e + \delta_e, \delta_e \in [0, d_e], e \in E, \sum_{e\in E} \delta_e \leq \Gamma\}, \tag{1.6}$$

where $\Gamma \geq 0$ is a a fixed parameter that can be seen as a budget of an adversary, and represents the maximum total increase of the element costs from their nominal values (an adversary is a malicious entity whose goal is to prevent the decision maker from achieving their optimization goal). Note that similarly to $\mathcal{U}_1^I(\Gamma)$ for sufficiently large Γ, $\mathcal{U}_2^I(\Gamma)$ reduces to the traditional interval uncertainty representation $\mathcal{U}^I$ and if $\Gamma = 0$ then $U_1^I(\Gamma)$ contains only the nominal scenario.

The last uncertainty set we will discuss in this thesis is a special case of $\mathcal{U}_2^I(\Gamma)$ called *polyhedral uncertainty set* $\mathcal{U}_3^I(\Gamma)$. It is defined as follows:

$$\mathcal{U}_3^I(\Gamma) = \{S = (c_e^S)_{e\in E} : c_e^S = c_e + \delta_e, e \in E, (\delta_e)_{e\in E} \subseteq \mathcal{V}, \sum_{e\in E} \delta_e \leq \Gamma\}, \tag{1.7}$$

where $\mathcal{V} \subseteq \mathbb{R}^n$ is a polyhedron described by some additional linear constraints involving $\delta_{e\in E}$. To ensure that $S \in \mathcal{U}_3^I(\Gamma)$, we assume that $\mathbf{0} \in \mathcal{V}$, where $\mathbf{0}$ is a zero vector of size n. One can use $\mathcal{V}$ to model some additional relationships among the uncertainty of the costs. For example, a constraint $\sum_{i\in A} \delta_i \leq \Gamma_A$ represents the situation in which a subset of the costs has its own budget $\Gamma_A \leq \Gamma$. Other constraints of the form $\alpha_{ij}\delta_i \leq \delta_j \leq \beta_{ij}\delta_i$, for some fixed $\alpha_{ij} \leq \beta_{ij}$, model a possible correlation between δ_i and δ_j. Note that if $\mathcal{V} = \mathbb{R}^n$, then we get $\mathcal{U}_2^I(\Gamma)$ [58]. As $\mathcal{U}_3^I(\Gamma) \subseteq \mathcal{U}_2^I(\Gamma)$, the set $\mathcal{U}_2^I(\Gamma)$ is a relaxation of $\mathcal{U}_3^I(\Gamma)$.

This section does not cover all possible methods to model uncertainty. The interested reader may also want to check ellipsoidal uncertainty sets [7, 30, 31, 63] among the others.

1.2.2 Robust approach to combinatorial optimization problems under uncertainty

Risk-averse decision makers may be interested only in the worst case scenario. Robust optimization framework was initially proposed in [64] and further discussed in a number of papers, e.g., [5, 7, 8, 50, 71]). In most common robust optimization models we are

seeking a solution that minimizes the largest cost, which is also called *min-max* criterion or a solution that minimizes maximum deviation from the optimum, which is also known as *min-max regret* criterion (savage criterion, maximum opportunity loss) over scenario set. The above criteria are known and frequently used in the cases where exact values of parameters are not known and probability distribution over scenarios in $\mathcal{U}$ can not be estimated.

We will now denote the total cost under scenario S by $f(X,S) = \sum_{e \in X} c_e^S$ and by $f^*(\hat{S})$ we denote the cost of an optimal solution under a fixed scenario $\hat{S}$. Now, robust combinatorial optimization problems can be defined as follows:

$$\text{MinMax } \mathcal{P} : \min_{X \in \Phi} F(X) = \min_{X \in \Phi} \max_{S \in \mathcal{U}} f(X,S), \tag{1.8}$$

$$\text{MinMax Regret } \mathcal{P} : \min_{X \in \Phi} Z(X) = \min_{X \in \Phi} \max_{S \in \mathcal{U}} \{f(X,S) - f^*(S)\}. \tag{1.9}$$

Here $F(X)$ and $Z(X)$, $X \in \Phi$ stand for respectively the maximum cost of a solution X over all scenarios and the maximum regret of the solution X over all scenarios.

The robust approach has its disadvantages. A serious drawback of the min-max criterion is that it is extremely conservative and takes into consideration only the worst-case scenario. Additionally, selected solution may not be Pareto optimal [46]. This may lead to a so called *drowning effect*, i.e., when only one extremely bad scenario is taken into account, leaving the rest of scenarios completely ignored [25]. In turn, in the min-max regret criterion we minimize the maximum opportunity lost (see, e.g., [55]), in other words the cost of the solution compared to the optimal solution. Decision makers should be aware that the solution selected with respect to this criterion may have a large solution cost (this feature is also known as the *price of robustness*, see, e.g., [9]).

In order to mitigate the above drawbacks of the robust optimization several modifications to the model were proposed, the *recoverable robust* optimization is one of them.

1.2.3 Multi-stage combinatorial optimization

In practice some applications of combinatorial optimization problems have a multi-stage nature. In other words the decision maker may posses an ability to slightly alter an initial solution at a later stage of the problem solution. It is rarely possible to exchange the whole solution with a different one, but only to replace a given fraction of the elements in the current solution. One of the models which take such a situation into account is the *recoverable model*.

We will first define a class of *incremental problems* (Inc $\mathcal{P}$) [22]. Let $X \in \Phi$ be an initial feasible solution. A new feasible solution $Y \in \Phi$ is an increment from X by the amount allowed by a distance function

$$d(X,Y) : \Phi \times \Phi \to \mathbb{Z}_{\geq 0}.$$

Function $d(X,Y)$ is called an *incremental function*. It defines a neighborhood $\Phi_X^k \subseteq \Phi$ of solution X, where $k \in \{0, \ldots, n-1\}$ is a *recovery parameter*:

$$\Phi_X^k = \{Y \in \Phi : d(X,Y) \leq k\}. \tag{1.10}$$

We will also use the following definition of neighborhood for a fixed $\alpha \in [0,1]$:

$$\Phi_X^\alpha = \{Y \in \Phi : d(X,Y) \leq \alpha|X|\}. \tag{1.11}$$

Consequently, $Y \in \Phi_X, \Phi_X \in \{\Phi_X^k, \Phi_X^\alpha\}$ is called an *incremental solution*.

Neighborhood Φ_X is called an *element inclusion neighborhood* if the incremental function is defined as

$$d(X,Y) = |Y \setminus X|, \tag{1.12}$$

i.e., d returns a number of elements in the incremental solution Y and not in the starting solution X. Neighborhood Φ_X is called an *element exclusion neighborhood* if incremental function is defined as

$$d(X,Y) = |X \setminus Y|, \tag{1.13}$$

i.e., d returns a number of elements in the starting solution X and not in the incremental solution Y. Finally, *symmetric difference neighborhood* uses the incremental function defined as

$$d(X,Y) = X \oplus Y = (X \setminus Y) \cup (Y \setminus X). \tag{1.14}$$

It is worth noting that all the above neighborhoods, namely the element inclusion, the element exclusion and the symmetric difference neighborhoods are equivalent if $|X| = m$, $m \in \{1, \ldots, n\}$ for each $X \in \Phi$. We will call a problem $\mathcal{P}$ possessing such a property an *equal cardinality* problem. For Φ_X^k and Φ_X^α respectively we can then write

$$\Phi_X^k = \{Y \in \Phi : |X \cap Y| \geq m - k\} \tag{1.15}$$

and

$$\Phi_X^\alpha = \{Y \in \Phi : |X \cap Y| \geq m(1-\alpha)\}, \tag{1.16}$$

where $l = \lceil m(1-\alpha) \rceil$ is a fixed integer, independent from solution X.

The *incremental problem* INC $\mathcal{P}$ is defined as follows:

$$\text{INC } \mathcal{P} : \min_{Y \in \Phi_X} \sum_{e \in Y} c_e. \tag{1.17}$$

In the incremental problem we seek for the cheapest solution within a neighborhood $\Phi_X \in \{\Phi_X^k, \Phi_X^\alpha\}$ of a fixed solution $X \in \Phi$. Şeref *et al.* in [22] examine a class of incremental network problems and show that the incremental version of some basic network problems (for example, the shortest path) can be already NP-hard for some natural neighborhood definitions.

The *recoverable model* consists of two stages. We are given costs C_e, $e \in E$ of a first stage solution X. The total cost of the *first stage* solution $X \in \Phi$ is equal to $\sum_{e \in X} C_e$. Additionally a neighborhood Φ_X of the first stage solution X is defined. Each solution Y in this neighborhood Φ_X is within some predefined distance $d(X,Y)$ from the first stage solution X. In the *second stage* the first stage X solution can be replaced with a solution in its neighborhood. The second stage costs are designated as c_e, $e \in E$. Similarly, the cost of the second stage solution is equal to $\sum_{e \in Y} c_e$. The goal is to compute a pair of

solutions, namely the *first stage solution* and the *second stage solution* in the neighborhood of X, which minimize the total first and second stage costs. We, thus, seek the first stage solution $X \in \Phi$ and the second stage solution $Y \in \Phi_X$, so that the total cost of X and Y for respectively C_e and c_e is minimum. The *recoverable* problem (REC $\mathcal{P}$ for short) can be stated formally as follows:

$$\text{REC } \mathcal{P}: \min_{X\in\Phi}\left(\sum_{e\in X} C_e + \min_{Y\in\Phi_X}\sum_{e\in Y} c_e\right). \tag{1.18}$$

Note that the recoverable problems can be seen as a generalization of the class of incremental problems INC $\mathcal{P}$ (1.17).

1.2.4 Recoverable robust combinatorial optimization

Let us now turn our attention to *recoverable robust* models for combinatorial optimization problems with uncertain element costs under scenario uncertainty representation.

Let us assume that the second stage costs are uncertain, then the robust approach can be utilized (see Section 1.2.2 and [5, 50]). In this model we look for pairs of solutions which minimize the total cost in the worst case. Recoverable robustness was first proposed in [53] and it is similar to the *adjustable robustness* from the field of mathematical programming [6]. It was also previously discussed in [13, 14, 15, 17, 47].

Similarly to robust models, recoverable robust models also utilize scenario sets. In these models the second stage cost of Y under the scenario $S \in \mathcal{U}$ is equal to $\sum_{e\in Y} c_e^S$. The goal is to find a pair of the first stage solution $X \in \Phi$ and the second stage solution $Y \in \Phi_X^k$ so that the sum of the first and the second stage costs $\sum_{e\in X} C_e + \sum_{e\in Y} c_e^S$ in the worst case is minimum:

$$\text{RR } \mathcal{P}: \min_{X\in\Phi}\left(\sum_{e\in X} C_e + \max_{S\in\mathcal{U}}\min_{Y\in\Phi_X}\sum_{e\in Y} c_e^S\right). \tag{1.19}$$

The robust recoverable versions of some basic combinatorial problems such as the shortest path, minimum $s-t$ cut, minimum knapsack, and minimum matroid base, were first studied in [13, 14], where several complexity results for them were obtained. Recently the robust recoverable versions of the selection and traveling salesman problems were discussed in [17, 18, 47].

The recoverable robust problem generalizes the recoverable problem (1.18). It is enough to set $\mathcal{U}$ so that it contains single scenario S. Assume that neighborhood of X contains only one solution, which is X itself. For example, this property holds for neighborhood Φ_X^α when $\mathcal{P}$ is an equal cardinality problem and $\alpha = 0$. If additionally the first stage costs $C_e = 0$ for each $e \in E$, than RR $\mathcal{P}$ can be rewritten as

$$\min_{X\in\Phi}\max_{S\in\mathcal{U}}\sum_{e\in X} c_e^S,$$

which is a traditional *single-stage robust min-max problem* (see (1.8)).

The recoverable robust problem also generalizes the following *evaluation problem*:

$$\textsc{Eval}(X)\ \mathcal{P}:\ \sum_{e\in X} C_e + \max_{S\in\mathcal{U}} \min_{Y\in\Phi_X} \sum_{e\in Y} c_e^S, \tag{1.20}$$

where the inner maximization problem

$$\textsc{Adv}(X)\ \mathcal{P}:\ \max_{S\in\mathcal{U}} \min_{Y\in\Phi_X} \sum_{e\in Y} c_e^S \tag{1.21}$$

is called the *adversarial problem* [58].

Observe that solving RR $\mathcal{P}$ consists of minimizing $\textsc{Eval}(X)$ over $X \in \Phi$. The evaluation problem generalizes the adversarial problem, which in turn generalizes the incremental one, as we get the latter by fixing $C_e = 0$ for each $e \in E$:

$$\begin{aligned}\text{RR } \mathcal{P}:\ \min_{X\in\Phi} \textsc{Eval}(X) &= \min_{X\in\Phi} \left(\sum_{e\in X} C_e + \textsc{Adv}(X) \right) \\ &= \min_{X\in\Phi} \left(\sum_{e\in X} C_e + \max_{S\in\mathcal{U}} \textsc{Inc}(X,S) \right).\end{aligned}$$

1.2.5 Other interesting models

Two-stage robust model A *two-stage robust* model is another example of a multi-stage robust models. We define it as follows. Suppose that $X \in \Phi_1$ (Φ_1 is a set of partial solutions defined for a problem) is a partial solution that we can form in the first stage. As before, by C_e and c_e^S we denote first stage costs and second stage costs under scenario $S \in \mathcal{U}$, respectively. The cost of a first-stage solution is equal to $\sum_{e\in X} C_e$, $X \in \Phi_1$. Then, after a scenario $S \in \mathcal{U}$ is revealed, decision maker completes the problem solution to $X \cup Y \in \Phi$ with a cost $\sum_{e\in Y} c_e^S$. The *two-stage robust* problem is formally defined as follows:

$$\text{2SR } \mathcal{P}:\ \min_{X\in\Phi_1} \left\{ \sum_{e\in X} C_e + \max_{S\in\mathcal{U}} \min_{\{Y: X\cup Y\in\Phi\}} \sum_{e\in Y} c_e^S \right\}. \tag{1.22}$$

The 2SR $\mathcal{P}$ problem has been applied to a number of various combinatorial problems $\mathcal{P}$: the matching problem [49], the minimum spanning tree problem [45], selected network problems [48] and the selection problem [47]. It was shown that under the interval uncertainty representation $\mathcal{U}^I$, the problem is polynomially solvable, when $\mathcal{P}$ is polynomially solvable. On the other hand, under the discrete uncertainty representation $\mathcal{U}^D$, 2SR $\mathcal{P}$ is NP-hard for all considered basic problems. Some approximation ratios for the scenario set $\mathcal{U}^D$ were achieved by applying randomized algorithms [45, 49, 47].

A key difference between the two-stage robust model and the recoverable robust model is that the former takes into account costs of the selected items only once, while the latter takes them into account once in first stage and once in the second stage with the possibility of replacing a set of items from the first to the second stage, controlled by the recovery parameter.

Rent-recoverable robust model In a *rent-recoverable robust* model we are given a *rental factor* $\alpha \in (0,1)$ and *inflation factor* $\beta \geq 0$ [13]. We first define the *rent cost* of solution $X \in \Phi$ under scenario $S \in \mathcal{U}$ by

$$f_R(X,S) = \alpha f(X,S)$$

and the *implementation cost* of solution $X \in \Phi$ under scenario S as

$$f_I(X,S) = \min_{Y \in \Phi}\{(1-\alpha)f(Y,S) + (\alpha+\beta)f(Y \setminus X, S)\},$$

where $f(X,S) = \sum_{e \in X} c_e^S$ is a cost of solution X under scenario S. Now let us define overall rent cost by

$$f_{Rent}(X) = \max_{S \in \mathcal{U}}\{f_R(X,S) + f_I(X,S)\}.$$

Thus, the rent-recoverable robust combinatorial problem (P) can be defined as follows:

$$\text{RENT RR } \mathcal{P}: \ \min_{X \in \Phi} f_{Rent}(X) = \min_{X \in \Phi} \max_{S \in \mathcal{U}}\{f_R(X,S) + f_I(X,S)\}. \tag{1.23}$$

The reader may check paper by Kasperski *et al.* [44] to learn more about rental-recoverable robust model.

Min-max-min robust model The reader may also be interested in a discussion of a *min-max-min* approach by Buchheim and Kurtz in [12], i.e., a two-stage robust model where no first stage variables exist. This model, instead of looking for a single worst-case optimal solution, attempts to calculate a number of solutions. Once the actual scenario is revealed, the best of them is getting adopted. The phase of calculating solutions for a number of scenarios is interpreted as a robust-prepossessing phase. This phase can be performed early and its results stored and reused multiple times when the true scenarios are revealed.

1.3 The computational complexity of problems

The computational complexity of the incremental (1.17), adversarial (1.21), recoverable (1.18), and recoverable robust (1.19) problems is higher than the complexity of the generic deterministic problem $\mathcal{P}$. It follows that all this problems are NP-hard if $\mathcal{P}$ is already NP-hard. Even the incremental version of $\mathcal{P}$ can be much harder than $\mathcal{P}$ itself. As an example, it has been shown in [22], that the incremental shortest path problem for the element exclusion neighborhood (1.13) is NP-hard and hard to approximate. On the other hand, the incremental shortest path problem with the element inclusion neighborhood (1.12) is polynomially solvable [22]. The incremental minimum assignment problem is equivalent to the minimum exact matching problem, for which no polynomial time algorithm is known [22]. In turn the incremental selection problem can be solved efficiently [47]. This problem posses a matroidal structure, its set of feasible solutions Φ is composed of all bases of an

uniform matroid [61]. Notice that matroidal problems have the equal cardinality property, as each matroid base has the same number of elements. Still, the recoverable version of the shortest path problem is NP-hard and hard to approximate for both element inclusion and element exclusion neighborhoods [14, 22].

Let us take a look at the spanning tree problem variants. Şeref *et al.* in [22] consider *incremental spanning tree* (denoted by Inc ST). In this problem we are given an initial spanning tree X and we seek a spanning tree $Y \in \Phi_X^k$ whose total cost is minimal. Note that this problem is the inner problem for RR ST, where X is fixed and $\mathcal{U}$ contains only one scenario. Authors have shown in [22] that Inc ST for the element inclusion neighborhood (1.12) can be solved in strongly polynomial time by applying the Lagrangian relaxation technique. Authors also demonstrate several interesting practical applications of the incremental network optimization.

As for the *robust recoverable spanning tree* problem (denoted by RR ST), its complexity depends on the way scenario set $\mathcal{U}$ is defined. For discrete scenario representation $\mathcal{U}^D$ (1.3) the problem is known to be NP-hard even for two scenarios, $K = 2$, and any constant recovery parameter $k \in \{0, \ldots, n-1\}$ [44]. Furthermore, it becomes strongly NP-hard and not at all approximable when both K and k are both part of the input [44]. On the other hand, complexity of RR ST with scenario set $\mathcal{U}^I$ is open up to this moment and no strongly polynomial time combinatorial algorithm for Rec ST has been known.

Let us consider *interval uncertainty representation* (1.4). In [13] a polynomial algorithm for the *recoverable robust matroid basis* problem under scenario set $\mathcal{U}^I$ was constructed, provided that the recovery parameter k is constant. In consequence, RR ST under $\mathcal{U}^I$ is also polynomially solvable for constant k. Unfortunately, the algorithm proposed in [13] is exponential in k. Interestingly, the corresponding recoverable robust version of the shortest path problem (Φ is replaced with the set of all $s-t$ paths in G) has been proven to be strongly NP-hard and not at all approximable even if $k = 2$ [14]. The problem RR ST for scenario set $\mathcal{U}_1^I(\Gamma)$ is known to be strongly NP-hard when Γ is a part of input [58]. In fact, the inner adversarial problem Adv ST is already strongly NP-hard (it is equivalent to the problem of finding Γ most vital edges) [28, 54, 58]. Interestingly, the corresponding Min-Max ST problem with $\mathcal{U}_1^I(\Gamma)$ is polynomially solvable [8].

Consider now the problem RR $\mathcal{P}$ under the uncertainty set $\mathcal{U}_2^I(\Gamma)$ (see (1.6)). Obviously, this problem is not easier than Rec $\mathcal{P}$. That is why it is useful to characterize its the complexity when the underlying recoverable problem is polynomially solvable. It has been shown that RR $\mathcal{P}$ is polynomially solvable under $\mathcal{U}_2^I(\Gamma)$, when $\mathcal{P}$ is the selection problem [18]. However, the complexity of RR $\mathcal{P}$ under $\mathcal{U}_2^I(\Gamma)$ for other matroidal problems, in particular for the minimum spanning tree, remains open. Even though the corresponding evaluation problem can be solved in polynomial time [58].

Table 1.1 contains a compilation of the known complexity results for basic, polynomially solvable, problems $\mathcal{P}$ with comparison to their corresponding incremental, recoverable, evaluation and recoverable robust versions under different uncertainty sets.

Under $\mathcal{U}^D$, $|\mathcal{U}^D| = K$, the evaluation problem has the same complexity as the incremental problem, because it reduces to solving K incremental problems Inc $\mathcal{P}$ for each $S \in \mathcal{U}$. The single-stage robust min-max problem under discrete uncertainty representa-

Table 1.1: Summary of complexity results for basic problems $\mathcal{P}$: SEL - the selection problem, ST - the minimum spanning tree problem, SP_I - the shortest path problem with the element inclusion neighborhood, SP_E - the shortest path problem with the element exclusion neighborhood; **P** - polynomially solvable, **H** - NP-hard.

			$\mathcal{U}_1^I(\Gamma)$		$\mathcal{U}_2^I(\Gamma)$		$\mathcal{U}^D$	
$\mathcal{P}$	Inc	Rec	Eval	RR	Eval	RR	Eval	RR
SEL	**P** [47]	**P** [47]	**P** [18]	?	**P** [18]	**P** [18]	**P** [47]	**H** [3, 47]
ST	**P** [22]	?	**H** [54, 58]	**H** [54, 58]	**P** [58]	?	**P** [22]	**H** [50]
SP_I	**P** [22]	**H** [14]	**H** [14, 4, 58]	**H** [14, 4, 58]	**P** [58]	**H** [14]	**P** [22]	**H** [14, 50]
SP_E	**H** [22]	**H** [22]	**H** [22]	**H** [22]	**H** [22]	**H** [22]	**H** [22]	**H** [22]

tion has been extensively discussed and all the negative results obtained in this area (see, e.g., [3, 50]) remain valid for RR $\mathcal{P}$. All the negative results presented in Table 1.1 remain true even if $K = 2$. Observe that the complexity of the RR $\mathcal{P}$ version of the selection problem under $\mathcal{U}_1^I(\Gamma)$ and the minimum spanning tree problem under $\mathcal{U}_2^I(\Gamma)$ is not known. The complexity of the selection problem under $\mathcal{U}_3^I(\Gamma)$ (1.7) is not known. However, the problem is NP-hard when the uncertainty set is any polyhedron. It is enough to observe that the single-stage robust min-max problem for two scenarios S_1 and S_2 is equivalent to the single-stage robust min-max problem under $\mathcal{U}' = \mathrm{conv}\{S_1, S_2\}$ (see Definition 1.2) and the former problem is known to be NP-hard [3]. But, it is not clear that $\mathcal{U}'$ can be represented as (1.7).

1.4 Additional definitions

In this section we will provide some important definitions useful in this thesis

Definition 1.1 (Linear span)**.** *Let $\mathcal{V}$ be a vector space over some field $\mathcal{K}$. The elements of $\mathcal{V}$ are called vectors and elements of $\mathcal{K}$ are called scalars. Then for vectors $\boldsymbol{x}_1, \ldots, \boldsymbol{x}_n \in \mathcal{V}$ and scalars $\alpha_1, \ldots, \alpha_n \in \mathcal{K}$ the linear combination of those vectors with those scalars as coefficients is defined by:*

$$\sum_{i=1}^{n} \alpha_i \boldsymbol{x}_i.$$

A linear span of a set of vectors $\boldsymbol{x}_1, \ldots, \boldsymbol{x}_n$ is a set of all finite linear combinations of those vectors:

$$\mathrm{span}(\{\boldsymbol{x}_1, \ldots, \boldsymbol{x}_n\}) = \{\sum_{i=1}^{n} \alpha_i \boldsymbol{x}_i \ : \ \alpha_i \in \mathcal{K}\}.$$

Definition 1.2 (Convex hull)**.** *Given $\boldsymbol{x}_1, \ldots, \boldsymbol{x}_n$ from a real vector space $\mathcal{V}$. A convex combination of those vectors with the real numbers $\alpha_1, \ldots, \alpha_n$ is defined by*

$$\sum_{i=1}^{n} \alpha_i \boldsymbol{x}_i,$$

where $\alpha_i \geq 0$ *and* $\alpha_1 + \alpha_2 + \cdots + \alpha_n = 1$.

A convex hull of a set of vectors $\boldsymbol{x}_1, \ldots, \boldsymbol{x}_n$ *is a set of all convex combinations of those vectors:*

$$\mathrm{conv}(\{\boldsymbol{x}_1, \ldots, \boldsymbol{x}_n\}) = \{\sum_{i=1}^{n} \alpha_i \boldsymbol{x}_i \ : \ \alpha_i \geq 0, \alpha_1 + \cdots + \alpha_n = 1, \alpha_i \in \mathbb{R}\}.$$

Definition 1.3 (Approximation algorithm)**.** *An* α*-approximation algorithm for minimization problem is a polynomial-time algorithm that for all instances of the problem produces a solution whose value is within a factor of* α *of the value of an optimal solution (see [67]).*

Parameter α *is called* approximation ratio *or a* performance guarantee *of an algorithm.*

Definition 1.4 (Laminar family)**.** *Given a set* V*, then a collection of subsets of set* V*,* $\mathcal{L} \subseteq 2^V$ *is called* laminar *if no two sets* $A,\ B \in \mathcal{L}$ *are* intersecting*, i.e., when at least one of the sets* $A \cap B$ *or* $A \setminus B$ *or* $B \setminus A$ *is empty (see, e.g., [51]).*

Definition 1.5 (Vertex solution)**.** *Let* $\Phi = \{\boldsymbol{x} \ : \ \boldsymbol{A}\boldsymbol{x} = \boldsymbol{b}, \boldsymbol{x} \geq 0\} \subseteq \mathbb{R}^n$*. Then* $\boldsymbol{x} \in \mathbb{R}^n$ *is a* vertex solution *(or* extreme point solution*) of* Φ *if there does not exist a non-zero vector* $\boldsymbol{y} \in \mathbb{R}^n$ *such that* $\boldsymbol{x} + \boldsymbol{y}, \boldsymbol{x} - \boldsymbol{y} \in \Phi$ *(see, e.g., [51]).*

Vertex solutions can be seen as corner points of the set of feasible solutions.

Definition 1.6 (Matroid)**.** *A* matroid M *is an ordered pair* $(E, \mathcal{I})$ *consisting of a finite set* E*, called* ground set*, and a collection* $\mathcal{I}$ *of subsets of* E *satisfying conditions (see, e.g., [60]):*

i. $\emptyset \in \mathcal{I}$*,*

ii. if $I \in \mathcal{I}$ *and* $I' \subseteq I$*, then* $I' \in \mathcal{I}$*,*

iii. If $I_1,\ I_2 \in \mathcal{I}$ *and* $|I_1| < |I_2|$*, then there is an element* e *of* $I_2 \setminus I_1$ *such that* $I_1 \cup \{e\} \in \mathcal{I}$*.*

The elements of $\mathcal{I}$ *are called* independent sets *of matroid* M*.*
A rank function *of* M*,* $r_M : 2^E \to \mathbb{Z}_{\geq 0}$*, is defined by* $r_M(U) = \max_{W \subseteq U, W \in \mathcal{I}} |W|$*.*
A basis *of* M *is a maximal under inclusion element of* $\mathcal{I}$*.*
The following operations can be defined on a matroid M*:*

- Deletion e from M*,* $M \backslash e = (E_{M \backslash e}, \mathcal{I}_{M \backslash e})$*, is a matroid obtained by deleting* $e \in E$ *from* M *defined by* $E_{M \backslash e} = E \setminus \{e\}$ *and* $\mathcal{I}_{M \backslash e} = \{U \subseteq E \setminus \{e\} \ : \ U \in \mathcal{I}\}$*.*

  The rank function of $M \backslash e$ *is given by* $r_{M \backslash e}(U) = r_M(U)$ *for all* $U \subseteq E \setminus \{e\}$*.*

- Contraction e from M*,* $M/e = (E_{M/e}, \mathcal{I}_{M/e})$*, is a matroid obtained by contracting* $e \in E$ *in* M*, defined by* $E_{M/e} = E \setminus \{e\}$*; and* $\mathcal{I}_{M/e} = \{U \subseteq E \setminus \{e\} \ : \ U \cup \{e\} \in \mathcal{I}\}$ *if* $\{e\}$ *is independent and* $\mathcal{I}_{M/e} = \mathcal{I}$*, otherwise.*

  The rank function of M/e *is given by* $r_{M/e}(U) = r_M(U) - r_M(\{e\})$ *for all* $U \subseteq E \setminus \{e\}$*.*

Definition 1.7 (Fully Polynomial Time Approximation Scheme (FPTAS))**.** *An algorithm $\mathcal{A}$ is an FPTAS for an optimization problem $\mathcal{P}$, if for a given input I for a problem $\mathcal{P}$ and $\epsilon > 0$, algorithm $\mathcal{A}$ finds in time polynomial of the size of input I and of $\frac{1}{\epsilon}$, a solution S for an the input I that satisfies*

$$(f^*(I) - f(S)) \leq \epsilon f^*(I),$$

where $f(S)$ is a value of the solution S and $f^*(I)$ is the optimal value of a solution for the input I (see, e.g., [66]).

Definition 1.8 (Integral polyhedron)**.** *A polyhedron $P = \{\boldsymbol{x} \in \mathbb{R}^n \ : \ \boldsymbol{Ax} \leq \boldsymbol{b}\}$ is an integral polyhedron if $\arg\min_{\boldsymbol{x} \in P} \boldsymbol{c}^T \boldsymbol{x}$ has integer components for every $\boldsymbol{c} \in \mathbb{R}^n$ (see, e.g., [1]).*

Definition 1.9 (Unimodularity)**.** *A square integer matrix M is called unimodular if its determinant is equal to +1 or -1.*

A matrix M is called a totally unimodular if every its square non-singular submatrix is unimodular. Equivalently, a matrix M is totally unimodular if every square submatrix has determinant -1, 0 or +1. Note that in this case matrix M is not required to be square (see, e.g., [61]).

Chapter 2

The recoverable robust spanning tree problem

2.1 Introduction

This chapter begins a main part of the thesis. In this chapter we investigate in it the recoverable robust version of the *minimum spanning tree* problem (RR ST). Deterministic version of this problem is thoroughly researched and can be solved in polynomial time by several well known algorithms (see, e.g., [1, 61]).

The RR ST problem has been previously discussed in a number of papers [13, 14, 15, 17, 53, 58]. It is a special case of the robust spanning tree problem with incremental recourse considered in [58]. Furthermore, if $k = 0$ and $C_e = 0$ for each $e \in E$, then the problem is equivalent to the robust min-max spanning tree problem investigated in [45, 50].

Let us recall that the complexity of the RR ST with the interval uncertainty representation $\mathcal{U}^I$ under the element inclusion neighborhood (1.12) is open to date (see Section 1.3). First, we will show that RR ST for scenario set $\mathcal{U}^I$ is polynomially solvable (see Section 2.2). We will apply a technique called the *iterative relaxation*, whose framework was described in [51]. The idea is to construct a linear programming relaxation of the problem and show that at least one variable in each optimum vertex solution is integer. Such a variable allows us to add an edge to the partial spanning tree solution being built and recursively solve the relaxation of the smaller problem. This technique, however, does not imply directly a strongly polynomial algorithm for RR ST, since it requires the solution of a linear program. This issue will be addressed in the second part of the chapter.

In the second part of the chapter we first consider a *recoverable spanning tree problem* (Rec ST). Recall that Rec ST is a generalization of the *incremental spanning tree problem* (Inc ST). Note that Inc ST can be seen as the Rec ST problem with a fixed first stage spanning tree, whereas in Rec ST both the first and the second stage trees are unknown. Şeref *et al.* have shown in [22] that Inc ST can be solved in strongly polynomial time by applying the Lagrangian relaxation technique. Authors also demonstrate in [22] several interesting practical applications of the incremental network optimization. On the other

hand, no strongly polynomial time combinatorial algorithm for REC ST is known to exist to date. Thus the design of such such an algorithm for this problem is one of the main results of this chapter. We will also apply this algorithm for solving RR ST under scenario set $\mathcal{U}^I$ in strongly polynomial time. Moreover, we will show how the algorithm for REC ST can be used to obtain several approximation results for RR ST under scenario sets $\mathcal{U}_1^I(\Gamma)$ and $\mathcal{U}_2^I(\Gamma)$.

The recoverable spanning tree problem can be generalized by considering its robust version. We are given a connected undirected graph $G = (V, E)$, where $|V| = n$ and $|E| = m$. Recall the definition of RR $\mathcal{P}$ (1.19) from Section 1.2.4. Here Φ is a set of all *spanning trees* of G. Hence, the problem RR ST can be formally stated as follows:

$$\text{RR ST}: \min_{X\in\Phi}\left(\sum_{e\in X} C_e + \max_{S\in\mathcal{U}} \min_{Y\in\Phi_X^k} \sum_{e\in Y} c_e^S\right). \tag{2.1}$$

Note that Φ_X^k can be seen as a neighborhood of X containing all spanning trees which can be obtained from X by exchanging up to k edges (see (1.12) and (1.13)).

If $C_e = 0$ for each $e \in E$ and $k = 0$, then RR ST is equivalent to the following *min-max spanning tree* problem (1.8), examined in [2, 50, 45], in which we seek a spanning tree that minimizes the largest cost over all scenarios:

$$\text{MIN-MAX ST}: \min_{X\in\Phi}\max_{S\in\mathcal{U}}\sum_{e\in X} c_e^S. \tag{2.2}$$

If $C_e = 0$ for each $e \in E$ and $k = n-1$, then RR ST becomes the following *adversarial problem* [58] (see (1.21)) in which an *adversary* wants to find a scenario which leads to the greatest increase in the cost of the minimum spanning tree:

$$\text{ADV ST}: \max_{S\in\mathcal{U}}\min_{Y\in\Phi}\sum_{e\in Y} c_e^S. \tag{2.3}$$

It is worthwhile to mention that MIN-MAX ST under discrete scenario uncertainty representation U^D is NP hard even when $K = 2$ and becomes strongly NP-hard and not approximable within $O(\log^{1-\epsilon} n)$ for any $\epsilon > 0$ unless NP $\subseteq$ DTIME$(n^{\text{poly}\log n})$, when K is a part of input [50, 45]. It admits an FPTAS (see Definition 1.7), when K is a constant [2] and is approximable within $O(\log^2 n)$, when K is a part of the input [45]. The ADV ST problem, under scenario set $\mathcal{U}^D$, is polynomially solvable, since it boils down to solving K traditional minimum spanning tree problems.

Recall that for the computational complexity of RR ST for scenario set $\mathcal{U}_2^I(\Gamma)$ is still open. We only know that its special cases, namely MIN-MAX ST and ADV ST, are polynomially solvable [58].

2.2 Recoverable robust spanning tree problem

In this section we will use the iterative relaxation method [51] to construct a polynomial algorithm for RR ST under scenario set $\mathcal{U}^I$. Notice that in this case, the formulation (2.1)

can be rewritten as follows:

$$\text{RR ST}: \min_{X\in\Phi}\left(\sum_{e\in X} C_e + \min_{Y\in\Phi_X^k}\sum_{e\in Y}(c_e+d_e)\right). \tag{2.4}$$

In the recoverable spanning tree problem (2.4) we need to find a pair of spanning trees $X \in \Phi$ and $Y \in \Phi_X^k$. Since $|X| = |Y| = |V| - 1$, the problem (2.4) is equivalent the following mathematical programming problem:

$$\begin{array}{ll} \min & \sum_{e\in X} C_e + \sum_{e\in Y}(c_e+d_e) \\ \text{s.t.} & |X\cap Y| \geq |V| - 1 - k, \\ & X, Y \in \Phi. \end{array} \tag{2.5}$$

Let V_X and V_Y be subsets of vertices V, and E_X and E_Y be subsets of edges E, which induce connected graphs (multigraphs) $G_X = (V_X, E_X)$ and $G_Y = (V_Y, E_Y)$, respectively. Let E_Z be a subset of E such that $E_Z \subseteq E_X \cup E_Y$ and $|E_Z| \geq L$ for some fixed integer L. We will use $E_X(U)$ (resp. $E_Y(U)$) to denote the set of edges that has both endpoints in a given subset of vertices $U \subseteq V_X$ (resp. $U \subseteq V_Y$).

Let us consider the following linear program that we will substantially use in the algorithm for RR ST, denoted by $LP_{RRST}(E_X, V_X, E_Y, V_Y, E_Z, L)$:

$$\min \sum_{e\in E_X} C_e x_e + \sum_{e\in E_Y}(c_e+d_e)y_e \tag{2.6}$$

$$\text{s.t.} \quad \sum_{e\in E_X} x_e = |V_X| - 1, \tag{2.7}$$

$$\sum_{e\in E_X(U)} x_e \leq |U| - 1, \qquad \forall U \subset V_X, \tag{2.8}$$

$$-x_e + z_e \leq 0, \qquad \forall e \in E_X \cap E_Z, \tag{2.9}$$

$$\sum_{e\in E_Z} z_e = L, \tag{2.10}$$

$$z_e - y_e \leq 0, \qquad \forall e \in E_Y \cap E_Z, \tag{2.11}$$

$$\sum_{e\in E_Y} y_e = |V_Y| - 1, \tag{2.12}$$

$$\sum_{e\in E_Y(U)} y_e \leq |U| - 1, \qquad \forall U \subset V_Y, \tag{2.13}$$

$$x_e \geq 0, \qquad \forall e \in E_X, \tag{2.14}$$

$$z_e \geq 0, \qquad \forall e \in E_Z, \tag{2.15}$$

$$y_e \geq 0, \qquad \forall e \in E_Y. \tag{2.16}$$

The one can observe that we can fix $E_X = E_Z = E_Y = E$, $V_X = V_Y = V$, $L = |V| - 1 - k$, then the linear program $LP_{RRST}(E_X, V_X, E_Y, V_Y, E_Z, L)$ becomes a linear programming relaxation of (2.5). Indeed, the binary variables $x_e, y_e, z_e \in \{0, 1\}$ indicate then the spanning

trees X and Y and their common part $X \cap Y$, respectively. Moreover, the constraint (2.10) takes the form of equality, instead of the inequality, since the variables z_e, $e \in E_Z$, are not present in the objective function (2.6). Problem $LP_{RRST}(E_X, V_X, E_Y, V_Y, E_Z, L)$ has exponentially many constraints. However, the constraints (2.7), (2.8) and (2.12), (2.13) are the spanning tree ones for graphs $G_X = (V_X, E_X)$ and $G_Y = (V_Y, E_Y)$, respectively. Fortunately, there exists a polynomial time separation oracle over such constraints [56]. Clearly, separating over the remaining constraints, i.e., (2.9), (2.10) and (2.11) can be done in a polynomial time. In consequence, an optimal vertex solution to the problem can be found in polynomial time. It is also worth noting that, alternatively, one may rewrite the spanning tree constraints: (2.7), (2.8) and (2.12), (2.13) in an equivalent "compact" form, with polynomial number of variables and constraints, that can be more attractive from the computational point of view (see, for instance, the directed multicommodity flow model [56]). However, throughout this section will use the model (2.6)-(2.16), since it is easier to use for proving the properties of $LP_{RRST}(E_X, V_X, E_Y, V_Y, E_Z, L)$.

Let us focus now on a vertex solution $(\boldsymbol{x}, \boldsymbol{z}, \boldsymbol{y}) \in \mathbb{R}_{\geq 0}^{|E_X|\times|E_Z|\times|E_Y|}$ of the linear programming problem $LP_{RRST}(E_X, V_X, E_Y, V_Y, E_Z, L)$. If $E_Z = \emptyset$, than the only constraints being left in (2.6)-(2.16) are the spanning tree constraints. Thus $\boldsymbol{x}$ and $\boldsymbol{y}$ are 0-1 incidence vectors of the spanning trees X and Y, respectively (see [56, Theorem 3.2]).

We now turn to the more advanced case, when $E_X \neq \emptyset$, $E_Y \neq \emptyset$ and $E_Z \neq \emptyset$. We first reduce the sets E_X, E_Y and E_Z by removing all edges e with $x_e = 0$, or $y_e = 0$, or $z_e = 0$. Removing these edges does not change the feasibility and the cost of the vertex solution $(\boldsymbol{x}, \boldsymbol{z}, \boldsymbol{y})$. Note that V_X and V_Y remain unaffected. From now on, we can assume that the variables corresponding to all edges from E_X, E_Y and E_Z are positive, i.e., $x_e > 0$, $e \in E_X$, $y_e > 0$, $e \in E_Y$ and $z_e > 0$, $e \in E_Z$. Hence the constraints (2.14), (2.15) and (2.16) are not taken into account, since they are not tight with respect to $(\boldsymbol{x}, \boldsymbol{z}, \boldsymbol{y})$. It is possible, after reducing E_X, E_Y, and E_Z, to characterize $(\boldsymbol{x}, \boldsymbol{z}, \boldsymbol{y})$ by $|E_X| + |E_Z| + |E_Y|$ constraints that are linearly independent and tight with respect to $(\boldsymbol{x}, \boldsymbol{z}, \boldsymbol{y})$.

Let $\mathcal{F}(\boldsymbol{x})$ and $\mathcal{F}(\boldsymbol{y})$ defined in the following way:

$$\mathcal{F}(\boldsymbol{x}) = \{U \subseteq V_X \ : \ \sum_{e \in E_X(U)} x_e = |U| - 1\},$$

$$\mathcal{F}(\boldsymbol{y}) = \{U \subseteq V_Y \ : \ \sum_{e \in E_Y(U)} y_e = |U| - 1\}$$

stand for the sets of subsets of nodes that indicate the tight constraints (2.7), (2.8) and (2.12), (2.13) for $\boldsymbol{x}$ and $\boldsymbol{y}$, respectively. Similarly we define the sets of edges, $\mathcal{E}(\boldsymbol{x}, \boldsymbol{z})$ and $\mathcal{E}(\boldsymbol{z}, \boldsymbol{y})$ that indicate the tight constraints (2.9) and (2.11) with respect to $(\boldsymbol{x}, \boldsymbol{z}, \boldsymbol{y})$:

$$\mathcal{E}(\boldsymbol{x}, \boldsymbol{z}) = \{e \in E_X \cap E_Z \ : \ -x_e + z_e = 0\},$$

$$\mathcal{E}(\boldsymbol{z}, \boldsymbol{y}) = \{e \in E_Y \cap E_Z \ : \ z_e - y_e = 0\}.$$

Let $\chi_X(W)$, $W \subseteq E_X$ denote the characteristic vector in $\{0,1\}^{|E_X|} \times \{0\}^{|E_Z|} \times \{0\}^{|E_Y|}$ that has 1 if $e \in W$ and 0 otherwise. In a similar manner we denote by $\chi_Z(W)$, $W \subseteq E_Z$

and $\chi_Y(W)$, $W \subseteq E_Y$ the characteristic vectors in $\{0\}^{|E_X|} \times \{0,1\}^{|E_Z|} \times \{0\}^{|E_Y|}$ and $\{0\}^{|E_X|} \times \{0\}^{|E_Z|} \times \{0,1\}^{|E_Y|}$ respectively, both having 1 if $e \in W$ and 0 otherwise.

Observe that the number of subsets in $\mathcal{F}(\boldsymbol{x})$ and $\mathcal{F}(\boldsymbol{y})$ can be exponential. Let $\mathcal{L}(\boldsymbol{x})$ (resp. $\mathcal{L}(\boldsymbol{y})$) be a *maximal laminar subfamily* of $\mathcal{F}(\boldsymbol{x})$ (resp. $\mathcal{F}(\boldsymbol{y})$) (see Definition 1.4). The following lemma is an extension of [51, Lemma 4.1.5] It allows us to select certain subsets that indicate linearly independent tight constraints from $\mathcal{F}(\boldsymbol{x})$ and $\mathcal{F}(\boldsymbol{y})$. In the following lemma *span* denotes a linear span (see Definition 1.1).

Lemma 2.1. *For $\mathcal{L}(\boldsymbol{x})$ and $\mathcal{L}(\boldsymbol{y})$ the following equalities hold:*

$$\operatorname{span}(\{\chi_X(E_X(U)) : U \in \mathcal{L}(\boldsymbol{x})\}) = \operatorname{span}(\{\chi_X(E_X(U)) : U \in \mathcal{F}(\boldsymbol{x})\}),$$
$$\operatorname{span}(\{\chi_Y(E_Y(U)) : U \in \mathcal{L}(\boldsymbol{y})\}) = \operatorname{span}(\{\chi_Y(E_Y(U)) : U \in \mathcal{F}(\boldsymbol{y})\}).$$

Proof. The proof is the same as that for the spanning tree in [51, Lemma 4.1.5]. □

A trivial verification shows that the following observation is true:

Observation 2.1. *$V_X \in \mathcal{L}(\boldsymbol{x})$ and $V_Y \in \mathcal{L}(\boldsymbol{y})$.*

We are now ready to give a characterization of a vertex solution.

Lemma 2.2. *Let $(\boldsymbol{x}, \boldsymbol{z}, \boldsymbol{y})$ be a vertex solution of $LP_{RRST}(E_X, V_X, E_Y, V_Y, E_Z, L)$ such that $x_e > 0$, $e \in E_X$, $y_e > 0$, $e \in E_Y$ and $z_e > 0$, $e \in E_Z$. Then there exist laminar families $\mathcal{L}(\boldsymbol{x}) \neq \emptyset$ and $\mathcal{L}(\boldsymbol{y}) \neq \emptyset$ and subsets $E(\boldsymbol{x}, \boldsymbol{z}) \subseteq \mathcal{E}(\boldsymbol{x}, \boldsymbol{z})$ and $E(\boldsymbol{z}, \boldsymbol{y}) \subseteq \mathcal{E}(\boldsymbol{z}, \boldsymbol{y})$ that must satisfy the following:*

(i) $|E_X| + |E_Z| + |E_Y| = |\mathcal{L}(\boldsymbol{x})| + |E(\boldsymbol{x}, \boldsymbol{z})| + |E(\boldsymbol{z}, \boldsymbol{y})| + |\mathcal{L}(\boldsymbol{y})| + 1$,

(ii) the vectors in $\{\chi_X(E_X(U)) : U \in \mathcal{L}(\boldsymbol{x})\} \cup \{\chi_Y(E_Y(U)) : U \in \mathcal{L}(\boldsymbol{y})\} \cup \{-\chi_X(\{e\}) + \chi_Z(\{e\}) : e \in E(\boldsymbol{x}, \boldsymbol{z})\} \cup \{\chi_Z(\{e\}) - \chi_Y(\{e\}) : e \in E(\boldsymbol{z}, \boldsymbol{y})\} \cup \{\chi_Z(E_Z)\}$ *are linearly independent.*

Proof. The vertex $(\boldsymbol{x}, \boldsymbol{z}, \boldsymbol{y})$ can be uniquely characterized by any set of linearly independent constraints with the cardinality of $|E_X| + |E_Z| + |E_Y|$, chosen from among the constraints (2.7)-(2.13), tight with respect to $(\boldsymbol{x}, \boldsymbol{z}, \boldsymbol{y})$. We construct such set by choosing a maximal subset of linearly independent tight constraints that characterizes $(\boldsymbol{x}, \boldsymbol{z}, \boldsymbol{y})$. Lemma 2.1 shows that there exist maximal laminar subfamilies $\mathcal{L}(\boldsymbol{x}) \subseteq \mathcal{F}(\boldsymbol{x})$ and $\mathcal{L}(\boldsymbol{y}) \subseteq \mathcal{F}(\boldsymbol{y})$ such that

$$\operatorname{span}(\{\chi_X(E_X(U)) : U \in \mathcal{L}(\boldsymbol{x})\}) = \operatorname{span}(\{\chi_X(E_X(U)) : U \in \mathcal{F}(\boldsymbol{x})\})$$

and

$$\operatorname{span}(\{\chi_Y(E_Y(U)) : U \in \mathcal{L}(\boldsymbol{y})\}) = \operatorname{span}(\{\chi_Y(E_Y(U)) : U \in \mathcal{F}(\boldsymbol{y})\})).$$

Observation 2.1 implies $\mathcal{L}(\boldsymbol{x}) \neq \emptyset$ and $\mathcal{L}(\boldsymbol{y}) \neq \emptyset$. Moreover, it is evident that

$$\begin{aligned}&\operatorname{span}(\{\chi_X(E_X(U)) : U \in \mathcal{L}(\boldsymbol{x})\} \cup \{\chi_Y(E_Y(U)) : U \in \mathcal{L}(\boldsymbol{y})\})\\ &= \operatorname{span}(\{\chi_X(E_X(U)) : U \in \mathcal{F}(\boldsymbol{x})\} \cup \{\chi_Y(E_Y(U)) : U \in \mathcal{F}(\boldsymbol{y})\}).\end{aligned}$$

Thus $\mathcal{L}(\boldsymbol{x}) \cup \mathcal{L}(\boldsymbol{y})$ indicate certain linearly independent tight constraints that have been already included in the set constructed. We add (2.10) to the set constructed. Obviously, it still consists of linearly independent constraints. We complete forming the set by choosing a maximal number of tight constraints from among the ones (2.9) and (2.11), such that they form a linearly independent set with the constraints previously selected. We characterize these constraints by the sets of edges $E(\boldsymbol{x}, \boldsymbol{z}) \subseteq \mathcal{E}(\boldsymbol{x}, \boldsymbol{z})$ and $E(\boldsymbol{z}, \boldsymbol{y}) \subseteq \mathcal{E}(\boldsymbol{z}, \boldsymbol{y})$. Therefore, the vectors in $\{\chi_X(E_X(U)) : U \in \mathcal{L}(\boldsymbol{x})\} \cup \{\chi_Y(E_Y(U)) : U \in \mathcal{L}(\boldsymbol{y})\} \cup \{-\chi_X(\{e\}) + \chi_Z(\{e\}) : e \in E(\boldsymbol{x}, \boldsymbol{z})\} \cup \{\chi_Z(\{e\}) - \chi_Y(\{e\}) : e \in E(\boldsymbol{z}, \boldsymbol{y})\} \cup \{\chi_Z(E_Z)\}$ are linearly independent and represent the constructed maximal set of independent tight constraints, with the cardinality of $|\mathcal{L}(\boldsymbol{x})| + |E(\boldsymbol{x}, \boldsymbol{z})| + |E(\boldsymbol{z}, \boldsymbol{y})| + |\mathcal{L}(\boldsymbol{y})| + 1$, that uniquely describe $(\boldsymbol{x}, \boldsymbol{z}, \boldsymbol{y})$. Hence $|E_X| + |E_Z| + |E_Y| = |\mathcal{L}(\boldsymbol{x})| + |E(\boldsymbol{x}, \boldsymbol{z})| + |E(\boldsymbol{z}, \boldsymbol{y})| + |\mathcal{L}(\boldsymbol{y})| + 1$, which establishes the lemma. □

Lemma 2.3. *Let $(\boldsymbol{x}, \boldsymbol{z}, \boldsymbol{y})$ be a vertex solution of $LP_{RRST}(E_X, V_X, E_Y, V_Y, E_Z, L)$ such that $x_e > 0$, $e \in E_X$, $y_e > 0$, $e \in E_Y$ and $z_e > 0$, $e \in E_Z$. Then there is an edge $e' \in E_X$ with $x_{e'} = 1$ or an edge $e'' \in E_Y$ with $y_{e''} = 1$.*

Proof. On the contrary, suppose that $0 < x_e < 1$ for every $e \in E_X$ and $0 < y_e < 1$ for every $e \in E_Y$. Constraints (2.9) and (2.11) lead to $0 < z_e < 1$ for every $e \in E_Z$. By Lemma 2.2, there exist laminar families $\mathcal{L}(\boldsymbol{x}) \neq \emptyset$ and $\mathcal{L}(\boldsymbol{y}) \neq \emptyset$ and subsets $E(\boldsymbol{x}, \boldsymbol{z}) \subseteq \mathcal{E}(\boldsymbol{x}, \boldsymbol{z})$ and $E(\boldsymbol{z}, \boldsymbol{y}) \subseteq \mathcal{E}(\boldsymbol{z}, \boldsymbol{y})$ indicating linearly independent constraints which uniquely define $(\boldsymbol{x}, \boldsymbol{z}, \boldsymbol{y})$, namely

$$\sum_{e \in E_X(U)} x_e = |U| - 1, \qquad \forall U \in \mathcal{L}(\boldsymbol{x}), \tag{2.17}$$

$$-x_e + z_e = 0, \qquad \forall e \in E(\boldsymbol{x}, \boldsymbol{z}), \tag{2.18}$$

$$\sum_{e \in E_Z} z_e = L, \tag{2.19}$$

$$z_e - y_e = 0, \qquad \forall e \in E(\boldsymbol{z}, \boldsymbol{y}), \tag{2.20}$$

$$\sum_{e \in E_Y(U)} y_e = |U| - 1, \qquad \forall U \in \mathcal{L}(\boldsymbol{y}). \tag{2.21}$$

We will arrive to a contradiction with Lemma 2.2(i) by applying a *token counting argument*, frequently used in [51].

We give exactly two tokens to each edge in E_X, E_Z and E_Y. Thus we use $2|E_X| + 2|E_Z| + 2|E_Y|$ tokens. We then redistribute these tokens to the tight constraints (2.17)-(2.21) as follows. For $e \in E_X$ the first token is assigned to the constraint indicated by the smallest set $U \in \mathcal{L}(\boldsymbol{x})$ containing its two endpoints (see (2.17)) and the second one

is assigned to the constraint represented by e (see (2.18)) if $e \in E(\boldsymbol{x}, \boldsymbol{z})$. Similarly, for $e \in E_Y$ the first token is assigned to the constraint indicated by the smallest set $U \in \mathcal{L}(\boldsymbol{y})$ containing its both endpoints (see (2.21)) and the second one is assigned to the constraint represented by e (see (2.20)) if $e \in E(\boldsymbol{z}, \boldsymbol{y})$. Each $e \in E_Z$ assigns the first token to the constraint corresponding to e (see (2.18)) if $e \in E(\boldsymbol{x}, \boldsymbol{z})$; otherwise to the constraint (2.19). The second token is assigned to the constraint indicated by e (see (2.20)) if $e \in E(\boldsymbol{z}, \boldsymbol{y})$.

Claim 2.1. *Each of the constraints (2.18) and (2.20) receives exactly two tokens. Each of the constraints (2.17) and (2.21) collects at least two tokens.*

The first part of Claim 2.1 is obvious. In order to show the second part we apply the same reasoning as [51, Proof of Lemma 4.2.1]. Consider the constraint represented by any subset $U \in \mathcal{L}(\boldsymbol{x})$. We say that U is the *parent* of a subset $C \in \mathcal{L}(\boldsymbol{x})$ and C is the *child* of U if U is the smallest set containing C. Let $C_1, \ldots, C_\ell$ be the children of U. The constraints corresponding to these subsets are as follows

$$\sum_{e \in E_X(U)} x_e = |U| - 1, \tag{2.22}$$

$$\sum_{e \in E_X(C_k)} x_e = |C_k| - 1, \ \forall k \in [\ell]. \tag{2.23}$$

Subtracting (2.23) for every $k \in [\ell]$ from (2.22) yields:

$$\sum_{e \in E_X(U) \setminus \bigcup_{k \in [\ell]} E_X(C_k)} x_e = |U| - \sum_{k \in [\ell]} |C_k| + \ell - 1. \tag{2.24}$$

Equation (2.24) holds, since the sets $C_1, \ldots, C_\ell$ are the children of U and all these sets are in laminar family $\mathcal{L}(\boldsymbol{x})$. Observe that

$$E_X(U) \setminus \bigcup_{k \in [\ell]} E_X(C_k) \neq \emptyset.$$

Otherwise, this leads to a contradiction with the linear independence of the constraints. Since the right hand side of (2.24) is integer and $0 < x_e < 1$ for every $e \in E_X$,

$$|E_X(U) \setminus \bigcup_{k \in [\ell]} E_X(C_k)| \geq 2.$$

Hence U receives at least two tokens. The same arguments apply to the constraint represented by any subset $U \in \mathcal{L}(\boldsymbol{y})$. This proves the claim.

Claim 2.2. *Either constraint (2.19) collects at least one token and there are at least two extra tokens left or constraint (2.19) receives no token and there are at least three extra tokens left.*

To prove the claim we need to consider several nested cases:

1. Case: $E_Z \setminus E(\boldsymbol{x}, \boldsymbol{z}) \neq \emptyset$. Since $E_Z \setminus E(\boldsymbol{x}, \boldsymbol{z}) \neq \emptyset$, at least one token is assigned to constraint (2.19). We have yet to show that there are at least two token left.

   (a) Case: $E_Z \setminus E(\boldsymbol{z}, \boldsymbol{y}) = \emptyset$. Subtracting (2.20) for every $e \in E(\boldsymbol{z}, \boldsymbol{y})$ from (2.19) gives:

   $$\sum_{e \in E(\boldsymbol{z}, \boldsymbol{y})} y_e = L. \tag{2.25}$$

      i. Case: $E_Y \setminus E(\boldsymbol{z}, \boldsymbol{y}) = \emptyset$. Thus $L = |V_Y| - 1$, since $(\boldsymbol{x}, \boldsymbol{z}, \boldsymbol{y})$ is a feasible solution. By Observation 2.1, $V_Y \in \mathcal{L}(\boldsymbol{y})$ and (2.25) has the form of constraint (2.21) for V_Y, which contradicts the linear independence of the constraints.

      ii. Case: $E_Y \setminus E(\boldsymbol{z}, \boldsymbol{y}) \neq \emptyset$. Thus $L < |V_Y| - 1$. Since the right hand side of (2.25) is integer and $0 < y_e < 1$ for every $e \in E_Y$, $|E_Y \setminus E(\boldsymbol{z}, \boldsymbol{y})| \geq 2$. Hence, there are at least two extra tokens left.

   (b) Case: $E_Z \setminus E(\boldsymbol{z}, \boldsymbol{y}) \neq \emptyset$. Consequently, $|E_Z \setminus E(\boldsymbol{z}, \boldsymbol{y})| \geq 1$ and thus at least one token left over, i.e at least one token is not assigned to constraints (2.20). Therefore, yet one additional token is required.

      i. Case: $E_Y \setminus E(\boldsymbol{z}, \boldsymbol{y}) = \emptyset$. Consider the constraint (2.21) corresponding to V_Y. Adding (2.20) for every $e \in E(\boldsymbol{z}, \boldsymbol{y})$ to this constraint yields:

      $$\sum_{e \in E(\boldsymbol{z}, \boldsymbol{y})} z_e = |V_Y| - 1. \tag{2.26}$$

      Obviously $|V_Y| - 1 < L$. Since L is integer and $0 < z_e < 1$ for every $e \in E_Z$, $|E_Z \setminus E(\boldsymbol{z}, \boldsymbol{y})| \geq 2$. Hence there are at least two extra tokens left.

      ii. Case: $E_Y \setminus E(\boldsymbol{z}, \boldsymbol{y}) \neq \emptyset$. One can see immediately that at least one token left over, i.e at least one token is not assigned to constraints (2.20).

   Summarizing the above cases, constraint (2.19) collects at least one token and there are at least two extra tokens left.

2. Case: $E_Z \setminus E(\boldsymbol{x}, \boldsymbol{z}) = \emptyset$. Subtracting (2.18) for every $e \in E(\boldsymbol{x}, \boldsymbol{z})$ from (2.19) gives:

   $$\sum_{e \in E(\boldsymbol{x}, \boldsymbol{z})} x_e = L. \tag{2.27}$$

   Thus constraint (2.19) receives no token. We yet need to show that there are at least three extra tokens left.

   (a) Case: $E_X \setminus E(\boldsymbol{x}, \boldsymbol{z}) = \emptyset$. Therefore $L = |V_X| - 1$, since $(\boldsymbol{x}, \boldsymbol{z}, \boldsymbol{y})$ is a feasible solution. By Observation 2.1, $V_X \in \mathcal{L}(\boldsymbol{x})$ and (2.27) has the form of constraint (2.17) for V_X, which contradicts with the linear independence of the constraints.

(b) Case: $E_X \setminus E(\boldsymbol{x}, \boldsymbol{z}) \neq \emptyset$ Thus $L < |V_X| - 1$. Since the right hand side of (2.27) is integer and $0 < x_e < 1$ for every $e \in E_X$, $|E_X \setminus E(\boldsymbol{x}, \boldsymbol{z})| \geq 2$. Consequently, there are at least two extra tokens left. Yet at least one token is required.

i. Case: $E_Z \setminus E(\boldsymbol{z}, \boldsymbol{y}) = \emptyset$. Reasoning is the same as in Case 1a.

ii. Case: $E_Z \setminus E(\boldsymbol{z}, \boldsymbol{y}) \neq \emptyset$. Reasoning is the same as in Case 1b.

Accordingly, constraint (2.19) receives no token and there are at least three extra tokens left.

Thus the claim is proved. The method of assigning tokens to constraints (2.17)-(2.21) and Claims 2.1 and 2.2 now show that either

$$2|E_X| + 2|E_Z| + 2|E_Y| - 2 \geq 2|\mathcal{L}(\boldsymbol{x})| + 2|E(\boldsymbol{x}, \boldsymbol{z})| + 2|E(\boldsymbol{z}, \boldsymbol{y})| + 2|\mathcal{L}(\boldsymbol{y})| + 1$$

or

$$2|E_X| + 2|E_Z| + 2|E_Y| - 3 \geq 2|\mathcal{L}(\boldsymbol{x})| + 2|E(\boldsymbol{x}, \boldsymbol{z})| + 2|E(\boldsymbol{z}, \boldsymbol{y})| + 2|\mathcal{L}(\boldsymbol{y})|.$$

The above inequalities lead to $|E_X|+|E_Z|+|E_Y| > |\mathcal{L}(\boldsymbol{x})|+|E(\boldsymbol{x}, \boldsymbol{z})|+|E(\boldsymbol{z}, \boldsymbol{y})|+|\mathcal{L}(\boldsymbol{y})|+1$. This contradicts Lemma 2.2(i). □

It remains to verify two cases: $E_X = \emptyset$ and $|V_X| = 1$; $E_Y = \emptyset$ and $|V_Y| = 1$. We consider only the first one, because the second case is symmetrical. The constraints (2.10), (2.11) and the inclusion $E_Z \subseteq E_Y$ yield

$$\sum_{e \in E_Z} y_e \geq L. \tag{2.28}$$

Lemma 2.4. *Let $\boldsymbol{y}$ be a vertex solution of linear program: (2.6), (2.12), (2.13), (2.16) and (2.28) such that $y_e > 0$, $e \in E_Y$. Then there is an edge $e' \in E_Y$ with $y_{e'} = 1$. Moreover, using $\boldsymbol{y}$ one can construct a vertex solution of $LP_{RRST}(\emptyset, V_X, E_Y, V_Y, E_Z, L)$ with $y_{e'} = 1$ and the cost of $\boldsymbol{y}$.*

Proof. Similarly as in the proof Lemma 2.2, we construct a maximal subset of linearly independent tight constraints that characterize $\boldsymbol{y}$ and get: $|E_Y| = |\mathcal{L}(\boldsymbol{y})|$ if (2.28) is not tight or adding (2.28) makes the subset dependent; $|E_Y| = |\mathcal{L}(\boldsymbol{y})| + 1$ otherwise. In the first case the spanning tree constraints define $\boldsymbol{y}$ and, in consequence, $\boldsymbol{y}$ is integral (see [56, Theorem 3.2]). Consider the second case and assume, on the contrary, that $0 < y_e < 1$ for each $e \in E_Y$. Thus

$$\sum_{e \in E_Z} y_e = L, \tag{2.29}$$

$$\sum_{e \in E_Y(U)} y_e = |U| - 1, \qquad \forall U \in \mathcal{L}(\boldsymbol{y}). \tag{2.30}$$

We assign two tokens to each edge in E_Y and redistribute $2|E_Y|$ tokens to constraints (2.29) and (2.30) in the following way. The first token is given to the constraint indicated

by the smallest set $U \in \mathcal{L}(\boldsymbol{y})$ containing its two endpoints and the second one is assigned to (2.29). Since $0 < y_e < 1$ and L is integer, similarly as in the proof Lemma 2.3, one can show that each of the constraints (2.30) and (2.29) receives at least two tokens. If $E_Y \setminus E_Z = \emptyset$ then $L = |V_Y| - 1$, since $\boldsymbol{y}$ is a feasible solution - a contradiction with the linear independence of the constraints. Otherwise ($E_Y \setminus E_Z \neq \emptyset$), at least one token is left. Hence $2|E_Y| - 1 = 2|\mathcal{L}(\boldsymbol{y})| + 2$ and so $|E_Y| > |\mathcal{L}(\boldsymbol{y})| + 1$, a contradiction.

By (2.28) and the fact that there are no variables z_e, $e \in E_Z$, in the objective (2.6), it is obvious that using $\boldsymbol{y}$ one can construct $\boldsymbol{z}$ satisfying (2.10) and, in consequence, a vertex solution of $LP_{RRST}(\emptyset, V_X, E_Y, V_Y, E_Z, L)$ with $y_{e'} = 1$ and the cost of $\boldsymbol{y}$. □

Algorithm 2.1 Algorithm for RR ST

$E_X \leftarrow E$, $E_Y \leftarrow E$, $E_Z \leftarrow E$, $V_X \leftarrow V$, $V_Y \leftarrow V$, $L \leftarrow |V| - 1 - k$, $X \leftarrow \emptyset$, $Y \leftarrow \emptyset$, $Z \leftarrow \emptyset$
while $|V_X| \geq 2$ **or** $|V_Y| \geq 2$ **do**
  Find an optimal vertex solution $(\boldsymbol{x}^*, \boldsymbol{z}^*, \boldsymbol{y}^*)$ of $LP_{RRST}(E_X, V_X, E_Y, V_Y, E_Z, L)$
  for all $e \in E_Z$ with $z_e^* = 0$ **do** $E_Z \leftarrow E_Z \setminus \{e\}$
  end for
  for all $e \in E_X$ with $x_e^* = 0$ **do** $E_X \leftarrow E_X \setminus \{e\}$
  end for
  for all $e \in E_Y$ with $y_e^* = 0$ **do** $E_Y \leftarrow E_Y \setminus \{e\}$
  end for
  if there exists edge $e' \in E_X$ with $x_{e'}^* = 1$ **then**
    $X \leftarrow X \cup \{e'\}$
    contract edge $e' = \{u, v\}$ by deleting e and identifying its endpoints u and v in graph $G_X = (V_X, E_X)$, induced by V_X and E_X, which is equivalent to: $|V_X| \leftarrow |V_X| - 1$
    and $E_X \leftarrow E_X \setminus \{e'\}$
  end if
  if there exists edge $e' \in E_X$ with $y_{e'}^* = 1$ **then**
    $Y \leftarrow Y \cup \{e'\}$
    contract edge $e' = \{u, v\}$ by deleting e and identifying its endpoints u and v in graph $G_Y = (V_Y, E_Y)$, induced by V_Y and E_Y, which is equivalent to: $|V_Y| \leftarrow |V_Y| - 1$
    and $E_Y \leftarrow E_Y \setminus \{e'\}$
  end if
  if there exists edge $e' \in E_Z$ such that $e' \in X \cap Y$ **then**
    $Z \leftarrow Z \cup \{e'\}$
    $L \leftarrow L - 1$
    $E_Z \leftarrow E_Z \setminus \{e'\}$
  end if
  return X, Y, Z
end while

We are now ready to give the main result of this section.

Theorem 2.1. *Algorithm 2.1 solves RR ST in polynomial time.*

Proof. Lemmas 2.3 and 2.4 and the case when $E_Z = \emptyset$ (see the comments in this section) ensure that Algorithm 2.1 terminates after performing $O(|V|)$ iterations (Steps 3-25). Let $\mathrm{OPT_{LP}}$ denote the optimal objective function value of $LP_{RRST}(E_X, V_X, E_Y, V_Y, E_Z, L)$, where $E_X = E_Z = E_Y = E$, $V_X = V_Y = V$, $L = |V| - 1 - k$. Hence $\mathrm{OPT_{LP}}$ is a lower bound

on the optimal objective value of RR ST. It is not difficult to show that after the termination of the algorithm X and Y are two spanning trees in G such that $\sum_{e\in X} C_e+\sum_{e\in Y}(c_e+d_e) \leq \mathrm{OPT}_{\mathrm{LP}}$. It remains to show that $|X\cap Y| \geq |V|-k-1$. By induction on the number of iterations of Algorithm 2.1 one can easily show that at any iteration the inequality $L+|Z| = |V|-1-k$ is satisfied. Accordingly, if, after the termination of the algorithm, $L=0$ holds, then we are done. Suppose, on the contrary that $L \geq 1$ (L is integer). Since $\sum_{e\in E_Z} z^*_e \geq L$, $|E_Z| \geq L \geq 1$ and E_Z is the set with edges not belonging to $X\cap Y$. Consider any $e' \in E_Z$. Of course $z^*_{e'} > 0$ and it remained positive during the course of Algorithm 2.1. Moreover, at least one of the constraints $-x_{e'}+z_{e'} \leq 0$ or $z_{e'}-y_{e'} \leq 0$ is still present in the linear program (2.6)-(2.16). Otherwise, since $z^*_{e'} > 0$, Steps 10-14, Steps 16-20 and, in consequence, Steps 22-25 for e' must have been executed during the course of Algorithm 2.1 and e' must have been included to Z, a contradiction with the fact $e' \notin X\cap Y$. Since the above constraints are present, $0 < x^*_{e'} < 1$ or $0 < y^*_{e'} < 1$. Thus $e' \in E_X$ or $e' \in E_Y$, which contradicts the termination of Algorithm 2.1. □

2.3 Combinatorial algorithm for recoverable spanning tree problem

In this section we construct a combinatorial algorithm for Rec ST with strongly polynomial running time. Since $|X| = n-1$ for each $X \in \Phi$, Rec ST (1.23) is equivalent to the following mathematical programming problem:

$$\begin{array}{ll} \min & \displaystyle\sum_{e\in X} C_e + \sum_{e\in Y} c_e \\ \text{s.t.} & |X\cap Y| \geq L, \\ & X, Y \in \Phi, \end{array} \tag{2.31}$$

where $L = n-1-k$. Problem (2.31) can be expressed as the following MIP model:

$$Opt = \min \sum_{e\in E} C_e x_e + \sum_{e\in E} c_e y_e \tag{2.32}$$

$$\text{s.t.} \quad \sum_{e\in E} x_e = n-1, \tag{2.33}$$

$$\sum_{e\in E(U)} x_e \leq |U|-1, \qquad \forall U \subset V, \tag{2.34}$$

$$\sum_{e\in E} y_e = n-1, \tag{2.35}$$

$$\sum_{e\in E(U)} y_e \leq |U|-1, \qquad \forall U \subset V, \tag{2.36}$$

$$x_e - z_e \geq 0, \qquad \forall e \in E, \tag{2.37}$$

$$y_e - z_e \geq 0, \qquad \forall e \in E, \tag{2.38}$$

$$\sum_{e\in E} z_e \geq L, \tag{2.39}$$

$$x_e, y_e, z_e \geq 0, \text{ integer} \qquad \forall e \in E, \tag{2.40}$$

where $E(U)$ stands for the set of edges that have both endpoints in $U \subseteq V$. We first apply the Lagrangian relaxation (see, e.g., [1]) to (2.32)-(2.40) by relaxing the cardinality constraint (2.39) with a nonnegative multiplier θ. We also relax the integrality constraints (2.40). We thus get the following linear program (with the corresponding dual variables which will be used later):

$$\begin{aligned}
\phi(\theta) = \min & \sum_{e\in E} C_e x_e + \sum_{e\in E} c_e y_e - \theta \sum_{e\in E} z_e + \theta L && && (2.41)\\
\text{s.t. } & \sum_{e\in E} x_e = n-1, && && [\mu],\\
& -\sum_{e\in E(U)} x_e \geq -(|U|-1), && \forall U \subset V, && [w_U],\\
& \sum_{e\in E} y_e = n-1, && && [\nu],\\
& -\sum_{e\in E(U)} y_e \geq -(|U|-1), && \forall U \subset V, && [v_U],\\
& x_e - z_e \geq 0, && \forall e \in E, && [\alpha_e],\\
& y_e - z_e \geq 0, && \forall e \in E, && [\beta_e],\\
& x_e, y_e, z_e \geq 0, && \forall e \in E. &&
\end{aligned}$$

For any $\theta \geq 0$, the Lagrangian function $\phi(\theta)$ is a lower bound on Opt. It is well-known that $\phi(\theta)$ is concave and piecewise linear. By the optimality test (see, e.g., [1]), we obtain the following theorem:

Theorem 2.2. *Let $(x_e, y_e, z_e)_{e\in E}$ be an optimal solution to (2.41) for some $\theta \geq 0$, feasible to (2.33)-(2.40) and satisfying the complementary slackness condition $\theta(\sum_{e\in E} z_e - L) = 0$. Then $(x_e, y_e, z_e)_{e\in E}$ is optimal to (2.32)-(2.40).*

Let (X, Y), $X, Y \in \Phi$, be a pair of spanning trees of G (a *pair* for short). This pair corresponds to a feasible $0-1$ solution to (2.41), defined as follows: $x_e = 1$ for $e \in X$, $y_e = 1$ for $e \in Y$, and $z_e = 1$ for $e \in X \cap Y$; the values of the remaining variables are set to 0. From now on, by a pair (X, Y) we also mean a feasible solution to (2.41) defined as above. Given a pair (X, Y) with the corresponding solution $(x_e, y_e, z_e)_{e\in E}$, let us define the partition (E_X, E_Y, E_Z, E_W) of the set of the edges E in the following way:

$$\begin{aligned}
E_X &= \{e \in E : x_e = 1, y_e = 0\},\\
E_Y &= \{e \in E : y_e = 1, x_e = 0\},\\
E_Z &= \{e \in E : x_e = 1, y_e = 1\},\\
E_W &= \{e \in E : x_e = 0, y_e = 0\}.
\end{aligned}$$

Thus equalities: $X = E_X \cup E_Z$, $Y = E_Y \cup E_Z$ and $E_Z = X \cap Y$ hold. Our goal is to establish some sufficient optimality conditions for a given pair (X, Y) in the problem (2.41). The dual to (2.41) has the following form:

$$\begin{aligned}
\phi^D(\theta) = \max & -\sum_{U\subset V}(|U|-1)w_U + (n-1)\mu - \sum_{U\subset V}(|U|-1)v_U + (n-1)\nu + \theta L && (2.42)\\
\text{s.t.} \; & -\sum_{\{U\subset V\,:\,e\in E(U)\}} w_U + \mu \le C_e - \alpha_e, && \forall e\in E,\\
& -\sum_{\{U\subset V\,:\,e\in E(U)\}} v_U + \nu \le c_e - \beta_e, && \forall e\in E,\\
& \alpha_e + \beta_e \ge \theta, && \forall e\in E,\\
& w_U, v_U \ge 0, && U\subset V,\\
& \alpha_e, \beta_e \ge 0, && \forall e\in E.
\end{aligned}$$

Lemma 2.5. *The dual problem (2.42) can be rewritten as follows:*

$$\phi^D(\theta) = \max_{\{\alpha_e\ge 0, \beta_e\ge 0\,:\,\alpha_e+\beta_e\ge\theta,\; e\in E\}} \left(\min_{X\in\Phi}\sum_{e\in X}(C_e-\alpha_e) + \min_{Y\in\Phi}\sum_{e\in Y}(c_e-\beta_e)\right) + \theta L. \qquad (2.43)$$

Proof. Fix some α_e and β_e such that $\alpha_e+\beta_e \ge \theta$ for each $e\in E$ in (2.42). For these constant values of α_e and β_e, $e\in E$, using the dual to (2.42), we arrive to $\min_{X\in\Phi}\sum_{e\in X}(C_e-\alpha_e) + \min_{Y\in\Phi}\sum_{e\in Y}(c_e-\beta_e) + \theta L$ and the lemma follows. □

Lemma 2.5 allows us to establish the following result:

Theorem 2.3 (**Sufficient pair optimality conditions**). *A pair of spanning trees (X, Y) is optimal to (2.41) for a fixed $\theta \ge 0$ if there exist $\alpha_e \ge 0$, $\beta_e \ge 0$ such that $\alpha_e + \beta_e = \theta$ for each $e\in E$ and*

(i) X is a minimum spanning tree for the costs $C_e - \alpha_e$, Y is a minimum spanning tree for the costs $c_e - \beta_e$,

(ii) $\alpha_e = 0$ for each $e\in E_X$, $\beta_e = 0$ for each $e\in E_Y$.

Proof. By the primal-dual relation, the inequality $\phi^D(\theta) \le \phi(\theta)$ holds. Using (2.43), we obtain

$$\begin{aligned}
\phi^D(\theta) &\ge \sum_{e\in X}(C_e-\alpha_e) + \sum_{e\in Y}(c_e-\beta_e) + \theta L = \sum_{e\in E_X} C_e + \sum_{e\in E_Y} c_e + \sum_{e\in E_Z}(C_e + c_e - \theta) + \theta L\\
&= \sum_{e\in E_X} C_e + \sum_{e\in E_Z} C_e + \sum_{e\in E_Y} c_e + \sum_{e\in E_Z} c_e - \theta|E_Z| + \theta L\\
&= \sum_{e\in X} C_e + \sum_{e\in Y} c_e - \theta|E_Z| + \theta L = \phi(\theta).
\end{aligned}$$

The Weak Duality Theorem implies the optimality of (X, Y) in (2.41) for a fixed $\theta \ge 0$. □

Lemma 2.6. *A pair (X, Y), which satisfies the sufficient pair optimality conditions for $\theta = 0$, can be computed in polynomial time.*

Proof. Let X be a minimum spanning tree for the costs C_e and Y be a minimum spanning tree for the costs c_e, $e \in E$. Since $\theta = 0$, we set $\alpha_e = 0$, $\beta_e = 0$ for each $e \in E$. It is clear that (X, Y) satisfies the sufficient pair optimality conditions. □

Assume that (X, Y) satisfies the sufficient pair optimality conditions for some $\theta \geq 0$. If, for this pair, $|E_Z| \geq L$ and $\theta(|E_Z| - L) = 0$, then we are done, because by Theorem 2.2, the pair (X, Y) is optimal to (2.32)-(2.40). Suppose that $|E_Z| < L$ ((X, Y) is not feasible to (2.32)-(2.40)). We will now show a polynomial time procedure for finding a new pair (X', Y'), which satisfies the sufficient pair optimality conditions and $|E_{Z'}| = |E_Z| + 1$. This implies a polynomial time algorithm for the problem (2.32)-(2.40), since it is enough to start with a pair satisfying the sufficient pair optimality conditions for $\theta = 0$ (see Lemma 2.6) and repeat the procedure at most L times, i.e., until $|E_{Z'}| = L$.

Given a spanning tree T in $G = (V, E)$ and edge $e = \{k, l\} \notin T$, let us denote by $P_T(e)$ the unique path in T connecting nodes k and l. It is well known that for any $f \in P_T(e)$, $T' = T \cup \{e\} \setminus \{f\}$ is also a spanning tree in G. We will say that T' is the result of a *move* on T.

Consider a pair (X, Y) that satisfies the sufficient pair optimality conditions for some fixed $\theta \geq 0$. Set $C^*_e = C_e - \alpha_e$ and $c^*_e = c_e - \beta_e$ for every $e \in E$, where α_e and β_e, $e \in E$, are the numbers which satisfy the conditions in Theorem 2.3. Thus, by Theorem 2.3(i) and the *path optimality conditions* (see, e.g., [1]), we get the following conditions which must be satisfied by (X, Y):

$$\text{for every } e \notin X \qquad C^*_e \geq C^*_f \qquad \text{for every } f \in P_X(e), \tag{2.44a}$$

$$\text{for every } e \notin Y \qquad c^*_e \geq c^*_f \qquad \text{for every } f \in P_Y(e). \tag{2.44b}$$

We now build a so-called *admissible graph* $G^A = (V^A, E^A)$ in two steps. We first associate with each edge $e \in E$ a node v_e and include it to V^A, $|V^A| = |E|$. We then add arc (v_e, v_f) to E^A if $e \notin X$, $f \in P_X(e)$ and $C^*_e = C^*_f$. This arc is called an *X-arc*. We also add arc (v_f, v_e) to E^A if $e \notin Y$, $f \in P_Y(e)$ and $c^*_e = c^*_f$. This arc is called an *Y-arc*. We say that $v_e \in V^A$ is *admissible* if $e \in E_Y$, or v_e is reachable from a node $v_g \in V^A$, such that $g \in E_Y$, by a directed path in G^A. In the second step we remove from G^A all the nodes which are not admissible, together with their incident arcs. An example of an admissible graph is shown in Figure 2.1. Each node of this admissible graph is reachable from some node v_g, $g \in E_Y$. Note that the arcs (v_{e_7}, v_{e_6}) and $(v_{e_7}, v_{e_{10}})$ are not present in G^A, because v_{e_7} is not reachable from any node v_g, $g \in E_Y$. These arcs have been removed from G^A in the second step.

Observe that each X-arc $(v_e, v_f) \in E^A$ represents a move on X, namely $X' = X \cup \{e\} \setminus \{f\}$ is a spanning tree in G. Similarly, each Y-arc $(v_e, v_f) \in E^A$ represents a move on Y, namely $Y' = Y \cup \{f\} \setminus \{e\}$ is a spanning tree in G. Notice that the cost, with respect to C^*_e, of X' is the same as X and the cost, with respect to c^*_e, of Y' is the same

as Y. So, the moves indicated by X-arcs and Y-arcs preserve the optimality of X and Y, respectively. Observe that $e \notin X$ or $e \in Y$, which implies $e \notin E_X$. Also $f \in X$ or $f \notin Y$, which implies $f \notin E_Y$. Hence, no arc in E^A can start in a node corresponding to an edge in E_X and no arc in E^A can end in a node corresponding to an edge in E_Y. Observe also that $(v_e, v_f) \in E^A$ can be both X-arc and Y-arc only if $e \in E_Y$ and $f \in E_X$. Such a case is shown in Figure 2.1 (see the arc (v_{e_1}, v_{e_2})). Since each arc $(v_e, v_f) \in E^A$ represents a move on X or Y, e and f cannot both belong to E_W or E_Z.

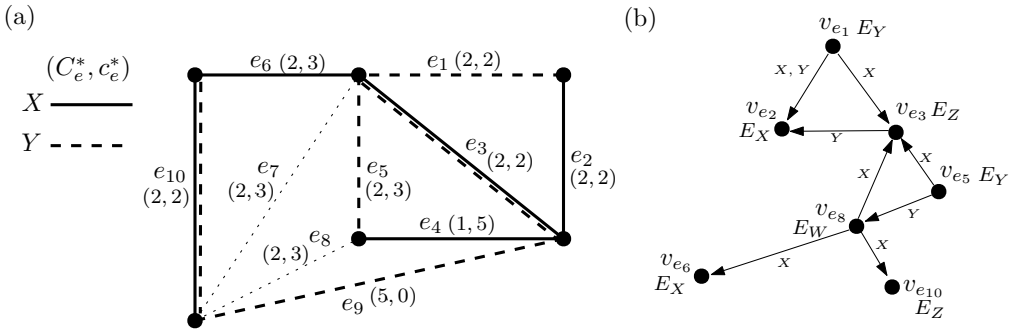


Figure 2.1: (a) A pair (X, Y) such that $X = \{e_2, e_3, e_4, e_6, e_{10}\}$ and $Y = \{e_1, e_3, e_5, e_9, e_{10}\}$. (b) The admissible graph G^A for (X, Y).

We will consider two cases: $E_X \cap \{e \in E : v_e \in V^A\} \neq \emptyset$ and $E_X \cap \{e \in E : v_e \in V^A\} = \emptyset$. The first case means that there is a directed path from v_e, $e \in E_Y$, to a node v_f, $f \in E_X$, in the admissible graph G^A and in the second case no such a path exists. We will show that in the first case it is possible to find a new pair (X', Y') which satisfies the sufficient pair optimality conditions and $|E_{Z'}| = |E_Z| + 1$. The idea will be to perform a sequence of moves on X and Y, indicated by the arcs on some suitably chosen path from v_e, $e \in E_Y$, to v_f, $f \in E_X$ in the admissible graph G^A. Let us formally handle this case.

Lemma 2.7. *If $E_X \cap \{e \in E : v_e \in V^A\} \neq \emptyset$, then there exists a pair (X', Y') with $|E_{Z'}| = |E_Z| + 1$, which satisfies the sufficient pair optimality conditions for θ.*

Proof. We begin by introducing the notion of a *cycle graph* $G(T) = (V^T, A^T)$, corresponding to a given spanning tree T of graph $G = (V, A)$. We build $G(T)$ as follows: we associate with each edge $e \in E$ a node v_e and include it to V^T, $|E| = |V^T|$; then we add arc (v_e, v_f) to A^T if $e \notin T$ and $f \in P_T(e)$. An example is shown in Figure 2.2.

Claim 2.3. *Given a spanning tree T of G, let $\mathcal{F} = \{(v_{e_1}, v_{f_1}), (v_{e_2}, v_{f_2}), \ldots, (v_{e_\ell}, v_{f_\ell})\}$ be a subset of arcs of $G(T)$, where all v_{e_i} and v_{f_i} (resp. e_i and f_i), $i \in [\ell]$, are distinct. If $T' = T \cup \{e_1, \ldots, e_\ell\} \setminus \{f_1, \ldots, f_\ell\}$ is not a spanning tree, then $G(T)$ contains a subgraph depicted in Figure 2.3, where $\{j_1, \ldots, j_\kappa\} \subseteq [\ell]$.*

Let us illustrate Claim 2.3 by using the sample graph in Figure 2.2. Suppose that $\mathcal{F} = \{(v_{e_1}, v_{f_5}), (v_{e_2}, v_{f_2}), (v_{e_3}, v_{f_3})\}$. Then $T' = T \cup \{e_1, e_2, e_3, \} \setminus \{f_5, f_2, f_3\}$ is not a spanning tree and $G(T)$ contains the subgraph composed of the following arcs (see Figure 2.2):

$$(v_{e_1}, v_{f_2}), (v_{e_2}, v_{f_2}), (v_{e_2}, v_{f_3}), (v_{e_3}, v_{f_3}), (v_{e_3}, v_{f_5}), (v_{e_1}, v_{f_5}).$$

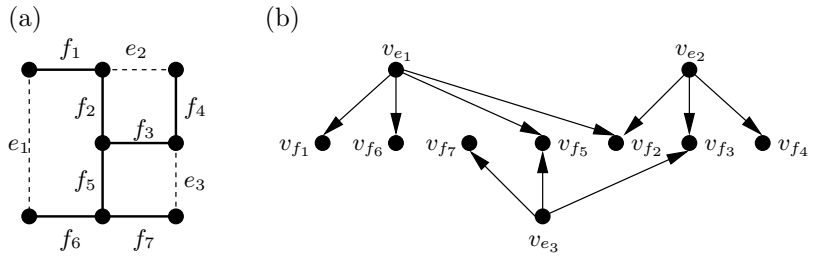


Figure 2.2: (a) A graph G with a spanning tree T (the solid lines). (b) The cycle graph $G(T)$.

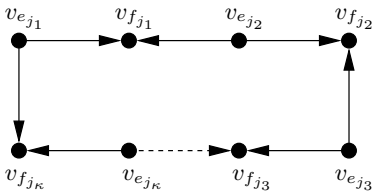


Figure 2.3: A subgraph of $G(T)$ from Claim 2.3.

Proof of Claim 2.3. We form T' by performing a sequence of moves consisting in adding edges e_i and removing edges $f_i \in P_T(e_i)$, $i \in [\ell]$. Suppose that, at some step, a cycle appears, which is formed by some edges from $\{e_1, \ldots, e_\ell\}$ and the remaining edges of T (not removed from T). Such a cycle must appear, since otherwise T' would be a spanning tree. Let us relabel the edges so that $\{e_1, \ldots, e_s\}$ are on this cycle, i.e., the first s moves consisting in adding e_i and removing f_i create the cycle, $i \in [s]$. An example of such a situation for $s = 4$ is shown in Figure 2.4. The cycle is formed by the edges $e_1, \ldots, e_4$ and the paths $P_{v_2v_3}$, $P_{v_4v_5}$ and $P_{v_1v_6}$ in T. Consider the edge $e_1 = \{v_1, v_2\}$. Because T is a spanning tree, $P_T(e_1) \subseteq P_{v_2v_3} \cup P_T(e_2) \cup P_T(e_3) \cup P_{v_4v_5} \cup P_T(e_4) \cup P_{v_1v_6}$. Observe that $f_1 \in P_T(e_1)$ cannot belong to any of $P_{v_2v_3}$, $P_{v_4v_5}$ and $P_{v_1v_6}$. If it would be contained in one of these paths, then no cycle would be created. Hence, f_1 must belong to $P_T(e_2) \cup P_T(e_3) \cup P_T(e_4)$. The above argument is general and, by using it, we can show that for each $i \in [s]$, $f_i \in P_T(e_j)$ for some $j \in [s] \setminus \{i\}$.

We are now ready to build a subgraph depicted in Figure 2.3. Consider a subgraph $G'(T)$ of the cycle graph $G(T)$ built as follows. The nodes of $G'(T)$ are $v_{e_1}, \ldots, v_{e_s}$, $v_{f_1}, \ldots, v_{f_s}$. Observe that $G'(T)$ has exactly $2s$ nodes, since all the edges $e_1, \ldots, e_s$, $f_1, \ldots, f_s$ are distinct by the assumption of the claim. For each $i \in [s]$ we add to $G'(T)$ two arcs, namely (v_{e_i}, v_{f_i}), $f_i \in P_T(e_i)$ and (v_{e_j}, v_{f_i}), $f_i \in P_T(e_j)$ for $j \in [s] \setminus \{i\}$ (see Figure 2.4). The resulting graph $G'(T)$ is bipartite and has exactly $2s$ arcs. In consequence $G'(T)$ (and thus $G(T)$) must contain a cycle which is of the form depicted in Figure 2.3. □

After this preliminary step, we can now return to the main proof. If $E_X \cap \{e \in E : v_e \in V^A\} \neq \emptyset$, then, by the construction of the admissible graph, there exists a directed path in G^A from a node v_e, $e \in E_Y$, to a node v_f, $f \in E_X$. Let P be a shortest such a path from v_e to v_f, i.e., a path consisting of the fewest number of arcs, called an *augmenting*

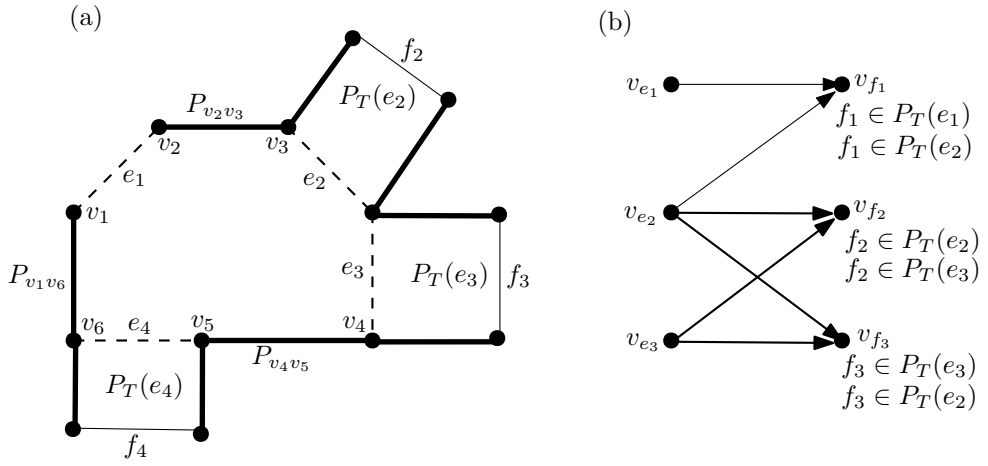


Figure 2.4: Illustration for the proof of Claim 2.3. (a) The bold lines represent paths in T (not necessarily disjoint); $f_1 \in P_T(e_2) \cup P_T(e_3) \cup P_T(e_4)$. (b) The subgraph $G'(T)$ with the corresponding cycle.

path. We need to consider the following cases:

1. The augmenting path P is of the form:

$$\begin{array}{ccc} E_Y & & E_X \\ v_e & \to & v_f \end{array}$$

   If (v_e, v_f) is X-arc, then $X' = X \cup \{e\} \setminus \{f\}$ is an updated spanning tree of G such that $|X' \cap Y| = |E_Z| + 1$. Furthermore X' is a minimum spanning tree for the costs C_e^* and the new pair (X', Y) satisfies the sufficient pair optimality conditions ($E_{X'} \subseteq E_X$, so condition (ii) in Theorem 2.3 is not violated). If (v_e, v_f) is Y-arc, then $Y' = Y \cup \{f\} \setminus \{e\}$ is an updated spanning tree of G such that $|X \cap Y'| = |E_Z| + 1$. Also Y' is a minimum spanning tree for the costs c_e^* and the new pair (X, Y') satisfies the sufficient pair optimality conditions. An example can be seen in Figure 2.1. There is a path $v_{e_1} \to v_{e_2}$ in the admissible graph. The arc (v_{e_1}, v_{e_2}) is both X-arc and Y-arc. We can thus choose one of the two possible moves $X' = X \cup \{e_1\} \setminus \{e_2\}$ or $Y' = Y \cup \{e_2\} \setminus \{e_1\}$, which results in (X', Y) or (Y', X).

2. The augmenting path P is of the form:

$$\begin{array}{lccccccccccccc} & E_Y & & E_Z & & E_W & & E_Z & & E_W & & E_Z & & E_X \\ (a)\ v_{e_1} & \overset{X}{\to} & v_{f_1} & \overset{Y}{\to} & v_{e_2} & \overset{X}{\to} & v_{f_2} & \overset{Y}{\to} & v_{e_3} & \overset{X}{\to} & v_{f_3} & \overset{Y}{\to} \cdots \overset{Y}{\to} & v_{e_\ell} & \overset{X}{\to} & v_{f_\ell} & \overset{Y}{\to} & v_{e_{\ell+1}} \\ & & & & & & & & & & & & & E_X & & & \\ (b) & & & & & & & & & & & & & \overset{X}{\to} & v_{f_\ell} & & \end{array}$$

   Let $X' = X \cup \{e_1, \ldots, e_\ell\} \setminus \{f_1, \ldots, f_\ell\}$. Let $Y' = Y \cup \{e_2, \ldots, e_{\ell+1}\} \setminus \{f_1, \ldots f_\ell\}$ for case (a), and $Y' = Y \cup \{e_2, \ldots, e_\ell\} \setminus \{f_1, \ldots f_{\ell-1}\}$ for case (b). We now have to show

that the resulting pair (X', Y') is a pair of spanning trees. Suppose that X' is not a spanning tree. Observe that the X-arcs $(v_{e_1}, v_{f_1}), \ldots, (v_{e_\ell}, v_{f_\ell})$ belong to the cycle graph $G(X)$. Thus, by Claim 2.3, the cycle graph $G(X)$ must contain a subgraph depicted in Figure 2.3, where $\{j_1, \ldots, j_\kappa\} \subseteq [\ell]$. An easy verification shows that all edges e_i, f_i, $i \in \{j_1, \ldots, j_\kappa\}$ must have the same costs with respect to C^*_e. Indeed, if some costs are different, then there exists an edge exchange which decreases the cost of X. This contradicts our assumption that X is a minimum spanning tree with respect to C^*_e. Finally, there must be an arc $(v_{e_{i'}}, v_{f_{i''}})$ in the subgraph such that $i' < i''$. Since $C^*_{e_{i'}} = C^*_{f_{i''}}$, the arc $(v_{e_{i'}}, v_{f_{i''}})$ is present in the admissible graph G^A. This leads to a contradiction with our assumption that P is an augmenting path. Now suppose that Y' is not a spanning tree. We consider only the case (a) since the proof of case (b) is just the same. For a convenience, let us number the nodes v_{e_i} on P from $i = 0$ to ℓ, so that $Y' = \{e_1, \ldots, e_\ell\} \setminus \{f_1, \ldots, f_\ell\}$. The arcs $(v_{e_1}, v_{f_1}), \ldots, (v_{e_\ell}, v_{f_\ell})$, which correspond to the Y-arcs $(v_{f_1}, v_{e_1}), \ldots, (v_{f_\ell}, v_{e_\ell})$ of P, belong to the cycle graph $G(Y)$. Hence, by Claim 2.3, $G(Y)$ must contain a subgraph depicted in Figure 2.3, where $\{i_1, \ldots, i_\kappa\} \subseteq [\ell]$. The rest of the proof is similar to the proof for X. Namely, the edges e_i and f_i for $i \in \{i_1, \ldots, i_\kappa\}$ must have the same costs with respect to c^*_e. Also, there must exist an arc $(v_{e_{i'}}, v_{f_{i''}})$ in the subgraph such that $i' > i''$. In consequence, the arc $(v_{f_{i''}}, v_{e_{i'}})$ belongs to the admissible graph, which contradicts the assumption that P is an augmenting path.

An example of the case (a) is shown in Figure 2.5. Thus $X' = X \cup \{e_1, e_2, e_3, e_4\} \setminus \{f_1, f_2, f_3, f_4\}$ and $Y' = Y \cup \{e_2, e_3, e_4, e_5\} \setminus \{f_1, f_2, f_3, f_4\}$. An example of the case (b) is shown in Figure 2.6. In this example X' is the same as in the previous case and $Y' = Y \cup \{e_2, e_3, e_4\} \setminus \{f_1, f_2, f_3\}$.

It is easy to verify that $|E_{Z'}| = |X' \cap Y'| = |E_Z| + 1$ holds (see also the examples in Figures 2.5 and 2.6). The spanning trees X' and Y' are optimal for the costs C^*_e and c^*_e, respectively. Furthermore, $E_{X'} \subseteq E_X$ and $E_{Y'} \subseteq E_Y$, so (X', Y') satisfies the sufficient pair optimality conditions (the condition (ii) in Theorem 2.3 is not violated).

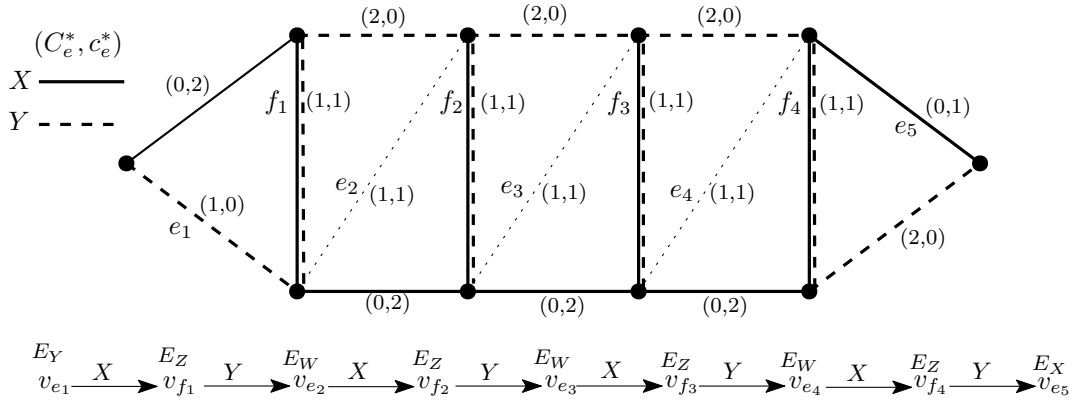


Figure 2.5: A pair (X, Y) and the corresponding admissible graph for the case 2a.

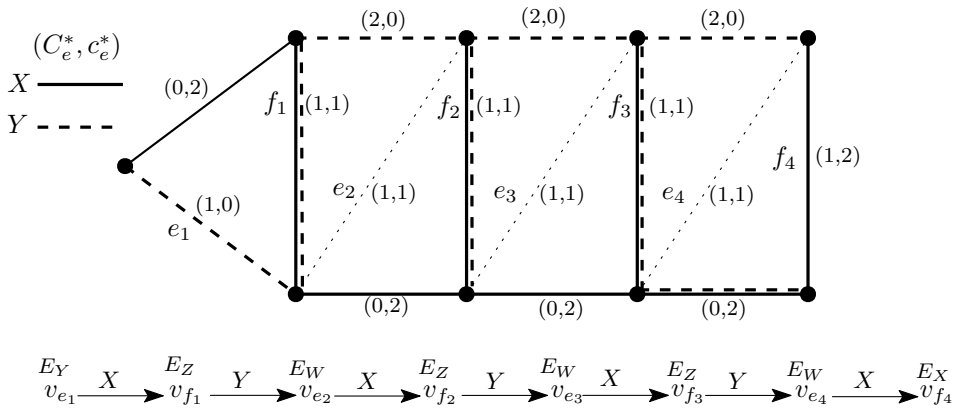


Figure 2.6: A pair (X, Y) and the corresponding admissible graph for the case 2b.

3. The augmenting path P is of the form

$$(a)\quad v_{e_1} \xrightarrow{Y} v_{f_1} \xrightarrow{X} v_{e_2} \xrightarrow{Y} v_{f_2} \xrightarrow{X} v_{e_3} \xrightarrow{Y} v_{f_3} \xrightarrow{X} \ldots \xrightarrow{X} v_{e_\ell} \xrightarrow{Y} v_{f_\ell} \xrightarrow{X} v_{e_{\ell+1}}$$

$$(b)\quad \xrightarrow{Y} v_{f_\ell} \xrightarrow{X}$$

Let $X' = X \cup \{f_1, \ldots, f_\ell\} \setminus \{e_2, \ldots, e_{\ell+1}\}$ for the case (a) and $X' = X \cup \{f_1, \ldots, f_{\ell-1}\} \setminus \{e_2, \ldots e_\ell\}$ for the case (b). Let $Y' = Y \cup \{f_1, \ldots, f_\ell\} \setminus \{e_1, \ldots e_\ell\}$. The proof that X' and Y' are spanning trees follows by the same arguments as for the symmetric case described in point 2. An example of the case (a) is shown in Figure 2.7. Thus $X' = X \cup \{f_1, f_2, f_3, f_4\} \setminus \{e_2, e_3, e_4, e_5\}$ and $Y' = Y \cup \{f_1, f_2, f_3, f_4\} \setminus \{e_1, e_2, e_3, e_4\}$. An example for the case (b) is shown in Figure 2.8. The spanning tree Y' is the same as in the previous case and $X' = X \cup \{f_1, f_2, f_3\} \setminus \{e_2, e_3, e_4\}$. The equality $|E_{Z'|} = |X' \cap Y'| = |E_Z| + 1$ holds. Also, the trees X' and Y' are optimal for the costs C_e^* and c_e^*, respectively, $E_{X'} \subseteq E_X$, $E_{Y'} \subseteq E_Y$, so (X', Y') satisfies the sufficient pair optimality conditions.

□

We now turn to the case $E_X \cap \{e \in E : v_e \in V^A\} = \emptyset$. Fix $\delta > 0$ (the precise value of δ will be specified later) and set:

$$C_e(\delta) = C_e^* - \delta, \qquad c_e(\delta) = c_e^* \qquad v_e \in V^A, \tag{2.45a}$$

$$C_e(\delta) = C_e^*, \qquad c_e(\delta) = c_e^* - \delta \qquad v_e \notin V^A. \tag{2.45b}$$

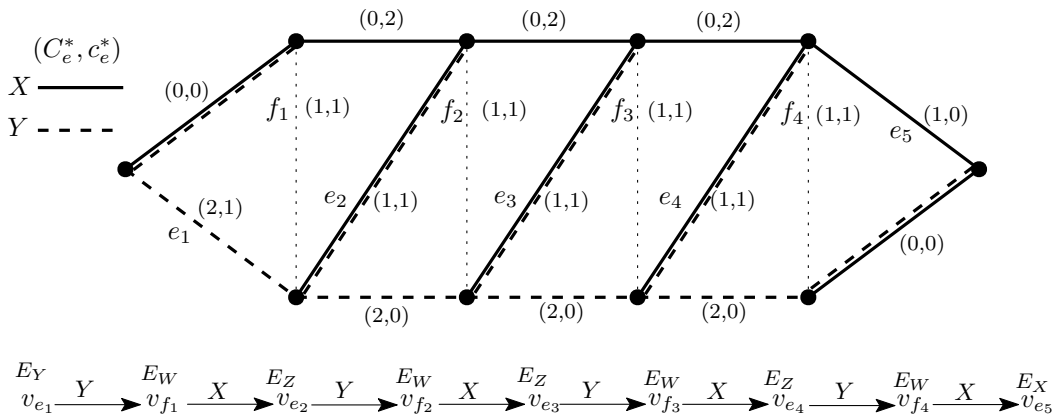


Figure 2.7: A pair (X, Y) and the corresponding admissible graph for the case 3a.

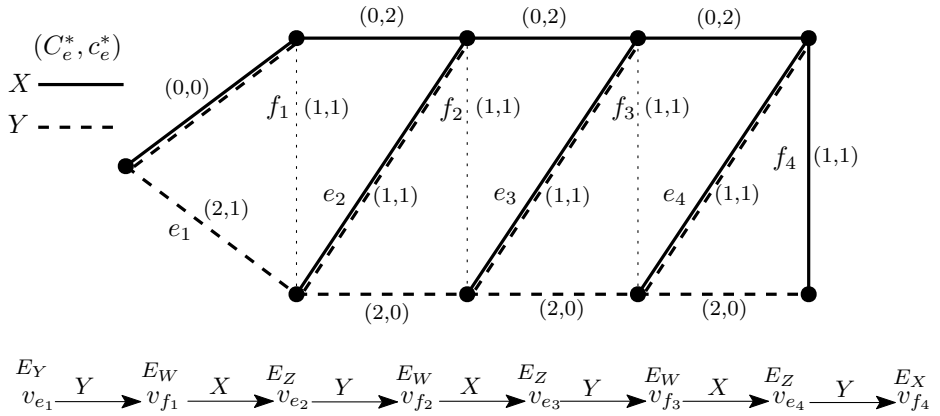


Figure 2.8: A pair (X, Y) and the corresponding admissible graph for the case 3b.

Lemma 2.8. *There exists a sufficiently small $\delta > 0$ such that the costs $C_e(\delta)$ and $c_e(\delta)$ satisfy the path optimality conditions for X and Y, respectively, i.e:*

$$\text{for every } e \notin X \qquad C_e(\delta) \geq C_f(\delta) \qquad \text{for every } f \in P_X(e), \tag{2.46a}$$

$$\text{for every } e \notin Y \qquad c_e(\delta) \geq c_f(\delta) \qquad \text{for every } f \in P_Y(e). \tag{2.46b}$$

Proof. If $C_e^* > C_f^*$ (resp. $c_e^* > c_f^*$), $e \notin X, f \in P_X(e)$ (resp. $e \notin Y, f \in P_Y(e)$), then there is $\delta > 0$, such that after setting the new costs (2.45) the inequality $C_e(\delta) \geq C_f(\delta)$ (resp. $c_e(\delta) \geq c_f(\delta)$) holds. Hence, one can choose a sufficiently small $\delta > 0$ such that after setting the new costs (2.45), all the strong inequalities are not violated. Therefore, for such a chosen δ it remains to show that all originally tight inequalities in (2.44) are preserved for the new costs. Consider a tight inequality of the form:

$$C_e^* = C_f^*,\ e \notin X,\ f \in P_X(e). \tag{2.47}$$

On the contrary, suppose that $C_e(\delta) < C_f(\delta)$. This is only possible when $C_e(\delta) = C_e^* - \delta$ and $C_f(\delta) = C_f^*$. Hence and from the construction of the new costs, we have $v_f \notin V^A$ (see (2.45b)) and $v_e \in V^A$ (see (2.45a)). By (2.47), we obtain $(v_e, v_f) \in E^A$. Thus $v_f \in V^A$, a contradiction. Consider a tight inequality of the form:

$$c_e^* = c_f^*,\ e \notin Y,\ f \in P_Y(e). \tag{2.48}$$

On the contrary, suppose that $c_e(\delta) < c_f(\delta)$. This is only possible when $c_e(\delta) = c_e^* - \delta$ and $c_f(\delta) = c_f^*$. Thus we deduce that $v_e \notin V^A$ and $v_f \in V^A$ (see (2.45)). From (2.48), it follows that $(v_f, v_e) \in E^A$ and so $v_e \in V^A$, a contradiction. □

We are now ready to give the precise value of δ. We do this by increasing the value of δ until some inequalities, originally not tight in (2.44), become tight. Namely, let $\delta^* > 0$ be the smallest value of δ for which an inequality originally not tight becomes tight. Obviously, it occurs when $C_e^* - \delta^* = C_f^*$ for $e \notin X$, $f \in P_X(e)$ or $c_f^* - \delta^* = c_e^*$ for $f \notin Y$, $e \in P_Y(f)$. By (2.45), $v_e \in V^A$ and $v_f \notin V^A$. Accordingly, if $\delta = \delta^*$, then at least one arc is added to G^A. Observe also that no arc can be removed from G^A - the admissibility of the nodes remains unchanged. It follows from the fact that each tight inequality for $v_e \in V^A$ and $v_f \in V^A$ is still tight. This leads to the following lemma.

Lemma 2.9. *If $E_X \cap \{e \in E : v_e \in V^A\} = \emptyset$, then (X, Y) satisfies the sufficient pair optimality conditions for each $\theta' \in [\theta, \theta + \delta^*]$.*

Proof. Set $\theta' = \theta + \delta$, $\delta \in [0, \delta^*]$. Lemma 2.8 implies that X is optimal for $C_e(\delta)$ and Y is optimal for $c_e(\delta)$. From (2.45) and the definition of the costs C_e^* and c_e^*, it follows that $C_e(\delta) = C_e - \alpha'_e$ and $c_e(\delta) = c_e - \beta'_e$, where $\alpha'_e = \alpha_e + \delta$ and $\beta'_e = \beta_e$ for each $v_e \in V^A$, $\alpha'_e = \alpha_e$ and $\beta'_e = \beta_e + \delta$ for each $v_e \notin V^A$. Notice that $\alpha'_e + \beta'_e = \alpha_e + \beta_e + \delta = \theta + \delta = \theta'$ for each $e \in E$. By (2.45), $c_e(\delta) = c_e$ for each $e \in E_Y$ (recall that $e \in E_Y$ implies $v_e \in V^A$), and thus $\beta_e = 0$ for each $e \in E_Y$. Since $E_X \cap \{e \in E : v_e \in V^A\} = \emptyset$, $C_e(\delta) = C_e^* = C_e$ holds for each $e \in E_X$, and so $\alpha_e = 0$ for each $e \in E_X$. We thus have shown that there exist $\alpha'_e, \beta'_e \geq 0$ such that $\alpha'_e + \beta'_e = \theta'$ for each $e \in E$ satisfying the conditions (i) and (ii) in Theorem 2.3, which completes the proof. □

We now describe a polynomial procedure that, for a given pair (X, Y) satisfying the sufficient pair optimality conditions for some $\theta \geq 0$, finds a new pair of spanning trees (X', Y'), which also satisfies the sufficient pair optimality conditions with $|E'_Z| = |E_Z| + 1$. We start by building the admissible graph $G^A = (V^A, E^A)$ for (X, Y). If this graph contains an augmenting path, then by Lemma 2.7, we are done. Otherwise, we determine δ^* and modify the costs by using (2.45). Lemma 2.9 shows that (X, Y) satisfies the sufficient pair optimality conditions for $\theta + \delta^*$. For δ^* some new arcs are added to the admissible graph G^A (all the previous arcs must be still present in G^A). Thus G^A is updated and we set $C_e^* := C_e(\delta^*)$, $c_e^* := c_e(\delta^*)$ for each $e \in E$, and $\theta := \theta + \delta^*$. We repeat this until there is an augmenting path in $G^A = (V^A, E^A)$. Note that such a path must appear after at most $m = |E|$ iterations, which follows from the fact that at some step a node v_e such that $e \in E_X$ must appear in G^A.

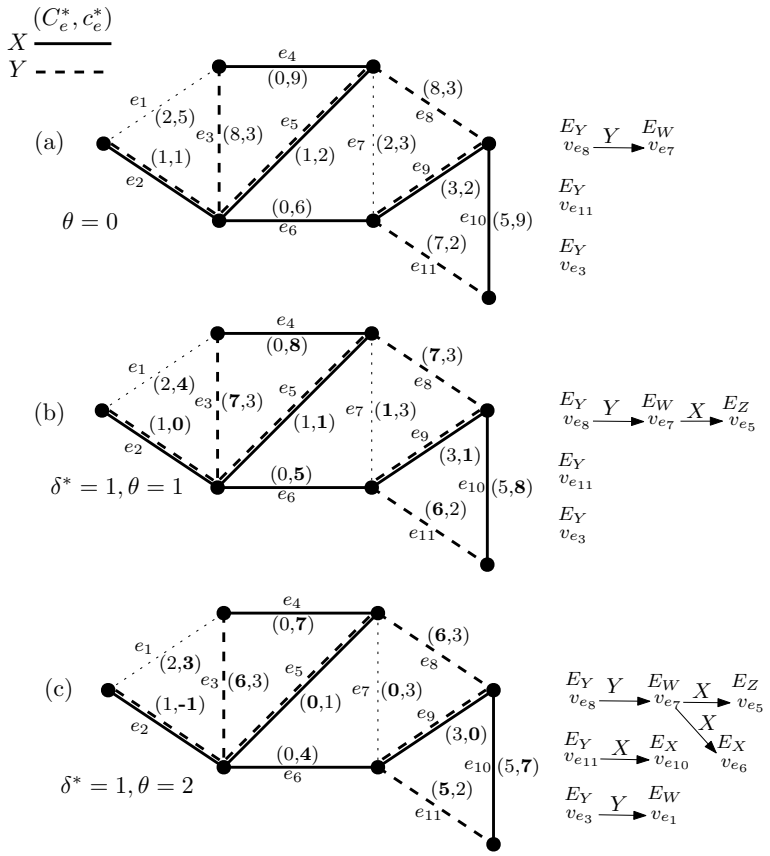


Figure 2.9: Sample computations, $X = \{e_2, e_4, e_5, e_6, e_9, e_{10}\}$ and $Y = \{e_2, e_3, e_5, e_8, e_9, e_{11}\}$.

Sample computations are shown in Figure 2.9. We start with the pair (X, Y), where $X = \{e_2, e_4, e_5, e_6, e_9, e_{10}\}$ and $Y = \{e_2, e_3, e_5, e_8, e_9, e_{11}\}$, which satisfies the sufficient pair optimality conditions for $\theta = 0$ (see Figure 2.9a). Observe that in this case it is enough to check that X is optimal for the costs $C_e^* = C_e$ and Y is optimal for the costs $c_e^* = c_e$, $e \in E$. For $\theta = 0$, the admissible graph does not contain any augmenting path. We thus have to modify the costs C_e^* and c_e^*, according to (2.45). For $\delta^* = 1$, a new inequality becomes tight and one arc is added to the admissible graph (see Figure 2.9b). The admissible graph still does not have an augmenting path, so we have to again modify the costs. For $\delta^* = 1$ some new inequalities become tight and three arcs are added to the admissible graph (see Figure 2.9c). Now the admissible graph has two augmenting paths (cases 1 and 3a, see the proof of Lemma 2.7). Choosing one of them, and performing the modification described in the proof of Lemma 2.7 we get a new pair (X', Y') with $|E_{Z'}| = |E_Z| + 1$.

Let us now estimate the running time of the procedure. The admissible graph has at most m nodes and at most mn arcs. It can be built in $O(nm)$ time. The augmenting path in the admissible graph can be found in $O(nm)$ time by applying the breath first search. Also the number of inequalities which must be analyzed to find δ^* is $O(nm)$. Since we have to update the cost of each arc of the admissible graph at most m times, until an

augmenting path appears, the required time of the procedure is $O(m^2n)$. We thus get the following result.

Theorem 2.4. *The* REC ST *problem is solvable in* $O(Lm^2n)$ *time, where* $L = n-1-k$.

2.4 Combinatorial algorithm for recoverable robust spanning tree problem

In this section we are concerned with the RR ST problem under the interval uncertainty representation, i.e., for the scenario sets $\mathcal{U}^I$, $\mathcal{U}_1^I(\Gamma)$, and $\mathcal{U}_2^I(\Gamma)$. Using the polynomial algorithm for REC ST, constructed in Section 2.3, we will provide a polynomial algorithm for RR ST under $\mathcal{U}^I$ and some approximation algorithms for a wide class of RR ST under $\mathcal{U}_1^I(\Gamma)$ and $\mathcal{U}_2^I(\Gamma)$. The idea will be to solve REC ST for a suitably chosen second stage costs. Recall

$$\textsc{Eval}(X) : \sum_{e\in X} C_e + \max_{S\in\mathcal{U}} \min_{Y\in\Phi_X^k} \sum_{e\in Y} c_e^S.$$

It is worth pointing out that under scenario sets $\mathcal{U}^I$ and $\mathcal{U}_2^I(\Gamma)$, the value of $\textsc{Eval}(X)$, for a given spanning tree X, can be computed in polynomial time [22, 58]. On the other hand, computing $\textsc{Eval}(X)$ under $\mathcal{U}_1^I(\Gamma)$ turns out to be strongly NP-hard [28, 58]. Given scenario $S = (c_e^S)_{e\in E}$, consider the following REC ST problem:

$$\min_{X\in\Phi}\left(\sum_{e\in X} C_e + \min_{Y\in\Phi_X^k}\sum_{e\in Y} c_e^S\right). \tag{2.49}$$

Problem (2.49) is equivalent to the formulation (1.23) for $S = (c_e)_{e\in E}$ and it is polynomially solvable, according to the result obtained in Section 2.3. As in the previous section, we denote by pair (X, Y) a solution to (2.49), where $X \in \Phi$ and $Y \in \Phi_X^k$. Given S, we call (X, Y) an *optimal pair under* S if (X, Y) is an optimal solution to (2.49).

The RR ST problem with scenario set $\mathcal{U}^I$ can be rewritten as follows:

$$\min_{X\in\Phi}\left(\sum_{e\in X} C_e + \max_{S\in\mathcal{U}^I}\min_{Y\in\Phi_X^k}\sum_{e\in Y} c_e^S\right) = \min_{X\in\Phi}\left(\sum_{e\in X} C_e + \min_{Y\in\Phi_X^k}\sum_{e\in E}(c_e+d_e)\right). \tag{2.50}$$

Thus (2.50) is (2.49) for $S = (c_e + d_e)_{e\in E} \in \mathcal{U}^I$. Hence and from Theorem 2.4 we immediately get the following theorem:

Theorem 2.5. *For scenario set* $\mathcal{U}^I$*, the* RR ST *problem is solvable in* $O((n-1-k)m^2n)$ *time.*

We now address RR ST under $\mathcal{U}_1^I(\Gamma)$ and $\mathcal{U}_2^I(\Gamma)$. Suppose that $c_e \geq \alpha(c_e + d_e)$ for each $e \in E$, where $\alpha \in (0, 1]$ is a given constant. This inequality means that for each edge $e \in E$ the nominal cost c_e is positive and $c_e + d_e$ is at most $1/\alpha$ greater than c_e. It is reasonable to assume that this condition will be true in many practical applications for not very large value of $1/\alpha$.

Lemma 2.10. *Suppose that $c_e \geq \alpha(c_e + d_e)$ for each $e \in E$, where $\alpha \in (0, 1]$, and let $(\hat{X}, \hat{Y})$ be an optimal pair under $\underline{S} = (c_e)_{e\in E}$. Then for the scenario sets $\mathcal{U}_1^I(\Gamma)$ and $\mathcal{U}_2^I(\Gamma)$ the inequality* $\textsc{Eval}(\hat{X}) \leq \frac{1}{\alpha}\textsc{Eval}(X)$ *holds for any $X \in \Phi$.*

Proof. We give the proof only for the scenario set $\mathcal{U}_1^I(\Gamma)$. The proof for $\mathcal{U}_2^I(\Gamma)$ is the same. Let $X \in \Phi$. The following inequality is satisfied:

$$\textsc{Eval}(X) = \sum_{e\in X} C_e + \max_{S\in\mathcal{U}_1^I(\Gamma)} \min_{Y\in\Phi_X^k} \sum_{e\in Y} c_e^S = \sum_{e\in X} C_e + \sum_{e\in Y^*} c_e^{S^*} \geq \sum_{e\in X} C_e + \sum_{e\in Y^*} c_e^{\underline{S}}.$$

Clearly, (X, Y^*) is a feasible pair to (2.49) under $\underline{S}$. From the definition of $(\hat{X}, \hat{Y})$ we get

$$\begin{aligned}\textsc{Eval}(X) \geq \sum_{e\in\hat{X}} C_e + \sum_{e\in\hat{Y}} c_e^{\underline{S}} = \sum_{e\in\hat{X}} C_e + \sum_{e\in\hat{Y}} c_e \geq \sum_{e\in\hat{X}} C_e + \sum_{e\in\hat{Y}} \alpha(c_e + d_e) \\ = \sum_{e\in\hat{X}} C_e + \alpha \sum_{e\in\hat{Y}} c_e^{\overline{S}},\end{aligned} \tag{2.51}$$

where $\overline{S} = (c_e + d_e)_{e\in E}$. Hence

$$\begin{aligned}\textsc{Eval}(X) &\geq \sum_{e\in\hat{X}} C_e + \alpha \max_{S\in\mathcal{U}_1^I(\Gamma)} \sum_{e\in\hat{Y}} c_e^S \geq \sum_{e\in\hat{X}} C_e + \alpha \max_{S\in\mathcal{U}_1^I(\Gamma)} \min_{Y\in\Phi_{\hat{X}}^k} \sum_{e\in Y} c_e^S \\ &\geq \alpha \left(\sum_{e\in\hat{X}} C_e + \max_{S\in\mathcal{U}_1^I(\Gamma)} \min_{Y\in\Phi_{\hat{X}}^k} \sum_{e\in Y} c_e^S \right) = \alpha\textsc{Eval}(\hat{X})\end{aligned}$$

and the lemma follows. □

The condition $c_e \geq \alpha(c_e + d_e)$, $e \in E$, in Lemma 2.10, can be weakened and, in consequence, the set of instances to which the approximation ratio of the algorithm applies can be extended. Indeed, from inequality (2.51) it follows that the bounds of the uncertainty intervals are only required to meet the condition $\sum_{e\in\hat{Y}} c_e \geq \alpha \sum_{e\in\hat{Y}} (c_e + d_e)$. This condition can be verified efficiently, since $\hat{Y}$ can be computed in polynomial time.

We now focus on RR ST for $\mathcal{U}_2^I(\Gamma)$. Define $D = \sum_{e\in E} d_e$ and suppose that $D > 0$ (if $D = 0$, then the problem is equivalent to Rec ST for the second stage costs c_e, $e \in E$). Consider scenario S' under which $c_e^{S'} = \min\{c_e + d_e, c_e + \Gamma\frac{d_e}{D}\}$ for each $e \in E$. Obviously, $S' \in \mathcal{U}_2^I(\Gamma)$, since $\sum_{e\in E} \delta_e \leq \sum_{e\in E} \Gamma\frac{d_e}{D} \leq \Gamma$. The following theorem provides another approximation result for RR ST with scenario set $\mathcal{U}_2^I(\Gamma)$:

Lemma 2.11. *Let $(\hat{X}, \hat{Y})$ be an optimal pair under S'. Then the following implications are true for scenario set $\mathcal{U}_2^I(\Gamma)$:*

(i) If $\Gamma \geq \beta D$, $\beta \in (0, 1]$, then $\textsc{Eval}(\hat{X}) \leq \frac{1}{\beta}\textsc{Eval}(X)$ *for any $X \in \Phi$.*

(ii) If $\Gamma \leq \gamma\textsc{Eval}(\hat{X})$, $\gamma \in [0, 1)$ then $\textsc{Eval}(\hat{X}) \leq \frac{1}{1-\gamma}\textsc{Eval}(X)$ *for any $X \in \Phi$.*

Proof. Let $X \in \Phi$. Since $S' \in \mathcal{U}_2^I(\Gamma)$, we get

$$\textsc{Eval}(X) = \sum_{e\in X} C_e + \max_{S\in\mathcal{U}_2^I(\Gamma)} \min_{Y\in\Phi_X^k} \sum_{e\in Y} c_e^S \geq \sum_{e\in X} C_e + \min_{Y\in\Phi_X^k} \sum_{e\in Y} c_e^{S'}. \tag{2.52}$$

We first prove implication (i). By (2.52) and the definition of $(\hat{X}, \hat{Y})$, we obtain

$$\begin{aligned}
\textsc{Eval}(X) &\geq \sum_{e\in\hat{X}} C_e + \sum_{e\in\hat{Y}} c_e^{S'} = \sum_{e\in\hat{X}} C_e + \sum_{e\in\hat{Y}} \min\{c_e + d_e, c_e + \Gamma\frac{d_e}{D}\} \\
&\geq \sum_{e\in\hat{X}} C_e + \sum_{e\in\hat{Y}} \min\{c_e + d_e, c_e + \beta d_e\} = \sum_{e\in\hat{X}} C_e + \sum_{e\in\hat{Y}} (c_e + \beta d_e) \\
&\geq \sum_{e\in\hat{X}} C_e + \beta \sum_{e\in\hat{Y}} c_e^{\overline{S}},
\end{aligned}$$

where $\overline{S} = (c_e + d_e)_{e\in E}$. The rest of the proof is the same as in the proof of Lemma 2.10. We now prove implication (ii). By (2.52) and the definition of $(\hat{X}, \hat{Y})$, we have

$$\begin{aligned}
\textsc{Eval}(X) &\geq \sum_{e\in\hat{X}} C_e + \sum_{e\in\hat{Y}} c_e^{S'} \geq \sum_{e\in\hat{X}} C_e + \sum_{e\in\hat{Y}} c_e^{\overline{S}} \geq \sum_{e\in\hat{X}} C_e + \max_{S\in\mathcal{U}_2^I(\Gamma)} \sum_{e\in\hat{Y}} c_e^S - \Gamma \\
&\geq \sum_{e\in\hat{X}} C_e + \max_{S\in\mathcal{U}_2^I(\Gamma)} \min_{Y\in\Phi_{\hat{X}}^k} \sum_{e\in Y} c_e^S - \Gamma = \textsc{Eval}(\hat{X}) - \Gamma.
\end{aligned}$$

If $\Gamma \leq \gamma\textsc{Eval}(\hat{X})$. Then $\textsc{Eval}(X) \geq \textsc{Eval}(\hat{X}) - \gamma\textsc{Eval}(\hat{X}) = (1-\gamma)\textsc{Eval}(\hat{X})$ and $\textsc{Eval}(\hat{X}) \leq \frac{1}{1-\gamma}\textsc{Eval}(X)$. □

Note that the value of $\textsc{Eval}(\hat{X})$ under $\mathcal{U}_2^I(\Gamma)$ can be computed in polynomial time [58]. In consequence, the constants β and γ can be efficiently determined for every particular instance of the problem. Clearly, we can assume that $d_e \leq \Gamma$ for each $e \in E$, which implies $D \leq m\Gamma$, where $m = |E|$. Hence, we can assume that $\Gamma \geq \frac{1}{m}D$ for every instance of the problem. We thus get from Lemma 2.11 (implication (i)) that $\textsc{Eval}(\hat{X}) \leq m\textsc{Eval}(X)$ for any $X \in \Phi$ and the problem is approximable within m. If α, β and γ are the constants from Lemmas 2.10 and 2.11, then the following theorem summarizes the approximation results:

Theorem 2.6. RR ST *is approximable within* $\frac{1}{\alpha}$ *under scenario set* $\mathcal{U}_1^I(\Gamma)$ *and it is approximable within* $\min\{\frac{1}{\beta}, \frac{1}{\alpha}, \frac{1}{1-\gamma}\}$ *under scenario set* $\mathcal{U}_2^I(\Gamma)$.

Observe that Lemma 2.10 and Lemma 2.11 hold of any sets Φ and Φ_X^k (the particular structure of these sets is not exploited). Hence the approximation algorithms can be applied to any problem for which the recoverable version (2.49) is polynomially solvable.

2.5 Recoverable robust matroid basis problem

The minimum spanning tree can be generalized to the following *minimum matroid basis problem*. Here we make an assumption that checking the independence of a set $A \subseteq E$ can be done in polynomial time (see Section 1.4). The cardinality of each basis equals $r_M(E)$. Let c_e be a cost specified for each element $e \in E$. We wish to find a basis of M of the minimum total cost, $\sum_{e \in M} c_e$. It is well known that the minimum matroid basis problem is polynomially solvable by a greedy algorithm (see, e.g., [26]). A spanning tree is a basis of a *graphic matroid*, in which E is a set of edges of a given graph and $\mathcal{I}$ is the set of all forests in G.

Assume now that the first stage cost of element $e \in E$ equals C_e and its second stage cost is uncertain and is modeled by interval $[c_e, c_e + d_e]$. The *recoverable robust matroid basis problem* (RR MB for short) under scenario set $\mathcal{U}^I$ can be stated similarly to RR ST. Indeed, it suffices to replace the set of spanning trees by the bases of M and $\mathcal{U}$ by $\mathcal{U}^I$ in the formulation (2.1) and, in consequence, in (2.4). Here and subsequently, Φ denotes the set of all bases of M. Likewise, RR MB under $\mathcal{U}^I$ is equivalent to the following problem:

$$\begin{array}{ll} \min & \sum_{e \in X} C_e + \sum_{e \in Y} (c_e + d_e) \\ \text{s.t.} & |X \cap Y| \geq r_M(E) - k, \\ & X, Y \in \Phi. \end{array} \tag{2.53}$$

Let $E_X, E_Y \subseteq E$ and $\mathcal{I}_X$, $\mathcal{I}_Y$ be collections of subsets of E_X and E_Y, respectively, (independent sets), that induce matroids $M_X = (E_X, \mathcal{I}_X)$ and $M_Y = (E_Y, \mathcal{I}_Y)$. Let E_Z be a subset of E such that $E_Z \subseteq E_X \cup E_Z$ and $|E_Z| \geq L$ for some fixed L. The following linear program, denoted by $LP_{RRMB}(E_X, \mathcal{I}_X, E_Y, \mathcal{I}_Y, E_Z, L)$, after setting $E_X = E_Y = E_Z = E$, $\mathcal{I}_X = \mathcal{I}_Y = \mathcal{I}$ and $L = r_M(E) - k$ is a relaxation of (2.53):

$$\min \sum_{e \in E_X} C_e x_e + \sum_{e \in E_Y} (c_e + d_e) y_e \tag{2.54}$$

$$\text{s.t.} \sum_{e \in E_X} x_e = r_{M_X}(E_X), \tag{2.55}$$

$$\sum_{e \in U} x_e \leq r_{M_X}(U), \qquad \forall U \subset E_X, \tag{2.56}$$

$$-x_e + z_e \leq 0, \qquad \forall e \in E_X \cap E_Z, \tag{2.57}$$

$$\sum_{e \in E_Z} z_e = L, \tag{2.58}$$

$$z_e - y_e \leq 0, \qquad \forall e \in E_Y \cap E_Z, \tag{2.59}$$

$$\sum_{e \in E_Y} y_e = r_{M_Y}(E_Y), \tag{2.60}$$

$$\sum_{e \in U} y_e \leq r_{M_Y}(U), \qquad \forall U \subset E_Y, \tag{2.61}$$

$$x_e \geq 0, \qquad \forall e \in E_X, \tag{2.62}$$

$$z_e \geq 0, \qquad \forall e \in E_Z, \tag{2.63}$$
$$y_e \geq 0, \qquad \forall e \in E_Y. \tag{2.64}$$

The indicator variables $x_e, y_e, z_e \in \{0,1\}$, $e \in E$, describing the bases X, Y and their intersection $X \cap Y$, respectively have been relaxed. Since there are no variables z_e in the objective function (2.54), we can use equality constraint (2.58), instead of the inequality one. The above linear program is solvable in polynomial time. The rank constraints (2.55), (2.56) and (2.60), (2.61) relate to matroids $M_X = (E_X, \mathcal{I}_X)$ and $M_Y = (E_Y, \mathcal{I}_Y)$, respectively, and a separation over these constraints can be carried out in polynomial time [23]. Obviously, a separation over (2.57), (2.58) and (2.59) can be done in polynomial time as well.

Consider a vertex solution $(\boldsymbol{x}, \boldsymbol{z}, \boldsymbol{y}) \in \mathbb{R}_{\geq 0}^{|E_X|\times|E_Z|\times|E_Y|}$ of $LP_{RRMB}(E_X, \mathcal{I}_X, E_Y, \mathcal{I}_Y, E_Z, L)$. Note that if $E_Z = \emptyset$, then the only rank constraints are left in (2.54)-(2.64). Consequently, $\boldsymbol{x}$ and $\boldsymbol{y}$ are 0-1 incidence vectors of bases X and Y of matroids M_X and M_Y, respectively (see [26]). Let us turn to other cases. Assume that $E_X \neq \emptyset$, $E_Y \neq \emptyset$ and $E_Z \neq \emptyset$. Similarly as in Section 2.2 we first reduce the sets E_X, E_Y and E_Z by removing all elements e with $x_e = 0$, or $y_e = 0$, or $z_e = 0$. Let $\mathcal{F}(\boldsymbol{x})$ and $\mathcal{F}(\boldsymbol{y})$ defined in the following way:

$$\mathcal{F}(\boldsymbol{x}) = \{U \subseteq E_X \,:\, \sum_{e\in U} x_e = r_{M_X}(U)\},$$
$$\mathcal{F}(\boldsymbol{y}) = \{U \subseteq E_Y \,:\, \sum_{e\in U} y_e = r_{M_Y}(U)\}$$

denote the sets of subsets of elements that indicate tight constraints (2.55), (2.56) and (2.60), (2.61) for $\boldsymbol{x}$ and $\boldsymbol{y}$, respectively. Similarly we define the sets of elements that indicate tight constraints (2.57) and (2.59) with respect to $(\boldsymbol{x}, \boldsymbol{z}, \boldsymbol{y})$, namely $\mathcal{E}(\boldsymbol{x}, \boldsymbol{z})$ and $\mathcal{E}(\boldsymbol{z}, \boldsymbol{y})$:

$$\mathcal{E}(\boldsymbol{x}, \boldsymbol{z}) = \{e \in E_X \cap E_Z \,:\, -x_e + z_e = 0\},$$
$$\mathcal{E}(\boldsymbol{z}, \boldsymbol{y}) = \{e \in E_Y \cap E_Z \,:\, z_e - y_e = 0\}.$$

Let $\chi_X(W)$, $W \subseteq E_X$, (resp. $\chi_Z(W)$, $W \subseteq E_Z$, and $\chi_Y(W)$, $W \subseteq E_Y$) denote the characteristic vector in $\{0,1\}^{|E_X|} \times \{0\}^{|E_Z|} \times \{0\}^{|E_Y|}$ (resp. $\{0\}^{|E_X|} \times \{0,1\}^{|E_Z|} \times \{0\}^{|E_Y|}$ and $\{0\}^{|E_X|} \times \{0\}^{|E_Z|} \times \{0,1\}^{|E_Y|}$) that has 1 if $e \in W$ and 0 otherwise.

We recall that a family $\mathcal{L} \subseteq 2^E$ is a *chain* if for any $A, B \in \mathcal{L}$, either $A \subseteq B$ or $B \subseteq A$ (see, e.g., [51]). Let $\mathcal{L}(\boldsymbol{x})$ (resp. $\mathcal{L}(\boldsymbol{y})$) be a *maximal chain subfamily* of $\mathcal{F}(\boldsymbol{x})$ (resp. $\mathcal{F}(\boldsymbol{y})$). The following lemma is a fairly straightforward adaptation of [51, Lemma 5.2.3] to the problem under consideration and its proof may be handled in much the same way.

Lemma 2.12. *For $\mathcal{L}(\boldsymbol{x})$ and $\mathcal{L}(\boldsymbol{y})$ the following equalities:*

$$\operatorname{span}(\{\chi_X(E_X(U)) \,:\, U \in \mathcal{L}(\boldsymbol{x})\}) = \operatorname{span}(\{\chi_X(E_X(U)) \,:\, U \in \mathcal{F}(\boldsymbol{x})\}),$$
$$\operatorname{span}(\{\chi_Y(E_Y(U)) \,:\, U \in \mathcal{L}(\boldsymbol{y})\}) = \operatorname{span}(\{\chi_Y(E_Y(U)) \,:\, U \in \mathcal{F}(\boldsymbol{y})\})$$

hold.

Algorithm 2.2 Algorithm for RR MB

$M_X = (E_X, \mathcal{I}_X) \leftarrow (E, \mathcal{I})$, $M_Y = (E_Y, \mathcal{I}_Y) \leftarrow (E, \mathcal{I})$, $L \leftarrow r_M(E) - k$, $X \leftarrow \emptyset$, $Y \leftarrow \emptyset$, $Z \leftarrow \emptyset$
while $E_X \neq \emptyset$ **or** $E_Y \neq \emptyset$ **do**
  Find an optimal vertex solution $(\boldsymbol{x}^*, \boldsymbol{z}^*, \boldsymbol{y}^*)$ of $LP_{RRMB}(E_X, \mathcal{I}_X, E_Y, \mathcal{I}_Y, E_Z, L)$
  for all $e \in E_Z$ with $z_e^* = 0$ **do** $E_Z \leftarrow E_Z \setminus \{e\}$
  end for
  for all $e \in E_X$ with $x_e^* = 0$ **do** $M_X \leftarrow M_X \backslash e$
  end for
  for all $e \in E_Y$ with $y_e^* = 0$ **do** $M_Y \leftarrow M_Y \backslash e$
  end for
  if there exists element $e \in E_X$ with $x_e^* = 1$ **then**
    $X \leftarrow X \cup \{e\}$
    $M_X \leftarrow M_X / e$
  end if
  if there exists edge $e' \in E_X$ with $y_{e'}^* = 1$ **then**
    $Y \leftarrow Y \cup \{e\}$
    $M_Y \leftarrow M_Y / e$
  end if
  if there exists edge $e' \in E_Z$ such that $e' \in X \cap Y$ **then**
    $Z \leftarrow Z \cup \{e\}$
    $L \leftarrow L - 1$
    $E_Z \leftarrow E_Z \setminus \{e\}$
  end if
  return X, Y, Z
end while

The next lemma, which characterizes a vertex solution, is analogous to Lemma 2.2. Its proof is based on Lemma 2.12 and is similar in spirit to the one of Lemma 2.2.

Lemma 2.13. *Let $(\boldsymbol{x}, \boldsymbol{z}, \boldsymbol{y})$ be a vertex solution of $LP_{RRMB}(E_X, \mathcal{I}_X, E_Y, \mathcal{I}_Y, E_Z, L)$ such that $x_e > 0$, $e \in E_X$, $y_e > 0$, $e \in E_Y$ and $z_e > 0$, $e \in E_Z$. Then there exist chain families $\mathcal{L}(\boldsymbol{x}) \neq \emptyset$ and $\mathcal{L}(\boldsymbol{y}) \neq \emptyset$ and subsets $E(\boldsymbol{x}, \boldsymbol{z}) \subseteq \mathcal{E}(\boldsymbol{x}, \boldsymbol{z})$ and $E(\boldsymbol{z}, \boldsymbol{y}) \subseteq \mathcal{E}(\boldsymbol{z}, \boldsymbol{y})$ that must satisfy the following:*

(i) $|E_X| + |E_Z| + |E_Y| = |\mathcal{L}(\boldsymbol{x})| + |E(\boldsymbol{x}, \boldsymbol{z})| + |E(\boldsymbol{z}, \boldsymbol{y})| + |\mathcal{L}(\boldsymbol{y})| + 1$,

(ii) the vectors in $\{\chi_X(E_X(U)) : U \in \mathcal{L}(\boldsymbol{x})\} \cup \{\chi_Y(E_Y(U)) : U \in \mathcal{L}(\boldsymbol{y})\} \cup \{-\chi_X(\{e\}) + \chi_Z(\{e\}) : e \in E(\boldsymbol{x}, \boldsymbol{z})\} \cup \{\chi_Z(\{e\}) - \chi_Y(\{e\}) : e \in E(\boldsymbol{z}, \boldsymbol{y})\} \cup \{\chi_Z(E_Z)\}$ *are linearly independent.*

Lemma 2.12 and (2.13) now lead to the next two ones and their proofs run as the proofs of Lemma 2.3 and 2.4.

Lemma 2.14. *Let $(\boldsymbol{x}, \boldsymbol{z}, \boldsymbol{y})$ be a vertex solution of $LP_{RRMB}(E_X, \mathcal{I}_X, E_Y, \mathcal{I}_Y, E_Z, L)$ such that $x_e > 0$, $e \in E_X$, $y_e > 0$, $e \in E_Y$ and $z_e > 0$, $e \in E_Z$. Then there is an element $e' \in E_X$ with $x_{e'} = 1$ or an element $e'' \in E_Y$ with $y_{e''} = 1$.*

We now turn to two cases: $E_X = \emptyset$; $E_Y = \emptyset$. Consider $E_X = \emptyset$, the second case is symmetrical. Observe that (Equation (2.58)) and (Equation (2.59)) and $E_Z \subseteq E_Y$ implies constraint (2.28).

Lemma 2.15. *Let $\boldsymbol{y}$ be a vertex solution of linear program: (Equation (2.54)), (Equation (2.60)), (Equation (2.61)), (Equation (2.64)) and (Equation (2.28)) such that $y_e > 0$, $e \in E_Y$. Then there is an element $e' \in E_Y$ with $y_{e'} = 1$. Moreover using $\boldsymbol{y}$ one can construct a vertex solution of $LP_{RRMB}(\emptyset, \emptyset, E_Y, \mathcal{I}_Y, E_Z, L)$ with $y_{e'} = 1$ and the cost of $\boldsymbol{y}$.*

We are thus led to the main result of this section. Its proof follows by the same arguments as for RR ST.

Theorem 2.7. *Algorithm 2.2 solves RR MB in polynomial time.*

2.6 Conclusion and open issues

In this chapter we have studied the recoverable robust spanning tree problem under the traditional interval uncertainty representation $\mathcal{U}^I$ and the budgeted interval uncertainty representations $\mathcal{U}_1^I(\Gamma)$ and $\mathcal{U}_2^I(\Gamma)$. We have shown that the recoverable robust version of the minimum spanning tree problem with interval uncertainty representation is polynomially solvable. The complexity of this problem was not known to date. In order to prove this result we have applied a technique called an iterative relaxation. Since polynomial time algorithm is based on solving linear programs, the next natural step to take is to design a polynomial time combinatorial algorithm for this problem.

In the second part of this chapter we have constructed a polynomial time combinatorial algorithm for the recoverable robust spanning tree problem. We have applied this algorithm for solving the problem under the interval uncertainty representation in polynomial time. Additionally, the algorithm was used to provide several approximation results for recoverable spanning tree problem under the discrete interval budgeted uncertainty representation $\mathcal{U}_1^I(\Gamma)$ and continuous interval budgeted uncertainty representation $\mathcal{U}_2^I(\Gamma)$.

Moreover, the first algorithm was generalized to the recoverable robust matroid basis problem with interval element costs. No polynomial time combinatorial algorithm for this problem was designed in this thesis. Lend *et al.* in [52] took on this task and continued our investigation of the recoverable robust matroid basis problem under the interval uncertainty representation and has proven the existence of a strongly polynomial time primal-dual algorithm for this problem.

There are still several open questions regarding discussed problems. As an example, the complexity of the robust spanning tree problem under the continuous interval budgeted uncertainty representation $\mathcal{U}_2^I(\Gamma)$ is an interesting open problem. It is possible that some of the algorithms designed in this chapter may be extended to solve this problem in polynomial time.

Chapter 3

Robust recoverable 0-1 optimization problems under polyhedral uncertainty

3.1 Introduction

The goal of this chapter is to provide a framework for solving robust recoverable 0-1 programming problems with a specified polyhedral uncertainty $\mathcal{U}_3^I(\Gamma)$ defined in Section 1.2.1, more specifically (1.7). In general, such problems can be very complex from a computational point of view. The underlying deterministic single-stage problem can be already NP-hard and hard to approximate. Adding recoverable robustness leads to a min-max-min 0-1 programming problem, which can be very difficult to solve. For a finite uncertainty set, a mixed integer programming (denoted by MIP) formulation can be built and solved by row and column generation techniques proposed, for example, in [72]. An uncertainty set is said to be finite if it contains a finite number of scenarios or if it can be replaced with such finite representation, e.g., replacing by the set of extreme points of a polytope for polyhedral uncertainty. However, for the problems examined in this chapter no finite representation of the considered uncertainty set is known. Hence, the solution method consisting in solving a MIP formulation can be hard to apply. In Section 3.3 we propose several lower bounds, which will be based on solving one or a sequence of special MIP formulations. The formulations can be solved for quite large instances by using modern solvers. We will then use these lower bounds to characterize the quality of some approximate solutions in Section 3.4. Finally, in Section 3.5, we will present the results of the experiments for robust recoverable versions of the knapsack and assignment problems. The tested instances have large number of variables, so the proposed approach can be attractive in practical applications.

3.1.1 On usage of a vector notation

A dot product between column vectors $\boldsymbol{a}, \boldsymbol{b} \in \mathbb{R}^n$ is denoted by $\boldsymbol{a} \cdot \boldsymbol{b}$ and is defined in the following way:

$$\boldsymbol{a} \cdot \boldsymbol{b} = \boldsymbol{a}^T \boldsymbol{b} = \sum_{i=1}^{n} a_i b_i.$$

For the purpose of clarity and the reader's convenience from now on we will be using a vector notation to define 0-1 optimization problems. The formulation (1.1) will be rewritten as

$$\begin{aligned} \mathcal{P}: \quad & \min \boldsymbol{C} \cdot \boldsymbol{x} \\ & \boldsymbol{x} \in \Phi \subseteq \{0,1\}^n, \end{aligned} \tag{3.1}$$

where $\boldsymbol{C} = (C_1, \ldots, C_n)^T$ is a vector of nonnegative costs and $\boldsymbol{x} = (x_e)_{e \in E}$, $|E| = n$ is a vector of binary decision variables.

From the above follows the following definitions of the problem discussed in Chapter 1:

$$\begin{aligned} \textsc{Inc}\,\mathcal{P}(\boldsymbol{c}, \boldsymbol{x}): & \min_{\boldsymbol{y} \in \Phi_{\boldsymbol{x}}^{\alpha}} \boldsymbol{c} \cdot \boldsymbol{y}, \\ \textsc{Eval}(\boldsymbol{x}): & \boldsymbol{C} \cdot \boldsymbol{x} + \max_{\boldsymbol{c} \in \mathcal{U}_3^I(\Gamma)} \min_{\boldsymbol{y} \in \Phi_{\boldsymbol{x}}^{\alpha}} \boldsymbol{c} \cdot \boldsymbol{y} = \boldsymbol{C} \cdot \boldsymbol{x} + \max_{\boldsymbol{c} \in \mathcal{U}_3^I(\Gamma)} \textsc{Inc}(\boldsymbol{c}, \boldsymbol{x}), \\ \textsc{Rec}\,\mathcal{P}(\boldsymbol{c}): & \min_{\boldsymbol{x} \in \Phi} \min_{\boldsymbol{y} \in \Phi_{\boldsymbol{x}}^{\alpha}} (\boldsymbol{C} \cdot \boldsymbol{x} + \boldsymbol{c} \cdot \boldsymbol{y}) = \min_{(\boldsymbol{x}, \boldsymbol{y}) \in \mathcal{Z}} (\boldsymbol{C} \cdot \boldsymbol{x} + \boldsymbol{c} \cdot \boldsymbol{y}), \\ \textsc{RR}\,\mathcal{P}: & \min_{\boldsymbol{x} \in \Phi} \max_{\boldsymbol{c} \in \mathcal{U}_3^I(\Gamma)} \min_{\boldsymbol{y} \in \Phi_{\boldsymbol{x}}^{\alpha}} (\boldsymbol{C} \cdot \boldsymbol{x} + \boldsymbol{c} \cdot \boldsymbol{y}), \end{aligned}$$

where $\Phi_{\boldsymbol{x}}^{\alpha}$ is a neighborhood of $\boldsymbol{x}$ (see (1.11)), $(\boldsymbol{x}, \boldsymbol{y})$ ($\boldsymbol{x} \in \Phi$ and $\boldsymbol{y} \in \Phi_{\boldsymbol{x}}^{\alpha}$) is a *feasible pair* of solutions and the set of all such feasible pairs is denoted by $\mathcal{Z}$.

3.2 Solving the problems by MIP formulations

The incremental and recoverable problems for the element exclusion neighborhood (1.11) can be solved by using the following MIP formulations (3.2a) and (3.2b), respectively:

$$\begin{aligned} \min \quad & \boldsymbol{c} \cdot \boldsymbol{y} \\ & \sum_{i \in \{1,\ldots,n\}} x_i (1 - y_i) \leq \alpha \sum_{i \in \{1,\ldots,n\}} x_i, \\ & \boldsymbol{y} \in \Phi; \end{aligned} \tag{3.2a}$$

$$
\begin{array}{lll}
\min & \boldsymbol{C}\cdot\boldsymbol{x}+\boldsymbol{c}\cdot\boldsymbol{y} & \\
& \sum_{i\in\{1,\ldots,n\}}(x_i-z_i)\leq\alpha\sum_{i\in\{1,\ldots,n\}}x_i, & \\
& z_i\leq x_i, & i\in\{1,\ldots,n\},\\
& z_i\leq y_i, & i\in\{1,\ldots,n\},\\
& z_i\geq x_i+y_i-1, & i\in\{1,\ldots,n\},\\
& z_i\in\{0,1\}, & i\in\{1,\ldots,n\},\\
& \boldsymbol{x},\boldsymbol{y}\in\Phi. &
\end{array}
\tag{3.2b}
$$

The MIP formulations can be simplified if $\mathcal{P}$ is an equal cardinality problem. The incremental and recoverable problems can be then formulated as follows:

$$
\begin{array}{lll}
\min & \boldsymbol{c}\cdot\boldsymbol{y} & \\
& \sum_{i\in\{1,\ldots,n\}}x_iy_i\geq\ell, & \\
& \boldsymbol{y}\in\Phi; &
\end{array}
\tag{3.3a}
$$

$$
\begin{array}{lll}
\min & \boldsymbol{C}\cdot\boldsymbol{x}+\boldsymbol{c}\cdot\boldsymbol{y} & \\
& \sum_{i\in\{1,\ldots,n\}}z_i\geq\ell, & \\
& z_i\leq x_i, & i\in\{1,\ldots,n\},\\
& z_i\leq y_i, & i\in\{1,\ldots,n\},\\
& z_i\geq 0, & i\in\{1,\ldots,n\},\\
& \boldsymbol{x},\boldsymbol{y}\in\Phi, &
\end{array}
\tag{3.3b}
$$

where $\ell=\lceil m(1-\alpha)\rceil$. Observe that we can drop the assumption that z_i is binary in (3.3b). The algorithms described in the next part of this chapter will be based on the assumption that the formulations (3.2) and (3.3) can be solved exactly in reasonable time. To this purpose one can use a good off-the-shelf MIP solver (e.g., Gurobi, CPLEX).

Consider now the evaluation problem, i.e., the problem of computing the value of $\textsc{Eval}(\boldsymbol{x})$. Given $\boldsymbol{x}$, the inner adversarial problem $\max_{S\in\mathcal{U}_3^I(\Gamma)}\textsc{Inc}(\boldsymbol{x},\boldsymbol{c}^S)$ can be represented as the following linear programming problem.

$$
\begin{array}{lll}
\max & t & \\
& t\leq\boldsymbol{c}^S\cdot\boldsymbol{y}, & \forall\boldsymbol{y}\in\Phi^\alpha_{\boldsymbol{x}},\\
& S\in\mathcal{U}_3^I(\Gamma). &
\end{array}
\tag{3.4}
$$

If t_{opt} is the optimal value of t, then $\textsc{Eval}(\boldsymbol{x})=\boldsymbol{C}\cdot\boldsymbol{x}+t_{opt}$. Notice that (3.4) is a linear programming problem, since $S\in\mathcal{U}_3^I(\Gamma)$ can be described by a system of linear constraints with real variables and the set $\Phi^\alpha_{\boldsymbol{x}}$ is finite. However, formulation (3.4) has an exponential number of constraints. If we replace $\Phi^\alpha_{\boldsymbol{x}}$ with a subset $\mathcal{Y}\subseteq\Phi^\alpha_{\boldsymbol{x}}$ of feasible solutions, then we get an upper bound on t_{opt}. Also the value of $\textsc{Inc}(\boldsymbol{x},\boldsymbol{c}^S)$, for any $S\in\mathcal{U}_3^I(\Gamma)$, is a lower bound on t_{opt}. Thus in order to find the value of t_{opt} with a given accuracy $\epsilon\geq 0$, we can use a relaxation (constraint generation) algorithm shown in the form of Algorithm 3.1.

Algorithm 3.1 solves a sequence of the incremental problems and relaxed problems (3.4). It is easily seen that the algorithm converges. This fact follows from the observation that

Algorithm 3.1 Compute EVAL($\boldsymbol{x}$) with accuracy ϵ, $\epsilon > 0$.

$UB \leftarrow \infty$
2: Choose an initial scenario $S_0 \in \mathcal{U}_3^I(\Gamma)$
Solve INC($\boldsymbol{x}, \boldsymbol{c}^{S_0}$) obtaining $\boldsymbol{y}^* \in \Phi_{\boldsymbol{x}}^{\alpha}$, $LB \leftarrow$ INC($\boldsymbol{x}, \boldsymbol{c}^{S_0}$)
4: $\mathcal{Y} \leftarrow \{\boldsymbol{y}^*\}$
while $\frac{UB-LB}{LB} > \epsilon$ **do**
6: Solve the formulation (3.4) with $\Phi_{\boldsymbol{x}}^{\alpha} = \mathcal{Y}$ obtaining $(\boldsymbol{c}^*, t^*)$, $UB \leftarrow t^*$
Solve INC($\boldsymbol{x}, \boldsymbol{c}^*$) obtaining $\boldsymbol{y}^* \in \Phi_{\boldsymbol{x}}^{\alpha}$
8: **if** $LB <$ INC($\boldsymbol{x}, \boldsymbol{c}^*$) **then** $LB \leftarrow$ INC($\boldsymbol{x}, \boldsymbol{c}^*$)
$\mathcal{Y} \leftarrow \mathcal{Y} \cup \{\boldsymbol{y}^*\}$
10: **end while**
return $\boldsymbol{C} \cdot \boldsymbol{x} + UB$.

the size of $\mathcal{Y}$ increases by one at each step 9 of the algorithm. Indeed, suppose that $\boldsymbol{y}^*$ is already present in $\mathcal{Y}$, so in formulation (3.4) solved in step 6. Then $LB = \text{INC}(\boldsymbol{x}, \boldsymbol{c}^*) = \boldsymbol{c}^* \cdot \boldsymbol{y}^* \geq \boldsymbol{c}^S \cdot \boldsymbol{y}^*$ for each $S \in \mathcal{U}_3^I(\Gamma)$. In consequence $LB \geq t^* = UB$ and the algorithm terminates.

Notice that each relaxed problem (3.4) is a linear programming problem, which can be solved efficiently. Hence, the running time of the algorithm relies on the complexity of solving the incremental problem. For larger instances the algorithm may converge slowly. However, we can terminate it after a specified time is exceeded. In this case we get an upper bound on EVAL($\boldsymbol{x}$). We can also improve the performance of the algorithm by choosing good initial cost scenario $\boldsymbol{c}^{S_0}$ in step 2. Such scenario will be proposed in Section 3.3.1.

Finally, focus on the most complex RR $\mathcal{P}$ problem, which can be represented as the following program:

$$\begin{array}{lll} \min & \boldsymbol{C} \cdot \boldsymbol{x} + \theta & \\ & \theta \geq \boldsymbol{c}^S \cdot \boldsymbol{y}_S, & S \in \mathcal{U}_3^I(\Gamma), \\ & \sum\limits_{i \in \{1,\ldots,n\}} x_i(1 - y_i^S) \leq \alpha \sum\limits_{i \in \{1,\ldots,n\}} x_i, & S \in \mathcal{U}_3^I(\Gamma), \\ & \boldsymbol{x}, \boldsymbol{y}_S \in \Phi, & S \in \mathcal{U}_3^I(\Gamma). \end{array} \tag{3.5}$$

Assume for a while that $\mathcal{U}_3^I(\Gamma)$ is explicitly given, for instance $\mathcal{U}_3^I(\Gamma) = \mathcal{U}^D$ or the uncertainty set can be replaced with its finite representation, for example by the extreme points of $\mathcal{U}_3^I(\Gamma)$ (see, e.g., [17, 72]), then (3.5) becomes a MIP formulation for RR $\mathcal{P}$. This formulation can have exponential numbers of variables and constraints and solving it requires special row and column generation techniques [72, 17]. For the problem discussed in this chapter the situation seems to be more complex, because it is difficult to replace $\mathcal{U}_3^I(\Gamma)$ with its finite equivalent representation. In particular, we cannot replace $\mathcal{U}_3^I(\Gamma)$ with the set of its extreme points. We will demonstrate this fact using the following simple example. Let $\Phi = \{(x_1, x_2) \in \{0,1\}^2 : x_1 + x_2 = 1\}$ and $\mathcal{U}_3^I(\Gamma) = \{(0 + \delta_1, 0 + \delta_2) : \delta_1, \delta_2 \in [0,1], \delta_1 + \delta_2 \leq 1\}$. When $\boldsymbol{C} = \boldsymbol{0}$ and $\alpha = 1$, the RR $\mathcal{P}$ reduces to the following adversarial problem:

$$\max_{(S_1,S_2) \in \mathcal{U}_3^I(\Gamma)} \min_{(x_1,x_2) \in \Phi} c^{S_1} x_1 + c^{S_2} x_2.$$

This problem has a unique solution $c^{S_1} = 0.5$, $c^{S_2} = 0.5$ with the objective value equal to 0.5. The set of extreme points of $\mathcal{U}_3^I(\Gamma)$ is $\{(0,0),(0,1),(1,0)\}$ and for each of these points the objective value is 0. In this chapter we do not consider a MIP formulation for RR $\mathcal{P}$. Instead, we will use the formulations (3.2), (3.3), (3.4) for the incremental, recoverable and evaluation problems to construct approximate solutions for RR $\mathcal{P}$.

3.3 Lower bounds

In this section we will propose several methods of computing a lower bound for the RR $\mathcal{P}$ problem. We will then use these lower bounds to evaluate the quality of the approximate solutions. We will denote by *opt* the optimal objective value in RR $\mathcal{P}$.

3.3.1 Adversarial lower bound

It is easy to check that for each cost scenario $S \in \mathcal{U}_3^I(\Gamma)$, the value of $\textsc{Rec}(\boldsymbol{c}^S)$ is a lower bound on *opt*. In consequence, the following adversarial problem can provide us the first general lower bound:

$$\textsc{Adv}: \max_{S\in\mathcal{U}_3^I(\Gamma)} \textsc{Rec}(\boldsymbol{c}^S) = \max_{S\in\mathcal{U}_3^I(\Gamma)} \min_{(\boldsymbol{x},\boldsymbol{y})\in\mathcal{Z}} (\boldsymbol{C}\cdot\boldsymbol{x} + \boldsymbol{c}^S\cdot\boldsymbol{y}).$$

Let us rewrite this problem as follows:

$$\begin{array}{ll} \max & t \\ & t \le \boldsymbol{C}\cdot\boldsymbol{x} + \boldsymbol{c}^S\cdot\boldsymbol{y}, \quad \forall(\boldsymbol{x},\boldsymbol{y})\in\mathcal{Z}, \\ & S\in\mathcal{U}_3^I(\Gamma). \end{array} \tag{3.6}$$

In order to solve (3.6) we will use a similar technique as for the model (3.4). The corresponding algorithm is shown in the form of Algorithm 3.2.

Algorithm 3.2 Compute ADV with accuracy ϵ, $\epsilon > 0$.

$UB \leftarrow \infty$
Choose an initial scenario $S_0 \in \mathcal{U}_3^I(\Gamma)$
Solve $\textsc{Rec}(\boldsymbol{c}^{S_0})$ obtaining $(\boldsymbol{x}^*,\boldsymbol{y}^*)\in\mathcal{Z}$ and $LB \leftarrow \textsc{Rec}(\boldsymbol{c}^{S_0})$
$\mathcal{Z}' \leftarrow \{(\boldsymbol{x}^*,\boldsymbol{y}^*)\}$
while $\frac{UB-LB}{LB} > \epsilon$ **do**
    Solve the formulation (3.6) with $\mathcal{Z} \leftarrow \mathcal{Z}'$ obtaining $(\boldsymbol{c}^*, t^*)$ and $UB \leftarrow t^*$.
    Solve $\textsc{Rec}(\boldsymbol{c}^*)$ obtaining $(\boldsymbol{x}^*,\boldsymbol{y}^*)\in\mathcal{Z}$
    if $LB < \textsc{Rec}(\boldsymbol{c}^*)$ **then** $LB \leftarrow \textsc{Rec}(\boldsymbol{c}^*)$
    $\mathcal{Z}' \leftarrow \mathcal{Z}' \cup \{(\boldsymbol{x}^*,\boldsymbol{y}^*)\}$.
end while
return LB.

To reiterate, (3.6) is a linear programming problem, which can be solved efficiently for a small subset $\mathcal{Z}' \subseteq \mathcal{Z}$. In order to prove that Algorithm 3.2 converges, one can use

the same argument as in Section 3.2. The running time of Algorithm 3.2 depends on the complexity of solving the recoverable problem, which is not easy in general. However, we can fix a limit for its running time, after which we still get a lower bound on *opt*.

For larger problems the relaxation algorithm may converge slowly. In order to speed up the computations we can start with a good heuristic initial scenario $S_0 = (c_i)_{i\in\{1,\dots,n\}} \in \mathcal{U}_3^I(\Gamma)$ (see (1.7)), computed as follows:

$$\begin{array}{lll} \max & v & \\ & c_i + \delta_i \geq \min\{c_i + d_i, v\}, & \\ & \sum_{i\in\{1,\dots,n\}} \delta_i \leq \Gamma, & \\ & 0 \leq \delta_i \leq d_i, & i \in \{1,\dots,n\}, \\ & \boldsymbol{\delta} \in \mathcal{V}. & \end{array} \tag{3.7}$$

Recall that c_i, $i \in \{1,\dots,n\}$, are the fixed nominal second stage costs (see 1.2.1). The idea of (3.7) is to uniformly distribute the budget among the smallest costs. This problem can be solved efficiently by using a binary search on $[0, V]$, where $V = \max_{i\in\{1,\dots,n\}}\{c_i + d_i\}$. If $\boldsymbol{\delta}^*$ is an optimal solution to (3.7), then $\boldsymbol{c}^{S_0} = \boldsymbol{c} + \boldsymbol{\delta}^*$. An illustration for the uncertainty set $\mathcal{U}_2^I(\Gamma)$ is shown in Figure 3.1. In this case, given $v \geq 0$, we fix $\delta_i = \max\{0, \min\{d_i, v - c_i\}\}$. We choose the maximum value of v for which the constraint $\sum_{i\in\{1,\dots,n\}} \delta_i \leq \Gamma$ is satisfied.

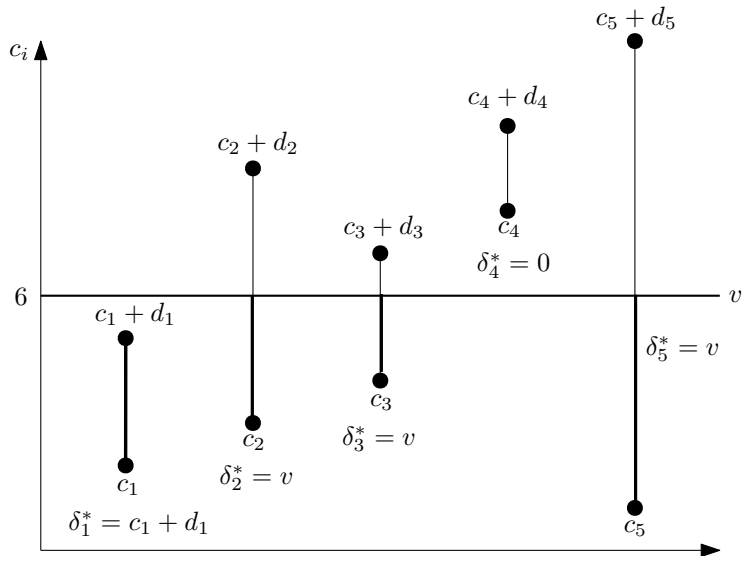


Figure 3.1: Computing the initial scenario $\boldsymbol{c}^{S_0}$ for a sample uncertainty set $\mathcal{U}_2^I(\Gamma)$ with $\Gamma = 13$.

For large problems we can use $\textsc{Rec}(\boldsymbol{c}^{S_0})$ as a starting lower bound on *opt*. We can then try to improve the lower bound by running Algorithm 3.2 for a given time limit.

The tightness of the adversarial lower bound is likely to depend on α. Observe that when $\alpha = 1$, then $\Phi_{\boldsymbol{x}}^{\alpha} = \Phi$ and RR $\mathcal{P}$ is equivalent to ADV. Also, for α close to 1, the adversarial lower bound should be closer to *opt*.

3.3.2 Cardinality selection constraint lower bound

In this section we will propose another lower bound which, contrary to the adversarial lower bound, can be computed by solving one MIP formulation. This lower bound should behave better than the adversarial lower bound for smaller values of α. In order to simplify the presentation we will use the uncertainty set $\mathcal{U}_2^I(\Gamma)$. A generalization to any set $\mathcal{U}_3^I(\Gamma)$ will be straightforward. The idea will be to relax the incremental problem by relaxing the structure of the neighborhood. We consider first the case when $\mathcal{P}$ is an equal cardinality problem. Let us replace the constraint $\boldsymbol{y} \in \Phi$ in (3.3a) with a weaker *cardinality (selection) constraint*, namely $y_1 + \cdots + y_n = m$, $m \in \{1, \ldots, n\}$. So, the second stage solution need not to be feasible. Only, the cardinality constraint must be satisfied. As the result, we get the following relaxation of the incremental problem:

$$\begin{aligned} \textsc{Inc}(\boldsymbol{x}, \boldsymbol{c}) \geq \textsc{Inc}'(\boldsymbol{x}, \boldsymbol{c}) = \min \quad & \boldsymbol{c} \cdot \boldsymbol{y} \\ & \sum_{i \in \{1,\ldots,n\}} x_i y_i \geq \ell, \\ & \sum_{i \in \{1,\ldots,n\}} y_i = m, \\ & y_i \in \{0,1\}, \qquad i \in \{1, \ldots, n\}. \end{aligned} \tag{3.8}$$

Since ℓ is integer and $\boldsymbol{x} \in \{0,1\}^n$ is fixed, we get the following equivalent problem:

$$\begin{aligned} \textsc{Inc}'(\boldsymbol{x}, \boldsymbol{c}) = \min \quad & \boldsymbol{c} \cdot \boldsymbol{y} \\ & \sum_{i \in \{1,\ldots,n\}} x_i y_i \geq \ell, \\ & \sum_{i \in \{1,\ldots,n\}} y_i = m, \\ & 0 \leq y_i \leq 1, \qquad i \in \{1, \ldots, n\}. \end{aligned} \tag{3.9}$$

The problems (3.8) and (3.9) have the same optimal objective values. In order to see this, let $V = \{i \in \{1, \ldots, n\} : x_i = 1\}$. Then an integral optimal solution to (3.9) can be formed by fixing first $y_i = 1$ for $t = \min\{\ell, |V|\}$ variables y_i, $i \in V$ with the smallest c_i and fixing then $y_i = 1$ for $m - t$ out of the remaining variables with the smallest c_i. Notice also that, given $\boldsymbol{x}$, the constraint matrix of (3.9) is totally unimodular (see Definition 1.9). Taking the dual to (3.9) we get

$$\begin{aligned} \textsc{Inc}'(\boldsymbol{x}, \boldsymbol{c}) = \max \quad & \sigma\ell + \rho m - \sum_{i \in \{1,\ldots,n\}} \alpha_i \\ & \sigma x_i + \rho - \alpha_i \leq c_i, \qquad i \in \{1, \ldots, n\}, \\ & \sigma \geq 0. \end{aligned}$$

Now the relaxed evaluation problem

$$\textsc{Eval}'(\boldsymbol{x}) = \max_{\boldsymbol{c} \in \mathcal{U}_2^I(\Gamma)} \textsc{Inc}'(\boldsymbol{x}, \boldsymbol{c})$$

can be formulated as follows

$$\begin{aligned}
\textsc{Eval}(\boldsymbol{x}) \geq \textsc{Eval}'(\boldsymbol{x}) = \quad & \boldsymbol{C}\cdot\boldsymbol{x} + \max \sigma\ell + \rho m - \sum_{i\in\{1,\ldots,n\}} \alpha_i & \\
& \sigma x_i + \rho - \alpha_i \leq c_i + \delta_i, & i \in \{1,\ldots,n\},\\
& \sum_{i\in\{1,\ldots,n\}} \delta_i \leq \Gamma, & \\
& \delta_i \leq d_i, & i \in \{1,\ldots,n\},\\
& \delta_i \geq 0, & i \in \{1,\ldots,n\},\\
& \sigma \geq 0. &
\end{aligned}$$

Taking the dual to the inner maximization problem we get:

$$\begin{aligned}
\textsc{Eval}'(\boldsymbol{x}) = \quad & \boldsymbol{C}\cdot\boldsymbol{x} + \min \pi\Gamma + \sum_{i\in\{1,\ldots,n\}} c_i y_i + \sum_{i\in\{1,\ldots,n\}} u_i d_i & \\
& \sum_{i\in\{1,\ldots,n\}} x_i y_i \geq \ell, & \\
& \sum_{i\in\{1,\ldots,n\}} y_i = m, & \\
& y_i \leq 1, & i \in \{1,\ldots,n\},\\
& -y_i + \pi + u_i \geq 0, & i \in \{1,\ldots,n\},\\
& \pi \geq 0, & \\
& u_i, y_i \geq 0, & i \in \{1,\ldots,n\}.
\end{aligned}$$

Finally, we obtain

$$\begin{aligned}
\min_{\boldsymbol{x}\in\Phi} \textsc{Eval}'(\boldsymbol{x}) = \quad & \min \boldsymbol{C}\cdot\boldsymbol{x} + \pi\Gamma + \sum_{i\in\{1,\ldots,n\}} c_i y_i + \sum_{i\in\{1,\ldots,n\}} u_i d_i & \\
& \sum_{i\in\{1,\ldots,n\}} x_i y_i \geq \ell, & \\
& \sum_{i\in\{1,\ldots,n\}} y_i = m, & \\
& y_i \leq 1, & i \in \{1,\ldots,n\},\\
& -y_i + \pi + u_i \geq, 0 & i \in \{1,\ldots,n\},\\
& \pi \geq 0, & \\
& \boldsymbol{x} \in \Phi, & \\
& u_i, y_i \geq 0, & i \in \{1,\ldots,n\}.
\end{aligned} \tag{3.10}$$

Since

$$opt = \min_{\boldsymbol{x}\in\Phi} \textsc{Eval}(\boldsymbol{x}) \geq \min_{\boldsymbol{x}\in\Phi} \textsc{Eval}'(\boldsymbol{x}),$$

the MIP formulation (3.10) gives us a lower bound on *opt*. Notice that the constraint $\sum_{i\in\{1,\ldots,n\}} x_i y_i \geq \ell$ can be linearized by using standard techniques (here it is enough to introduce $z_i \geq 0$, substitute $z_i = x_i y_i$ and add constraints $z_i \leq x_i$, $z_i \leq y_i$ for each $i \in \{1,\ldots,n\}$).

Let us now turn to the element exclusion neighborhood (1.13). Let us remove the constraint $\boldsymbol{y} \in \Phi$ in (3.2a) and rewrite this problem as follows:

$$\begin{aligned} \textsc{Inc}(\boldsymbol{x}, \boldsymbol{c}) \geq \textsc{Inc}'(\boldsymbol{x}, \boldsymbol{c}) = \min \quad & \boldsymbol{c} \cdot \boldsymbol{y} \\ & \sum_{i \in \{1,\dots,n\}} x_i(1 - y_i) \leq \alpha \sum_{i \in \{1,\dots,n\}} x_i, \\ & y_i \in \{0, 1\}, \qquad i \in \{1, \dots, n\} \end{aligned} \tag{3.11}$$

which can be rewritten as

$$\begin{aligned} \textsc{Inc}'(\boldsymbol{x}, \boldsymbol{c}) = \min \quad & \boldsymbol{c} \cdot \boldsymbol{y} \\ & \sum_{i \in \{1,\dots,n\}} x_i y_i \geq (1 - \alpha) \sum_{i \in \{1,\dots,n\}} x_i, \\ & y_i \in \{0, 1\}, \qquad i \in \{1, \dots, n\}. \end{aligned} \tag{3.12}$$

We cannot add the cardinality constraint, since the size of $\boldsymbol{y}$ is unknown. Also, the right hand side of the constraint need not to be integral. However, we can still obtain a lower bound on the incremental problem by solving the following relaxation of (3.11):

$$\begin{aligned} \textsc{Inc}(\boldsymbol{x}, \boldsymbol{c}) \geq \textsc{Inc}''(\boldsymbol{x}, \boldsymbol{c}) = \min \quad & \boldsymbol{c} \cdot \boldsymbol{y} \\ & \sum_{i \in \{1,\dots,n\}} x_i y_i \geq (1 - \alpha) \sum_{i \in \{1,\dots,n\}} x_i, \\ & y_i \leq 1, \qquad i \in \{1, \dots, n\}, \\ & y_i \geq 0, \qquad i \in \{1, \dots, n\}. \end{aligned}$$

Using similar reasoning as for the equal cardinality problem, we get the following MIP formulation:

$$\begin{aligned} \min_{\boldsymbol{x} \in \Phi} \textsc{Eval}''(\boldsymbol{x}) = \min \quad & \boldsymbol{C} \cdot \boldsymbol{x} + \pi\Gamma + \sum_{i \in \{1,\dots,n\}} c_i y_i + \sum_{i \in \{1,\dots,n\}} u_i d_i \\ & \sum_{i \in \{1,\dots,n\}} x_i y_i \geq (1 - \alpha) \sum_{i \in \{1,\dots,n\}} x_i, \\ & y_i \leq 1, \qquad i \in \{1, \dots, n\}, \\ & -y_i + \pi + u_i \geq 0, \qquad i \in \{1, \dots, n\}, \\ & \pi \geq 0, \\ & \boldsymbol{x} \in \Phi, \\ & u_i, y_i \geq 0, \qquad i \in \{1, \dots, n\}. \end{aligned} \tag{3.13}$$

The formulation (3.13) is a lower bound on *opt* for the element exclusion neighborhood (1.13). The terms $x_i y_i$ in (3.13) can be linearized by using standard techniques.

Observe that for the equal cardinality problem $\mathcal{P}$ the cardinality selection constraint lower bound is equal to *opt* when $\alpha = 0$, because $\Phi^0_{\boldsymbol{x}} = \{\boldsymbol{x}\}$. The same property is true for the element exclusion neighborhood under the additional assumption that $\boldsymbol{c} > \boldsymbol{0}$. In this case $\Phi^0_{\boldsymbol{x}}$ contains every solution $\boldsymbol{y}$ such that $\boldsymbol{y}$ is a superset of $\boldsymbol{x}$. Hence the cardinality selection constraint lower bound can be closer to the optimum for α close to 0.

3.3.3 Lagrangian lower bound

In this section we will construct another lower bound, which will be based on the Lagrangian relaxation technique (see, e.g., [1]). Contrary to the adversarial and selection lower bounds, this bound will be limited to a special class of problems. Namely, we will make two assumptions about the underlying problem $\mathcal{P}$. Firstly, we will assume that $\Phi = \{\boldsymbol{x} \in \{0,1\}^n : \boldsymbol{A}\boldsymbol{x} = \boldsymbol{b}\}$, where $\boldsymbol{A}$ is an $m \times n$ matrix, and the corresponding polyhedron $P_\Phi = \{\boldsymbol{x} : \boldsymbol{0} \leq \boldsymbol{x} \leq \boldsymbol{1}, \boldsymbol{A}\boldsymbol{x} = \boldsymbol{b}\}$ is integral (see Definition 1.8). This is true, for example, when $\boldsymbol{A}$ is a totally unimodular matrix (see Definition 1.9). Specifically, if $\boldsymbol{A}$ is totally unimodular and $\boldsymbol{b}$ is integral, then every extreme point of the feasible region P_Φ is integral and thus P_Φ is an integral polyhedron. We will also assume that $\mathcal{P}$ has the equal cardinality property, which will allow us to use the simplified neighborhood representation. An important problem, which satisfies both assumptions, is the minimum assignment problem. Again, we will consider the uncertainty set $\mathcal{U}_2^I(\Gamma)$, as the generalization to any $\mathcal{U}_3^I(\Gamma)$ is straightforward.

Let us introduce a Lagrangian multiplier $\mu \geq 0$ and consider the following Lagrangian relaxation of the incremental problem (3.3a):

$$\begin{aligned} \textsc{Inc}(\boldsymbol{x},\boldsymbol{c}) \geq \textsc{Inc}'(\boldsymbol{x},\boldsymbol{c},\mu) = \min \quad & \boldsymbol{c}\cdot\boldsymbol{y} + \mu(\ell - \sum_{i\in\{1,\ldots,n\}} x_i y_i) \\ & \boldsymbol{A}\boldsymbol{y} = \boldsymbol{b}, \\ & \boldsymbol{y} \in \{0,1\}^n. \end{aligned} \tag{3.14}$$

By the integrality property, (3.14) is equivalent to the following linear programming problem:

$$\begin{aligned} \textsc{Inc}'(\boldsymbol{x},\boldsymbol{c},\mu) = \min \quad & \sum_{i\in\{1,\ldots,n\}} (c_i - \mu x_i) y_i + \mu\ell \\ & \boldsymbol{A}\boldsymbol{y} = \boldsymbol{b}, \\ & y_i \leq 1, && i \in \{1,\ldots,n\}, \\ & y_i \geq 0, && i \in \{1,\ldots,n\}. \end{aligned} \tag{3.15}$$

Dualizing (3.15) yields

$$\begin{aligned} \textsc{Inc}'(\boldsymbol{x},\boldsymbol{c},\mu) = \max \quad & \sum_{i\in\{1,\ldots,m\}} \gamma_i b_i - \sum_{i\in\{1,\ldots,n\}} \beta_i + \mu\ell \\ & \boldsymbol{\gamma}\boldsymbol{A}_i - \beta_i \leq c_i - \mu x_i, && i \in \{1,\ldots,n\}, \\ & \beta_i \geq 0, && i \in \{1,\ldots,n\}, \end{aligned}$$

where $\boldsymbol{A}_i$ is the ith column of matrix $\boldsymbol{A}$ and $\boldsymbol{\gamma} = (\gamma_1,\ldots,\gamma_m)$. Now, given $\boldsymbol{x} \in \Phi$, we get

the following linear programming relaxation of the evaluation problem:

$$\begin{aligned}
\textsc{Eval}(\boldsymbol{x}) \geq \textsc{Eval}'(\boldsymbol{x},\mu) = \boldsymbol{C}\cdot\boldsymbol{x} + \max\quad & \sum_{i\in\{1,\dots,m\}} \gamma_i b_i - \sum_{i\in\{1,\dots,n\}} \beta_i + \mu\ell & \\
& \boldsymbol{\gamma}\boldsymbol{A}_i - \beta_i \leq c_i + \delta_i - \mu x_i, & i\in\{1,\dots,n\},\\
& \sum_{i\in\{1,\dots,n\}} \delta_i \leq \Gamma, & \\
& \delta_i \leq d_i, & i\in\{1,\dots,n\},\\
& \delta_i,\beta_i \geq 0, & i\in\{1,\dots,n\}.
\end{aligned}$$

After dualizing the inner maximization problem, we get the following equivalent formulation

$$\begin{aligned}
\textsc{Eval}'(\boldsymbol{x},\mu) = \boldsymbol{C}\cdot\boldsymbol{x} + \min\quad & \pi\Gamma + \sum_{i\in\{1,\dots,n\}} y_i(c_i-\mu x_i) + \sum_{i\in\{1,\dots,n\}} u_i d_i + & \mu\ell\\
& \boldsymbol{A}\boldsymbol{y} = \boldsymbol{b}, & \\
& -y_i + \pi + u_i \geq 0, & \\
& 0 \leq y_i \leq 1, & i\in\{1,\dots,n\},\\
& u_i \geq 0, & i\in\{1,\dots,n\}.
\end{aligned}$$

Finally, we have

$$\begin{aligned}
\min_{\boldsymbol{x}\in\Phi} \textsc{Eval}'(\boldsymbol{x},\mu) = \min\quad & \boldsymbol{C}\cdot\boldsymbol{x} + \pi\Gamma + \sum_{i\in\{1,\dots,n\}} y_i(c_i-\mu x_i) + & \textstyle\sum_{i\in\{1,\dots,n\}} u_i d_i + \mu\ell\\
& \boldsymbol{A}\boldsymbol{y} = \boldsymbol{b}, & \\
& -y_i + \pi + u_i \geq 0, & \\
& y_i \leq 1, & i\in\{1,\dots,n\},\\
& \boldsymbol{A}\boldsymbol{x} = \boldsymbol{b}, & \\
& u_i, y_i \geq 0, & i\in\{1,\dots,n\},\\
& \boldsymbol{x}\in\{0,1\}^n. &
\end{aligned} \tag{3.16}$$

The formulation (3.16) can be linearized, which results in the following linear MIP model:

$$\begin{aligned}
\min_{\boldsymbol{x}\in\Phi} \textsc{Eval}'(\boldsymbol{x},\mu) = \min\quad & \boldsymbol{C}\cdot\boldsymbol{x} + \pi\Gamma + \sum_{i\in\{1,\dots,n\}} y_i c_i - \mu \sum_{i\in\{1,\dots,n\}} z_i + & \textstyle\sum_{i\in\{1,\dots,n\}} u_i d_i + \mu\ell\\
& \boldsymbol{A}\boldsymbol{y} = \boldsymbol{b}, & \\
& -y_i + \pi + u_i \geq 0, & \\
& y_i \leq 1, & i\in\{1,\dots,n\},\\
& \boldsymbol{A}\boldsymbol{x} = \boldsymbol{b}, & \\
& z_i \leq x_i, & i\in\{1,\dots,n\},\\
& z_i \leq y_i, & i\in\{1,\dots,n\},\\
& u_i, y_i, z_i \geq 0, & i\in\{1,\dots,n\},\\
& \boldsymbol{x}\in\{0,1\}^n. &
\end{aligned} \tag{3.17}$$

Observe that (3.17) has only n binary variables. Let us denote

$$\textsc{Eval}^*(\mu) = \min_{\boldsymbol{x}\in\Phi} \textsc{Eval'}(\boldsymbol{x}, \mu).$$

Hence, for each $\mu \geq 0$, we get a lower bound on *opt*. The best lower bound can be computed by solving the following Lagrangian multipliers problem:

$$\max_{\mu\geq 0} \textsc{Eval}^*(\mu). \tag{3.18}$$

The problem (3.18) can be solved by applying a search method on the single parameter $\mu \geq 0$. One can also solve one problem (3.17) for a heuristically chosen value of μ, also obtaining a lower bound on *opt* (but possibly not the most tight one).

3.4 Upper bounds and approximate solutions

As we make no assumptions on the underlying problem $\mathcal{P}$, no general polynomial time approximation algorithm can exist for any problem discussed in this chapter. In this section we will explore the approximability of the robust recoverable problem, under the assumption that we can solve the incremental and recoverable problems in reasonable time. In general, these problems cannot be solved in polynomial time. However, good modern solvers can solve them to optimality for quite large instances (see Section 3.5). As in Section 3.3, we will use *opt* to denote the optimal objective value for RR $\mathcal{P}$.

Let us fix $\boldsymbol{x} \in \Phi$. By using well-known min-max relations, we get

$$\max_{S\in\mathcal{U}_3^I(\Gamma)} \min_{\boldsymbol{y}\in\Phi_{\boldsymbol{x}}^\alpha} \boldsymbol{c}^S \cdot \boldsymbol{y} \leq \min_{\boldsymbol{y}\in\Phi_{\boldsymbol{x}}^\alpha} \max_{S\in\mathcal{U}_3^I(\Gamma)} \boldsymbol{c}^S \cdot \boldsymbol{y}. \tag{3.19}$$

Using (3.19) we get

$$opt = \min_{\boldsymbol{x}\in\Phi} \max_{S\in\mathcal{U}_3^I(\Gamma)} \min_{\boldsymbol{y}\in\Phi_{\boldsymbol{x}}^\alpha} (\boldsymbol{C}\cdot\boldsymbol{x} + \boldsymbol{c}^S\cdot\boldsymbol{y}) \leq \min_{\boldsymbol{x}\in\Phi} \min_{\boldsymbol{y}\in\Phi_{\boldsymbol{x}}^\alpha} \max_{S\in\mathcal{U}_3^I(\Gamma)} (\boldsymbol{C}\cdot\boldsymbol{x} + \boldsymbol{c}^S\cdot\boldsymbol{y}) =$$

$$= \min_{(\boldsymbol{x},\boldsymbol{y})\in\mathcal{Z}} (\boldsymbol{C}\cdot\boldsymbol{x} + \max_{S\in\mathcal{U}_3^I(\Gamma)} \boldsymbol{c}^S\cdot\boldsymbol{y}).$$

Because $\mathcal{U}_3^I(\Gamma) \subseteq \mathcal{U}_2^I(\Gamma)$, we conclude that

$$UB = \min_{(\boldsymbol{x},\boldsymbol{y})\in\mathcal{Z}} (\boldsymbol{C}\cdot\boldsymbol{x} + \max_{S\in\mathcal{U}_2^I(\Gamma)} \boldsymbol{c}^S\cdot\boldsymbol{y}) \geq \min_{(\boldsymbol{x},\boldsymbol{y})\in\mathcal{Z}} (\boldsymbol{C}\cdot\boldsymbol{x} + \max_{S\in\mathcal{U}_3^I(\Gamma)} \boldsymbol{c}^S\cdot\boldsymbol{y}) \geq opt.$$

Hence UB is an upper bound on *opt*. Given $\boldsymbol{y}$, the inner $\max_{S\in\mathcal{U}_2^I(\Gamma)} \boldsymbol{c}^S\cdot\boldsymbol{y}$ problem can easily be solved by allocating the largest possible part of the budget Γ to the costs of the elements in $\boldsymbol{y}$. Hence, either the whole budget is allocated or the allocation is blocked by the upper bounds on the second stage costs of $\boldsymbol{y}$. We thus get

$$\max_{S\in\mathcal{U}_2^I(\Gamma)} \boldsymbol{c}^S\cdot\boldsymbol{y} = \min\{\boldsymbol{c}\cdot\boldsymbol{y} + \Gamma, (\boldsymbol{c}+\boldsymbol{d})\cdot\boldsymbol{y}\}.$$

In consequence

$$UB = \min_{(\boldsymbol{x},\boldsymbol{y})\in\mathcal{Z}}(\boldsymbol{C}\cdot\boldsymbol{x}+\min\{\boldsymbol{c}\cdot\boldsymbol{y}+\Gamma,(\boldsymbol{c}+\boldsymbol{d})\cdot\boldsymbol{y}\}) = \min_{(\boldsymbol{x},\boldsymbol{y})\in\mathcal{Z}}\min\{\boldsymbol{C}\cdot\boldsymbol{x}+\boldsymbol{c}\cdot\boldsymbol{y}+\Gamma,\boldsymbol{C}\cdot\boldsymbol{x}+(\boldsymbol{c}+\boldsymbol{d})\cdot\boldsymbol{y}\} =$$

$$= \min\{\textsc{Rec}(\boldsymbol{c})+\Gamma,\textsc{Rec}(\boldsymbol{c}+\boldsymbol{d})\}.$$

Hence, in order to compute UB, it is enough to solve two recoverable problems. We now investigate the quality of the solutions $(\underline{\boldsymbol{x}},\underline{\boldsymbol{y}})\in\mathcal{Z}$ and $(\overline{\boldsymbol{x}},\overline{\boldsymbol{y}})\in\mathcal{Z}$ obtained by computing $\textsc{Rec}(\boldsymbol{c})$ and $\textsc{Rec}(\boldsymbol{c}+\boldsymbol{d})$, respectively. We will choose the best of $\underline{\boldsymbol{x}}$ and $\overline{\boldsymbol{x}}$ as an approximate first stage solution to RR $\mathcal{P}$, i.e., we choose $\underline{\boldsymbol{x}}$ if $\textsc{Eval}(\underline{\boldsymbol{x}})\leq\textsc{Eval}(\overline{\boldsymbol{x}})$ and $\overline{\boldsymbol{x}}$, otherwise. Observe that

$$\textsc{Eval}(\underline{\boldsymbol{x}}) \leq \boldsymbol{C}\cdot\underline{\boldsymbol{x}} + \max_{S\in\mathcal{U}_3^I(\Gamma)}\boldsymbol{c}^S\cdot\underline{\boldsymbol{y}} \leq \boldsymbol{C}\cdot\underline{\boldsymbol{x}}+\boldsymbol{c}\cdot\underline{\boldsymbol{y}}+\Gamma = \textsc{Rec}(\boldsymbol{c})+\Gamma$$

and

$$\textsc{Eval}(\overline{\boldsymbol{x}}) \leq \boldsymbol{C}\cdot\overline{\boldsymbol{x}} + \max_{S\in\mathcal{U}_3^I(\Gamma)}\boldsymbol{c}^S\cdot\overline{\boldsymbol{y}} \leq \boldsymbol{C}\cdot\overline{\boldsymbol{x}}+(\boldsymbol{c}+\boldsymbol{d})\cdot\overline{\boldsymbol{y}} = \textsc{Rec}(\boldsymbol{c}+\boldsymbol{d}).$$

Hence

$$\min\{\textsc{Eval}(\underline{\boldsymbol{x}}),\textsc{Eval}(\overline{\boldsymbol{x}})\}\leq UB. \tag{3.20}$$

Since $\textsc{Rec}(\boldsymbol{c})$ is a lower bound on *opt* for each $S\in\mathcal{U}_3^I(\Gamma)$, we get

$$\frac{UB}{opt}\leq\frac{\min\{\textsc{Rec}(\boldsymbol{c}^S)+\Gamma,\textsc{Rec}(\boldsymbol{c}^S+\boldsymbol{d})\}}{\textsc{Rec}(\boldsymbol{c}^S)}=\rho(\boldsymbol{c}^S),\quad\forall S\in\mathcal{U}_3^I(\Gamma). \tag{3.21}$$

Notice that we get the smallest ratio $\rho^*=\min_{S\in\mathcal{U}_3^I(\Gamma)}\rho(\boldsymbol{c}^S)$ by choosing $S\in\mathcal{U}_3^I(\Gamma)$ maximizing $\textsc{Rec}(\boldsymbol{c})$, i.e., by solving the adversarial problem discussed in Section 3.3.1. Using (3.20) we get

$$\min\{\textsc{Eval}(\underline{\boldsymbol{x}}),\textsc{Eval}(\underline{\boldsymbol{x}})\}\leq\rho^*\cdot opt,$$

so the best of solutions $\underline{\boldsymbol{x}}$ and $\overline{\boldsymbol{x}}$ has an approximation ratio of ρ^*.

The value of ρ^* depends on the problem data. Furthermore, its precise evaluation requires solving recoverable and adversarial problems, which can be time-consuming. We now show several estimations of the ratio ρ^* from above, which can be computed more efficiently. We will use the following lemma:

Lemma 3.1. *Let $(\boldsymbol{x}^*,\boldsymbol{y}^*)\in\mathcal{Z}$ be an optimal solution to the recoverable problem for a fixed scenario $S_0\in\mathcal{U}_3^I(\Gamma)$. We then have*

$$\rho^*\leq\min\left\{\frac{\boldsymbol{C}\cdot\boldsymbol{x}^*+\boldsymbol{c}\cdot\boldsymbol{y}^*+\Gamma}{\boldsymbol{C}\cdot\boldsymbol{x}^*+\boldsymbol{c}^{S_0}\cdot\boldsymbol{y}^*},\frac{\boldsymbol{C}\cdot\boldsymbol{x}^*+(\boldsymbol{c}+\boldsymbol{d})\cdot\boldsymbol{y}^*}{\boldsymbol{C}\cdot\boldsymbol{x}^*+\boldsymbol{c}^{S_0}\cdot\boldsymbol{y}^*}\right\}. \tag{3.22}$$

Proof. Using (3.21), we get

$$\rho^*\leq\rho(\boldsymbol{c}^{S_0})=\min\left\{\frac{\textsc{Rec}(\boldsymbol{c})+\Gamma}{\textsc{Rec}(\boldsymbol{c}^{S_0})},\frac{\textsc{Rec}(\boldsymbol{c}+\boldsymbol{d})}{\textsc{Rec}(\boldsymbol{c}^{S_0})}\right\}.$$

Now (3.22) follows from the inequalities $\textsc{Rec}(\boldsymbol{c})\leq\boldsymbol{C}\cdot\boldsymbol{x}^*+\boldsymbol{c}\cdot\boldsymbol{y}^*$, $\textsc{Rec}(\boldsymbol{c}+\boldsymbol{d})\leq\boldsymbol{C}\cdot\boldsymbol{x}^*+(\boldsymbol{c}+\boldsymbol{d})\cdot\boldsymbol{y}^*$ and equality $\textsc{Rec}(\boldsymbol{c}^{S_0})=\boldsymbol{C}\cdot\boldsymbol{x}^*+\boldsymbol{c}^{S_0}\cdot\boldsymbol{y}^*$. □

Lemma 3.2. *If* $\frac{c_i+d_i}{c_i} \leq \sigma$, *for each* $i \in \{1,\ldots,n\}$, *then* $\rho^* \leq \sigma$

Proof. By setting $\boldsymbol{c}^{S_0} = \boldsymbol{c}^S$, $S \in \mathcal{U}_3^I(\Gamma)$ in (3.22), we get $(\boldsymbol{x}^*, \boldsymbol{y}^*) = (\underline{\boldsymbol{x}}, \underline{\boldsymbol{y}})$ and

$$\rho^* \leq \frac{\boldsymbol{C}\cdot\underline{\boldsymbol{x}} + (\boldsymbol{c}+\boldsymbol{d})\cdot\underline{\boldsymbol{y}}}{\boldsymbol{C}\cdot\underline{\boldsymbol{x}} + \boldsymbol{c}\cdot\underline{\boldsymbol{y}}} \leq \frac{\boldsymbol{C}\cdot\underline{\boldsymbol{x}} + (\boldsymbol{c}+\boldsymbol{d})\cdot\underline{\boldsymbol{y}}}{\boldsymbol{C}\cdot\underline{\boldsymbol{x}} + (1/\sigma)(\boldsymbol{c}+\boldsymbol{d})\cdot\underline{\boldsymbol{y}}} \leq \frac{\boldsymbol{C}\cdot\underline{\boldsymbol{x}} + (\boldsymbol{c}+\boldsymbol{d})\cdot\underline{\boldsymbol{y}}}{(1/\sigma)(\boldsymbol{C}\cdot\underline{\boldsymbol{x}} + (\boldsymbol{c}+\boldsymbol{d})\cdot\underline{\boldsymbol{y}})} = \sigma,$$

where the second inequality follows from $(\boldsymbol{c}+\boldsymbol{d})\cdot\underline{\boldsymbol{y}} \leq \sigma\boldsymbol{c}\cdot\underline{\boldsymbol{y}}$ and the third inequality follows from the fact that $\frac{1}{\sigma} \leq 1$. □

The value of σ in Lemma 3.2 can be interpreted as the maximal factor by which the second stage costs can increase. For example, when $\sigma = 2$, then the second stage costs can increase by at most 100% from their nominal values, and in this case $\rho^* \leq 2$. It is reasonable to assume that in many practical applications σ is not large, which results in a good approximation ratio.

Lemma 3.3. *If*

$$\Gamma \leq \beta \cdot \left(\min_{\boldsymbol{x}\in\Phi} \boldsymbol{C}\cdot\boldsymbol{x} + \max_{S\in\mathcal{U}_3^I(\Gamma)} \min_{\boldsymbol{y}\in\Phi} \boldsymbol{c}^S\cdot\boldsymbol{y} \right)$$

for $\beta \geq 0$ *then* $\rho^* \leq 1+\beta$.

Proof. Let $\boldsymbol{c}^{S_0} = \boldsymbol{c}^{S^*}$, $S^* \in \mathcal{U}_3^I(\Gamma)$ be a costs of a scenario maximizing $\textsc{Rec}(\boldsymbol{c})$ and let $(\boldsymbol{x}^*, \boldsymbol{y}^*) \in \mathcal{Z}$ be an optimal solution to the recoverable problem under $\boldsymbol{c}^{S^*}$. Using the first term in the minimum in (3.22) we get

$$\rho^* \leq 1 + \frac{\Gamma}{\boldsymbol{C}\cdot\boldsymbol{x}^* + \boldsymbol{c}^{S^*}\boldsymbol{y}^*}.$$

Since

$$\begin{aligned}\textsc{Rec}(\boldsymbol{c}^{S^*}) = \boldsymbol{C}\cdot\boldsymbol{x}^* + \boldsymbol{c}^{S^*}\boldsymbol{y}^* = \max_{S\in\mathcal{U}_3^I(\Gamma)} \left(\min_{\boldsymbol{x}\in\Phi} \boldsymbol{C}\cdot\boldsymbol{x} + \min_{\boldsymbol{y}\in\Phi_{\boldsymbol{x}}^{\alpha}} \boldsymbol{c}^S\cdot\boldsymbol{y} \right) \geq \\ \geq \max_{S\in\mathcal{U}_3^I(\Gamma)} \left(\min_{\boldsymbol{x}\in\Phi} \boldsymbol{C}\cdot\boldsymbol{x} + \min_{\boldsymbol{y}\in\Phi} \boldsymbol{c}^S\cdot\boldsymbol{y} \right) = \\ = \min_{\boldsymbol{x}\in\Phi} \boldsymbol{C}\cdot\boldsymbol{x} + \max_{S\in\mathcal{U}_3^I(\Gamma)} \min_{\boldsymbol{y}\in\Phi} \boldsymbol{c}^S\cdot\boldsymbol{y}\end{aligned}$$

the lemma follows. □

Lemma 3.3 shows that ρ^* is not large if the budget is not large in comparison with the first and second stage solution costs. The value of

$$\min_{\boldsymbol{x}\in\Phi} \boldsymbol{C}\cdot\boldsymbol{x}$$

can be computed in polynomial time if $\mathcal{P}$ is polynomially solvable. Also, the value of

$$\max_{S\in\mathcal{U}_3^I(\Gamma)} \min_{\boldsymbol{y}\in\Phi} \boldsymbol{c}^S\cdot\boldsymbol{y}$$

can be sometimes computed efficiently by dualizing the inner minimization problem and solving a resulting linear programming formulation.

The next two lemmas are valid only for the uncertainty set $\mathcal{U}_2^I(\Gamma)$.

Lemma 3.4. *Assume that the uncertainty set is $\mathcal{U}_2^I(\Gamma)$. If*

$$\Gamma \geq \beta \sum_{i\in\{1,\ldots,n\}} d_i = \beta D$$

for $\beta \in (0,1]$, then $\rho^* \leq \frac{1}{\beta}$.

Proof. Choose a cost vector $\boldsymbol{c}'$ of a scenario $S' \in \mathcal{U}_2^I(\Gamma)$ such that $c'_i = \min\{c_i + d_i, c_i + \Gamma\frac{d_i}{D}\}$ for $i \in \{1,\ldots,n\}$. It is clear that $S' \in \mathcal{U}_2^I(\Gamma)$, because $\sum_{i=1}^n \delta'_i \leq \sum_{i=1}^n \Gamma\frac{d_i}{D} = \Gamma$. Let $(\boldsymbol{x}^*, \boldsymbol{y}^*) \in \mathcal{Z}$ be an optimal solution to the recoverable problem under $\boldsymbol{c}'$. Consider the second term in the minimum in (3.22). Because $\boldsymbol{C}\cdot\boldsymbol{x}^* \geq 0$ and $(\boldsymbol{c}+\boldsymbol{d})\cdot\boldsymbol{y}^* \geq \boldsymbol{c}'\cdot\boldsymbol{y}^*$, we get the following estimation:

$$\rho^* \leq \frac{(\boldsymbol{c}+\boldsymbol{d})\cdot\boldsymbol{y}^*}{\boldsymbol{c}'\cdot\boldsymbol{y}^*} = \frac{\sum_{i\in\{1,\ldots,n\}}(c_i+d_i)y_i^*}{\sum_{i\in\{1,\ldots,n\}}\min\{c_i+d_i, c_i+\Gamma\frac{d_i}{D}\}y_i^*}.$$

Let $I_1 = \{i \in \boldsymbol{y}^* : c_i + d_i \leq c_i + \Gamma\frac{d_i}{D}\}$ and $I_2 = \boldsymbol{y}^* \setminus I_1$. We get

$$\rho^* \leq \frac{\sum_{i\in I_1\cup I_2}(c_i+d_i)}{\sum_{i\in I_1}(c_i+d_i) + \sum_{i\in I_2}(c_i+\Gamma\frac{d_i}{D})} \leq \frac{\sum_{i\in I_1\cup I_2}(c_i+d_i)}{\sum_{i\in I_1}(c_i+d_i) + \sum_{i\in I_2}(c_i+\beta d_i)}.$$

Because $\beta \in (0,1]$,

$$\rho^* \leq \frac{\sum_{i\in I_1\cup I_2}(c_i+d_i)}{\beta(\sum_{i\in I_1}(c_i+d_i) + \sum_{i\in I_2}(c_i+d_i))} = \frac{1}{\beta}.$$

□

Lemma 3.4 shows that ρ^* is not large if the budget Γ is not significantly smaller than D, which denotes the maximum amount of the uncertainty which can be allocated to the second stage item costs.

Lemma 3.5. *Assume that the uncertainty set is $\mathcal{U}_2^I(\Gamma)$. Let $q = |\{i \in \{1,\ldots,n\} : d_i > 0\}|$. Then, the inequality $\rho^* \leq q+1$ holds. Furthermore, if $\mathcal{P}$ is an equal cardinality problem and $d_i \geq \Gamma/n$ for each $i \in \{1,\ldots,n\}$, then $\rho^* \leq \frac{n}{m}+1$.*

Proof. Choose a cost vector $\boldsymbol{c}'$ of a scenario $S' \in \mathcal{U}_2^I(\Gamma)$ such that $c'_i = \min\{c_i+d_i, c_i+\frac{\Gamma}{q}\}$ for $i \in \{1,\ldots,n\}$. Indeed, $\boldsymbol{c}' \in \mathcal{U}_2^I(\Gamma)$, because $\sum_{i\in\{1,\ldots,n\}}\delta'_i = \sum_{\{i\in\{1,\ldots,n\}:d_i>0\}}\delta'_i \leq q\cdot\frac{\Gamma}{q} = \Gamma$. Let $(\boldsymbol{x}^*, \boldsymbol{y}^*) \in \mathcal{Z}$ be an optimal solution to the recoverable problem under $\boldsymbol{c}'$. Using the first term in the minimum in (3.22) we get

$$\rho^* \leq 1 + \frac{\Gamma}{\boldsymbol{C}\cdot\boldsymbol{x}^* + \boldsymbol{c}'\cdot\boldsymbol{y}^*} \leq 1 + \frac{\Gamma}{\boldsymbol{c}'\cdot\boldsymbol{y}^*} = 1 + \frac{\Gamma}{\sum_{i\in\{1,\ldots,n\}}\min\{c_i+d_i, c_i+\frac{\Gamma}{q}\}y_i^*}.$$

Let $I_1 = \{i \in \boldsymbol{y}^* : c_i + d_i \leq c_i + \frac{\Gamma}{q}\}$ and $I_2 = \boldsymbol{y}^* \setminus I_1$. We get

$$\rho^* \leq 1 + \frac{\Gamma}{\sum_{i \in I_1}(c_i + d_i) + \sum_{i \in I_2}(c_i + \frac{\Gamma}{q})}.$$

If $I_2 = \emptyset$, then we obtain $\rho^* = 1$ by using the second term in the minimum in (3.22). So $|I_2| \geq 1$ and we can estimate

$$\rho^* \leq 1 + \frac{\Gamma}{\frac{\Gamma}{q}} = 1 + q.$$

If $d_i \geq \frac{\Gamma}{n}$ for each $i \in \{1, \ldots, n\}$ and $\mathcal{P}$ is an equal cardinality problem, then $q = n$, $I_1 = \emptyset$, $|I_2| = m$ and

$$\rho^* \leq 1 + \frac{\Gamma}{\sum_{i \in I_2} \frac{\Gamma}{n}} = 1 + \frac{n}{m}.$$

□

We can now apply Lemma 3.5 to several special cases of problem $\mathcal{P}$ under $\mathcal{U}_2^I(\Gamma)$, with $d_i \geq \frac{\Gamma}{n}$ for each $i \in \{1, \ldots, n\}$. If $\mathcal{P}$ is the selection problem discussed in [18], then $\rho^* \leq 1 + \frac{n}{p}$. If $\mathcal{P}$ is the minimum spanning tree problem in a sparse graph, in which $|E| \leq \theta|V|$ for some constant $\theta \geq 1$, then $\rho^* \leq 1 + \frac{\theta|V|}{|V|-1} \sim 1 + \theta$ for large graphs. If $\mathcal{P}$ is the minimum assignment problem, then $\rho^* \leq 1 + \sqrt{n}$.

3.5 Experiments

In this section we will show the results of some experiments. We will test the lower bounds and approximate solutions using two problems, namely the assignment and the knapsack ones. The assignment problem is polynomially solvable and has the equal cardinality property. Hence we can apply all the lower bounds, proposed in Section 3.3, to this problem. On the other hand, the knapsack problem is NP-hard and does not possess the equal cardinality property. In consequence, only the adversarial and cardinality selection constraint lower bounds will be used for this problem. We will use scenario set $\mathcal{U}_2^I(\Gamma)$, i.e., the continuous budgeted uncertainty. The experiments were executed on a `2 GHz` computer equipped with 80 `Intel(R) Xeon(R) CPU E7-4850` processors. We used `IBM ILOG CPLEX 12.8.0.0` optimizer [38] to solve the MIP formulations.

The reader is encouraged to see Appendix A for technical information about the implementation of conducted experiments.

3.5.1 The minimum assignment problem

In this section we will show the results of experiments when $\mathcal{P}$ is the following assignment problem:

$$\begin{aligned} \min \quad & \sum_{i\in\{1,\dots,m\}} \sum_{j\in\{1,\dots,m\}} C_{ij}x_{ij} \\ & \sum_{i\in\{1,\dots,m\}} x_{ij} = 1, \qquad j \in \{1,\dots,m\}, \\ & \sum_{j\in\{1,\dots,m\}} x_{ij} = 1, \qquad i \in \{1,\dots,m\}, \\ & x_{ij} \in \{0,1\}, \qquad i,j \in \{1,\dots,m\}. \end{aligned} \tag{3.23}$$

The experiment was performed for $m \in \{10, 25, 100\}$, so the number of variables $n \in \{100, 625, 10\ 000\}$. The parameters were generated in the following way:

1. The first stage costs C_{ij}, nominal second stage costs c_{ij} are random integers uniformly distributed in $[1, 20]$.
2. The maximal deviations d_{ij} are random integers uniformly distributed in $[0, 100]$.
3. The budget $\Gamma = 0.1 \sum_{i,j\in\{1,\dots,n\}} d_{ij}$, hence it is equal to 10% of the total uncertainty of the second stage cots.
4. $\alpha \in \{0.1, 0.2, \dots, 0.9\}$.
5. The accuracy ϵ in Algorithm 3.1 and Algorithm 3.2 was set to 0.01 and both algorithms were terminated if the running time exceeds 600 seconds. The accuracy of computing the Lagrangian lower bound by a version of golden search method was set to 0.1. The maximal time of solving the problem (3.17) was set to 600 seconds. After this time the computations of the bound were terminated.

For each parameters setting, we have generated 10 random instances. In the first experiment we have computed, for every instance, the ratio

$$\rho(\boldsymbol{c}^{S_0}) = \frac{\min\{\textsc{Rec}(\boldsymbol{c}) + \Gamma, \textsc{Rec}(\boldsymbol{c}+\boldsymbol{d})\}}{\textsc{Rec}(\boldsymbol{c}^{S_0})}, \tag{3.24}$$

where $\boldsymbol{c}^{S_0}$ is the heuristic scenario proposed in Section 3.3.1. Computing this ratio requires solving three recoverable problems. Recall that the best of the solutions $\underline{\boldsymbol{x}}$ or $\overline{\boldsymbol{x}}$ has an approximation ratio at most $\rho(\boldsymbol{c}^{S_0})$.

The average ratios $\rho(\boldsymbol{c}^{S_0})$ for various m and the average times required to compute it for $m = 100$ are shown on Figures 3.2 and 3.3. Observe first, that $\rho(\boldsymbol{c}^{S_0})$ can be computed efficiently. The average time required to compute $\rho(\boldsymbol{c}^{S_0})$, for $m = 100$, is less than 25 seconds. It can be observed that the time is significantly smaller for larger α. The average ratios $\rho(\boldsymbol{c}^{S_0})$ are less than 2. Interestingly, the ratio $\rho(\boldsymbol{c}^{S_0})$ is smaller for larger m (for $m = 100$ the average ratios $\rho(\boldsymbol{c}^{S_0})$ are less than 1.2). This fact is true for the particular

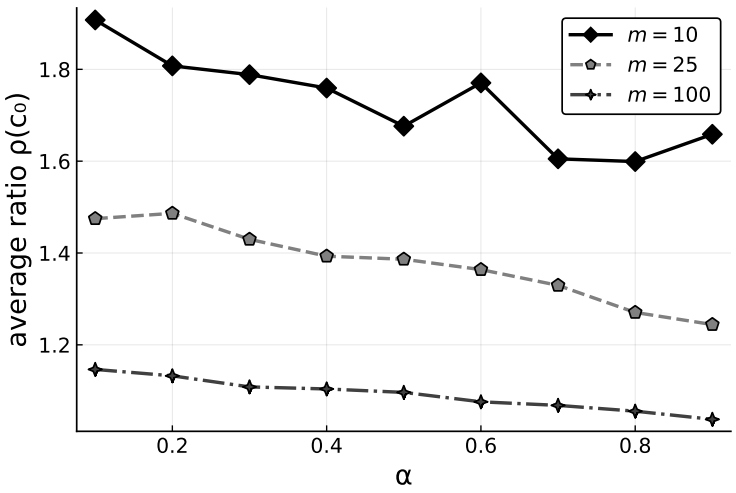


Figure 3.2: The average ratios $\rho(\boldsymbol{c}^{S_0})$ for the assignment problem with $m \in \{10, 25, 100\}$.

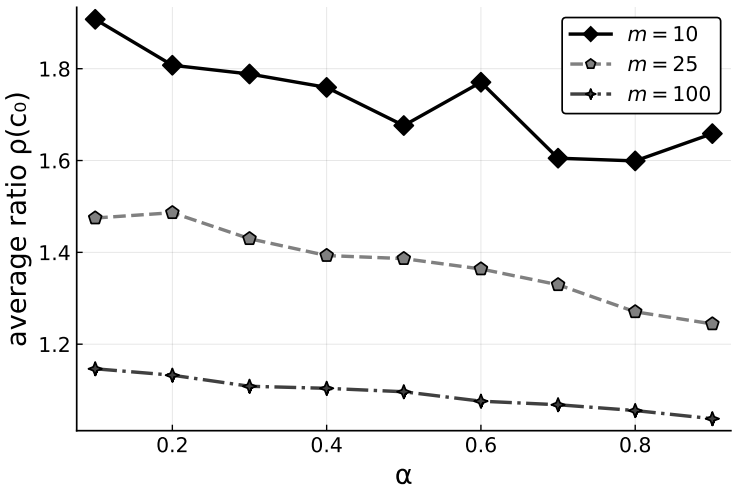


Figure 3.3: The average running times of computing $\rho(\boldsymbol{c}^{S_0})$ for the assignment problem with $m = 100$.

method of data generation and may be different for other settings (verifying this requires more tests).

We now investigate the cases $m = 10$ and $m = 25$ in more detail. For each instance we computed: the adversarial lower bound LB_{Adv} by executing Algorithm 3.2, the lower bound $LB_h = \textsc{Rec}(\boldsymbol{c}^S)$ for scenario $\boldsymbol{c}^S \in \mathcal{U}_3^I(\Gamma)$, proposed in Section 3.3.1, the cardinality selection constraint lower bound LB_{Sel} by solving the MIP formulation (3.13) constructed in Section 3.3.2 and the Lagrangian lower bound LB_{Lag} constructed in Section 3.3.3. Notice that $LB_h \leq LB_{Adv}$ for every instance, as we start Algorithm 3.2 from the initial scenario $\boldsymbol{c}^{S_0}$. We also computed the first stage solutions $\underline{\boldsymbol{x}}$ and $\overline{\boldsymbol{x}}$ (see Section 3.4), and the quantities

EVAL($\underline{\boldsymbol{x}}$) and EVAL($\overline{\boldsymbol{x}}$) by using Algorithm 3.1. We have computed the average ratios

$$\rho_k = \frac{\min\{\text{EVAL}(\underline{\boldsymbol{x}}), \text{EVAL}(\overline{\boldsymbol{x}})\}}{LB_k}, \tag{3.25}$$

for each lower bound LB_k. We also measured the average running times of computing the ratios.

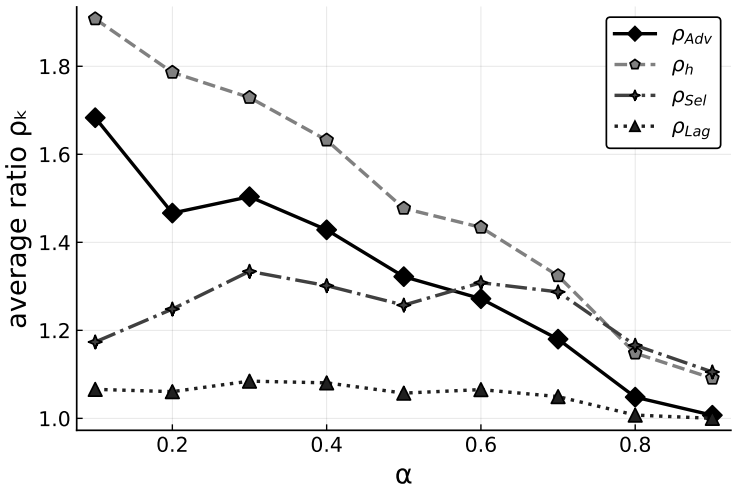


Figure 3.4: The average ratios ρ_k for the assignment problem with $m = 10$.

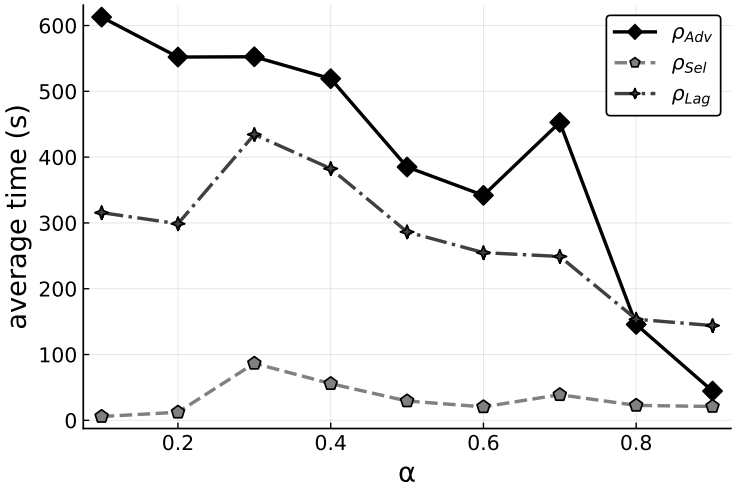


Figure 3.5: The average times of computing ρ_k for the assignment problem with $m = 10$.

The results for $m = 10$ are shown on Figures 3.4 and 3.5. For $m = 10$, all the quantities were solved to optimality, or with the assumed accuracy ϵ, when Algorithms 3.1 and 3.2 were used. One can observe in Figures 3.4 and 3.5 that one of $\underline{\boldsymbol{x}}$ or $\overline{\boldsymbol{x}}$ is always a good approximate solution. The best lower bound can be computed by using the Lagrangian

relaxation technique (the bound LB_{Lag}). On can also see in Figure 3.5 that the evaluation and all lower bounds can be computed in reasonable time. The best lower bound is LB_{Lag}. As one can expect, the lower bound LB_{Sel} is better than LB_{Adv} for smaller α and worse for larger. Notice, however, that LB_{Sel} can be computed very efficiently.

Figures 3.6 and 3.7 show the results for $m = 25$. We can still observe an improvement of LB_{Adv} over LB_h. For this case, not all solutions $\overline{\boldsymbol{x}}$ and $\underline{\boldsymbol{x}}$ were evaluated exactly. For some instances Algorithm 3.2 was terminated after the time of 600 seconds was exceeded. In this case we obtained upper bounds on Eval($\underline{\boldsymbol{x}}$) and Eval($\overline{\boldsymbol{x}}$). The Lagrangian lower bound LB_{Lag} was harder to compute than for $m = 10$. In Figure 3.7, in the brackets the number of instances, for which LB_{Lag} was computed successfully, is shown. But, as for $m = 10$, it outperforms all the remaining lower bounds and suggests that the approximate solutions behave well. The cardinality selection constraint lower bound LB_{Sel} outperforms LB_{Adv} for $\alpha \leq 0.5$. The time required to compute LB_{Sel} is again small.

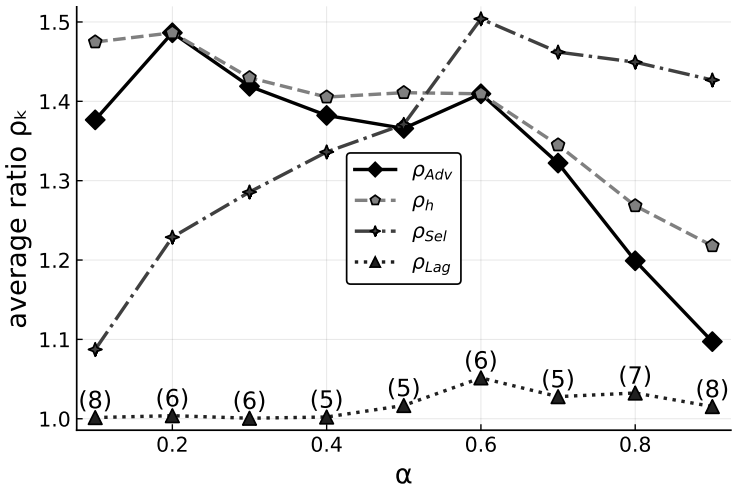


Figure 3.6: The average ratios ρ_k for the assignment problem with $m = 25$. The numbers in brackets denote the number of instances for which the value of LB_{Lag} was computed successfully.

3.5.2 The minimum knapsack problem

In this section we will show the results of experiments when $\mathcal{P}$ is the following minimum knapsack problem:

$$\begin{aligned} \min \quad & \sum_{i\in\{1,\ldots,n\}} C_i x_i \\ & \sum_{i\in\{1,\ldots,n\}} w_i x_i \geq W, \\ & x_i \in \{0,1\}, \qquad i \in \{1,\ldots,n\}. \end{aligned}$$

The test were performed for $n \in \{100, 400, 1000\}$, with the following parameter setting:

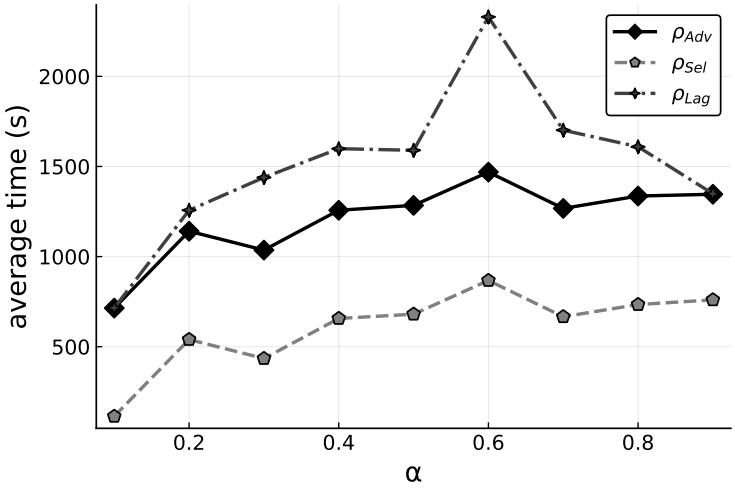


Figure 3.7: The average times of computing average ratios ρ_k for the assignment problem with $m = 25$.

- The first stage costs C_i, nominal second stage costs c_i, and weights w_i are random integers uniformly distributed in $[1, 20]$. The knapsack capacity $W = 0.3 \sum_{i\in\{1,\dots,n\}} w_i$.
- The maximal deviations d_i are random integers uniformly distributed in $[0, 100]$.
- The budget $\Gamma = 0.1 \sum_{i\in\{1,\dots,n\}} d_i$.
- $\alpha \in \{0.1, 0.2, \dots, 0.9\}$.
- The accuracy in Algorithm 3.1 and Algorithm 3.2 was set to 0.01. Algorithm 3.1 and Algorithm 3.2 were terminated after the time limit of 600 seconds was exceeded. Also, the time limit on the MIP formulation (3.13), for computing the cardinality selection constraint lower bound, was set to 600 seconds. If this time was exceeded, then an estimation from below for this lower bound was returned.

For each parameters settings, we have generated 10 random instances. In the first experiment we computed the ratio $\rho(\boldsymbol{c}^{S_0})$ by using (3.24). The average ratios and the average running time of computing them for $n = 1000$ are shown on Figures 3.8 and 3.9.

Observe first that the ratio $\rho(\boldsymbol{c}^{S_0})$ can be computed efficiently. The largest running times were observed for $\alpha = 0.3$. The average value of this ratio is less than 2.0 and for smaller n the figure is more chaotic. For $n = 1000$, the average value of $\rho(\boldsymbol{c}^{S_0})$ is close to 1.975 for all α. This behavior is different than for the assignment problem (see Figure 3.2), where the ratio is significantly smaller for larger instances and slightly decreases when α increases.

We next considered the case with $n = 100$. Figures 3.10 and 3.11 shows the average ratios ρ_{Adv}, ρ_h and ρ_{Sel} for $n = 100$ (see (3.25)) and the average times for computing these ratios for various α. One can observe that the approximation algorithm proposed in

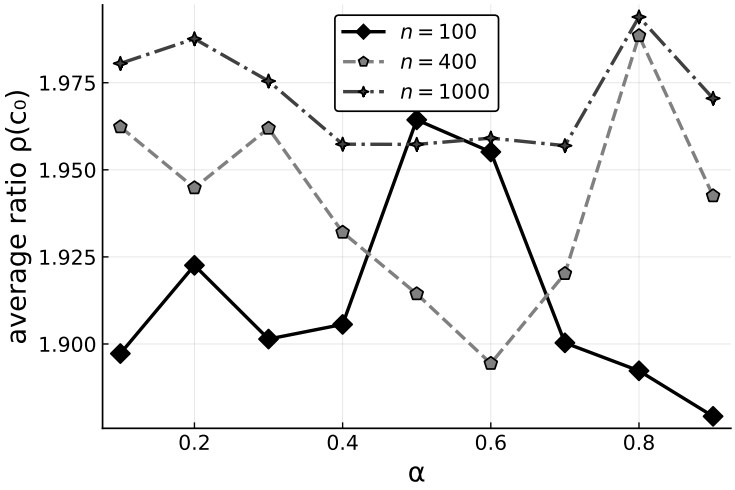


Figure 3.8: The average ratios $\rho(\boldsymbol{c}^{S_0})$ for the knapsack problem with $n \in \{100, 400, 1000\}$.

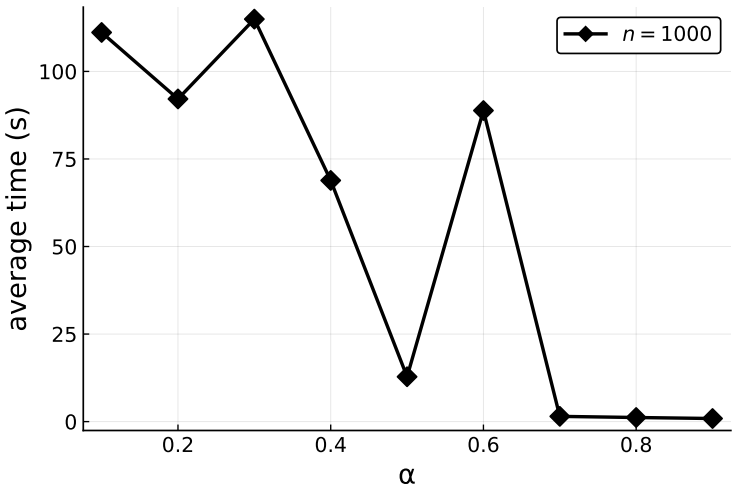


Figure 3.9: The average times of computing $\rho(\boldsymbol{c}^{S_0})$ for the knapsack problem with $n = 1000$.

Section 3.4 performs well for the tested instances. By using better of LB_{Adv} and LB_{Sel}, the average ratio for each α was not greater than 1.5. There is also an improvement of ρ_{Adv} over ρ_h. The cardinality selection constraint lower bound is better than the adversarial one for $\alpha < 0.4$ and worse for $\alpha > 0.4$. Observe that computing ρ_{Sel} for the knapsack problem is more time consuming than for the assignment.

On Figures 3.12 and 3.13 the results for $n = 400$ are shown. One can observe similar a relation between ρ_{Sel} and ρ_{Adv} as for the smaller problem with $n = 100$. However, the adversarial lower bound is now harder to compute and most instances were not solved to optimality (Algorithm 3.1 was terminated after the time of 600 seconds was exceeded). Observe that there is no significant improvement of ρ_{Adv} over ρ_h.

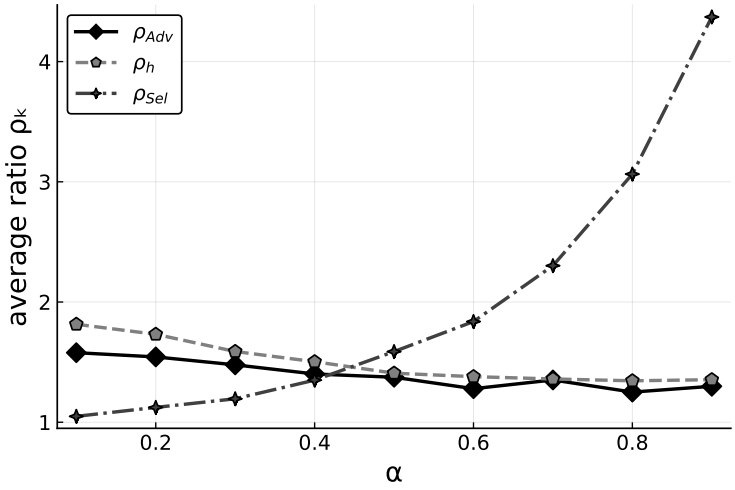


Figure 3.10: The average ratios ρ_k for the knapsack problem with $n = 100$.

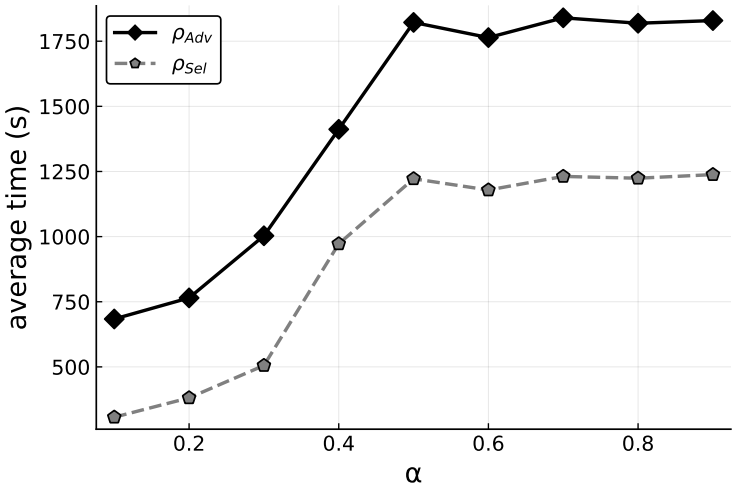


Figure 3.11: The average times of computing the average ratios ρ_k for the knapsack problem with $n = 100$.

3.5.3 Summary of the tests

Let us briefly summarize the results of the tests. For the assumed method of data generation, the ratio $\rho(\boldsymbol{c}^{S_0})$ is almost always not greater than 2. Furthermore $\rho(\boldsymbol{c}^{S_0})$ can be computed efficiently for quite large instances, with thousands of variables. This suggests that the best of solutions $\underline{\boldsymbol{x}}$, $\overline{\boldsymbol{x}}$ has, for the tested instances, the empirical approximation ratio less than 2. One can conclude that this ratio is indeed significantly smaller than 2, by using better lower bounds. However, computing these lower bounds is more time-consuming and can be done efficiently for smaller instances.

The solutions $\underline{\boldsymbol{x}}$, $\overline{\boldsymbol{x}}$ can be computed by solving the recoverable problem. For the assignment and knapsack problem, the recoverable problem is not particularly difficult to

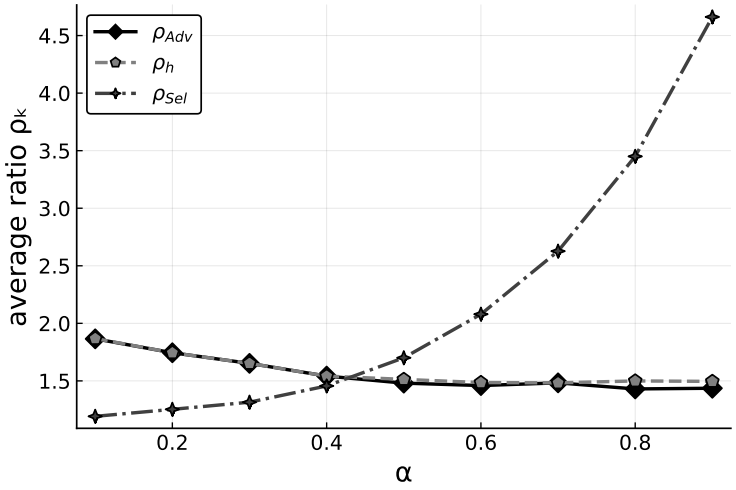


Figure 3.12: The average ratios ρ_k.

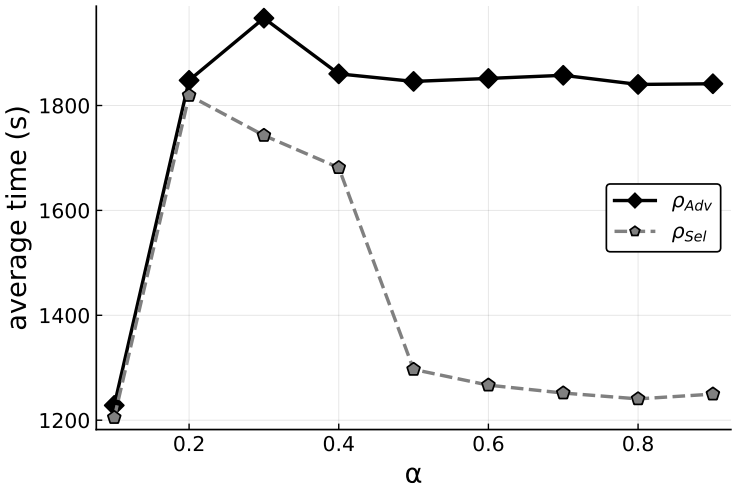


Figure 3.13: The average times for computing the average ratios ρ_k for the knapsack problem with $n = 400$. The time required to compute ρ_H is negligible.

solve by `CPLEX`. However, the evaluation problem is more difficult, as we have to use the relaxation algorithm to perform this task. For large instances, we should assume more time for executing Algorithm 3.1. We can then choose the solution among $\underline{\boldsymbol{x}}$, $\overline{\boldsymbol{x}}$, which has better upper bound on the value of Eval($\boldsymbol{x}$).

Notice that the techniques proposed in this chapter are general. For specific problem $\mathcal{P}$, the incremental, recoverable and evaluation problems can be solved by specialized algorithms (even in polynomial time). So, one can obtain better estimations for larger instances.

3.6 Conclusion and open issues

This chapter considers a general class of 0-1 optimization problems. Problems belonging to this class can be either polynomially solvable or NP-hard. The concept of recoverable robustness was applied to take into account the possibility of performing a recourse action on the current first-stage solution. In this chapter we use the polyhedral uncertainty representation which generalizes continuous interval budgeted uncertainty. Moreover, this uncertainty representation may lead to more tractable problems than the other uncertainty representations. Unfortunately, the resulting min-max-min problem can be still too complex to solve. Instead of solving the problem to optimality, we proposed to use some approximate solutions. The quality of these solutions can be estimated by using various lower bounds. One can apply this approach to relatively large instances of the recoverable version of any 0-1 programming problem $\mathcal{P}$. In this chapter we have shown the results of computational tests when $\mathcal{P}$ is the knapsack problem with up to 1000 variables or the minimum assignment problem with up to 10 000 variables.

Speaking of the open questions, one can try to solve the formulation (3.5) by using a row and column generation technique. The corresponding computational tests should be performed for various problems $\mathcal{P}$. Notice that even the incremental and recoverable versions of $\mathcal{P}$ can be nontrivial and interesting from the computational point of view. Solving the robust version can be done more efficiently if a polynomial algorithm is known for the incremental or recoverable problem. However, this is the case only for very specific problems, for example, such as the selection or the minimum spanning tree problems.

Chapter 4

Summary and conclusions

In this thesis we investigate the recoverable robust versions of a class of discrete combinatorial optimization problems with uncertain costs. The uncertainty is modelled by utilizing the discrete or interval scenario sets. The advantage of using the robust approach lies in the fact that it allows us to deal with an uncertainty of a problem parameters in an optimization problems, especially when no probability distribution of scenarios is known. On the other hand, this approach can be excessively risk averse in many practical applications generating a feasible solutions influenced by the worst case scenarios. In turn, the concept of recoverability allows us to change selected solution to some extent after uncertain parameters were realized. Those two concepts can be combined together giving an opportunity to control the degree of uncertainty.

This doctorate thesis consists of the two main parts. The first part, Chapter 2, deals with the recoverable robust spanning tree problem under interval uncertainty representation. The second part, Chapter 3, investigate recoverable robust 0-1 programming models.

In Chapter 2 we have studied the recoverable robust spanning tree problem under a number of interval uncertainty representations. It was shown that the problem is polynomially solvable under the interval uncertainty representation $\mathcal{U}^I$, thus resolving a problem which has been open to date in the literature. The idea has been also generalized to the recoverable robust matroid basis problem with interval element costs. Next we have constructed a polynomial time combinatorial algorithm for the recoverable robust spanning tree problem. Additionally, the algorithm has been used to provide several approximation results for recoverable spanning tree problem with the scenario sets with a budged constraint.

Chapter 3 considers a general class of 0-1 optimization problems without assumptions about their complexity. We have used the polyhedral uncertainty representation which the generalizes continuous interval budgeted uncertainty. Although, this uncertainty representation may lead to more tractable problems than other uncertainty representations, the resulting problems may still be too hard to solve. Thus, we proposed the usage of approximate solutions. The quality of these solutions can be estimated by using various lower bounds. In this chapter we have applied this approach by performing computational experiments on the knapsack problem with up to 1000 variables and the minimum assignment

problem with up to 10 000 variables.

There is still a number of open questions regarding discussed problems. As an example, the complexity of the recoverable robust spanning tree problem under the interval uncertainty representation with budgeted constraint is not yet known. The complexity of the recoverable robust matroid basis problem under different uncertainty representations is also an interesting topic of future research.

Bibliography

[1] R. K. Ahuja, T. L. Magnanti, and J. B. Orlin. *Network Flows: theory, algorithms, and applications*. Prentice Hall, Englewood Cliffs, New Jersey, 1993.

[2] H. Aissi, C. Bazgan, and D. Vanderpooten. Approximation of min-max and min-max regret versions of some combinatorial optimization problems. *European Journal of Operational Research*, 179:281–290, 2007.

[3] I. Averbakh. On the complexity of a class of combinatorial optimization problems with uncertainty. *Mathematical Programming*, 90:263–272, 2001.

[4] A. Bar-Noy, A. S. Khuller, and B. Schieber. The complexity of finding most vital arcs and nodes. Technical Report CS-TR-3539, Institute for Advanced Studies, University of Maryland, College Park, MD, 1995.

[5] A. Ben-Tal, L. El Ghaoui, and A. Nemirovski. *Robust optimization*. Princeton Series in Applied Mathematics. Princeton University Press, Princeton, NJ, 2009.

[6] A. Ben-Tal, A. Goryashko, E. Guslitzer, and A. Nemirovski. Adjustable robust solutions of uncertain linear programs. *Mathematical Programming A*, 99:351–376, 2004.

[7] A. Ben-Tal and A. Nemirovski. Robust solutions of uncertain linear programs. *Operations Research Letters*, 25:1–13, 1999.

[8] D. Bertsimas and M. Sim. Robust discrete optimization and network flows. *Mathematical Programming*, 98:49–71, 2003.

[9] D. Bertsimas and M. Sim. The price of robustness. *Operations research*, 52:35–53, 2004.

[10] D. Bienstock. Histogram models for robust portfolio optimization. *Journal of Computational Finance*, 11, 09 2007.

[11] J. R. Birge and F. Louveaux. *Introduction to Stochastic Programming*. Springer-Verlag, 1997.

[12] C. Buchheim and J. Kurtz. Min-max-min robust combinatorial optimization. *Mathematical Programming A*, 163:1–23, 2017.

[13] C. Büsing. *Recoverable robustness in combinatorial optimization.* PhD thesis, Technical University of Berlin, Berlin, 2011.

[14] C. Büsing. Recoverable robust shortest path problems. *Networks*, 59:181–189, 2012.

[15] C. Büsing, A. M. C. A. Koster, and M. Kutschka. Recoverable robust knapsacks: the discrete scenario case. *Optimization Letters*, 5:379–392, 2011.

[16] L. Cadarso and Ángel Marín. Recoverable robustness in rapid transit network design. *Procedia - Social and Behavioral Sciences*, 54:1288–1297, 2012. Proceedings of EWGT2012 - 15th Meeting of the EURO Working Group on Transportation, September 2012, Paris.

[17] A. Chassein and M. Goerigk. On the recoverable robust traveling salesman problem. *Optimization Letters*, 10:1479–1492, 2016.

[18] A. Chassein, M. Goerigk, A. Kasperski, and P. Zieliński. On recoverable and two-stage robust selection problems with budgeted uncertainty. *European Journal of Operational Research*, 265:423–436, 2018.

[19] S. Cicerone, G. D'Angelo, G. Di Stefano, D. Frigioni, A. Navarra, M. Schachtebeck, and A. Schöbel. *Recoverable Robustness in Shunting and Timetabling*, pages 28–60. Springer Berlin Heidelberg, Berlin, Heidelberg, 2009.

[20] S. Cicerone, G. Di Stefano, M. Schachtebeck, and A. Schöbel. Multi-stage recovery robustness for optimization problems: A new concept for planning under disturbances. *Information Sciences*, 190:107–126, 2012.

[21] T. Cormen, C. Leiserson, and R. Rivest. *Introduction to Algorithms.* MIT Press, 1990.

[22] O. Şeref, R. K. Ahuja, and J. B. Orlin. Incremental network optimization: theory and algorithms. *Operations Research*, 57:586–594, 2009.

[23] W. H. Cunningham. Testing membership in matroid polyhedra. *Journal of Combinatorial Theory, Series B*, 36:161–188, 1984.

[24] G. D'Angelo, G. Di Stefano, A. Navarra, and C. M. Pinotti. Recoverable robust timetables: An algorithmic approach on trees. *IEEE Transactions on Computers*, 60(3):433–446, 2011.

[25] D. Dubois and P. Fortemps. Computing improved optimal solutions to max–min flexible constraint satisfaction problems. *European Journal of Operational Research*, 118(1):95–126, 1999.

[26] J. Edmonds. Matroids and the greedy algorithm. *Mathematical Programming*, 1:127–136, 1971.

[27] D. Fischer, T. A. Hartmann, S. Lendl, and G. J. Woeginger. An investigation of the recoverable robust assignment problem. *CoRR*, abs/2010.11456, 2020.

[28] N. G. Frederickson and R. Solis-Oba. Increasing the weight of minimum spanning trees. *Journal of Algorithms*, 33:244–266, 1999.

[29] L. Galand and O. Spanjaard. Exact algorithms for OWA-optimization in multiobjective spanning tree problems. *Computers and Operations Research*, 39:1540–1554, 2012.

[30] L. E. Ghaoui and H. Lebret. Robust solutions to least-squares problems with uncertain data. *SIAM Journal on Matrix Analysis and Applications*, 18:1035–1064, 1997.

[31] L. E. Ghaoui, F. Oustry, and H. Lebret. Robust solutions to least-squares problems with uncertain data. *SIAM Journal on Optimization*, 9:33–52, 1998.

[32] M. Goerigk, S. Lendl, and L. Wulf. Recoverable robust representatives selection problems with discrete budgeted uncertainty. *CoRR*, abs/2008.12727, 2020.

[33] M. Hradovich, A. Kasperski, and P. Zieliński. Recoverable robust spanning tree problem under interval uncertainty representations. *Journal of Combinatorial Optimization*, 34:554–573, 2017.

[34] M. Hradovich, A. Kasperski, and P. Zieliński. The recoverable robust spanning tree problem with interval costs is polynomially solvable. *Optimization Letters*, 11:17–30, 2017.

[35] M. Hradovich, A. Kasperski, and P. Zieliński. Risk averse scheduling with scenarios. In N. Kliewer, J. F. Ehmke, and R. Borndörfer, editors, *Operations Research Proceedings 2017*, pages 435–441, Cham, 2018. Springer International Publishing.

[36] M. Hradovich, A. Kasperski, and P. Zieliński. Robust recoverable 0–1 optimization problems under polyhedral uncertainty. *European Journal of Operational Research*, 278(1):136 – 148, 2019.

[37] IBM. Ibm cplex optimizer. https://www.ibm.com/analytics/cplex-optimizer.

[38] IBM ILOG CPLEX Optimization Studio. CPLEX User's manual. https://www.ibm.com.

[39] Julia programming language community. Julia home page. https://julialang.org/.

[40] Julia programming language community. Julia v0.6 manual. https://docs.julialang.org/en/v0.6/manual/modules/.

[41] JuMP Community. Cplex.jl. https://github.com/jump-dev/CPLEX.jl.

[42] JuMP Community. Jump home page. https://jump.dev/.

[43] JuMP Community. Jump source code repository. https://github.com/jump-dev/JuMP.jl.

[44] A. Kasperski, A. Kurpisz, and P. Zieliński. Recoverable robust combinatorial optimization problems. In *Operations Research Proceedings 2012*, pages 147–153, 2014.

[45] A. Kasperski and P. Zieliński. On the approximability of robust spanning problems. *Theoretical Computer Science*, 412:365–374, 2011.

[46] A. Kasperski and P. Zieliński. *Robust Discrete Optimization Under Discrete and Interval Uncertainty: A Survey*, pages 113–143. Springer International Publishing, 2016.

[47] A. Kasperski and P. Zieliński. Robust recoverable and two-stage selection problems. *Discrete Applied Mathematics*, 233:52–64, 2017.

[48] A. Kasperski and P. Zieliński. Robust two-stage network problems. In K. F. Dörner, I. Ljubic, G. Pflug, and G. Tragler, editors, *Operations Research Proceedings 2015*, pages 35–40, Cham, 2017. Springer International Publishing.

[49] I. Katriel, C. Kenyon-Mathieu, and E. Upfal. Commitment under uncertainty: two-stage matching problems. *Theoretical Computer Science*, 408:213–223, 2008.

[50] P. Kouvelis and G. Yu. *Robust Discrete Optimization and its Applications*. Kluwer Academic Publishers, 1997.

[51] L. C. Lau, R. Ravi, and M. Singh. *Iterative Methods in Combinatorial Optimization*. Cambridge University Press, 2011.

[52] S. Lendl, B. Peis, and V. Timmermans. Matroid bases with cardinality constraints on the intersection. *Mathematical Programming*, Mar 2021.

[53] C. Liebchen, M. E. Lübbecke, R. H. Möhring, and S. Stiller. The concept of recoverable robustness, linear programming recovery, and railway applications. In *Robust and Online Large-Scale Optimization*, volume 5868 of *Lecture Notes in Computer Science*, pages 1–27. Springer-Verlag, 2009.

[54] K. Lin and M. S. Chern. The most vital edges in the minimum spanning tree problem. *Information Processing Letters*, 45:25–31, 1993.

[55] R. D. Luce and H. Raiffa. *Games and Decisions: Introduction and Critical Survey*. Dover Publications Inc., 1989.

[56] T. L. Magnanti and L. A. Wolsey. Optimal Trees. In M. O. Ball, T. L. Magnanti, C. L. Monma, and G. L. Nemhauser, editors, *Network Models, Handbook in Operations Research and Management Science*, volume 7, pages 503–615. North-Holland, Amsterdam, 1995.

[57] Mikita Hradovich. Robrecsolver.jl. https://github.com/nikagra/RobRecSolver.jl.

[58] E. Nasrabadi and J. B. Orlin. Robust optimization with incremental recourse. *CoRR*, abs/1312.4075, 2013.

[59] W. Ogryczak and T. Śliwiński. On solving linear programs with the ordered weighted averaging objective. *European Journal of Operational Research*, 148(1):80–91, 2003.

[60] J. G. Oxley. *Matroid theory*. Oxford University Press, 1992.

[61] C. H. Papadimitriou and K. Steiglitz. *Combinatorial optimization: algorithms and complexity*. Dover Publications Inc., 1998.

[62] B. Roy. Robustness in operational research and decision aiding: A multi-faceted issue. *European Journal of Operational Research*, 200:629–638, 2010.

[63] M. Sim. *Robust optimization*. PhD thesis, Massachusetts Institute of Technology, 2004.

[64] A. L. Soyster. Convex Programming with Set- Inclusive Constraints and Applications to Inexact Linear Programming. *Operations Research*, 21:1154–1157, 1973.

[65] M. H. van der Vlerk. Stochastic programming bibliography. http://www.eco.rug.nl/mally/spbib.html, 1996–2007.

[66] V. Vazirani. *Approximation Algorithms*. Springer Berlin Heidelberg, 2013.

[67] D. P. Williamson and D. B. Shmoys. *The Design of Approximation Algorithms*. Cambridge University Press, 2010.

[68] R. R. Yager. On ordered weighted averaging aggregation operators in multi-criteria decision making. *IEEE Transactions on Systems, Man and Cybernetics*, 18:183–190, 1988.

[69] R. R. Yager, J. Kacprzyk, and G. Beliakov, editors. *Recent developments in the Ordered Weighted Averaging operators: Theory and Practice*. Springer, 2011.

[70] C.-S. Yu and H.-L. Li. A robust optimization model for stochastic logistic problems. *International journal of production economics*, 64(1-3):385–397, 2000.

[71] G. Yu and P. Kouvelis. Complexity results for a class of min-max problems with robust optimization applications. In P. M. Pardalos, editor, *Complexity in Numerical Optimization*. World Scientyfic, 1993.

[72] B. Zeng and L. Zhao. Solving two-stage robust optimization problems using a column and constraint generation method. *Operation Research Letters*, 41:457–461, 2013.

[73] E. Álvarez Miranda, I. Ljubić, S. Raghavan, and P. Toth. The recoverable robust two-level network design problem. *INFORMS Journal on Computing*, 27(1):1–19, 2015.

Appendix A

Julia, JuMP and RobRecSolver

In this appendix we briefly introduce *Julia* programming language, *JuMP* modeling language for mathematical optimization in Julia and *RobRecSolver.jl* package written in Julia and containing implementations of algorithms necessary for conducting experiments in Chapter 3.

A.1 Introduction

A.1.1 Julia programming language

Julia[39] is an open source general purpose programming language with dynamic typing. It was first presented in 2012 and has grown rapidly since then. The language is designed to be very effective at numerical and scientific computing, which is achieved among other things thanks to type inference and just-in-time compilation. Julia also has a large ecosystem of libraries in different scientific domains including, but not limited to machine learning, data science, differential equations and optimization tools each supported by its own community. Julia's syntax is somewhat similar to that of MATLAB's, but it has a number of major syntactic and functional differences.

The reader may want to check Julia's home page [39], especially *Learn* and *Documentation* sections for more information about the Julia programming language.

A.1.2 Mathematical optimization with JuMP

JuMP (abbr. Julia Mathematical Programming) [42] is a package of Julia programming language introducing a modeling language for mathematical optimization. JuMP requires a solver to be installed in order to be able to solve optimization problems. Several solvers, e.g., *CPLEX* or *Gurobi* require a commercial license and these solvers should be installed separately. There is also a number of solvers available for free under open source licenses, e.g., *GLPK*. JuMP providers wrappers which communicate with solvers using available APIs, for example lower-level C programming language APIs. The whole process of package

installation and configuration is described on the JuMP package's home page [42].

JuMP uses Julia's metaprogramming feature to define a modeling language for mathematical optimization problems. Lets write the following problem:

$$\begin{aligned} \min \quad & 12x + 20y \\ \text{s.t.} \quad & 6x + 8y \geq 100 \\ & 7x + 12y \geq 120 \\ & x \geq 0 \\ & 0 \leq y \leq 3 \end{aligned} \tag{A.1}$$

using JuMP modeling language:

Listing A.1: Simple model

```
using JuMP
using GLPK
model = Model(GLPK.Optimizer)
@variable(model, x >= 0)
@variable(model, 0 <= y <= 3)
@objective(model, Min, 12x + 20y)
@constraint(model, c1, 6x + 8y >= 100)
@constraint(model, c2, 7x + 12y >= 120)
print(model)
optimize!(model)
@show termination_status(model)
@show primal_status(model)
@show dual_status(model)
@show objective_value(model)
@show value(x)
@show value(y)
@show shadow_price(c1)
@show shadow_price(c2)
nothing #hide
```

Note that the example above is based on the examples available in JuMP GitHub source code repository [43].

I invite the reader to review a *Quickstart Guide* and *Documentation* sections on the JuMP home page for more information about this package.

A.2 Installation and configuration

RobRecSolver.jl [57] is a Julia programming language package containing source code for conducting experiments described in Section 3.5. Source code utilizes Julia v0.6 since that was the latest version of Julia working correctly with JuMP at the moment. *RobRecSolver.jl*

package also has its own documentation and unit tests. We invite the reader to visit GitHub repository for the *RobRecSolver.jl* package [57] for more details.

A.2.1 Getting Julia

JuMP and consequently *RobRecSolver.jl* require Julia programming language of the version `0.6`. One can build Julia from the source code or use the binaries. Download links and more detailed instructions are available on the Julia website [39].

A.2.2 Getting CPLEX Optimizer

RobRecSolver.jl utilizes *CPLEX.jl* package [41] which in turn requires a working installation of IBM CPLEX Optimizer [37] with a commercial license. The license is free for faculty members and graduate teaching assistants. IBM CPLEX Optimizer must be downloaded and installed separately.

A.2.3 Installing RobRecSolver

RobRecSolver.jl repository organization follows standard Julia's convention of package layout and can be utilized by a Julia's builtin package manager, called *Pkg*. To install it, use `Pkg.clone` command:

```
julia> Pkg.clone("https://github.com/nikagra/RobRecSolver.jl.git")
```

Since *RobRecSolver.jl* contains REQUIRE file, that file will be used to determine which registered packages *RobRecSolver.jl* depends on, and they will be automatically installed.

A.2.4 Updating RobRecSolver

In order to update package run the following sequence of commands (`;` symbol at the start of the Julia's REPL enters shell mode):

```
julia> cd(Pkg.dir("RobRecSolver"))
julia> ;
shell> git fetch --all --tags --prune && git checkout tags/<version>
julia> Pkg.resolve()
```

A.2.5 RobRecSolver.jl

Package consists of several functions implementing algorithms described in Section 3.5 as well as some utility functions. The easiest way to experiment with them is to use package in Julia's interactive session (or REPL which stands for read-eval-print loop). For example:

```
$ julia
               _
   _       _ _(_)_     |  A fresh approach to technical computing
  (_)     | (_) (_)    |  Documentation: https://docs.julialang.org
   _ _   _| |_  __ _   |  Type "?help" for help.
```

```
  | | | | | | |/ _‘ |  |
  | | |_| | | | (_| |  |  Version 0.6.3 (2018-05-28 20:20 UTC)
 _/ |\__’_|_|_|\__’_|  |  Official http://julialang.org/ release
|__/                   |  x86_64-w64-mingw32

julia> 1 + 1
2
```

Interactive session may be useful while prototyping programs since it outputs result of each evaluation and allows to check intermediate results.

To leave interactive session enter `exit()` command or press `Ctrl+D`.

Alternatively, instead of running interactive session, one can evaluate source file, which uses `.jl` filename extension by convention:

```
$ julia main.jl arg
```

In the example above Julia passes argument `arg` to a script stored in source file named `main.jl` and then executes it in a non-interactive mode. Arguments passed to the script are available within the script in a global constant `ARGS`. The name of the source file is also store in global constant under the name `PROGRAM_FILE`.

See *Julia Scripting* section of the Julia's manual [40] for more information about writing scripts in Julia.

To start using *RobRecSolver.jl* library one need to import it first with a `using` keyword, then call function they are interested in, i.e., `incrementalProblem`:

```
julia> using RobRecSolver.Experiments
julia> runExperiments([100, 400, 1000], [10, 25, 100])
```

Check *Julia Modules* section of the manual page [40] for more detailed information about using modules in Julia.

A.2.6 Additional Types and Functions

There is a number of types and helper functions defined to facilitate implementation of algorithms described in Section 3.5. `ProblemDescriptor` is one of such types. It serves as a basic interface, defining size of the problem or whether it has equal cardinality property among the other properties. There are two subtypes of `ProblemDescriptor`, namely `KnapsackProblemDescriptor` and `AssignmentProblemDescriptor` for each problem discussed in Chapter 3:

```
using RobRecSolver

n = 5
knapsackProblemDescriptor = KnapsackProblemDescriptor(n)
property = hasEqualCardinalityProperty(knapsackProblemDescriptor)
println("KnapsackProblemDescriptor.hasEqualCardinalityProperty: $property")

assignmentProblemDescriptor = AssignmentProblemDescriptor(n)
property = hasEqualCardinalityProperty(assignmentProblemDescriptor)
println("AssignmentProblemDescriptor.hasEqualCardinalityProperty: $property")
println("AssignmentProblemDescriptor.getCardinality: $(getCardinality(
    assignmentProblemDescriptor))")
```

Upon executing example above one will get the following output:

```
KnapsackProblemDescriptor.hasEqualCardinalityProperty: false
AssignmentProblemDescriptor.hasEqualCardinalityProperty: true
AssignmentProblemDescriptor.getCardinality: 5
```

Function `initialScenario` is another example of a helper function. It searches for a good heuristic initial scenario in order to speed up some computations. Its behavior is described in the Section 3.3.1. See example below on how to use this function:

```
using RobRecSolver

c = [2, 3]
d = [8, 9]
Γ = 10
s = initialScenario(c, d, Γ)
println("Initial scenario is ", s)
```

In this example `c` is a vector of the second stage costs, `d` is a vector of the maximum deviations of the costs from their nominal values and Γ is a budget or the amount of uncertainty, which can be allocated to the second stage costs.

Upon running the sequence of commands above one will receive the following output:

```
Initial scenario is [7.50073, 7.50073]
```

The functions `loadProperties` and `getProperty` allow to customize package parameters like solver time limits or logging for different algorithms.

Function `loadProperties` loads properties stored in an INI file from the specified file location. To change default location set `ROBRECSOLVER_CONFIG` environment variable either in Julia REPL or in `/.julia/config/startup.jl` and then reload RobRecSolver package:

```
julia> ENV["ROBRECSOLVER_CONFIG"] = "<path_to_file>"
julia> Pkg.reload("RobRecSolver")
```

Use default properties file `Pkg.dir("RobRecSolver")/conf/config.ini` as a reference. There is an extract from it below:

```
; Problem properties
[main]
lagrangianLowerBound.cplexLogging=0
lagrangianLowerBound.epsilon=0.000001
lagrangianLowerBound.overallTimeLimit=1800
lagrangianLowerBound.subproblemTimeLimit=600
```

See Appendix A.3.5 for full contents.

In order to reset the configuration changes simply delete the environment variable and reload RobRecSolver package.

Function `getProperty` returns value for the key from previously loaded properties file:

```
using RobRecSolver

ϵ = getProperty("evaluationProblem.epsilon", parameterType = Float64)
timeLimit = getProperty("evaluationProblem.timeLimit")
```

In the example above the value for property `evaluationProblem.epsilon` of type `Float64` is stored in variable a ϵ. Then the value for property `evaluationProblem.timeLimit` of type `Int` (default) is stored in a variable `timeLimit`. If properties section is not specified, section called `main` is used by default.

A.3 Problems

A.3.1 Incremental and Recoverable Problems

Section 3.2 presents MIP formulations for incremental and recoverable problems for element exclusion neighborhood as well as their simplified versions for equal cardinality problem.

Both versions of incremental problems are solved by a `incrementalProblem` function. Here is an example of solving incremental problem for the minimum knapsack problem:

```
julia> using RobRecSolver
julia> n = 3
julia> α = 0.5
julia> c = [1, 2, 3]
julia> x = [0, 1, 1]
julia> w = [1, 2, 2]
julia> W = 3
julia> X = getKnapsackConstraints(w, W)
julia> problemDescriptor = KnapsackProblemDescriptor(n)
julia> incrementalProblem(c, α, x, X, problemDescriptor)
```

In this example we first import the *RobRecSolver.jl* package. Then we define a number of variables, namely `c` for a vector of nonnegative nominal second stage costs, `x` for the first stage solution, variable α for a fixed number belonging to `[0,1]` as described in Chapter 3. We also define a variable `w` for a vector of item weights and `W` for the knapsack capacity. The set of feasible solutions is prepared by a function `getKnapsackConstraints`. It is represented as a list of anonymous functions each of which for a given vector of JuMP variables returns a JuMP linear constraint. Variable `problemDescriptor` is an instance of the type `KnapsackProblemDescriptor` which defines some useful properties of a problem, i.e., its size or whether it has equal cardinality property. Last step is to call the function `incrementalProblem`. It will return a tuple containing vector of the second stage solutions and the objective value.

Let us solve the recoverable minimum assignment problem using a `recoverableProblem` function:

```
julia> using RobRecSolver
julia> m = 2
julia> α = 1.0
julia> C = [1 2; 3 1]
julia> c = [5 3; 2 4]
julia> X = getAssignmentConstraints(m)
julia> problemDescriptor = AssignmentProblemDescriptor(m)
julia> recoverableProblem(C, c, X, α, problemDescriptor)
```

As in the previous example we first define some auxiliary variables. Here `C` is a vector of nonnegative first stage costs, `C` is a vector of nonnegative nominal second stage costs, `X` is a

set of feasible solutions, α is fixed number belonging to `[0, 1]` and `problemDescriptor` is an instance of the `AssignmentProblemDescriptor` type. This example will return a tuple consisting of a vector of the first stage solutions, a vector of the second stage solutions and the objective value.

A.3.2 Evaluation Problem

Let us take a look at a function named `evaluationProblem`, which implements **Algorithm 1** in Section 3.2. Here is an example of how one can use it for the minimum knapsack problem:

```
julia> using RobRecSolver
julia> n = 2
julia> α = 1.0
julia> C = [4, 3]
julia> c = [2, 3]
julia> d = [8, 9]
julia> Γ = 9
julia> x = [0, 1]
julia> w = [1, 2]
julia> W = 1
julia> X = getKnapsackConstraints(w, W)
julia> problemDescriptor = KnapsackProblemDescriptor(n)
julia> evaluationProblem(C, c, d, Γ, α, x, X, problemDescriptor)
D-  2 constraints was added to this evaluation problem                Debug
    evaluation_problem.jl:1
10.0
```

We first define a size of the problem `n`, a parameter α, a vector of the first stage costs `C`, a vector of a nonnegative nominal second stage costs `C`, a vector of the maximum deviations of the costs from their nominal values `d`, a budget Γ, a set of feasible solutions `X` and `problemDescriptor` being an instance of the type `KnapsackProblemDescriptor`. We also define a vector of item weights `w` and knapsack capacity `W`. The last step is to call `evaluationProblem` passing all necessary arguments.

A.3.3 Lower Bounds

Section 3.3 of the publication contains algorithms and MIP formulations to calculate adversarial, lagrangian and cardinality selection constraint lower bounds. Corresponding functions from *RobRecSolver.jl* package to calculate this lower bounds are respectively `adversarialProblem`, `lagrangianLowerBound` and `selectionLowerBound`. All of these functions have very similar signatures, so as an example let us take a closer look to `adversarialProblem`. This function implements the Algorithm 3.2 for calculating adversarial lower bound in the Section 3.2. An example of the source file solving it for the minimum knapsack problem is provided below:

```
using RobRecSolver

n = 2
α = 0.5

w = [1, 2]
```

```
W = 1
X = getKnapsackConstraints(w, W)

C = [1, 3]
c = [3, 1]
d = [2, 2]
Γ = 2

problemDescriptor = KnapsackProblemDescriptor(n)
result = adversarialProblem(C, c, d, Γ, X, α, problemDescriptor)
println("Adversarial lower bound is ", result)
```

Assuming the code above is saved as `adv.jl`, running the program will return the following output:

```
$ julia adv.jl
D-  3 constraints was added to this adversarial problem              Debug
    adversarial_problem.jl:1
Adversarial lower bound is 5.0
```

In this program we first define a size of the problem `n`, a parameter α, a vector of the first stage costs `C`, a vector of the second stage costs `C`, a vector of the maximal deviations of the costs from their nominal values `d`, a budget Γ, a set of feasible solutions `X` and `problemDescriptor` being an instance of the type `KnapsackProblemDescriptor`. Then we call an `adversarialProblem` function passing all necessary arguments and printing out result. Note that depending on package settings it also may print some additional logs.

A.3.4 Experiments

RobRecSolver.jl package also contains `Experiments` submodule which can serve as a reference on how to use a core package functionality in the experiments. Note that almost all functions in `RobRecSolver.Experiments` are highly customized to serve purposes of Chapter 3. Never the less let us take a closer look at functions presented here.

`RobRecSolver.Experiments.runExperiments` is an entry point of the experiments. This function accepts a list of minimum knapsack problem sizes `ns`, a list of minimum assignment problem sizes `ms` and optionally a list of values of parameter α called α`s` and a number of instances to be generated for each value of α called `numberOfInstances`. By default α`s` has values $0.1, 0.2, ..., 0.9$ and `numberOfInstances` equals 5:

```
using RobRecSolver.Experiments

runExperiments([100, 400, 1000], [10, 25, 100])
```

Check Chapter 3 for more information about the scope of the experiments.

`RobRecSolver.Experiments.saveCsv` is a helper function developed to save experiment results as CSV files. It saves data described by `columnNames` argument using data passed to `data` argument to CSV file with name `filename`:

```
using RobRecSolver

Experiments.saveCsv("item_prices.csv", ["milk" 100; "ham" 250], ["item", "price"])
```

The above script will save a CSV file `item_prices.csv` with the following content:

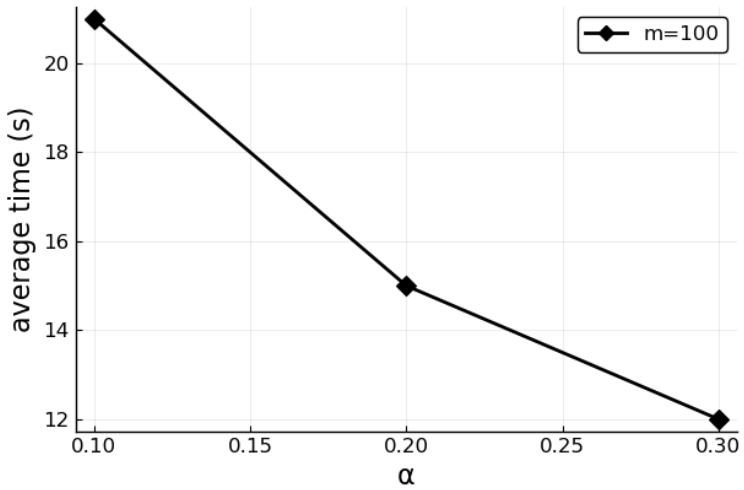
```
item,price
milk,100
ham,250
```

Function `RobRecSolver.Experiments.drawAndSavePlot` is a function used to draw plots used in Chapter 3. It accepts a number of parameteres and is heavily based on a `PyPlot` backend. As an example. the following code snippet:

```
using RobRecSolver, Plots

pyplot()
Experiments.drawAndSavePlot("plot.pdf", [0.1, 0.2, 0.3], [21, 15, 12], "α", "average time
    (s)", "m=100")
```

will produce the following plot:



A.3.5 Properties File

```
; Problem properties
[main]
lagrangianLowerBound.cplexLogging=0
lagrangianLowerBound.epsilon=0.000001
lagrangianLowerBound.overallTimeLimit=1800
lagrangianLowerBound.subproblemTimeLimit=600

selectionLowerBound.timeLimit=600
selectionLowerBound.cplexLogging=0

adversarialProblem.timeLimit=600
adversarialProblem.cplexLogging=0
adversarialProblem.epsilon=0.01

evaluationProblem.timeLimit=600
evaluationProblem.cplexLogging=0
evaluationProblem.epsilon=0.01

incrementalProblem.cplexLogging=0
```

```
minimumAssignmentProblem.cplexLogging=0

minimumKnapsackProblem.cplexLogging=0

recoverableProblem.cplexLogging=0
```