K-Adaptability in Stochastic Optimization

Dissertation
zur Erlangung des akademischen Grades eines
Doktors der Naturwissenschaften
(Dr. rer. nat.)

Der Fakultät für Mathematik der
Technischen Universität Dortmund
vorgelegt von

Jonas Prünte

am 19. Mai 2020

Dissertation

K-Adaptability in Stochastic Optimization

Fakultät für Mathematik
Technische Universität Dortmund

Erstgutachter: Prof. Dr. Christoph Buchheim

Zweitgutachter: Prof. Dr. Arie Koster

Tag der mündlichen Prüfung: 11.09.2020

Acknowledgement

This work has been supported by the German Research Foundation within the Research Training Group 1855. Special thanks go to my supervisor Christoph Buchheim for his support and advice that was elementary for this thesis. Additionally, I want to thank Enrico Malaguti and Michele Monaci for the fruitful discussions we had in my period in Bologna, which led to very interesting results presented in this thesis. At last I want to thank my family and my girlfriend Federica Bomboi, who support me every day in my private life and make my life better.

Abstract

In this thesis we investigate a new strategy for addressing stochastic optimization problems. Stochastic optimization deals with problems that are affected by uncertainty. For e.g. a Vehicle Routing Problem, in which customers have to be served by trucks, the travel times are influenced by the current traffic situation. It is assumed that the distribution of the parameters affected by the uncertainty is known and the goal is to optimize the expected value of a given function. We introduce the new Min-E-Min approach, in which not only one solution is computed, but K different solutions. Our approach is based on the assumption that in many real-world optimization problems, the realization of the uncertain parameters can be observed before the solution is applied and therefore the best solution among the K precomputed solutions can be chosen. On the other hand, obtaining a solution in real time is often not possible due to a lack of time and therefore precomputation of solutions is essential. The classical stochastic optimization problem is the special case of the Min-E-Min Problem where K is equal to one, and hence the solution value of the Min-E-Min Problem is at least as good as the one of the classical stochastic optimization problem. The Min-E-Min methodology can be applied to every optimization problem e.g. the Vehicle Routing Problem or the Knapsack Problem. We call the certain sub problem inside the Min-E-Min Problem the underlying problem.

We investigate the complexity of the Min-E-Min Problem for discrete distributions by analyzing NP-hardness, parameterized complexity, and approximation properties in detail. We show that if K is part of the input, the problem is NP-hard and W[2]-hard parameterized by K, even if the set of feasible solutions of the underlying problem is polynomial in the input size. For fixed K, we show NP-hardness if K is at least two. Additionally, we show that there is no polynomial time constant factor approximation algorithm for the Min-E-Min Problem, unless P=NP. On the other hand, we prove that an approximation guarantee of an algorithm used for solving the underlying problem can be preserved if the problem is solved by an oracle based algorithm. We further demonstrate that for the Min-E-Min Problem with a continuous distribution even the evaluation of the objective function is #P-hard. Nevertheless, we

show that these problems can be solved approximately by using a sufficiently large set of samples following the same distribution.
In addition to the complexity results, we propose different exact algorithms for solving the Min-E-Min Problem including a Branch-and-price algorithm. In an extensive computational study, we conclude that the different algorithms perform differently according to the underlying problem and to the parameters of the instances, and that in different settings a different algorithm might be the best choice. We develop a basic heuristic and different variants of it and demonstrate that they can solve the Min-E-Min Problem fast and precisely. At last, we consider some problems that are related to the Min-E-Min Problem and investigate their complexity.

Zusammenfassung

In dieser Dissertation untersuchen wir eine neue Strategie, um mit stochastischen Optimierungsproblemen umzugehen. Die stochastische Optimierung beschäftigt sich mit Problemen mit unsicheren Parametern. Unter der Annahme, dass die Verteilung dieser unsicheren Parameter bekannt ist, soll der Erwartungswert der Zielfunktion optimiert werden. Wir führen den Min-E-Min-Ansatz ein, bei dem anstatt einer Lösung K Lösungen berechnet werden. Nachdem die Realisierung der unsicheren Parameter beobachtet worden ist, wird die für die Realisierung beste Lösung ausgewählt.

Wir untersuchen die Komplexität des Min-E-Min-Problems, indem wir NP-Schwere, parametrisierte Komplexität und Approximationseigenschaften detailliert analysieren. Wir zeigen, dass das Problem NP-schwer und W[2]-schwer ist, im Fall, dass K Teil des Inputs ist, selbst wenn die Menge der zulässigen Lösungen polynomiell von der Eingabegröße abhängt. Für festes K bleibt das Problem NP-schwer, wenn K größer als zwei ist. Zusätzlich beweisen wir, dass die Existenz eines polynomiellen Algorithmus mit einer konstanten Approximationsgüte für das Min-E-Min-Problem impliziert, dass P und NP gleich sind. Für das Min-E-Min-Problem mit kontinuierlicher Verteilung zeigen wir, dass allein das Evaluieren der Zielfunktion #P-schwer ist. Nichtsdestotrotz beweisen wir, dass man solche Probleme lösen kann, indem man die kontinuierliche Verteilung mit einer ausreichend großen Anzahl an Stichproben, welche derselben Verteilung folgen, diskretisiert.

Neben den Komplexitätsresultaten stellen wir auch verschiedene exakte Algorithmen, unter anderem einen Branch-and-price-Algorithmus, zur Lösung des Problems vor. In einer ausführlichen experimentellen Untersuchung können wir belegen, dass je nach Situation jeder der vorgestellten Algorithmen am sinnvollsten sein kann. Darüber hinaus stellen wir verschiedene Varianten einer von uns entwickelten Heuristik vor und demonstrieren, dass diese das Min-E-Min-Problem schnell und genau lösen können. Zu guter Letzt betrachten wir auch verwandte Probleme und untersuchen ihre Komplexität.

Contents

Chapter 1

Introduction

Consider a general combinatorial optimization problem of the form

$$\begin{aligned} \min \quad & \xi^\top x \\ \text{s.t.} \quad & x \in X(\theta) \ , \end{aligned} \tag{P}$$

where $X(\theta)$ is a compact subset of $\mathbb{R}^n$ and describes the set of feasible solutions and $\xi \in \mathbb{R}^n$ defines a linear objective function (when investigating computational complexity, we will assume $\xi \in \mathbb{Q}^n$). In the following we will denote Problem (P) also as the underlying problem. In practice, the vector ξ in Problem (P) is often unknown or uncertain. As an example, consider the Shortest Path Problem in a road network. In this case, the time needed to traverse an edge highly depends on the traffic situation, which is not known exactly in advance. The uncertainty can also arise in the constraints e.g. in a Maximum Flow Problem with uncertain lower and upper bounds on the edges. To model the uncertainty, we define a set of scenarios L. Every scenario i consists of an objective vector ξ_i, a vector of parameters θ_i that influences the feasible region $X(\theta)$ and a probability p_i that this scenario materializes. The structure of the feasible set remains the same in all scenarios, only some parameters defining the feasible set are affected by the uncertainty. To clarify this, let us consider a Maximum Flow Problem. In a Maximum Flow Problem the structure of the feasible set is defined by a graph, a source vertex and a sink vertex and this structure remains the same in all scenarios. The uncertainty affects the costs of the arcs via ξ and the capacities of the arcs via θ. Note that also uncertainty affecting the structure of the network e.g. a failure of an edge in a Shortest Path Problem can be modeled with this approach by introducing

a sufficiently high cost ξ_i in the dimension corresponding to an arc failing in scenario i. In this study we are interested in minimizing the expected value of the objective function over all instances. This leads to the following objective function:

$$E_{\xi,\theta}(\min_{x\in X(\theta)} \xi^\top x), \tag{Em}$$

where the objective vectors ξ and the vector of parameters θ are considered as random variables.

Suppose that there is an oracle for Problem (P) that gets the set of scenarios and the structure of the problem $X(\theta)$ as an input and returns the optimal solution for Problem (P). This oracle can be e.g. an algorithm that solves a Maximum Flow Problem and gets as input the costs of the arcs, the capacities of the arcs, the graph, the source node and the sink node. In the case of a discrete set of scenarios L, which are explicitly given as a list, the value (Em) can be computed in oracle-polynomial time because all of the scenarios induce optimization problems that can be solved separately in oracle-polynomial time and afterwards the expected value can be computed in linear time. If the set of scenarios L is not discrete, the evaluation of Problem (Em) in general becomes #P-hard for $K \geq 2$, as we will show in Section 3.3, and therefore no oracle-polynomial algorithms for computing the objective value can be found, unless P=NP.

The aim of this thesis is to extend Problem (Em) by integrating the K-adaptability approach: instead of one single solution, it is allowed to compute a set of up to K feasible solutions $\mathcal{X}_K$ such that the best one of them, determined separately in each scenario, is optimal in expectation. It is not necessary that every solution in $\mathcal{X}_K$ is feasible in every scenario. We only require that a solution that is chosen for a scenario is feasible in this scenario. The resulting problem is the min-E-min Problem:

$$\begin{aligned} \min \quad & E_{\xi,\theta}(\min_{x\in X(\theta)\cap\mathcal{X}_K} \xi^\top x) \\ \text{s.t.} \quad & |\mathcal{X}_K| \leq K \\ & \mathcal{X}_K \subset \mathbb{R}^n \, . \end{aligned} \tag{mEm}$$

If there is a scenario with no feasible solution among the K selected solutions, we assume that the minimum corresponding to this scenario takes the value ∞.

The evaluation of the objective function for a given solution for this problem is already #P-hard (see again Section 3.3) and therefore we focus here on the version of the problem where L is a discrete set with $l := |L|$ and is given explicitly as the input of the problem together with the probabilities $p_i > 0$, the objective vectors ξ_i and the parameters θ_i. Moreover, the feasible set is given by explicit constraints, where the coefficients depend on θ_i. Because the probabilities p_i are greater than zero and because $X(\theta)$ is bounded, a scenario without feasible solution lets the objective value of the whole problem become ∞. To simplify the notation, we denote $X(\theta_i)$ by X_i and integrate the probabilities p_i into the objective vectors ξ_i. Finally, we define the Discrete min-E-min Problem as:

$$\begin{aligned} \min \quad & \sum_{i=1}^{l} \min_{x \in X_i \cap \mathcal{X}_K} \xi_i^\top x \\ \text{s.t.} \quad & |\mathcal{X}_K| \leq K \\ & \mathcal{X}_K \subset \mathbb{R}^n \ . \end{aligned} \tag{dmEm}$$

Applications of our approach arise whenever the scenario materializes before a decision has to be taken, but by lack of time or flexibility we cannot solve Problem (P) for this scenario from scratch. In a practical context, decisions often have to be implemented by human users, e.g., truck drivers, in this case it is preferable to provide solutions from a small pool of candidate solutions instead of producing an entirely new solution for every scenario [6, 35]. The latter may confuse the human user and, in the worst case, he/she will not accept the solution. Another application for our problem is disaster management, where the user has to prepare escape plans for an evacuation that have to be trained by the staff [20]. It is clear that only training a small number of such plans is realistic. In other cases, the alternative solutions may have to be prepared physically, e.g., by establishing links in a network, so that a small set of candidate solutions is again preferable. Last but not least, the underlying Problem (P) may just be too hard computationally to be solved from scratch in real-time after the scenario is known.

Optimization problems that involve uncertainty have been widely studied in the literature. The two most used approaches for dealing with uncertainty are Robust Optimization [1] and Stochastic Optimization [4]. In Robust Optimization the goal is to find a solution that is always feasible and optimal in the worst-case. This approach leads to very conservative solutions [3]. This phenomenon is often called the "Price of Robustness". The aim of Stochastic Optimization is to optimize the expected value. Some variants of Stochastic Optimization deal with chance constraints [9], which means that the solution does not have to be feasible in all of the scenarios but in a fixed percentage of them. Another difference between the two approaches is that Stochastic Optimization requires knowledge about the probabilities in form of a probability distribution, whereas for Robust Optimization it is in general sufficient to know the relevant scenarios without understanding the probabilities. The advantage of Stochastic Optimization is clearly that in average the solution value is better and therefore it is a more natural approach if a solution has to be used several times. On the other hand, if it is required that the solution value does not exceed a certain threshold, Robust Optimization is the right approach. A good example for that is the problem of deciding when to start to go to the airport before taking a flight. For most of the passengers it is more important to be able to catch the flight in all possible scenarios than spending in average less time waiting because of a too early arrival. Therefore optimizing the worst case instead of the average case is the suitable choice for that problem. On the other hand stochastic optimization is favorable if the same optimization problem has to be solved many times. Consider a company that delivers to the same customers every day. The travel time of the delivery vehicles, which influences the costs of the company, has to be minimized, but in practice it is affected by uncertainty. In this application, it is more important for the company to have low accumulated costs over all days instead of minimizing the maximum cost on one day.

The K-adaptability approach mentioned above was first presented by Bertsimas and Caramanis [2] for Robust Optimization. The difference to Prob-

lem (dmEm) is that instead of optimizing the expected value the worst case is used. This approach is sometimes called min-max-min robustness, it has been further investigated by Hanasusanto et al., who used the K-adaptability approach for approximating two-stage robust binary problems [20] and two-stage distributionally robust programs with binary recourse decisions [21]. Buchheim and Kurtz introduced K-adaptability to robust problems without first stage, investigated the complexity for convex uncertainty sets and proposed an exact oracle-based algorithm for solving their problem [7]. They also presented complexity results for the discrete min-max-min problem for several underlying, combinatorial problems [6]. Subramanyam et al. [35] proposed a Branch-and-bound algorithm for solving K-adaptability problems in two-stage mixed-integer robust optimization. Chassein et al. [10] investigated the min-max-min problem for budgeted uncertainty sets. Even though Problem (dmEm) and the min-max-min Problem seem to be very similar, there are major differences in the complexity and especially in the methods. Nevertheless, some of our complexity results can be transferred to the min-max-min Problem to prove results that have not been present in the literature. The connection of Problem (dmEm) to the min-max-min Problem is discussed in Chapter 3.2.4.

Problem (dmEm) was studied in my own original research papers Buchheim and Prünte [8] and Malaguti, Monaci and Prünte [29] and the results of this thesis have partially been published in these papers. In Buchheim and Prünte [8] we focused on the variant, in which the uncertainty only occurs in the objective function ($X_i = X$ for all $i \in L$). For this setting we investigated the complexity in terms of NP-hardness, approximability and parameterized complexity with mostly negative results. Only in the special case, where $l - K$ is fixed, we presented an oracle-polynomial algorithm. Besides that, we examined different exact algorithms and a heuristic and compared them in a computational evaluation. These algorithms can be found in Section 4.1 to 4.3 and in Chapter 5. In Malaguti, Monaci et Prünte [29] we focused on the variant presented here as Problem (dmEm) and answered open complexity questions. Additionally we described new algorithms for solving Problem (dmEm). We

developed a Branch-and-price technique that outperforms the other algorithms for Problem (dmEm) in most of the settings (see Section 4.5) and improved the heuristic with unsupervised machine learning techniques (see Section 5.2) and refinement strategies (see Section 5.4). Apart from that, to the best of our knowledge, there are no other studies that investigate Problem (dmEm).

The main contribution of this thesis is the definition of a new problem, Problem (mEm), and its extensive investigation. We show that the problem is NP-hard and W[2]-hard and that there is no constant factor approximation algorithm for it. Additionally, we prove that even the evaluation of a fixed solution is #P-hard if the uncertainty follows a continuous distribution. Linked to that we demonstrate that by replacing a continuous distribution by a discrete set of samples that follow the same distribution (and under some further weak assumptions) we can create a discrete version of (mEm) with a solution that approximates the solution of the original problem if the number of samples is sufficiently large. Another positive complexity result that we show is that an approximation guarantee of an algorithm that is used for the underlying problem can be preserved for the (mEm) Problem. Our tests suggest that in the average case the approximation factor is even better for Problem (mEm) than for the underlying problem. Another important contribution of this thesis is the investigation of different exact and heuristic methods for solving Problem (mEm) and a detailed computational study, in which the effectiveness of each algorithm in various different settings is tested. Besides that we introduce two new problems that are strongly related to Problem (mEm), namely the Min-E-Min Completion Problem and the Stochastic Two-Stage Infrastructure Problem, and examine their complexity. We also transfer some of our complexity results to the min-max-min Problem, resulting in new insights about this problem that have not been gained in the literature before.

In the next chapter, we will explain different concepts that are used in this study and are beneficial for the understanding of the following chapters. Among other topics we introduce fpt algorithms, the W-hierarchy, and #P-

hardness. Chapter 3 is about the complexity of the min-E-min Problem and is divided into two parts. Section 3.2 is the main part of this chapter and we focus on Problem (dmEm) and explore the NP-hardness, the hardness of approximability and the parameterized complexity of the problem. Section 3.3 serves to understand why we do not consider further the continuous version of the min-E-min Problem by showing that even evaluating a fixed solution is #P-hard and that approximating the distribution with a discrete set of scenarios is a reasonable approach. Exact algorithms are presented in Chapter 4 and heuristics in Chapter 5. An extensive computational study of all of the presented algorithms can be found in Chapter 6, in which we investigate which algorithms have the best performance for which underlying problems and in which settings. We present and discuss other variants of the min-E-min Problem and connections to other problems from the literature in Section 7. Chapter 8 concludes.

Chapter 2

Preliminaries

2.1 Notation

In the following, we denote the set of base vectors of $\mathbb{Q}^n$ by $\{\mathbf{e}_d | 1 \leq d \leq n\}$, where $\mathbf{e}_d$ is a vector consisting of a one in dimension d and zeros in all other dimensions. We denote the n-dimensional vector consisting of n zeros by $\mathbf{0}_n$ and consisting of n ones by $\mathbf{1}_n$.

2.2 Complexity Concepts

This section serves as a recapitulation of well-known complexity concepts and an introduction for lesser-known ones. We start with the classical NP-hardness. Afterwards, we we review the terms about approximation that are relevant for this thesis. We also discuss parameterized complexity, in which the impact of a selected parameter on the running time is determined. We finish this chapter by discussing the complexity of counting problems.

2.2.1 NP-Hardness

In this section, we shortly present the basic information about NP-hardness. Therefore, we start by defining a decision problem.

Definition 2.1. A *decision problem* $\mathcal{P} = (X, Y)$ consists of a set X and a subset $Y \subseteq X$. We call X the set of instances and Y the set of yes-instances.

For a given instance $x \in X$, the decision problem is to determine, whether it is a yes-instance (i.e. $x \in Y$) or a no-instance (i.e. $x \in X \setminus Y$).

An important example of a decision problem is the Boolean Satisfiability Problem.

Definition 2.2. A *boolean variable* is a variable that can take only the values zero (*false*) and one (*true*). A *boolean formula* $f : \{0,1\}^n \to \{0,1\}$ consists of n boolean variables that are linked with the following operations:

- $\wedge$ (*and-operator*) : an operator that returns the minimum value of two boolean variables
- $\vee$ (*or-operator*) : an operator that returns the maximum value of two boolean variables
- $\neg$ (*negation-operator*) : an operator that returns one for false variables and zero for true variables.

A *literal* is a variable or a negated variable. A *conjunction* is a set of literals that are connected with the and-operator. Similarly, a *disjunction* is a set of literals connected with the or-operator. A formula is in *conjunctive normal form* if it consists of a conjunction of disjunctions and in *disjunctive normal form* if it consists of a disjunction of conjunctions. A formula is *satisfied* for an assignment of variables, if the result of the formula is one. The *Boolean Satisfiability Problem (SAT)* is to decide for a given boolean formula f if there exists an assignment of the variables such that f is satisfied. In the following we will assume that the given formula is in conjunctive normal form.

Decision problems can be categorized into complexity classes based on their tractability properties. An important question for every specific decision problem is whether it can be decided in polynomial time.

Definition 2.3. The class P consists of all problems $\mathcal{P} = (X, Y)$ for which we can decide for every instance $x \in X$ in polynomial time if it is a yes-instance or a no-instance.

Another desirable quality that a decision problem can possess is if every proof that an instance of this problem is a yes-instance can be verified in polynomial time.

Definition 2.4. Let $\mathcal{P} = (X, Y)$ be a decision problem and $x \in X$ be an instance of it. A *certificate* for x is a proof that $x \in Y$ holds.

For an instance of SAT, which is a boolean formula f, a certificate c is an assignment of the variables of f such that the formula is satisfied.

Definition 2.5. The class *NP* consists of all problems $\mathcal{P} = (X, Y)$ for which we can verify every certificate for every instance $x \in X$ in polynomial time.

Since all problems in P can be decided in polynomial time, no certificate is needed. Therefore, we have

$$\mathrm{P} \subseteq \mathrm{NP}.$$

The question whether there exists a problem in NP that is not in P, is one of the most important unsolved problems in the field of mathematics.

Definition 2.6. Let $\mathcal{P}_1 = (X_1, Y_1)$ and $\mathcal{P}_2 = (X_2, Y_2)$ be two decision problems. An algorithm for deciding every instance $x_1 \in X_1$ by applying an oracle that can decide every instance $x_2 \in X_2$ is called a *Turing reduction*.

A Turing reduction that can be executed in polynomial time assuming that every single use of the oracle is possible in constant time is called a *Cook reduction*. If there is a Cook reduction from a decision problem $\mathcal{P}_1 = (X_1, Y_1)$ to a decision problem $\mathcal{P}_2 = (X_2, Y_2)$ we write

$$\mathcal{P}_1 \leq_T^P \mathcal{P}_2.$$

A special case of the Turing reductions that is used more frequently is the many-one reduction.

Definition 2.7. Let $\mathcal{P}_1 = (X_1, Y_1)$ and $\mathcal{P}_2 = (X_2, Y_2)$ be two decision problems. The function $f : X_1 \to X_2$ is called a *many-one reduction* if the following condition holds:

$$x \in Y_1 \iff f(x) \in Y_2 \ \forall \ x \in X_1.$$

A many-one reduction that can be executed in polynomial time is called a *Karp reduction*. If there exists a Karp reduction from a decision problem $\mathcal{P}_1 = (X_1, Y_1)$ to a decision problem $\mathcal{P}_2 = (X_2, Y_2)$ we write

$$\mathcal{P}_1 \leq_K \mathcal{P}_2.$$

Definition 2.8. A decision problem $\mathcal{P}_1 = (X_1, Y_1)$ is called *NP-hard* if for every problem $\mathcal{P}_2 = (X_2, Y_2) \in \text{NP}$ we have

$$\mathcal{P}_2 \leq_T^P \mathcal{P}_1.$$

If for two decision problems $\mathcal{P}_1 = (X_1, Y_1)$ and $\mathcal{P}_2 = (X_2, Y_2)$

$$\mathcal{P}_2 \leq_T^P \mathcal{P}_1$$

holds, we can draw the following conclusions:

1. If P_1 can be decided in polynomial time, we can decide P_2 in polynomial time.

2. If P_2 is NP-hard, also P_1 is NP-hard.

A decision problem that is in NP and NP-hard is called *NP-complete*. In 1971, Cook [12] found a polynomial time reduction from the computation of a general non-deterministic Turing machine to SAT, which implies that SAT is NP-hard. Building on that, many other decision problems were proven to be NP-hard by reduction from SAT or another NP-hard problem.

2.2.2 Approximation

The topic of this section is approximation. Before we introduce this term, we have to define an optimization problem.

Definition 2.9. An *optimization problem* $O = (X, F(x), c(x, y), goal)$ is a quadruple consisting of the following parts:

1. A set of instances X.

2. A set of feasible solutions $F(x)$ for every instance $x \in X$.

3. A function $c : X \times F(x) \to \mathbb{R}$.

4. An optimization goal $goal \in \{\min, \max\}$.

The *optimal solution value* of O for an instance x is defined by

$$OPT(x) = goal\{c(x, y) \mid y \in F(x)\}.$$

Now we can define an optimization algorithm.

Definition 2.10. Let $O = (X, F(x), c(x, y), goal)$ be an optimization problem. An *optimization algorithm* $\mathcal{A}$ computes for every instance $x \in X$ with $F(x) \neq \emptyset$ a solution $y \in F(x)$ with solution value $\mathcal{A}(x) := goal\{c(x, y)\}$.

A special class of algorithms are approximation algorithms.

Definition 2.11. Let $O = (X, F(x), c(x, y), goal)$ be an optimization problem. An α-*approximation algorithm* $\mathcal{A}$ is an algorithm with polynomial running time that computes for every instance x and for $\alpha \geq 1$ a solution with

$$\mathcal{A}(x) \leq \alpha\, OPT(x), \text{ if } goal = \min$$

$$\mathcal{A}(x) \geq \frac{1}{\alpha}\, OPT(x), \text{ if } goal = \max.$$

An α-approximation algorithm with $\alpha = 1$ is called *exact algorithm* and its solution is called *optimal solution.*

A class of problems that will occur in this thesis is APX, the class of problems with a constant factor approximation algorithm.

Definition 2.12. The complexity class APX consists of all optimization problems that admit α-approximation algorithms with $\alpha \in \mathcal{O}(1)$. These algorithms are called *constant factor approximation algorithms.*

2.2.3 Parameterized Complexity

In this section we deal mostly with NP-hard problems. The parameterized complexity theory deals with the question which parameters contribute more than others to the complexity of problems. The goal is to identify a parameter that is alone responsible for an exponential running time of an algorithm or prove that for some problems an algorithm whose exponential part of the running time depends only on one parameter cannot exist under some reasonable assumptions. If such an algorithm exists, instance, in which the identified parameter is small can be solved in reasonable time.

Definition 2.13. Let $\mathcal{P}$ be a decision or optimization problem and let X be the set of instances of $\mathcal{P}$. We call a function $k : X \to \mathbb{R}$ a *parameter.*

A parameter can be e.g. the number of vertices in a graph problem or the number of variables in SAT. These numbers are clearly dependent on the given instance of the problem.

Definition 2.14. Let $\mathcal{P}$ be a decision or optimization problem, let X be the set of instances of $\mathcal{P}$ and $k(x)$ be a parameter. We call the tuple $(\mathcal{P}, k(x))$ a *parameterized problem*. If we want to specify the parameter, we refer to $(\mathcal{P}, k(x))$ as $\mathcal{P}$ parameterized by $k(x)$.

Now we have everything we need to define the algorithms indicated in the beginning of the section.

Definition 2.15. A *fixed-parameter tractable (fpt) algorithm* for a parameterized problem $(\mathcal{P}, k(x))$ is an exact algorithm that has a running time of $\mathcal{O}(p(|x|)f(k(x)))$ for every instance $x \in X$ of $\mathcal{P}$, where $p()$ is a polynomial, $f()$ is a computable function and $|x|$ is the input length of x.

The set of problems that can be solved with an fpt algorithm form a complexity class.

Definition 2.16. The class *FPT* consists of all parameterized problems that admit an fpt algorithm.

The problem SAT restricted to formulas in conjunctive normal form parameterized by the number of variables p is in FPT since it can be solved by a brute-force algorithm with running time in $\mathcal{O}(2^p m)$, where m is the number of disjunctions.

Definition 2.17. Let $(\mathcal{P}_1 = (X_1, Y_1), k_1(x))$ and $(\mathcal{P}_2 = (X_2, Y_2), k_2(x))$ be two parameterized decision problems. A function $R : X_1 \to X_2$ is called *fpt many-one reduction* if the following conditions are fulfilled for every instance $x \in X_1$:

1. $x \in Y_1 \iff R(x) \in Y_2$.
2. $R(x)$ is computable in $\mathcal{O}(p(|x|)f(k_1(x))$, where $p()$ is a polynomial, $f()$ is a computable function and $|x|$ is the input length of x.
3. $k_2(R(x)) \leq g(k(x))$, where $g()$ is a computable function.

In the following we refer to fpt many-one reductions by *fpt reductions* or *parameterized reductions*.

Definition 2.18. A *boolean circuit* is an acyclic, directed graph, in which:

- one node with out-degree 0 is labeled as output node.
- every other node with in-degree 0 is labeled an input node or a boolean constant.
- every other node with in-degree 1 is labeled as a negation node.
- every other node is labeled as and node or as or node.

A boolean circuit can be interpreted as a boolean formula, where the input nodes are the variables and the output node takes value one if the formula is true and zero otherwise.

Definition 2.19. We call a boolean circuit $\mathcal{C}$ *k-satisfiable* if there exists a truth assignment that sets exactly k variables to one. The parameterized problem *Weighted Circuit Satisfiability (WCS)* is to decide for a given circuit, whether it is k-satisfiable, where k is the parameter under consideration.

Definition 2.20. Let C be a class of circuits. WCS restricted to circuits in C is called *WCS[C]*.

We want to define some features of boolean circuits that help us to define different classes among them.

Definition 2.21. The *depth* of a circuit is the maximum length of a path from an input node to the output node. A *small node* is a node with in-degree lower or equal to two and a *large node* is a node with in-degree greater than two. The *weft* of a circuit is the maximum number of large nodes on a path from an input node to the output node. The class of circuits with weft at most t and depth at most d is called $C_{t,d}$. A parameterized problem $(\mathcal{P}, k(x))$ belongs to *W[t]* if there is a parameterized reduction from $(\mathcal{P}, k(x))$ to WCS$[C_{t,d}]$ for some fixed $d \geq 1$.

Note that every large node can be replaced by a set of small nodes. This operation possibly enlarges the depth of the circuit. If d is fixed, only large

nodes with a constant in-degree can become small nodes. Therefore, only nodes with an in-degree that depends on the input size are always large nodes and hence the definition of large nodes is equivalent if the two in the definition is replaced by any other constant that is larger than one.

Definition 2.22. The complexity class *W[P]* consists of all parameterized problems $(\mathcal{P}, k(x))$ that can be solved by an algorithm that is allowed to guess $k(x)$ elements of the solution nondeterministically and afterwards deterministically verify that the solution is feasible.

Corollary 2.23. *The following statements are true:*

- *Every problem in P, the class of problems that can be solved in polynomial time, parameterized by an arbitrary parameter, is in FPT.*
- $FPT = W[0] \subseteq W[1] \subseteq \ldots W[t] \subseteq W[t+1] \subseteq \ldots W[P]$.
- *Every parameterized problem in $W[P]$ without the parameter is in NP.*

Since every polynomial time algorithm is also an fpt algorithm, the first statement follows. The proof of FPT = W[0] can be found in [16]. From the definition of the class W[t] it follows that W[t+1] contains W[t]. WCS$[C_{t,d}]$ can be solved by nondeterministically guessing k elements of the set of variables that will be assigned with value one and then verifying the result and hence W[P] contains W[t] for every t. The class NP can be seen as the class of problems that can be solved by an algorithm that is allowed to guess every element of the solution nondeterministically and afterwards deterministically verify that the solution is feasible. Therefore, the third statement follows. If one of the relations between the classes in the second statement can be proven to be strict, we can conclude that P$\neq$NP.

Definition 2.24. A parameterized problem $(\mathcal{P}, k(x))$ is *W[t]-hard* if we can reduce every parameterized problem in W[t] to $(\mathcal{P}, k(x))$ by an fpt reduction.

Now we want to identify which problems are W[t]-hard. For this purpose, we need to define some terms.

Definition 2.25. We want to define classes of boolean formulas recursively:

- $\Gamma_{0,d} := \{\lambda_1 \wedge \cdots \wedge \lambda_c \mid c \in [0, \ldots, d], \lambda_1, \ldots, \lambda_c \text{ are literals}\}$.

- $\Delta_{0,d} := \{\lambda_1 \vee \cdots \vee \lambda_c \mid c \in [0, \ldots, d], \lambda_1, \ldots, \lambda_c \text{ are literals}\}$.
- $\Gamma_{t+1,d} := \{\bigwedge_{i \in I} \delta_i \mid I \text{ is a finite nonempty index set, } \delta_i \in \Delta_{t,d} \; \forall \; i \in I\}$.
- $\Delta_{t+1,d} := \{\bigvee_{i \in I} \gamma_i \mid I \text{ is a finite nonempty index set, } \gamma_i \in \Gamma_{t,d} \; \forall \; i \in I\}$.

With this definition we can see that $\Gamma_{1,d}$ is the set of formulas in d-disjunctive normal form, which means that every conjunction contains at most d literals. Similarly, $\Delta_{1,d}$ is the set of formulas in d-conjunctive normal form, which means that every disjunction contains at most d literals. The set $\Gamma_{2,1}$ is the set of all formulas in disjunctive normal form and $\Delta_{2,1}$ the set of all formulas in conjunctive normal form.

Definition 2.26. We call a boolean formula f *k-satisfiable* if there exists a truth assignment that sets exactly k variables to one. The parameterized problem *Weighted Satisfiability (WSAT)* is to decide for a given formula, whether it is k-satisfiable.

Definition 2.27. Let F be a class of boolean formulas. WSAT restricted to formulas in F is called *WSAT[F]*.

Corollary 2.28. *WSAT[$\Delta_{t+1,d}$] is W[t]-hard for $d \geq 1$ and $t > 0$.*

The proof of this corollary, which is far too long for this thesis, can be found in [16]. Building on that, one can prove that the Clique Problem parameterized by the size of the clique is W[1]-hard and that the Dominating Set Problem parameterized by the size of the dominating set is W[2]-hard, making them the most famous members of these classes. Afterwards, many other problems were proven to be W[1]-hard or W[2]-hard using these problems for reductions.

2.2.4 Complexity of Counting Problems

In this section we introduce counting problems and the important complexity class #P.

Definition 2.29. Let $\mathcal{P}(X, Y)$ be a decision problem. For every instance $x \in X$ we define the *count* of x $count_{\mathcal{P}}(x)$ as the number of certificates for x.

The count of an instance x of SAT is the number of variable assignments that satisfy the given boolean formula.

Definition 2.30. Let $\mathcal{P}(X, Y)$ be a decision problem. For every instance $x \in X$, the corresponding *counting problem* $\#\mathcal{P}$ is to compute $count_{\mathcal{P}}(x)$.

The counting problem corresponding to SAT is #SAT.

Definition 2.31. The counting problem *#SAT* is to compute the number of truth assignments for a given boolean formula f.

Now we can define the most important complexity class for counting problems.

Definition 2.32. The class $\#P$ consists of all counting problems, whose corresponding decision problems are in NP.

Similar to the reductions that we have seen before, one can define reductions that can be used for counting problems.

Definition 2.33. Let $\mathcal{P}_1 = (X_1, Y_1)$ and $\mathcal{P}_2 = (X_2, Y_2)$ be two decision problems. A *polynomial time many-one counting reduction* consists of two functions $f : X_1 \to X_2$ and $g : \mathbb{Z}_+ \to \mathbb{Z}_+$ with the following condition:

$$count_{\mathcal{P}_1}(x) = g(count_{\mathcal{P}_2}(f(x))) \; \forall \; x \in X_1.$$

In the following we refer to a polynomial time many-one counting reduction by *counting reduction*. A special case of counting reductions are parsimonious reductions.

Definition 2.34. A *parsimonious reduction* is a counting reduction, in which the function g is the identity function.

Definition 2.35. Let $\mathcal{P}_1 = (X_1, Y_1)$ be a decision problem. We call $\mathcal{P}_1$ *#P-hard* if for every problem $\mathcal{P}_2 = (X_2, Y_2) \in \#\mathrm{P}$ there exists a counting reduction from $\mathcal{P}_2$ to $\mathcal{P}_1$.

A problem that is #P-hard and is a member of #P is called *#P-complete*. The reduction used in the theorem of Cook [12] is parsimonious and therefore it follows that #SAT is #P-hard. Many counting problems with corresponding NP-hard problems are proven to be #P-hard, e.g. the problem #Knapsack [17]. But also problems in P can have a corresponding #P-hard counting

problem like the Perfect Matching Problem [36]. Since every decision problem $\mathcal{P} = (X, Y)$ can be reformulated to the question, whether for a given instance $x \in X$ the $count_{\mathcal{P}}(x)$ is at least one, it follows that by solving the corresponding counting problem $\#\mathcal{P}$ also $\mathcal{P}$ is solved. Therefore, every decision problem that has a corresponding counting problem that can be solved in polynomial time is in P. Note that this does not mean that NP-hardness of a problem $\mathcal{P}$ implies #P-hardness of the corresponding counting problem $\#\mathcal{P}$. Nevertheless, as far as we know, there is no known NP-hard problem with a corresponding counting problem that is not #P-hard [26].

2.3 Mixed Integer Quadratic Programming

In this section we want to present different techniques to deal with Mixed Integer Quadratic Programs. We will see in Chapter 4 that solving such programs is necessary for some of the exact solutions methods for (dmEm).

Definition 2.36. Given the matrices $Q \in \mathbb{R}^{n\times n}, A \in \mathbb{R}^{m\times n}$ and $E \in \mathbb{R}^{p\times m}$ and the vectors $c \in \mathbb{R}^n, b \in \mathbb{R}^m$ and $f \in \mathbb{R}^p$. Then the following program is called a *Quadratic Program*:

$$\begin{aligned} \min \quad & \tfrac{1}{2}x^\top Qx + c^\top x \\ \text{s.t.} \quad & Ax \leq b \\ & Bx = d \\ & x \in \mathbb{R}^n \end{aligned} \tag{QP}$$

If the matrix Q is positive semidefinite, the (IQP) is convex and therefore every local optimum is a global optimum.
The most popular algorithms for solving convex QP's are Active-Set-Methods, the Barrier algorithm or the QP simplex method. We will focus now on the Barrier algorithm because it is the default algorithm for solving QP's in the MILP-solver Cplex, which we will use later for our experiments. The barrier algorithm is an algorithm for general continuous non-linear optimization problems.

Definition 2.37. Let $f : \mathbb{R}^n \to \mathbb{R}$ and $g_i : \mathbb{R}^n \to \mathbb{R}$ for all $1 \leq i \leq m$. The following program is called a *non-linear Program*:

$$\begin{aligned} \min \quad & f(x) \\ \text{s.t.} \quad & g_i(x) \leq 0 \quad \forall\, 1 \leq i \leq m \\ & x \in \mathbb{R}^n \end{aligned} \tag{NLP}$$

In the following we assume that the functions f and g_i are all convex and twice differentiable for all $1 \leq i \leq m$. In the case of QP's these criteria are met. We assume additionally that $\{x \in \mathbb{R}^n : g_i(x) < 0 \ \forall\ 1 \leq i \leq m\}$ is non-empty. The idea of the Barrier algorithm is to solve a family of optimization problems defined on the inside of the feasible region of the NLP, in which solutions that are close to the boundaries of the feasible region are punished. For this purpose, we need a barrier function $b(x)$ that is twice differentiable and that is approaching infinity when x is approaching to the boundary of its feasible region. A commonly used barrier function is the log barrier function

$$b(x) = \sum_{i=1}^{m} -\ln(-g_i(x)).$$

Using a parameter τ, which decreases over time, the influence of the barrier function on the whole objective function can be decreased.

Definition 2.38. The *barrier problem* is defined as follows:

$$\begin{aligned} \min \quad & B_\tau(x) \\ \text{s.t.} \quad & g_i(x) < 0 \quad \forall\, 1 \leq i \leq m, \end{aligned} \tag{BP_τ}$$

where $B_\tau(x) = f(x) + \tau\, b(x)$.

The following corollary shows that the barrier problem can be solved by solving a system of non-linear equations.

Corollary 2.39. *BP_τ has a unique solution x so that*

$$\nabla B_\tau(x) = 0$$

holds, where $\nabla B_\tau(x)$ is the gradient of $B_\tau(x)$.

The corollary shows that (BP_τ) is much easier to solve than (NLP). We can solve it e.g. with the Newton method. Now we can define a basic version of the Barrier algorithm.

Basic Barrier algorithm

1: Set $i = 0$ and choose $\tau_i > 0$.
2: **repeat**
3: Compute solution x of (BP_τ) with $\tau = \tau_i$.
4: Set $i = i + 1$ and choose $0 < \tau_i < \tau_{i+1}$.
5: **until** x is optimal solution of (NLP).
6: **return** x.

The termination condition that x is an optimal solution of NLP is just theoretical. In practice a commonly used condition is the following:

$$m\,\tau_i \leq \epsilon,$$

where $\epsilon > 0$. One can show that this condition implies that x fulfills the Karush-Kuhn-Tucker conditions except for a tolerance of ϵ, which is a sufficient optimality criterion for convex problems. Note that for solving (BP_τ) in Step 3 the solution of the previous iteration is used and hence the conditioning of the Hesse matrix $\nabla^2 B_\tau(x)$, which gets worse if τ decreases, is important. Therefore, it is not preferable to start with a value of τ that is too small.

If some of the variables x are restricted to be integers, the resulting program is called *Mixed Integer Quadratic Program (MIQP)*. We want to discuss the two most relevant strategies for solving $MIQP$'s.
The first strategy is to linearize all products of variables. Let xy be a product of two variables x and y. If one of the variables is binary and the other is bounded and greater or equal to zero, the quadratic terms in the objective function can be handled by introducing an additional variable z that replaces the product xy. Let us assume that y is binary and x is bounded with maximum value $\bar{x}$. By introducing the following linear constraints, one can get rid of the non-linearities:

$$\begin{aligned} z &\geq 0 \\ x &\geq z \\ y\,\bar{x} &\geq z \\ z &\geq x - (1-y)\,\bar{x} \end{aligned} \tag{2.1}$$

If y takes the value zero, the first and the third constraint force z to take the same value and the second and third constraint do not become invalid. If y takes the value one, the second and the fourth constraint ensure that z is equal to x. Note that this linearization works for both continuous and integer variables x. If y is not binary but a bounded integer, binary expansion can be used. The resulting problem is a Mixed Integer Linear Program that can be solved by standard Branch-and-cut methods.

Another way of solving quadratic problems is to use the Branch-and-bound method without linearization. In every node of the Branch-and-bound tree quadratic relaxations are computed by dropping the integrality constraints and solving the resulting QP as discussed above. If the matrix Q is not positive semidefinite and all variables are binary, the problem can be convexified by increasing $Q_{i,i}$ by a value v_i and subtracting v_i from c_i for all $1 \leq i \leq n$, so that the objective function of the original problem remains unchanged. If the values v_i are sufficiently large, the matrix Q will be positive semidefinite after this operation and a global optimum can be found. If some variables are bounded integers, they can be replaced using binary expansion. If some of the variables cannot be replaced by binaries, it might be the case that the operation discussed before is sufficient if it is just used for the binary variables. This depends on the structure of the matrix Q. If this is not the case, relaxations cannot be solved to global optimality. To obtain a global optimal solution the spatial Branch-and-bound method has to be used, in which the feasible set of the problem is subdivided into regions, for which convex relaxations can be computed.

2.4 Column Generation

Column generation is a method for solving linear programs. The idea is to solve the given problem iteratively with a subset of the variables, check after

every solution if it is optimal for the original problem and, if not, add additional variables of the original problem that can possibly improve the objective value. Therefore, it is most effective for problems with a large set of variables.

2.4.1 Basic Algorithm

Let us call the following linear program the *master problem*:

$$\begin{aligned} \min \quad & \sum_{i=1}^{n} c_i x_i & & \\ \text{s.t.} \quad & \sum_{i=1}^{n} a_{j,i} x_i \geq b_j & \forall\ 1 \leq j \leq m & \qquad \text{(MP)} \\ & x_i \geq 0 & \forall\ 1 \leq i \leq n & \end{aligned}$$

If we restrict the set of variables to all variables with an index in

$$I \subseteq \{1, \ldots, n\},$$

the resulting program is called the *restricted master problem*:

$$\begin{aligned} \min \quad & \sum_{i \in I} c_i x_i & & \\ \text{s.t.} \quad & \sum_{i \in I} a_{j,i} x_i \geq b_j & \forall\ 1 \leq j \leq m & \qquad \text{(RMP)} \\ & x_i \geq 0 & \forall\ i \in I & \end{aligned}$$

The dual of the master problem is:

$$\begin{aligned} \max \quad & \sum_{j=1}^{m} b_j y_j & & \\ \text{s.t.} \quad & \sum_{j=1}^{m} a_{j,i} y_j \leq c_i & \forall\ 1 \leq i \leq n & \qquad \text{(DMP)} \\ & y_j \geq 0 & \forall\ 1 \leq j \leq m & \end{aligned}$$

If after adding an additional variable of the original problem a constraint in the dual of the restricted master problem is violated, the primal solution of the restricted master problem may not be optimal for the original problem. The constraint in the dual master problem corresponding to the i-th variable is violated if

$$r_i := c_i - \sum_{j=1}^{m} a_{j,i} y_j < 0$$

holds, where y is the set of dual variables that we received by solving the restricted master problem. We call r_i the *reduced cost* of variable i. If a variable with negative reduce cost is found, it can be added to the restricted master problem to possibly improve the objective value. If no variable with negative reduced cost exists, the objective value cannot be improved and therefore the solution is optimal. Given the dual optimal solution of the restricted master problem y, one can formulate the following *pricing problem* to find variables with negative reduced cost:

$$\begin{aligned} \min \quad & c_i - \sum_{j=1}^{m} a_{j,i} y_j \\ \text{s.t.} \quad & i \in \{1, \ldots, n\} \end{aligned} \tag{PP}$$

With these definitions we can define the basic column generation method, which is illustrated in the following scheme:

Basic Column generation Method

1: Find feasible solution x' and set $I := \{i \mid x'_i \neq 0\}$.
2: **repeat**
3: Solve restricted master problem on I and receive primal solution x and dual solution y.
4: Solve the pricing problem using y and receive index i.
5: **if** $r_i < 0$ **then**
6: Add i to I.
7: **end if**
8: **until** $r_i \geq 0$.
9: **return** x.

The feasible solution in the first line of the algorithm can either be found by using a heuristic or by introducing a dummy variable that ensures feasibility of all constraints but has a sufficiently high cost so that it will not be in the optimal solution. One can improve the basic column generation method by using a heuristic pricing to find new variables and just solve the pricing problem exactly, when the heuristic pricing does not find a variable with negative reduced cost. In the best case, when the heuristic pricing always identifies an existing variable with negative reduced cost, the pricing problem has to be solved just once. Of course, this is just an improvement if the heuristic pricing

is significantly faster than solving the exact pricing problem.
Note that the pricing problem still needs to check all variables, which can be problematic if the number of variables is very high. If columns

$$a_i := \{a_{j,i} \mid 1 \leq j \leq m\}$$

corresponding to a variable are given implicitly by set $\mathcal{A}$, over which one can optimize, and by a function $c(a_i)$ that computes the cost of variable c_i, the pricing problem can be solved without enumeration. In this case it is not even necessary to enumerate all variables. Problems with a high number of variables that seemed to be intractable can often be solved in this way. In this case, we call the pricing problem *implicitly*:

$$\begin{aligned} \min \quad & c(a) - \sum_{j=1}^{m} a_j y_j \\ \text{s.t.} \quad & a \in \mathcal{A} \end{aligned} \tag{IPP}$$

This structure $\mathcal{A}$ is often given in combinatorial optimization problems e.g. the Cutting Stock Problem or the Vehicle Routing Problem. To understand this property better, we want to examine the Vehicle Routing Problem in the following as an example.

2.4.2 Example: Vehicle Routing Problem

Definition 2.40. Given a directed graph $G = (V, A)$, a node $d \in V$, a cost function $c : A \to \mathbb{R}_+$ and an integer k, the *Vehicle Routing Problem (VRP)* asks for a set of at most k cycles that each contain d such that every node $v \in V \setminus \{d\}$ is contained in at least one cycle and such that the sum of the costs of all selected arcs is minimal. We call d *depot*, $V \setminus \{d\}$ the set of *customers* and the k cycles *routes*.

In practice, there can be additional constraints that only depend on each route independently e.g. time window or capacity constraints. We denote the set of feasible routes by $\mathcal{R}$. Now we introduce an IP for solving the VRP:

$$\begin{aligned} \min \quad & \sum_{r \in \mathcal{R}} p_r x_r \\ \text{s.t.} \quad & \sum_{r \in \mathcal{R}} \alpha_{r,v} x_r \geq 1 \quad \forall\, v \in V \setminus \{d\} \\ & x_r \in \{0, 1\} \qquad \forall\, r \in \mathcal{R}, \end{aligned} \tag{IPVRP}$$

where the constant $\alpha_{r,v}$ is the number of times route r contains customer v and p_r is the total cost of route r. The relaxed version of this program resulting from replacing the second set of constraints by

$$x_r \in [0,1] \; \forall \; r \in \mathcal{R}$$

can be solved with column generation. For computing new columns, one has to solve the following pricing problem:

$$\begin{aligned} \min \quad & p_r - \sum_{v \in V \setminus \{d\}} \alpha_{r,v} y_v \\ \text{s.t.} \quad & r \in \mathcal{R}, \end{aligned} \tag{IPPVRP}$$

where y is the given dual solution of the restricted master problem.

Theorem 2.41. *The program IPPVRP can be formulated as a Shortest Path Problem with Resource Constraints.*

Proof. The *Shortest Path Problem with Resource Constraints* consists of finding a shortest path according to a given cost function from one specified node to another specified node in a graph such that given constraints are fulfilled. Let $dest(a) \in V$ be the destination of arc $a \in A$. We define a new cost function

$$c_{sp}(a) := c(a) - y_{dest(a)}.$$

The cost of a path according to c_{sp} in G from the depot to the depot is now equivalent to the term that is minimized in the objective function. We require that all the constraints for a feasible route also have to hold in the Shortest Path Problem with Resource Constraints that we develop and hence we can solve IPPVRP with the corresponding reduction. □

The Shortest Path Problem with Resource Constraints is NP-hard but in general much faster to solve than enumerating all feasible routes. Note that only problems with strong duality can be solved exactly using column generation. Therefore, to solve IPVRP, column generation can only be used to compute relaxations that have to be embedded in a Branch-and-bound algorithm.

2.5 Clustering

In this section, we introduce the k-Clustering Problem, which we use later in Chapter 5. In particular, we present a well-known heuristic solution approach for it.

Definition 2.42. Let $X := \{x_1, \ldots, x_n\}$ be a set of d-dimensional vectors. The goal of the *k-Clustering Problem* is to partition X into k sets $S_1, \ldots, S_k$, which are called *clusters*, such that

$$\min_S \sum_{i=1}^{k} \sum_{x \in S_i} ||x - c_i||^2$$

is minimized, where

$$c_i := \frac{1}{|S_i|} \sum_{x \in S_i} x$$

is called the *center* of cluster i.

The k-Clustering Problem was proven to be NP-complete by Garey et al. [19]. An algorithm for finding local optima was proposed by Lloyd [27]:

k-means algorithm

1: Choose an arbitrary partition $S := \{S_1, \ldots, S_k\}$ of X.
2: **repeat**
3: Set $S' := S$.
4: Compute the center c_i for every cluster $S_i \in S'$.
5: Recompute the clusters by assigning every vector $x \in X$ to its closest center: $S_i = \{x \in X \mid ||x - c_i|| \leq ||x - c_j|| \; \forall \; 1 \leq j \leq k\} \; \forall \; 1 \leq i \leq k$.
6: Remove every x that is in more than one cluster from every cluster except for the lexicographically first cluster that contains x.
7: $S := \{S_1, \ldots, S_k\}$
8: **until** $S = S'$
9: **return** S

The k-means algorithm can be used as a fast heuristic algorithm for solving the k-Clustering Problem.

2.6 Partitions

In a solution of Problem (dmEm), each scenario j is *covered* by at least one of the solutions $\mathcal{X}_K = \{x_1, \ldots, x_K\}$, namely $\operatorname{argmin}\{\xi_j^\top x | x \in X_j \cap X_K\}$. Combining all scenarios that are covered by the same solution into a subset leads to a partition of the set of scenarios into at most K parts. In the following, we will call this the *partition induced by* $\mathcal{X}_K$. If one scenario is covered by more than one solution, we assign it lexicographically. Conversely, if such a partition $\{1, \ldots, l\} = \bigcup_{j=1}^{K} I_j$ is given, one can construct an *induced set of solutions* $x_1, \ldots, x_K$ by choosing

$$x_j \in \operatorname*{argmin}_{x \in \widetilde{X}_j} \sum_{i \in I_j} \xi_i^\top x \text{ with } \widetilde{X}_j := \bigcap_{i \in I_j} X_i$$

and solving problem (P) with objective function $\sum_{i \in I_j} \xi_i^\top x$ and feasible set $\widetilde{X}_j$. If there is more than one optimal solution for a subset of scenarios, we decide to choose the first solution in a lexicographic order with respect to the entries of the vector. Note that in the case of discrete scenarios and a certain feasible region X, the induced set of solutions can be computed by summing up all objective vectors and using the oracle on this aggregated objective vector.
In the case of an uncertain feasible region that is described by θ_i in scenario i this may also be true and an aggregated θ may be obtained, e.g., in a Maximum Flow Problem the capacity on each arc that is given to the oracle is the minimum capacity of this arc in all scenarios inside the regarded subset. For other underlying problems it is not possible to use the oracle, e.g. for a Knapsack Problem a Multidimensional Knapsack Problem has to be solved instead to obtain the set of solutions induced by a given partition.

For a given set of solutions, building the induced partition and then the induced set of solutions does not necessarily lead to the original set of solutions. Also starting with a given partition in general does not produce the original partition again. The idea of computing induced partitions and solution sets alternately gives rise to a heuristic approach to Problem (dmEm), which is discussed in Chapter 5 below.

2.7 Combinatorial Optimization Oracles

In this section we want to introduce problems that we use later for our experimentation in Chapter 6 and present the algorithms we used for solving them.

2.7.1 The Minimum Spanning Tree Problem

We start with a classical problem of combinatorial optimization, the Spanning Tree Problem.

Definition 2.43. A *tree* is an undirected, connected graph that contains no cycles. A *spanning* tree of a graph $G = (V, E)$ is a subgraph that is a tree and contains all vertices of V.

Definition 2.44. Given a graph $G = (V, E)$ and a cost function $c : E \to \mathbb{R}$, the *Spanning Tree Problem (ST)* asks for a spanning tree $T = (V, \bar{E})$ of G with minimum cost $\sum_{e \in \bar{E}} c_e$.

We solved occurring Spanning Tree Problems with the well-known *algorithm of Kruskal* [25].

Algorithm of Kruskal

1: Sort all edges by decreasing cost in list $L := \{e_1, \ldots, e_m\}$.
2: $\bar{E} = \emptyset$, $i = 1$.
3: **repeat**
4: **if** $\bar{E} \cup \{e_i\}$ contains no cycle **then**
5: Add e_i to $\bar{E}$.
6: $i = i + 1$.
7: **end if**
8: **until** $|\bar{E}| = |V| - 1$.
9: **return** $T = (V, \bar{E})$.

The running time of the algorithm of Kruskal is dominated by the sorting and hence is in $\mathcal{O}(|E|log|E|)$. Therefore, the Spanning Tree Problem belongs to the class P.

2.7.2 The Maximum Flow Problem

In this section we introduce the Maximum Flow Problem.

Definition 2.45. Given a directed graph $G = (V, A)$, two vertices $s, t \in V$ and a capacity function $u : A \to \mathbb{R}$, a function $f : A \to \mathbb{R}_+$ is called a *s-t-flow* if the following conditions are met:

1. $f_a \leq u_a \; \forall \; a \in A$.
2. $\sum_{a:=(w,v)\in A} f_a = \sum_{a:=(v,w)\in A} f_a \; \forall \; v \in V \setminus \{s, t\}$

The first set of constraints is called *capacity constraints* and the second set of constraints is called *flow conservation constraints*.

Definition 2.46. Given a directed graph $G = (V, A)$, two vertices $s, t \in V$ and a flow $f : A \to \mathbb{R}$, the *value* of the flow f is defined by

$$\sum_{a:=(s,w)\in A} f_a.$$

Definition 2.47. Given a directed graph $G = (V, A)$, two vertices $s, t \in V$ and a capacity function $u : A \to \mathbb{R}$, the *Maximum Flow Problem (MFP)* consists in finding the flow with the maximum value.

Although there are good combinatorial algorithms for solving the Maximum Flow Problem like the Edmonds-Karp algorithm [15], which has a running time in $\mathcal{O}(|V||A|^2)$, we experienced good results by solving the following linear program instead:

$$\begin{array}{lrcll} \max & \sum\limits_{a:=(s,w)\in A} f_a & & & \\ \text{s.t.} & f_a & \leq & u_a & \forall\ a \in A \\ & \sum\limits_{a:=(w,v)\in A} f_a & = & \sum\limits_{a:=(v,w)\in A} f_a & \forall\ v \in V \setminus \{s,t\} \\ & f_a & \in & \mathbb{R}_+ & \forall\ a \in A \end{array} \tag{LPMFP}$$

2.7.3 Knapsack Problems

Now we focus on Knapsack Problems.

Definition 2.48. Given a set of items $S := \{1, 2, \ldots, n\}$, a cost function $c : S \to \mathbb{R}_+$, m weight functions $a_j : S \to \mathbb{R}_+$ for all $1 \leq j \leq m$ and m positive numbers b_i. The *Boolean Multidimensional Knapsack Problem (MKP)* asks for a set $I \subseteq S$ such that

$$\sum_{i \in I} c_i$$

is maximal and that for all $1 \leq j \leq m$

$$\sum_{i \in I} (a_j)_i \leq b_j$$

holds.

Definition 2.49. The version of MKP with $m = 1$ is called *Boolean Knapsack Problem (1KP)* and the version with $m = 0$ *Unconstrained Binary Optimization Problem (UCB).*

In our experiments we solve MKP and KP with the following IP:

$$\begin{array}{lll} \max & c^\top x & \\ \text{s.t.} & a_j^\top x \leq b_j & \forall\ 1 \leq j \leq m \\ & x \in \{0,1\}^n, & \end{array} \tag{IPMKP}$$

where $x \in \{0,1\}^n$ is a vector that indicates which items are selected. It takes in dimension i the value one if and only if the i-th item is selected. Although there exists a pseudo-polynomial algorithm for (1KP) with running time in $\mathcal{O}(|S|b)$, we made better experiences by solving MKPIP with the MILP-solver Cplex instead. Problem (1KP) and Problem (MKP) with fixed m are known to be weakly NP-hard [17]. If the number of dimensions m is part of the input, Problem (MKP) is strongly NP-hard, which can be seen e.g. by the IP formulation of the Independent Set Problem. Because Problem (UCB) is much easier to solve, we used the following polynomial time algorithm.

Algorithm for (UCB)

```
1: x = 0_n
2: for all i ∈ I do
3:    if c_i ≥ 0 then
4:       x_i = 1.
5:    end if
6: end for
7: return x.
```

2.7.4 The Traveling Salesman Problem

One of the most important combinatorial problems with applications in the field of logistics is the Traveling Salesman Problem. It models the problem of minimizing the total distance covered by a salesman that has to visit n cities and return to his starting point. It can be applied to real-world delivery problems. We can define this problem formally as follows:

Definition 2.50. Given an undirected complete graph $K_n = (V, E)$ and a cost function $c : E \to \mathbb{R}_+$, the *Traveling Salesman Problem (TSP)* consists in finding a connected subgraph $T = (V, \bar{E})$ such that $\sum_{e:=(u,v)\in\bar{E}} c_e$ is minimal and that every vertex $v \in V$ has exactly two incident edges in $\bar{E}$.

The Traveling Salesman Problem is known to be NP-hard and the general version of the problem does not admit a constant factor approximation algorithm [17]. If the cost function c obeys the triangle inequality, the resulting problem is called Metric Traveling Salesman Problem. The Metric Traveling

Salesman Problem can be solved approximately with a guarantee of 1.5 by the *algorithm of Christofides* [11].

Algorithm of Christofides

1: Compute a Minimum Spanning Tree $T = (V, \bar{E}_T)$.
2: Compute the set of vertices $\bar{V}$ with odd degree in T.
3: Compute a Minimum Weighted Perfect Matching $M = (\bar{V}, \bar{E}_M)$ on the graph induced by $\bar{V}$.
4: Compute an Euler Tour U on the graph $H = (V, \bar{E}_T \cup \bar{E}_M)$.
5: Remove repeated vertices in U by introducing short-cuts.
6: **return** E.

A Minimum spanning tree can be constructed in polynomial time as seen before. Also a Minimum-Weighted Perfect Matching can be computed in polynomial time e.g., with the Blossom algorithm of Edmonds [14], and an Euler Tour can be computed with the algorithm of Hierholzer [23]. Therefore, the algorithm of Christofides has a polynomial running time.
The Traveling Salesman Problem can be solved exactly by solving the following IP:

$$\begin{aligned} \max \quad & c^\top x \\ \text{s.t.} \quad & \sum_{e \in E} x_e = |V| \\ & \sum_{e \in \delta(S)} x_e \geq 2 \quad \forall\, \emptyset \neq S \subset V, \end{aligned} \tag{IPTSP}$$

where $x \in \{0,1\}^{|E|}$ is a vector that indicates which edges are traveled in the tour. We denote by $\delta(S)$ the cut induced by S, which is the set of edges with exactly one end point in S. The second set of constraints is called subtour constraints. The problem with these constraints is that their number is exponential in the input size. Therefore, in our implementation we separated them by solving a Minimum Cut Problem. A Minimum Cut Problem can be solved in polynomial time by the algorithm of Stoer and Wagner [34].

Chapter 3

Complexity

Detailed knowledge about the complexity of an optimization problem is important for deciding if an investigation of a certain kind of algorithm is rewarding. If a problem is e.g. NP-hard, the search for a polynomial time algorithm can be stopped, unless one believes in P=NP and wants to prove it. With the adequate proof technique similar results can be stated for certain kinds of approximation algorithms, and a problem that is W[t]-hard with $t > 0$ does not admit an fpt algorithm unless the W-hierarchy collapses. But we also want to present positive results about the preservation of approximation factors of the underlying problem into the (mEm) Problem.
In the first section of this chapter we present results that are valid for the discrete and continuous version of (mEm). Afterwards we discuss the complexity of Problem (dmEm) by showing NP-hardness, inapproximability and W-hardness. The last section deals with the continuous version of the problem and underlines why solving this problem with discretization is an appropriate option in practice.

3.1 General Min-E-Min Problem

We start the investigation of complexity of (mEm) with a positive result. We will show that multiplicative and additive approximation factors of the underlying problem can be transferred into the Problem (mEm). This can be relevant in practice because it means that high quality solutions of (mEm) can be computed in a reasonable time even if the underlying problem is NP-

hard and would require a huge amount of time to be solved exactly. In that case a fast approximation algorithm or a fast heuristic can be used as an oracle, every time the underlying problem has to be solved. If (mEm) is solved exactly for this oracle and the oracle provides a solution close to the optimal solution of the underlying problem, the solution will be close to the optimal solution of (mEm) assuming that the problem corresponding to the induced set of solutions (see Section 2.6) is solved with an exact algorithm. For the convenience of the reader and to have a clean definition of approximation, we assume in the following that the underlying problem is a minimization problem and that all possible optimal solution values of the underlying problem are non-negative. The cases in which the underlying problem is a maximization problem and/or all possible optimal solution values are non-positive can be handled with analogous arguments.

Theorem 3.1. *Solving* (mEm) *with an α-approximation algorithm for the underlying problem, gives rise to an α-approximation algorithm for* (mEm).

Proof. Consider an arbitrary partition $\widetilde{P} = \{s_1, s_2, \ldots, s_K\}$ of the scenarios. Let $val_\alpha(\widetilde{P})$ be the sum of the solution values of every subset contained in $\widetilde{P}$ assuming that a solution value of a subset is obtained by using an α-approximation algorithm. In addition, let $val_1(\widetilde{P})$ denote the solution value when solving the oracle to optimality. For a subset of scenarios s, we define $val_\alpha(s)$ in the same way.
By definition of an α-approximation, for each subset s of scenarios in the partition, we have $val_\alpha(s) \leq \alpha \cdot val_1(s)$, and hence

$$val_\alpha(\widetilde{P}) = \sum_{s \in \widetilde{P}} val_\alpha(s) \leq \sum_{s \in \widetilde{P}} \alpha \cdot val_1(s) = \alpha \cdot val_1(\widetilde{P})$$

Denote now by $val^*_\alpha = \min_P val_\alpha(P)$ the best solution value, over all possible partitions, using the α-approximation algorithm. We have

$$val^*_\alpha \leq val_\alpha(\widetilde{P}) \leq \alpha \cdot val_1(\widetilde{P}) \tag{3.1}$$

Finally, let val^*_1 denote the optimal solution value for the problem if the underlying problem is solved to optimality.
Observe that (3.1) is valid for any partition $\widetilde{P}$ and hence we get $val^*_\alpha \leq \alpha \cdot val^*_1$, which concludes the proof. □

Note that Theorem 3.1 does not state that we have a polynomial time α-approximation algorithm for (mEm) if we have a polynomial time α-approximation for the oracle because in general (mEm) can not be solved in oracle polynomial time and is even NP-hard for underlying problems that are solvable in polynomial time, as we will see in Section 3.2.1.

Theorem 3.2. *Solving* (mEm) *with an absolute approximation algorithm for the oracle with guarantee c, we have an absolute approximation algorithm for* (mEm) *with guarantee Kc.*

Proof. The proof is similar to that of Theorem 3.1. The only difference is that, for each subset s of scenarios, the algorithm has an absolute approximation guarantee, i.e., the associated solution value is

$$val_\alpha(s) \leq c + val_1(s)$$

where c is a positive constant. Using the approximation algorithm for all subsets of scenarios we get

$$val_\alpha(\widetilde{P}) = \sum_{s\in\widetilde{P}} val_\alpha(s) \leq \sum_{s\in\widetilde{P}} (c + val_1(s)) = K \cdot c + val_1(\widetilde{P}).$$

□

Note that in the case of an absolute approximation, we do not need to make the assumption that all the optimal values of the oracle are positive or negative.

To conclude this section, we now want to show a property of (dmEm) that implies that it does not change the optimal solution value if the set of feasible solutions of Problem (dmEm) is replaced by its convex hull.

Theorem 3.3. *The objective function of Problem* (dmEm) *is concave. Therefore, the set of optimal solutions always contains an extreme point of the feasible set if the latter is convex.*

Proof. Let $f(\mathcal{X}_K) := E_\xi \min_{x\in\mathcal{X}_K} \xi^\top x$ be the objective function of Problem (dmEm). For every objective vector ξ, for $\lambda \in [0,1]$ and for two given, feasible solutions of Problem (dmEm) $\mathcal{X}_K := \{x_1, \ldots, x_K\}, \tilde{\mathcal{X}}_K := \{\tilde{x_1}, \ldots, \tilde{x_k}\}$, we have

$$\begin{aligned} \min_{j\in\{1,\ldots,K\}} \xi^\top(\lambda x_j + (1-\lambda)\tilde{x_j}) &\geq \min_{j\in\{1,\ldots,K\}} \xi^\top \lambda x_j + \min_{j\in\{1,\ldots,K\}} \xi^\top (1-\lambda)\tilde{x_j} \\ &= \lambda \min_{j\in\{1,\ldots,K\}} \xi^\top x_j + (1-\lambda) \min_{j\in\{1,\ldots,K\}} \xi^\top \tilde{x_j}. \end{aligned}$$

Because this holds for every realization of ξ, it also holds for the expected value. Therefore we have

$$f(\lambda\mathcal{X}_K + (1-\lambda)\tilde{\mathcal{X}}_K) \geq \lambda f(\mathcal{X}_K) + (1-\lambda) f(\tilde{\mathcal{X}}_K)$$

and hence f is concave. □

Theorem 3.3 shows in particular that the complexity results for Problem (dmEm) presented in the following sections generally also hold for convex feasible sets.

3.2 Discrete Min-E-Min Problem

For investigating the complexity of the Problem (dmEm), we analyze three different aspects: NP-hardness, hardness of approximation, and parameterized complexity. We distinguish between the problem variant where the parameter K is part of the input and the variant where it is fixed. Additionally we discuss the complexity of verifying whether a feasible solution exists and the hardness of a version of (dmEm), in which some of the K solutions are already fixed. Besides that, we use complexity results of Problem (dmEm) to prove new hardness results for the Min-Max-Min Optimization Problem. For the convenience of the reader in the following the term NP-hard is used for expressing that a problem is strongly NP-hard because in this thesis no weak NP-hardness results are shown. It will turn out that many results even hold when X_i is equal in every scenario and has polynomial size, in which case the underlying certain problem (P) could even be solved by enumeration. Table 3.1 summarizes the complexity results for the optimization variant and Table 3.2 illustrates the results for the feasibility problem.

Property	**Complexity result**	**Proof**
K is part of the input	NP-hard, not in APX	Theorem 3.5 & Theorem 3.11
K is not part of the input	NP-hard	Theorem 3.7
$K \geq 3$ is not part of the input	Not in APX	Theorem 3.12
$l - K$ fixed and the set of feasible solutions X is certain	Oracle-Polynomial time solvable	Theorem 4.1
Parameter K	W[2]-hard	Theorem 3.15
Parameter $n - K$	W[1]-hard	Theorem 3.16
$K = 2$ and one solution is fixed	NP-hard	Theorem 3.19

Table 3.1: Summary of complexity results of Problem (dmEm)

Property	Complexity result	Proof
K is part of the input	NP-hard	Theorem 3.8
K is not part of the input	NP-hard	Theorem 3.9
Parameter K	W[2]-hard	Theorem 3.17

Table 3.2: Summary of complexity results for the problem of deciding whether Problem (dmEm) has a feasible solution

3.2.1 NP-Hardness

We first show that the problem is NP-complete even in very restricted cases. First note that we have

Theorem 3.4. *If membership in X_i can be tested in polynomial time, Problem* (dmEm) *belongs to NP.*

Proof. For all the K solutions membership in X_i has to be verified. The objective value can be obtained by comparing K values obtained by vector multiplication, for a polynomial number of scenarios, and by summing them up. □

In the following, we distinguish between two variants of the problem: we first consider the number of solutions K as part of the input, afterwards we investigate the problem for fixed K. For the former case, we can show strong NP-hardness even when the underlying problem is certain and has a polynomial number of feasible solutions, i.e., when the set X_i is equal for all scenarios i and can be specified by an explicit list in the input. This shows that the hardness in this case already stems from the exponentially many possible assignments of scenarios to the K chosen solutions. Finally we show that finding a feasible solution is already NP-hard. This result holds if K is part of the input and if K is fixed.

Theorem 3.5. *If K is part of the input, the Problem* (dmEm) *is NP-hard. This remains true even if $X_i = X$ for all $i \in L$ and $|X|$ is polynomial.*

Proof. We use a reduction from the Dominating Set Problem. Figure 3.1 illustrates an example of the construction. A dominating set D in a graph G is a subset of the vertex set with the property that every vertex of G belongs to D or has a neighbor in D. The Dominating Set Decision Problem asks whether

a dominating set with at most K vertices exists; this problem is NP-complete and W[2]-hard for parameter K [13].
For the reduction, let $G = (V, E)$ be the given graph. We define $l = |V|$ scenarios as follows. For every $v \in V$, we define an objective vector $\xi_v \in \mathbb{Q}^V$ by

$$(\xi_v)_u := \begin{cases} -1 & \text{if } u = v \text{ or } u \text{ is a neighbor of } v, \\ 0 & \text{otherwise.} \end{cases}$$

For every scenario i we choose $X_i = X$ and X as the set of basis vectors in $\mathbb{Q}^V$. We claim that we can find a dominating set of size K in G if and only if we can find a solution for this instance of the Problem (dmEm) with value equal to $-|V|$.
First assume that there exists a dominating set $D \subseteq V$ with $|D| = K$. Then, by construction, for each $v \in V$ there exists a $u \in D$ with $(\xi_v)_u = -1$. Hence $\{\mathbf{e}_u \mid u \in D\}$ is a solution set for the Problem (dmEm) with

$$\sum_{v \in V} \min_{u \in D} \{\xi_v^\top \mathbf{e}_u\} = -|V| \ .$$

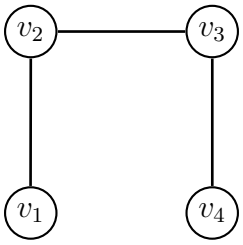
Conversely, if Problem (dmEm) has a set of solutions $x_1, \dots x_K \in X$ with value equal to $-|V|$, then for every objective vector ξ_v there must exist a basis vector $\mathbf{e}_u \in \mathcal{X}_K = \{x_1, \dots x_K\}$ with $(\xi_v)^\top \mathbf{e}_u = -1$. Therefore the set of vertices $\{u \in V \mid \mathbf{e}_u \in \mathcal{X}_K\}$ is a dominating set of size K in G.

□

Together with Theorem 3.4, this shows that the Discrete Min-E-Min Decision Problem is NP-complete and that the Discrete Min-E-Min Problem is NP-equivalent in general, provided that membership in X can be tested in polynomial time. This remains true even if $|X|$ is polynomial.

Remark 3.6. Theorem 3.3 can be used inside the proof of Theorem 3.5 to show that Problem (dmEm) is NP-hard even if the set of feasible solutions is the standard simplex. Hence, also for underlying problems with a convex feasible set Problem (dmEm) is in general NP-hard.

In the following, we focus on complexity results for fixed parameter K. Clearly, for $K = 1$ the problem is as easy as the underlying problem (P). However, starting from $K = 2$, the problem becomes NP-hard in general:



The set of vertices $\{v_2, v_3\}$ form a dominating set.

$$\xi_1 = \begin{pmatrix} -1 \\ -1 \\ 0 \\ 0 \end{pmatrix} \qquad \xi_2 = \begin{pmatrix} -1 \\ -1 \\ -1 \\ 0 \end{pmatrix}$$

$$\xi_3 = \begin{pmatrix} 0 \\ -1 \\ -1 \\ -1 \end{pmatrix} \qquad \xi_4 = \begin{pmatrix} 0 \\ 0 \\ -1 \\ -1 \end{pmatrix}$$

We obtain the solution of Problem (dmEm) as $x_1 = (0,1,0,0)^\top$ and $x_2 = (0,0,1,0)^\top$ with an objective value of -4.

Figure 3.1: Example of the construction of the proof of Theorem 3.5

Theorem 3.7. *For any fixed $K \geq 2$, the Discrete Min-E-Min Decision Problem is NP-hard, even for $X = \{0,1\}^n$.*

Proof. First we show NP-hardness for $K \geq 3$ by reduction from the Vertex Coloring Problem. Figure 3.2 illustrates an example of the construction. A K-vertex coloring of a graph $G = (V, E)$ is an assignment of K different colors to all vertices in V such that no edge in E connects two vertices of the same color. It is NP-complete to decide whether a given graph admits a K-vertex coloring when $K \geq 3$ [18]. In fact, the problem remains NP-complete even when a vertex is allowed to have more than one color. This follows from the fact that a graph admits a K-vertex coloring with more than one color per vertex allowed if and only if it admits a K-vertex coloring without multiple colors, since all but one color can be removed from every vertex without making the coloring infeasible.

Given a graph $G = (V, E)$, we consider one scenario for every vertex. We use the feasible set $X_i = X = \{0,1\}^V$ for all $i \in L$ and define an objective vector $\xi_v \in \mathbb{Q}^V$ for each vertex $v \in V$ by

$$(\xi_v)_u := \begin{cases} -1 & \text{if } u = v, \\ 1 & \text{if } u \text{ is a neighbor of } v, \\ 0 & \text{otherwise.} \end{cases}$$

Color j is assigned to vertex v if the solution x_j has value 1 in the dimension corresponding to v. Now for a given set of solutions $x_1, \ldots, x_K \in X$ we have $\sum_{v \in V} \min\{\xi_v^\top x_1, \ldots, \xi_v^\top x_K\} \leq -|V|$ iff the sets $V_j := \{v \in V \mid (x_j)_v = 1\}$ for $j \in \{1, \ldots, K\}$ are independent sets covering V. Indeed, if the sets were not independent, one vertex and one of its neighbors would belong to the same set V_j. Therefore, the corresponding scenario multiplied with the best x_j would be larger than -1 and the total objective value could not reach $-|V|$. This implies the result in the first case, since a covering of V by K independent sets is the same as a K-vertex coloring (with multiple colors allowed).
For the case $K = 2$, we reduce from the NP-complete decision variant of the Maximum Cut Problem [18]. Figure 3.3 illustrates an example of the construction. For given value q, the latter asks whether there exists a subset $W \subseteq V$ such that the cardinality of the induced cut is at least q, i.e., such that $|\delta(W)| \geq q$. We use the same construction as before to obtain scenarios, except that we set the objective vectors to

$$(\xi_v)_u := \begin{cases} -\deg_G(v) & \text{if } u = v, \\ 1 & \text{if } u \text{ is a neighbor of } v, \\ 0 & \text{otherwise.} \end{cases}$$

Then we claim that G has a cut of cardinality at least q if and only if there exist $x_1, x_2 \in \{0,1\}^V$ with $\sum_{v \in V} \min\{\xi_v^\top x_1, \xi_v^\top x_2\} \leq -2q$. Indeed, if $W \subseteq V$ with $|\delta(W)| \geq q$, we can set $(x_1)_v = 1$ and $(x_2)_v = 0$ if $v \in W$ and $(x_1)_v = 0$ and $(x_2)_v = 1$ otherwise. Then the minimum of the two terms is

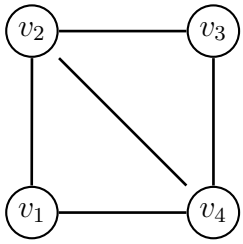
$$\begin{aligned} \xi_v^\top x_1 &= -\deg_G(v) + |N(v) \cap W|, \text{ if } v \in W \\ \xi_v^\top x_2 &= -\deg_G(v) + |N(v) \setminus W|, \text{ otherwise} \end{aligned}$$

and hence

$$\sum_{v \in V} \min\{\xi_v^\top x_1, \xi_v^\top x_2\} \leq -2|E| + 2|E(W)| + 2|E(V \setminus W)| = -2|\delta(W)| \, .$$

Conversely, if $\sum_{v \in V} \min\{\xi_v^\top x_1, \xi_v^\top x_2\} \leq -2q$ for some $x_1, x_2 \in \{0,1\}^V$, we define V_1 as above and obtain $|\delta(V_1)| \geq q$.

□



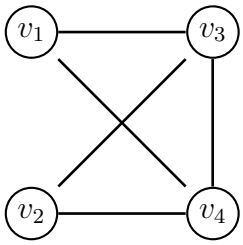
$$\xi_1 = \begin{pmatrix} -1 \\ 1 \\ 0 \\ 1 \end{pmatrix} \qquad \xi_2 = \begin{pmatrix} 1 \\ -1 \\ 1 \\ 1 \end{pmatrix}$$

$$\xi_3 = \begin{pmatrix} 0 \\ 1 \\ -1 \\ 1 \end{pmatrix} \qquad \xi_4 = \begin{pmatrix} 1 \\ 1 \\ 1 \\ -1 \end{pmatrix}$$

$V_1 := \{v_1, v_3\}$, $V_2 := \{v_2\}$, and $V_3 := \{v_4\}$ is a feasible coloring with three colors.

We obtain the solution of Problem (dmEm) as $x_1 = (1,0,1,0)^\top$, $x_2 = (0,1,0,0)^\top$ and $x_3 = (0,0,0,1)^\top$ with an objective value of -4.

Figure 3.2: Example of the construction of the proof of Theorem 3.7 for $K \geq 3$



$$\xi_1 = \begin{pmatrix} -2 \\ 0 \\ 1 \\ 1 \end{pmatrix} \qquad \xi_2 = \begin{pmatrix} 0 \\ -2 \\ 1 \\ 1 \end{pmatrix}$$

$$\xi_3 = \begin{pmatrix} 1 \\ 1 \\ -3 \\ 1 \end{pmatrix} \qquad \xi_4 = \begin{pmatrix} 1 \\ 1 \\ 1 \\ -3 \end{pmatrix}$$

The set of vertices $W := \{v_1, v_2\}$ induces a cut with value 4.

We obtain the solution of Problem (dmEm) as $x_1 = (1,1,0,0,)^\top$ and $x_2 = (0,0,1,1)^\top$ with value -8.

Figure 3.3: Example of the construction of the proof of Theorem 3.7 for $K = 2$

Different from the case of an unbounded K, we cannot expect a construction with a polynomially sized feasible set X anymore in Theorem 3.7, since for fixed K the number of solutions $|X|^K$ is polynomial, so that the problem could be solved efficiently by enumeration.

Now we focus on the problem of finding a feasible solution of Problem (dmEm). For many underlying problems there is a natural solution that is always fea-

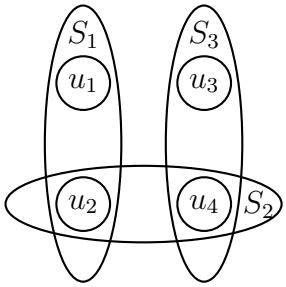
sible e.g. an empty knapsack for the Knapsack Problem. For other problems this is not always ensured. Consider the Maximum Flow Problem with lower and upper bounds. Because of the lower bounds the 0-flow is not necessarily feasible in all scenarios and therefore it is possible that Problem (dmEm) does not have a feasible solution if K is too small. Obviously, for $K = l$ a feasible solution can always be found if each scenario is a feasible instance of the underlying problem.

Theorem 3.8. *Deciding whether Problem* (dmEm) *has a feasible solution is NP-hard if K is part of the input, even if all the coefficients are binary and $|X_i|$ is polynomial in the input size for all $i \in L$.*

Proof. We prove the statement by reduction from the Set Cover Problem. Figure 3.4 illustrates an example of the construction. Given a positive integer K, a set $U = \{u_1, u_2, \ldots, u_l\}$ of items, and a collection $S = \{s_1, s_2, \ldots, s_n\}$ of subsets of U, the Set Cover Problem asks if there exists a subcollection of S with cardinality at most K, so that every item is contained in at least one of the subsets.
We will show that, given an instance of the Set Cover Problem, we can define an instance of Problem (dmEm) that is feasible if and only if the instance of the Set Cover Problem has a positive answer. For the reduction, we define a (dmEm) instance with $n := |S|$ dimensions and $l := |U|$ scenarios. For each scenario i, the feasible set X_i is the set of base vectors of $\mathbb{R}^n$ that satisfies a single constraint of the form $a_i^\top x \geq 1$, where the coefficient of each dimension d is defined as:

$$(a_i)_d := \begin{cases} 1 & \text{if } u_i \in s_d, \\ 0 & \text{otherwise.} \end{cases}$$

The set of solutions $x_1, \ldots, x_K$ of Problem (dmEm) induces the set cover as follows: if the d-th base vector is in the set of solutions, then item set s_d belongs to the set cover. With this definition, it holds that the i-th item is included in at least one subset if and only if there exists a selected subset s_d such that $u_i \in s_d$; this means that the associated coefficient $(a_i)_d$ is 1, i.e., the d-th solution is feasible for scenario i. Hence we have a set cover if and only if, for every scenario i, at least one solution among $x_1, \ldots, x_K$ is feasible. □



$$a_1 = \begin{pmatrix} 1 \\ 0 \\ 0 \end{pmatrix} \qquad a_2 = \begin{pmatrix} 1 \\ 1 \\ 0 \end{pmatrix}$$

$$a_3 = \begin{pmatrix} 0 \\ 0 \\ 1 \end{pmatrix} \qquad a_4 = \begin{pmatrix} 0 \\ 1 \\ 1 \end{pmatrix}$$

$\{S_1, S_3\}$ is a set cover.

We obtain the feasible solution of (dmEm) as $x_1 = (1,0,0)^\top$ and $x_2 = (0,0,1)^\top$.

Figure 3.4: Example of the construction of the proof of Theorem 3.8

We now discuss again the case of fixed K.

Theorem 3.9. *Deciding whether Problem* (dmEm) *has a feasible solution is NP-hard if $K \geq 2$ is not part of the input.*

Proof. We split the proof into two cases: $K = 2$ and $K \geq 3$. For $K \geq 3$, we show NP-hardness by reduction from the Vertex Coloring Problem. Figure 3.5 illustrates an example of the construction.
Given a graph $G = (V, E)$ and an integer $K \geq 3$, we define a (dmEm) instance with $n := |V|$ binary variables and $l := |V|$ scenarios. For each scenario i, there is a single constraint of the form $a_i^\top x \leq b$, and the coefficient of each dimension d is given by

$$(a_i)_d := \begin{cases} -1 & \text{if } d = i, \\ 1 & \text{if } (d,i) \in E, \\ 0 & \text{otherwise.} \end{cases}$$

Finally, we set $b := -1$ in every scenario.
A solution of Problem (dmEm) $x_1, \ldots, x_K$ induces the vertex coloring as follows: for each solution x_i, all vertices that correspond to a dimension d with value $(x_i)_d = 1$ receive the color i. This solution satisfies the coloring constraint as scenario constraints forbid any two neighbors to be taken in the same solution. Finally, feasibility of each scenario in at least one solution implies that every vertex belongs to a color class.

Conversely, given a solution of the Vertex Coloring Problem, we can define solutions $x_1, \ldots, x_K$ as follows: the j-th solution includes all variables that are associated to vertices belonging to the j-th color class. Remember that each color class corresponds to a stable set. Thus, a feasible solution for the j-th scenario is induced by the color class including vertex j, as by definition this color class cannot include any neighbor of j. Since all vertices receive a color, then all scenarios are satisfied by at least one solution.

For $K = 2$, we reduce from the NP-complete decision variant of the Maximum Cut Problem. Figure 3.6 illustrates an example of the construction. We set the number of scenarios l and the number of dimensions n to $2|V|$. Let ϵ be $\frac{1}{|V|+1}$. The set of feasible solutions is set to

$$X_i = \{0,1\}^V \times [\epsilon, |V| - 1]^V$$

for every scenario i and we add additional constraints that are described in the following. For every scenario $i \leq |V|$, we define a $|V|$-dimensional vector a_i as follows:

$$(a_i)_d := \begin{cases} \deg_G(v_i) & \text{if } d = i, \\ -1 & \text{if } v_d \text{ is a neighbor of } v_i, \\ 0 & \text{otherwise.} \end{cases}$$

Each of the first $|V|$ scenarios gets two equality constraints. The first constraint for all scenarios i with $i \leq |V|$ is:

$$\sum_{d=1}^{|V|} (a_i)_d x_d = x_{i+|V|}.$$

The second constraint for all scenarios i with $i \leq |V|$ is:

$$\sum_{d=|V|+1}^{2|V|} x_d \geq q.$$

All scenarios i with $i > |V|$ get only one constraint:

$$x_i = \epsilon.$$

The set of solutions $\{x_1, x_2\}$ of Problem (dmEm) with $K = 2$ induces the cut $\delta(W)$ as follows: v_d is in W if and only if $(x_1)_d = 1$.
Assume we have a feasible solution of Problem (dmEm). For every dimension $d \leq |V|$, either $(x_1)_d$ or $(x_2)_d$ has to be one because otherwise

$$\sum_{h=1}^{|V|} (a_d)_h x_h \leq 0$$

and therefore in order to fulfill scenario d also $x_{d+|V|} \leq 0 < \epsilon$ holds which is a contradiction to its domain. If $(x_1)_d$ and $(x_2)_d$ are equal to one in dimension d, we can set $(x_1)_d$ to 0 if scenario d is covered by x_2 and $(x_2)_d$ to 0 if scenario d is covered by x_1 without making the solution infeasible. Therefore without loss of generality we can assume that x_1 and x_2 are complementary in all dimensions $d \leq |V|$ and with the definition of a_i,

$$\sum_{d=1}^{|V|} (a_i)_d (x_j)_d$$

is the number of neighbors of v_i that are not in the same subset if scenario i is covered by x_j. Because of the first constraint of the first half of scenarios, $(x_1)_{i+|V|}$ is now equal to the number of neighbors of vertex i that are not in W if v_i is in W (respectively $(x_2)_{i+|V|}$ for neighbors in W of vertices not in W). Because constraint $x_{i+|V|} = \epsilon$ holds, we know that $(x_1)_{i+|V|}$ or $(x_2)_{i+|V|}$ has to be equal to ϵ. If v_i is not in W, $(x_2)_{i+|V|}$ takes the value

$$\sum_{d=1}^{|V|} (a_i)_d (x_2)_d$$

that cannot reach ϵ by construction and therefore $(x_1)_{i+|V|} = \epsilon$ (respectively $(x_2)_{i+|V|} = \epsilon$ if v_i is in W). Now we can rewrite

$$\sum_{d=|V|+1}^{2|V|} (x_1)_d = \sum_{\substack{i=1 \\ v_i \in W}}^{|V|} (x_1)_{i+|V|} + \sum_{\substack{i=1 \\ v_i \neq W}}^{|V|} (x_1)_{i+|V|}.$$

The first sum of the righthand side of the equation is the sum over all vertices in W of the number of neighbors that are not in W, which is the definition of the size of the cut. The second sum of the righthand side of the equation is

a value between 0 and $|V|\epsilon < 1$. Again, by the same arguments we can prove that

$$\sum_{d=|V|+1}^{2|V|} (x_2)_d$$

is equal to the size of the cut plus an additional term between 0 and $|V|\epsilon < 1$. Because of the second constraint of the first half of scenarios

$$\sum_{d=|V|+1}^{2|V|} (x_1)_d$$

has to be greater or equal to q. Because of the integrality the ϵ term has no influence on this constraint and we conclude that the size of the cut has to have at least the size of q.

Conversely, assume we have a cut $\delta(W)$ of size q or more. We will now construct a feasible solution for Problem (dmEm) step by step. We set $(x_1)_d = 1$ and $(x_2)_d = 0$ if vertex v_d is in W and $(x_1)_d = 0$ and $(x_2)_d = 1$ otherwise. We set

$$(x_1)_{d+|V|} = \sum_{d=1}^{|V|} (a_i)_d (x_1)_d$$

and $(x_2)_{d+|V|} = \epsilon$ if vertex v_i is in W and $(x_1)_{d+|V|} = \epsilon$ and

$$(x_2)_{d+|V|} = \sum_{d=1}^{|V|} (a_i)_d (x_2)_d$$

otherwise. Therefore all scenarios from the second half admit either x_1 or x_2 as feasible solution. The first constraint of the first half of solutions is fulfilled by x_1 if v_i is in W and by x_2 if v_i is not in W. So the scenario i is covered by x_1 if v_i is in W and by x_2 otherwise. Now we have to show that the second constraint holds in all of the scenarios plugging in the solution that covers the scenario. By our construction so far

$$\sum_{d=|V|+1}^{2|V|} (x_1)_d = \sum_{\substack{j=i \\ v_i \in W}}^{|V|} (x_1)_{i+|V|} + \sum_{\substack{i=1 \\ v_i \neq W}}^{|V|} (x_1)_{i+|V|}$$

is the size of the cut plus a term greater than 0. We know that the cut has at least size of q and therefore also the inequality

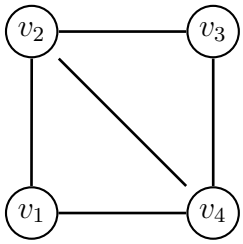
$$\sum_{d=|V|+1}^{2|V|} x_d \geq q$$

holds. Again by symmetry, we get the same result for the scenarios covered by x_2. □

Remark 3.10. The proof of Theorem 3.9 still holds, if we require the domain of the feasible set X_i to be binary instead of $\{0,1\}^V \times [\epsilon, |V|-1]^V$. In a first step the domain is transformed to

$$\{0,1\}^V \times \{1, |V|+1, |V|+2, \ldots, |V|^2-1\}^V.$$

This is possible by scaling the values of a_i and q with $(|V|+1)$, so that the lowest possible value is changed from ϵ to 1. In the proof of Theorem 3.9, all other values of a variable belonging to the set X_i in dimension $d > |V|$ are integers and therefore it is sufficient to restrict the feasible set to all integers between $|V|+1$ and $(|V|-1)(|V|+1) = |V|^2-1$. Afterwards, the desired result can be achieved by binary expansion.



$$a_1 = \begin{pmatrix} -1 \\ 1 \\ 0 \\ 1 \end{pmatrix} \qquad a_2 = \begin{pmatrix} 1 \\ -1 \\ 1 \\ 1 \end{pmatrix}$$

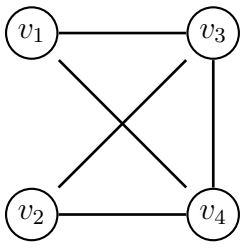
$$a_3 = \begin{pmatrix} 0 \\ 1 \\ -1 \\ 1 \end{pmatrix} \qquad a_4 = \begin{pmatrix} 1 \\ 1 \\ 1 \\ -1 \end{pmatrix}$$

$V_1 := \{v_1, v_3\}$, $V_2 := \{v_2\}$ and $V_3 := \{v_4\}$ is a feasible coloring with three colors.	We obtain the feasible solution of Problem (dmEm) as $x_1 = (1,0,1,0)^\top$, $x_2 = (0,1,0,0)^\top$ and $x_3 = (0,0,0,1)^\top$.

Figure 3.5: Example of the construction of the proof of Theorem 3.9 for $K \geq 3$

3.2.2 Hardness of Approximation

Having shown that Problem (dmEm) is NP-hard, we next investigate its approximability. Again, we first consider the case of K being part of the input and then the case of a fixed K. Polynomial time approximation algorithms bound



$W = \{v_1, v_2\}$ induces a cut with value q = 4.

Scenario 1:

$$2x_1 - x_3 - x_4 = x_5$$
$$x_5 + x_6 + x_7 + x_8 \geq 4$$

Scenario 5:

$$x_5 = \varepsilon$$

Scenario 2:

$$2x_2 - x_3 - x_4 = x_6$$
$$x_5 + x_6 + x_7 + x_8 \geq 4$$

Scenario 6:

$$x_6 = \varepsilon$$

Scenario 3:

$$-x_1 - x_2 + 3x_3 - x_4 = x_7$$
$$x_5 + x_6 + x_7 + x_8 \geq 4$$

Scenario 7:

$$x_7 = \varepsilon$$

Scenario 4:

$$-x_1 - x_2 - x_3 + 3x_4 = x_8$$
$$x_5 + x_6 + x_7 + x_8 \geq 4$$

Scenario 8:

$$x_8 = \varepsilon$$

$x_1 = (1, 1, 0, 0, 2, 2, \varepsilon, \varepsilon)^T$ and $x_2 = (0, 0, 1, 1, \varepsilon, \varepsilon, 2, 2)^T$ is a feasible solution for mEm because x_1 is feasible in scenario 1,2,7 and 8 and x_2 in the others.

Figure 3.6: Example of the construction of the proof of Theorem 3.9 for $K = 2$

the worst case ratio between the computed solution and the optimal solution. Among these, constant factor approximation algorithms are particularly interesting because they guarantee that this ratio is not worse than a constant, whereas for other approximation algorithms the approximation guarantee increases with the input size. We show that Problem (dmEm) is not in APX if K is part of the input or if $K \geq 3$ is fixed.

Theorem 3.11. *If K is part of the input, the Problem* (dmEm) *does not belong to APX unless $P = NP$, even if X_i is equal in every scenario, has polynomial size, and the objective vectors are restricted to be non-negative.*

Proof. Assume that there exists an α-approximation algorithm for the Prob-

lem (dmEm), for some $\alpha > 1$. Then we claim that the NP-complete decision variant of the Vertex Cover Problem can be solved in polynomial time, which implies P = NP. For given number K, the latter problem asks whether there exists a set of K vertices in a graph such that every edge is incident to at least one vertex in this set.

Given a graph $G = (V, E)$, let $X_i = X$ be the set of basis vectors in $\mathbb{Q}^V$ for every scenario i and define an objective vector $\xi_e \in \mathbb{Q}^V$ with $\xi_e \geq 0$ for each $e \in E$ by

$$(\xi_e)_v := \begin{cases} 1 & \text{if } v \in e, \\ (\alpha - 1)|E| + 2 & \text{otherwise.} \end{cases}$$

We claim that there exists a Vertex Cover of size K if and only if the given α-approximation algorithm finds a solution for (dmEm) with value $|E|$ or less. Indeed, if $U \subseteq V$ is a vertex cover of G with $|U| = K$, we can consider the K solutions $\{\mathbf{e}_v \mid v \in U\} \subseteq X$, for which we obtain

$$\sum_{e \in E} \min_{v \in U} \ \xi_e^\top \mathbf{e}_v \leq |E| \ .$$

Otherwise, if no such vertex cover exists, we have

$$\sum_{e \in E} \min\{\xi_e^\top x_1, \ldots, \xi_e^\top x_K\} \geq (\alpha - 1)|E| + 2 + (|E| - 1) = \alpha|E| + 1$$

for all $x_1, \ldots, x_K \in X$. This means that the optimal solution value of (dmEm) is larger than $|E|$ and therefore in the optimal solution there exists one edge that is not covered. Hence, there is no Vertex Cover of size K or larger. □

As argued above, in case of fixed K, we cannot expect the same result for a feasible set of polynomial size. For showing NP-hardness in Theorem 3.7, we thus used $X_i = \{0, 1\}^n$ in every scenario i. However, when restricting ourselves to non-negative objective functions, the feasibility of the zero vector obviously makes the Problem (dmEm) trivial, as the zero vector is necessarily optimal then. For this reason, we now consider a slightly changed underlying problem, by introducing one more variable that is fixed to one. This is equivalent to allowing an additional constant term in each scenario. Clearly, the underlying problem (P) remains trivial with this adaptation. Nevertheless, we can now show:

Theorem 3.12. *For any fixed $K \geq 3$, we cannot decide whether the optimal value of Problem* (dmEm) *is zero, even if all feasible solutions have non-negative objective value and $X_i = \{x \in \{0,1\}^n \mid x_n = 1\}$ for every scenario i.*

Proof. Assume on contrary that we can decide this question. We claim that the existence of a vertex coloring with K colors in a given graph can be decided in polynomial time then. Given a graph $G = (V, E)$, let $X_i = X = \{0,1\}^V \times \{1\}$ for every scenario and define an objective vector $\xi_v \in \mathbb{Q}^V \times \mathbb{Q}$ for each vertex $v \in V$ by

$$(\xi_v)_u = \begin{cases} -1 & \text{if } u = v, \\ 1 & \text{if } u \in V \setminus \{v\} \text{ is a neighbor of } v, \\ 0 & \text{if } u \in V \setminus \{v\} \text{ is not a neighbor of } v, \\ 1 & \text{otherwise (i.e. in the last component).} \end{cases}$$

Now for all $x \in X$ and all $v \in V$ we have $\xi_v^\top x \geq 0$ by construction. It is easy to verify that the optimal value of Problem (dmEm) for this instance is zero if and only if G admits a vertex coloring with K colors. □

Because every constant factor approximation algorithm has a value of zero if and only if an instance has an optimal solution value of zero, we obtain the following result:

Corollary 3.13. *For fixed $K \geq 3$, Problem* (dmEm) *does not belong to APX unless $P = NP$, even if all feasible solutions have non-negative objective value and $X_i = \{x \in \{0,1\}^n \mid x_n = 1\}$ for every scenario i.*

For the case $K = 2$, we do not know whether Problem (dmEm) belongs to APX. However, we obtain the following weaker result, which follows from the corresponding well-known result for the Maximum Cut Problem [22] and the construction in the proof of Theorem 3.7

Theorem 3.14. *For $K = 2$, approximating Problem* (dmEm) *with a factor better than $\frac{17}{16}$ is NP-hard.*

Proof. Following the proof of Theorem 3.7 we showed that we can construct a cut with value of q if the solution value of (dmEm) has a value of $-2q$. Therefore, an approximation algorithm for (dmEm) would imply an approximation algorithm for the Maximum Cut Problem with the same factor. □

3.2.3 Parameterized Complexity

For the complexity results obtained so far, the main distinction was made between the two variants of Problem (dmEm) where the number K of solutions is fixed or part of the input. In order to further investigate the role of K, we now ask for the existence of fpt algorithms.
In Theorem 3.7, we have shown that Problem (dmEm) is NP-hard for any fixed $K \geq 2$, using $X = \{0,1\}^n$. This already implies that there cannot exist an fpt algorithm in the parameter K in general, unless P = NP. On the other hand, if X is of polynomial size, the problem can be solved by enumeration for fixed K, leading to a running time of $O(|X|^K)$. However, this does not yield an fpt algorithm. In fact, by having a closer look at the NP-hardness proof of Theorem 3.5, we can show that even in case of a polynomially large X there likely does not exist any fpt algorithm in parameter K, by proving that the problem is W[2]-hard even in this special case.

Theorem 3.15. *The decision variant of Problem* (dmEm) *is $W[2]$-hard for parameter K, even if $X_i = X$ for every scenario i and $|X|$ is polynomial.*

Proof. The Dominating Set Problem is one of the most well-known W[2]-hard problems when considering the size of the set as parameter [13]. The reduction used in the proof of Theorem 3.5 is a parameterized reduction for K because we use the same K for both problems. This proves the statement. □

It follows from Theorem 3.15 that an fpt algorithm for Problem (dmEm) for parameter K cannot exist, unless W[0]=W[1]=W[2]. A slightly weaker result can be obtained when considering the parameter $n-K$ instead, where n is the number of dimensions in the underlying problem:

Theorem 3.16. *Problem* (dmEm) *is $W[1]$-hard for parameter $n-K$, even if $X_i = X$ for every scenario i and $|X|$ is polynomial.*

Proof. We construct a parameterized reduction from the decision variant of the Set Cover Problem. Let $X_i = X$ for every scenario i and X consist of all basis vectors in $\mathbb{Q}^n$ and define, for every $u \in U$, an objective vector $\xi_u \in \mathbb{Q}^n$ by

$$(\xi_u)_j = \begin{cases} -1 & \text{if } u \in s_j, \\ 0 & \text{otherwise.} \end{cases}$$

Then, by construction, any set cover $\{s_{i_1}, \ldots, s_{i_K}\}$ of size K induces a set of solutions $\{\mathbf{e}_{i_1}, \ldots, \mathbf{e}_{i_K}\} \subseteq X$ with

$$\sum_{u \in U} \min_{j \in \{1,\ldots,K\}} \xi_u^\top \mathbf{e}_{i_j} = -|U| \ .$$

Conversely, for every solution $\mathbf{e}_{i_1}, \ldots, \mathbf{e}_{i_K}$ with value $-|U|$, the corresponding sets $s_{i_1}, \ldots, s_{i_K}$ cover U. Since this is a parameterized reduction for $n-K$ and the Set Cover Problem is W[1]-hard for parameter $n-K$ [5], we obtain the desired result. □

The following theorem shows that also determining whether a feasible solution exists, is already W[2]-complete parameterized by K.

Theorem 3.17. *Deciding whether Problem* (dmEm) *has a feasible solution is W[2]-complete parameterized by K, even if all the coefficients are binary and $|X_i|$ is polynomial in the input size for all $i \in L$.*

Proof. The Set Cover Problem is known to be W[2]-complete parameterized by K. This was proven by reduction from the Dominating Set Problem [32]. In the proof of Theorem 3.8 we use the same K for Problem (dmEm) and the Set Cover Problem and therefore the reduction is parameterized. □

3.2.4 Connection to Min-Max-Min Robustness

We can adapt some of our proofs in order to obtain hardness results for a related problem in robust optimization, the so-called Min-Max-Min Optimization Problem [6]: instead of the expected value, one asks for the worst case and additionally one assumes that the uncertainty only affects the objective function, resulting in the problem

$$\begin{aligned} \min \quad & \max_{i \in \{1,\ldots,l\}} \min_{x \in X \cap \mathcal{X}_K} \xi_i^\top x \\ \text{s.t.} \quad & |\mathcal{X}_K| \leq K \ . \end{aligned} \tag{mmm}$$

Buchheim and Kurtz [7] showed that in case of discrete uncertainty Problem (mmm) is NP-hard for fixed K for the following underlying problems: Shortest Path, Spanning Tree, Bipartite Matching. We can show that Problem (mmm) is strongly NP-hard when K is part of the input, even if the number of feasible solutions of the underlying problem $|X|$ is a polynomial.

For proving this, we use the same reduction as in the proof of Theorem 3.5. The only difference is that we find a Dominating Set if and only if we can find a solution for (mmm) with value equal to -1.

Moreover, with the same reduction as in the proof of Theorem 3.11, we can show that Problem (mmm) is not in APX when K is part the input, even if $|X|$ is a polynomial. The only change here is that we can simplify the construction to

$$(\xi_e)_v := \begin{cases} 1 & \text{if } v \in e, \\ \alpha + 1 & \text{otherwise.} \end{cases}$$

Also in the case of fixed $K \geq 3$ we can show that Problem (mmm) is not in APX by adapting the proof of Theorem 3.12. Finally, also the results of Theorem 3.15 and Theorem 3.16 can be carried over to Problem (mmm) using the same reductions and hence Problem (mmm) is W[2]-hard for parameter K and W[1]-hard for parameter $n - K$.

3.2.5 The Min-E-Min Completion Problem

In this section, we introduce a problem that we call the Min-E-Min Completion Problem, and that is closely related to the Discrete Min-E-Min Problem. In particular, this problem is a variant of the Discrete Min-E-Min Problem arising when $\overline{K}$ solutions have been fixed and are part of the input. The objective is to determine the remaining $K_f := K - \overline{K}$ *free* solutions; without loss of generality, we may assume the free solutions to be the first K_f ones. We denote the set of fixed solutions, which are given as input, by $\mathcal{X}_{\overline{K}}$ and the set of solutions that have to be determined by $\mathcal{X}_{K_f}$. The vector ξ_i is the objective vector in scenario i. Using this notation, the Min-E-Min Completion Problem can be formulated as follows:

$$\begin{array}{ll} \min & \displaystyle\sum_{i=1}^{l} \min_{x \in X_i \cap (\mathcal{X}_{K_f} \cup \mathcal{X}_{\overline{K}})} \xi_i^\top x \\ \text{s.t.} & |\mathcal{X}_{K_f}| \leq K_f \\ & \mathcal{X}_{K_f} \subset \mathbb{R}^n \, . \end{array} \tag{mEmC}$$

Problem (mEmC) is equivalent to a subproblem appearing in a solution approach for (dmEm) based on column generation (see Section 4.5.1). In the following we discuss the complexity of this problem.

Theorem 3.18. *Problem* (mEmC) *is NP-hard and W*[2]*-hard parameterized by* K_f *if* $1 < K_f < l$.

Proof. It is easy to see that an instance of (dmEm) with parameter K can be modeled as an instance of the Min-E-Min Completion Problem with parameter $K_f = K$ and $\overline{K} = 1$, using a dummy solution that is infeasible (or high-costly) for all scenarios. The result follows from the complexity of Problem (dmEm) (see Theorem 3.5, Theorem 3.7 and Theorem 3.15). □

The case $K_f = 1$ is giving more interesting results because in this case (mEmC) turns out to be NP-hard, though Problem (dmEm) is not.

Theorem 3.19. *Problem* (mEmC) *is NP-hard even if* $K_f = \overline{K} = 1$, *if the uncertainty affects the objective function only, and if* $X_i = \{0,1\}^n$ *for all* $1 \leq i \leq l$.

Proof. We prove the statement by reduction from the Maximum Cut Problem. Given a graph $G = (V, E)$, we define an instance of (mEmC) with $\overline{K} = K_f = 1$ as follows. There are $n = |V|$ binary variables and $l = 2|E|$ scenarios, all with no explicit constraints, i.e., $X_i = \{0,1\}^n$ for all $1 \leq i \leq l$. In particular, for each edge $e_i = (u, v) \in E$, there are two scenarios numbered as i and $|E| + i$. For the former, the objective function coefficients for the variables are

$$(\xi_i)_d := \begin{cases} -1 & \text{if } d = u, \\ 2 & \text{if } d = v, \\ 0 & \text{otherwise,} \end{cases}$$

while for the latter the objective function is defined as follows

$$(\xi_{|E|+i})_d := \begin{cases} 2 & \text{if } d = u, \\ -1 & \text{if } d = v, \\ 0 & \text{otherwise.} \end{cases}$$

Finally, the fixed solution is a vector of zeros, hence, it has a zero cost for every scenario.

We now show that the graph G has a cut with value q or more if and only if there exists a solution for the Min-E-Min Completion Problem with value of $-q$ or less. Assume that G has a cut with value q, i.e., there exists

a partition $(W, V \setminus W)$ of its vertices such that $\delta(W) \geq q$. Define a solution for (mEmC) by setting x_1 in dimension d to one if the vertex corresponding to d is in W and 0 otherwise. Since $\delta(W) \geq q$, there are at least q scenarios for which this solution takes value -1. Because there exists a solution with zero value for all the remaining scenarios, the value of (mEmC) is at least $-q$. Conversely, assume now that there exists a solution of Problem (mEmC) with value $-q$. Define set W to include all the vertices associated with variables that take value 1 in the free solution. As (mEmC) has value $-q$, there are q scenarios whose value is -1. Every such scenario is associated with an edge that must have exactly one endpoint in W (otherwise, it would have a positive cost) and the proof is complete. □

3.3 Continuous Min-E-Min Problem

This subsection serves for showing that Problem (mEm) with a continuous distribution of ξ and θ is in general so hard that solving this problem without discretization requires techniques that are beyond the scope of this thesis. In fact we show that even for a given solution, evaluating the objective function is hard and therefore even meta heuristics, which are in general very fast and widely used in practice to deal with large problems, become inefficient. In addition we show that the solution of a new problem, in which the distribution is replaced by a discrete set of N samples, converges to the original solution for $N \to \infty$ if the set of feasible solutions is certain. Combining the proof of Lemma 2 and Theorem 1 in [21] results in the following statement:

Corollary 3.20. *Given $a \in \mathbb{Z}^n$ and $b \in \mathbb{Z}$, computing*

$$\mathcal{Q}(a, b) = E_{\xi}(\max\{a^{\top}\tilde{\xi} - b, 0\})$$

is weakly #P-hard (see Definition 2.35) if $\tilde{\xi} \sim \mathcal{U}[0, 1]^n$, i.e. each $\tilde{\xi}_i$ is independently uniformly distributed on $[0, 1]$.

We will use this for proving the following theorem.

Theorem 3.21. *Evaluating the objective function of Problem* (mEm) *for a fixed solution is #P-hard for general continuous distributions even if $K = 2$.*

Proof. We reduce the problem of calculating $\mathcal{Q}(a, b)$ to the evaluation of Problem (mEm). Let $a \in \mathbb{Z}^n$ and $b \in \mathbb{Z}$ be our input. For our reduction we choose $\tilde{\xi} \sim \mathcal{U}[0,1]^n$ and

$$\xi = \begin{pmatrix} \tilde{\xi} \\ 1 \end{pmatrix}.$$

We set the two fixed solutions for Problem (mEm) to

$$x_1 = \begin{pmatrix} -a \\ 0 \end{pmatrix}$$

and

$$x_2 = \begin{pmatrix} 0 \\ -b \end{pmatrix}$$

hence $x_1, x_2 \in \mathbb{Z}^{n+1}$. Now we reformulate the objective function of Problem (mEm):

$$\begin{aligned} E_\xi(\min\{\xi^\top x_1, \xi^\top x_2\}) &= E_\xi(\min\{-a^\top \tilde{\xi}, -b\}) \\ &= -E_\xi(\max\{a^\top \tilde{\xi}, b\}) \\ &= -b - E_\xi(\max\{a^\top \tilde{\xi} - b, 0\}) \\ &= -b - \mathcal{Q}(a, b) \end{aligned}$$

Therefore $\mathcal{Q}(a, b)$ can be computed by computing $E_\xi(\min\{\xi^\top x_1, \xi^\top x_2\})$, and the statement follows. □

Theorem 3.21 implies that it is not possible for every solution to evaluate the objective function for Problem (mEm) with a continuous distribution in polynomial time, unless P=NP. Therefore it is highly unlikely that there exists an efficient algorithm for Problem (mEm) in the continuous case.

In the following we want to show that Problem (mEm) with a certain feasible set X and a continuous distribution can be solved approximately by sampling. We show that if the number of samples goes to infinity, the optimal solution value and the optimal solution of the discretized problem converge to their counterparts of the original problem. The only assumptions we make are that a feasible solution of Problem (mEm) exists, that the set of feasible solutions X

is bounded, and that $E_\xi(||\xi||_2)$ is finite, from which it follows that also $E_\xi(\xi)$ is finite. These additional assumptions are not very restrictive, in fact every well defined instance of Problem (mEm) should meet them.

Definition 3.22. Let $\bar{\xi}$ be a realization of ξ. Define

$$F(\mathcal{X}_K, \bar{\xi}) := \min_{x \in \mathcal{X}_K} (\bar{\xi})^\top x$$

and let

$$f(\mathcal{X}_K) := E_\xi(F(\mathcal{X}_K, \xi))$$

be the original inner optimization problem of Problem (mEm) and

$$\hat{f}_N(\mathcal{X}_K) := \frac{1}{N} \sum_{i=1}^{N} F(\mathcal{X}_K, \hat{\xi}_i)$$

the inner optimization problem if the original distribution is discretized by N samples $\hat{\xi}_i$ which are independently identically distributed and following the same distribution as ξ. Let

$$S := \operatorname*{argmin}_{\mathcal{X}_K \in X^K} f(\mathcal{X}_K)$$

be the set of optimal solutions of the original problem and

$$v := \min_{\mathcal{X}_K \in X^K} f(\mathcal{X}_K)$$

as its optimal value. In the same way we define

$$\hat{S}_N := \operatorname*{argmin}_{\mathcal{X}_K \in X^K} \hat{f}_N(\mathcal{X}_K)$$

as the set of optimal solutions of the discretized problem and

$$\hat{v}_N := \min_{\mathcal{X}_K \in X^K} \hat{f}_N(\mathcal{X}_K)$$

be its optimal value.

In the following we identify $\mathcal{X}_K$ with the vector in $\mathbb{R}^{K \cdot n}$ resulting from appending all vectors in $\mathcal{X}_K$. In particular, we consider

$$||\mathcal{X}_K||_p := \sqrt[p]{\sum_{h=1}^{K} \sum_{d=1}^{n} |(x_h)_d|^p}.$$

Lemma 3.23. *The following statement holds for all $\mathcal{X}_K, \bar{\mathcal{X}}_K \in X^K$, for all realizations $\bar{\xi}$ of ξ and for all $\delta > 0$:*

$$||\mathcal{X}_K - \bar{\mathcal{X}}_K||_2 \leq \delta \Rightarrow |F(\mathcal{X}_K, \bar{\xi}) - F(\bar{\mathcal{X}}_K, \bar{\xi})| \leq \delta ||\bar{\xi}||_2,$$

which means that $F(\mathcal{X}_K, \bar{\xi})$ is Lipschitz continuous in $\mathcal{X}_K$ with a Lipschitz constant $||\bar{\xi}||_2$.

Proof. Let $\mathcal{X}_K := \{x_1, \ldots, x_K\}$ and $\bar{\mathcal{X}}_K := \{\bar{x}_1, \ldots, \bar{x}_K\}$. Assume without loss of generality that $F(\mathcal{X}_K, \bar{\xi}) \geq F(\bar{\mathcal{X}}_K, \bar{\xi})$ and that

$$j = \underset{1 \leq h \leq K}{\operatorname{argmin}} \{(\bar{\xi})^\top \bar{x}_h | \bar{x}_h \in \bar{\mathcal{X}}_K\}.$$

We can derive

$$\begin{aligned}
|F(\mathcal{X}_K, \bar{\xi}) - F(\bar{\mathcal{X}}_K, \bar{\xi})| &= |\min_{x \in \mathcal{X}_K} (\bar{\xi})^\top x - \min_{\bar{x} \in \bar{\mathcal{X}}_K} (\bar{\xi})^\top \bar{x}| \\
&\overset{*}{\leq} |(\bar{\xi})^\top x_j - (\bar{\xi})^\top \bar{x}_j| \\
&= |(\bar{\xi})^\top (x_j - \bar{x}_j)| \\
&\overset{**}{\leq} ||\bar{\xi}||_2 \, ||(x_j - \bar{x}_j)||_2 \\
&= ||\bar{\xi}||_2 \sqrt{\sum_{d=1}^{n} (x_j - \bar{x}_j)_d^2} \\
&\overset{***}{\leq} ||\bar{\xi}||_2 \sqrt{\sum_{h=1}^{K} \sum_{d=1}^{n} (x_h - \bar{x}_h)_d^2} \\
&= ||\bar{\xi}||_2 \, ||\mathcal{X}_K - \bar{\mathcal{X}}_K||_2 \\
&\leq \delta ||\bar{\xi}||_2.
\end{aligned}$$

For $(*)$ we use the definition of x_j and the assumption that $F(\mathcal{X}_K, \bar{\xi}) \geq F(\bar{\mathcal{X}}_K, \bar{\xi})$ holds. Inequality $(**)$ follows from the Cauchy-Schwarz inequality and $(***)$ follows from the fact that no summand of

$$\sum_{h=1}^{K} \sum_{d=1}^{n} (x_h - \bar{x}_h)_d^2$$

has a negative value. □

Lemma 3.24. *The function $f(\mathcal{X}_K)$ is continuous in $\mathcal{X}_K$.*

Proof. Given $\epsilon > 0$, define $\delta := \frac{\epsilon}{E_\xi(||\xi||_2)+1}$. Because $E_\xi(||\xi||_2)$ is finite by assumption, we have $\delta > 0$. If

$$|\mathcal{X}_K - \bar{\mathcal{X}}_K| \leq \delta$$

then

$$\begin{aligned}
|f(\mathcal{X}_K) - f(\bar{\mathcal{X}}_K)| &= |E_\xi(F(\mathcal{X}_K, \xi)) - E_\xi(F(\bar{\mathcal{X}}_K, \xi))| \\
&\leq E_\xi(|F(\mathcal{X}_K, \xi) - F(\bar{\mathcal{X}}_K, \xi)|) \\
&\overset{*}{\leq} E_\xi(\delta||\xi||_2) \\
&= \delta E_\xi(||\xi||_2) \\
&\leq \delta E_\xi(||\xi||_2) + 1 \\
&= \epsilon.
\end{aligned}$$

For (*) we use Lemma 3.23 and the fact that the expected value is monotone. □

Lemma 3.25. *For Problem* (mEm) *with a certain feasible set X, $\hat{f}_N(\mathcal{X}_K)$ converges pointwise on X^K to $f(\mathcal{X}_K)$ with probability 1 for $N \to \infty$.*

Proof. Following Shapiro et al. [33], we need to show that $C := X^K$ is compact, that for almost all realizations $\bar{\xi}$ of ξ, $F(\mathcal{X}_K, \bar{\xi})$ is continuous in $\mathcal{X}_K$, and that there exists a measurable function g with

$$|F(\mathcal{X}_K, \bar{\xi})| \leq g(\bar{\xi})$$

for all $\mathcal{X}_K \in C$. First note that C is compact because the set of feasible solutions X is compact. Lemma 3.23 implies that $F(\mathcal{X}_K, \bar{\xi})$ is continuous in $\mathcal{X}_K$ for all realizations $\bar{\xi}$ of ξ. Let m_d be the maximum absolute value that a feasible solution $x \in X$ can have in dimension d, which is finite because X is bounded, and define

$$g(\bar{\xi}) := \sum_{d=1}^{n} m_d \, |\bar{\xi}_d|.$$

Now we have $|F(\mathcal{X}_K, \bar{\xi})| \leq g(\bar{\xi})$ for all $\mathcal{X}_K \in C$ and almost all realizations $\bar{\xi}$ of ξ and $g(\bar{\xi})$ is continuous and hence measurable. □

Remark 3.26. For Problem (mEm) with uncertain constraints, we cannot prove a similar statement because the crucial requirement that $F(\mathcal{X}_K, \bar{\xi})$ is continuous in $\mathcal{X}_K$ with probability 1 is not met. A small change in $\mathcal{X}_K$ can make the current minimum solution x infeasible and that can cause a jump in the value of $F(\mathcal{X}_K, \bar{\xi})$.

Theorem 3.27. *If there exists a feasible solution for Problem* (mEm) *with certain, bounded feasible set X and if ξ follows a distribution such that $E_\xi(||\xi||_2)$ is finite, the following statements are true:*

1. *For $N \to \infty$, $\hat{v}_N$ converges to v with probability 1.*
2. *For $N \to \infty$, $\hat{S}_N$ converges to S with probability 1, meaning that*

$$\max\{\sup_{s_1 \in S} \inf_{s_2 \in \hat{S}_N} ||s_1 - s_2||, \inf_{s_1 \in S} \sup_{s_2 \in \hat{S}_N} ||s_1 - s_2||\} \to 0$$

holds.

Proof. Building up on the result of Shapiro et al. [33], it is sufficient to show that there exists a compact set C such that:

1. The set of optimal solutions of the original problem S is nonempty and contained in C.
2. The function f is finite valued and continuous on C.
3. $\hat{f}_N$ converges to f with probability 1 for $N \to \infty$.
4. For large enough N, the set $\hat{S}_N$ is nonempty and contained in C.

Let $C := X^K$ be the set of K-element subsets of the feasible set of Problem (mEm) with certain constraints and therefore C is compact. Obviously S and $\hat{S}_N$ are contained in C. Because we know that a feasible solution exists, S and $\hat{S}_N$ are nonempty. Therefore the first and the fourth point are valid. The function f is continuous on C because of Lemma 3.24. Therefore and because X^K is compact, it follows that f is finite valued. Lemma 3.25 shows the third point. □

Remark 3.28. For (mEm) with uncertain constraints, we cannot prove a similar statement because for these problems the statement of Lemma 3.25, which is crucial for the proof, is not true.

Theorem 3.27 shows that Problem (mEm) with certain constraints can be solved by replacing the distribution of ξ by a discrete set of samples. If the number of samples that are used are large enough, the solution value and the optimal solution of the discretized version are close enough to their counterparts of the original problem. The resulting discrete problem can be solved by the methods described in this thesis.

Chapter 4

Exact Algorithms

Having shown that the Discrete Min-E-Min Problem is NP-hard in general even if the underlying problem (P) is tractable, one cannot expect an exact polynomial-time algorithm. Nevertheless, we will now propose several exact approaches for this task that we will investigate experimentally in Chapter 6.

The methods differ in the way they depend on the underlying problem (P): some of the approaches only need an oracle for computing the induced solution of a subset of scenarios (remember that this is not necessarily the same as an oracle for (P), see Section 2.6), others require an ILP formulation for the latter or even a complete enumeration of all elements of $\bigcup_{1\leq i\leq l} X_i$. Moreover, as we will see, their performance depends strongly on the parameters K, l, and n, and different methods turn out to be preferable in different settings.

We start by describing two types of enumeration algorithms and then propose an IQP-based compact formulation. Afterwards we present two Set Partitioning formulations and extend one of them to a Branch-and-price algorithm.

4.1 Enumeration of Feasible Solutions

The most straightforward method to solve the Discrete Min-E-Min Problem is to compute all subsets of $\bigcup_{1\leq i\leq l} X_i$ of size K, to calculate their objective values and check whether they are feasible, and to choose the best solution obtained. Clearly, such a complete enumeration is only reasonable when K and $|\bigcup_{1\leq i\leq l} X_i|$ are very small, whereas the number of scenarios l only has a minor impact on running time. An additional benefit of this solution method

is the fact that it is not necessary to solve the underlying problem, which can be an advantage when the latter is NP-hard. Nevertheless, in most relevant situations, this algorithm is impractical and so we do not consider it in the experimental chapter.

4.2 Enumeration of Partitions

Another way of solving Problem (dmEm) exactly is to compute all possible partitions of scenarios, to calculate the induced solutions, and to return the best one of these.

The induced set of solutions is clearly optimal for a given partition and therefore this strategy leads to an optimal solution. Moreover, it is easy to verify that it suffices to consider partitions without empty subsets. For fixed l or fixed $l - K$, this leads to an oracle polynomial time algorithm, in case the induced set of solutions can be computed by applying an oracle for (P).

Theorem 4.1. *The Discrete Min-E-Min Problem with a certain feasible set X can be polynomially reduced to the underlying certain problem* (P) *if either the number of scenarios l or the difference $l - K$ is fixed.*

Proof. The number of partitions of $\{1, \ldots, l\}$ into K non-empty subsets is the Sterling number of the second kind,

$$S_{l,K} = \frac{1}{K!} \sum_{j=0}^{K} (-1)^{K-j} \binom{K}{j} j^l \ .$$

If $K \geq l$, we can compute an optimal solution for each scenario independently, using the oracle for (P). In particular, when l is bounded, we may assume that K is bounded as well, and hence also $S_{l,K}$. For the second assertion, note that $S_{l,K} \in O(l^{2(l-K)})$ is polynomially bounded for fixed $l - K$ and all necessary partitions can be computed in polynomial time as we will see in the following. □

In case of uncertain constraints we do not necessarily get an oracle-polynomial time algorithm but obtain an algorithm that has to solve another problem a polynomial number of times e.g. a Multidimensional Knapsack Problem for the Knapsack Problem as underlying problem.

Enumerating all of these partitions is possible with a running time that is a polynomial of the number of computed partitions. To understand this we cite a Lemma from [31]:

Lemma 4.2. *For all $l \in \mathbb{N}$ there exists a bijection from the set of partitions of a set $\{s_1, \ldots, s_l\}$ to the set of vectors*

$$\{(z_i, z_2, \ldots, z_l) |\ z_1 = 1, z_i \leq \max_{j \leq i} z_j + 1, z_i \in \mathbb{Z}\},$$

where $z_i = j$ if and only if s_i belongs to the j-th subset of the partition.

Intuitively in order to avoid symmetric partitions, every element that has not been assigned at a certain time can only be placed in a subset that is not empty at that time or in the empty subset with the lowest index. This is achieved by the constraint $z_i \leq \max_{j \leq i} z_j + 1$. Following this rule, the first element that is placed, is always placed in the first subset. To achieve that the number of subsets is at most K, the additional condition $z_i \leq K$ has to hold. If it is also required that no subset is empty, before determining the value of z_i, it has to be checked if the number of remaining elements (dimensions d that have an undetermined value z_d) is equal to the number of remaining empty subsets (numbers from 1 to K that are not used at that time) and if this is true, the remaining dimensions of the vector are set following the rule $z_d = z_{d-1} + 1$ for all $d \geq i$. Having a vector that is describing a partition, it is easy to build the corresponding partition by placing the i-th element into the z_i-th subset. Orlov [31] presented a recursive algorithm to compute all vectors corresponding to partitions consisting of K non-empty subsets. In the following we present an algorithm that works without recursions and is faster in practice. This algorithm will work with a depth-first search, even though with a breadth-first search, the pseudo code of the algorithm is easier to understand. The advantage of the depth-first search for using it within an algorithm for Problem (dmEm) is that after enumerating any partition, we can obtain a solution for Problem (dmEm). Hence, in instances in which the computation of all relevant partitions is not possible within the time limit, at least some solutions can be produced by the algorithm. The running time of the algorithm using depth-first search and of the version using breadth-first search is comparable.

Figure 4.1 illustrates the algorithm. The variable c in the algorithm corresponds to the current dimension, in which the value has to be computed in that step, and the vector v is the current vector the algorithm is working with. The list of lists of vectors Q is supposed to store all vectors, in which the first i entries are already fixed in the i-th list. In line 5 the variable *used* corresponds to the number of subsets that are already used. In the lines 6 to 10 the case that the number of remaining elements $(l - c + 1)$ equals the number of empty subsets $(K - used)$ is described. In that case, every element is placed in its own subset. The opposite case is described in lines 12-25. In line 12 the maximum number max of the c-th entry of the current vector v is computed. In line 13-17 we create all possible extensions of v that can arise by fixing the c-th element except for one and store them in the correct list of Q. In line 18 we perform the last possible fixation and instead of storing it, we go on working with it (depth-first search). If all l entries of the current vector v are fixed, the corresponding partition and the induced set of solutions is computed (line 20) and v is set to null, so that the algorithm knows that it has to take a new current vector out of Q (line 30-31). If a list of Q is empty, when the algorithm wants to take a new current vector, c is decreased (line 28) and the algorithm tries it again. When all lists in Q are empty and the current vector v is null, c will reach the value 0 and the algorithm stops.

The Sterling number of the second kind increases very fast in K and l but decreases again when K comes closer to l. Therefore, this solution method is very effective for small K and l or for small $l - K$, whereas the number of dimensions of the underlying problem n does not affect the running time explicitly, but only via a potentially longer solution time for the underlying problem.

Algorithm for Problem (dmEm) **by enumeration of partitions**

1: $c = 1$, $v = \{0\}^l$
2: Let Q be a list of $l - 1$ lists of vectors
3: **while** $c > 0$ **do**
4: **if** $v \neq null$ **then**
5: $used = \max\{v_i | i <= c\}$
6: **if** $l - c + 1 = K - used$ **then**
7: **while** $c \leq l$ **do**
8: $v_c = used + 1$, $c++$, $used++$
9: **end while**
10: Go to line 20
11: **else**
12: $max = \min\{used + 1, K\}$
13: **for** $i = 1 \rightarrow i = max - 1$ **do**
14: Create a copy $\hat{v}$ of v
15: $\hat{v}_c = i$
16: Add $\hat{v}$ to the (c)-th list of Q
17: **end for**
18: $v_c = max$
19: **if** $c = l$ **then**
20: Use v to obtain a partition p and compute set of solutions induced by p and store it if the objective value is better than the current best
21: $v = null$
22: **else**
23: $c++$
24: **end if**
25: **end if**
26: **else**
27: **if** the c-th list of Q is empty **then**
28: $c--$
29: **else**
30: Remove the last element $\hat{v}$ of the c-th list of Q
31: $v = \hat{v}$
32: **end if**
33: **end if**
34: **end while**

Figure 4.1: Algorithm for Problem dmEm by enumeration of partitions.

4.3 IQP-based Branch-And-Bound

A more sophisticated approach than enumeration is to model the problem as an integer quadratic program (IQP). To this end, we use two types of variables: for $j \in \{1, \ldots, K\}$, the vector x_j describes the j-th solution to be computed. Moreover, we need additional variables $y_{ij} \in \{0, 1\}$ to model an assignment of scenarios to the solutions: if $y_{ij} = 1$, then the scenario i is covered by solution j, i.e., in scenario i one best solution out of $x_1, \ldots, x_K$ is x_j. The y-variables are necessary to correctly calculate the costs induced by the scenario and to select the corresponding constraints. We obtain the following IQP formulation of Problem (dmEm):

$$\begin{aligned} \min \quad & \sum_{i=1}^{l} \sum_{j=1}^{K} y_{ij} \xi_i^\top x_j & & \\ \text{s.t.} \quad & \sum_{j=1}^{K} y_{ij} = 1 & & \forall\ 1 \le i \le l \\ & y_{ij} = 1 \Rightarrow x_j \in X_i & & \forall\ 1 \le i \le l, \forall\ 1 \le j \le K \\ & y_{ij} \in \{0, 1\} & & \forall\ 1 \le i \le l, \forall\ 1 \le j \le K \end{aligned} \tag{IQP}$$

Note that the objective function of (IQP) is indeed quadratic, since both y_{ij} and x_j are variables here; for each scenario i and each solution j, we count the corresponding cost $\xi_i^\top x_j$ if and only if $y_{ij} = 1$. The first set of constraints in (IQP) ensures that each scenario is covered by exactly one of the solutions $x_1, \ldots, x_K$. The second set of constraints guarantees that each scenario is covered by a solution that is feasible for that scenario. Note that for problems with a certain feasible region these constraints do not depend on y_{ij} anymore and hence can be simplified to the linear constraints $x_j \in X$ for all $1 \le j \le K$. We now discuss possible methods for handling this kind of constraints when using a general-purpose MILP-solver. Assuming that the feasible set of each scenario i is a polyhedron $X_i = \{x \in X : a_{i,r}^\top x \ge b_{i,r}\ \forall\ 1 \le r \le m\}$ characterized by m linear constraints, the first option is to apply linearization, and replace the constraints by

$$a_{i,r}^\top x_j + BIGM(1 - y_{ij}) \ge b_{i,r}\ \forall\ i \in \{1, ..., l\}, j \in \{1, ..., K\}, r \in \{1, ..., m\}$$

where BIGM is a large enough coefficient. Note that BIGM always exists because the set of feasible solutions X_i is bounded in every scenario i.

A second option is to exploit the availability of commercial MILP-solvers to use the so-called indicator constraints, which are a modeling tool to express disjunctive conditions. We can then replace the second set of constraints by

$$y_{ij} = 1 \Rightarrow a_{i,r}^{\top} x_j \geq b_{i,r} \; \forall \; i \in \{1, ..., l\}, j \in \{1, ..., K\}, r \in \{1, ..., m\}$$

Preliminary computational experiments showed that, on our benchmark, the two approaches were comparable, though the former was more robust from a numerical viewpoint, and was adopted in subsequent experiments.
In Section 2.3 we presented two different ways of handling the quadratic objective function. It is possible to force Cplex to use one of these strategies by setting the parameter 'qtolin'. Setting 'qtolin' to one will force Cplex to use linearization, whereas setting it to zero results in the use of quadratic relaxations. In our experiments in Chapter 6, we tested the impact of using different values of 'qtolin' on the running time.
It is easy to verify that the integrality of x-variables does not need to be required in (IQP) as long as y-variables are binary and a complete polyhedral description of $\text{conv}(X)$ is given.

In the following, we want to present some possible additional improvements that have not been realized in our experimentation due to the fact that implementing them inside an off-the-shelf MILP-solver is not always possible or beneficial. The branching decisions only need to take y-variables into account. When all y-variables are fixed, an algorithm for computing the induced set of solutions can be used to compute the K solutions $x_1, \ldots, x_K$: the solution x_j can be computed with

$$x_j \in \operatorname*{argmin}_{x \in \widetilde{X}_j} \sum_{y_{ij}=1} \xi_i^{\top} x \text{ with } \widetilde{X}_j := \bigcap_{y_{ij}=1} X_i.$$

As long as some y-variables are not fixed yet, it is reasonable here to create K children instead of applying the standard binary branching: these children correspond to the possible values of the vector $(y_{i1}, y_{i2}, \ldots, y_{iK})$, where exactly one of the variables may be one while all others are zero.

Clearly, the problem (IQP) contains a high degree of symmetry. Part of the symmetry can be easily broken by requiring $y_{ij} = 0$ for $j > i$. Nevertheless, even though this less sophisticated symmetry reduction can be implemented easily, additional preliminary computations using Cplex suggested that in most

of the instances it is faster and more stable from a numerical point of view to let Cplex handle the symmetry reduction. It is also possible to reduce the number of children further such that the number of leaves of the Branch-and-bound tree on the last level (assuming that no branch has been cut before) is exactly $S_{l,K}$, by adapting the methods from Section 4.2. This is illustrated by Figure 4.2 for an example with $l = 4$ and $K = 3$, in which the node $y_{i,j}$ represents the fixation of $y_{i,j}$ to one and of $y_{i,h}$ to zero for all $1 \leq h \leq K$ with $h \neq j$. Every node $y_{i,j}$ can be seen as the decision to put scenario i in subset j. Let $p(y_{i,j})$ be the set of all vertices on the direct way from $y_{i,j}$ to $y_{1,1}$ including $y_{i,j}$, in the following denoted by predecessors. Let

$$j_{max}(y_{i,j}) := \max\{K, \max\{\ b \mid y_{a,b} \in p(y_{i,j})\} + 1\}.$$

Let $c(y_{i,j})$ be the set of children of node $y_{i,j}$. In order to break the symmetry we can define $c(y_{i,j}) := \{y_{i+1,1}, \ldots, y_{i+1,j_{max}(y_{i,j})})\}$. This rule ensures that every node has at most K children and if a scenario is placed into an empty subset, the index of this subset and the highest index of the subsets so far differ just by 1. Therefore, we consider for the next scenario just the subsets that are non-empty and one of the empty subsets instead of all of them, when we decide in which subset the next scenario has to be placed, thus symmetric solutions disappear. For obtaining solutions that have no empty subsets, we count in every node how many subsets are empty in the actual fixation and how many scenarios are not fixed yet and if these numbers are the same, each of the missing scenarios is placed into its own subset. The additional avoidance of empty subsets can be seen in Figure 4.3. The resulting maximum number of nodes on the lowest level is exactly $S_{l,K}$ if both rules are used and K^l if both rules are not used. In the example, instead of 81 nodes on the last level the improved branching tree has only six.

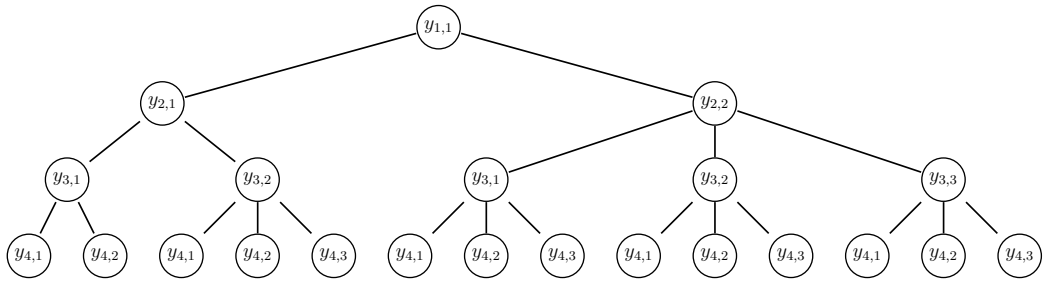


Figure 4.2: Example of the improved branching tree with symmetry breaking with four scenarios and $K = 3$.

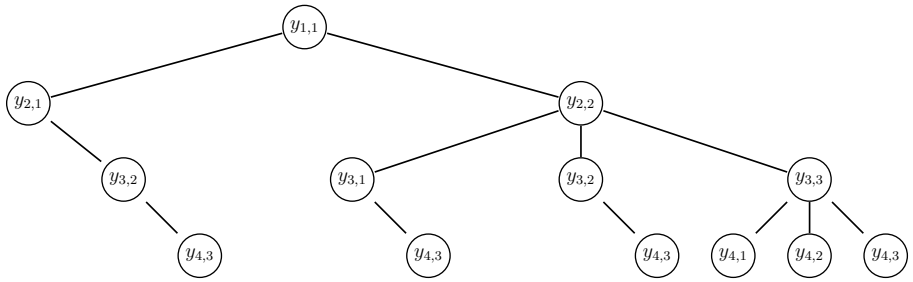


Figure 4.3: Example of the improved branching tree with symmetry breaking and avoidance of empty subsets with four scenarios and $K = 3$.

4.4 Set Partitioning Formulations

We now introduce two novel formulations for (dmEm). Both formulations are pure 0-1 linear programs and involve an exponential number of variables.

Formulation 1. Let $\mathcal{F}$ be the family of all subsets of the scenarios. Given subset $S \in \mathcal{F}$, the associated cost is determined by computing the solution induced by the scenarios in S. In the following, we will assume that an oracle is available for solving this problem. Note that as already mentioned above, this problem is not necessarily equivalent to the underlying problem, e.g., a Knapsack Problem can become a Multidimensional Knapsack Problem. For notational convenience, assume that $\mathcal{F}$ also contains the empty subset and that the associated cost is zero. Introducing a binary variable ϑ_t for each feasible

subset of scenarios $S_t \in \mathcal{F}$ and denoting its cost by c_t, we get:

$$\begin{aligned} \min \quad & \sum_{t=1}^{|\mathcal{F}|} c_t \, \vartheta_t \\ \text{s.t.} \quad & \sum_{t=1}^{|\mathcal{F}|} z_{it} \, \vartheta_t = 1 \quad \forall \; 1 \leq i \leq l \\ & \sum_{t=1}^{|\mathcal{F}|} \vartheta_t = K \\ & \vartheta_t \in \{0,1\} \qquad \forall \; 1 \leq t \leq |\mathcal{F}|, \end{aligned} \tag{SPP1}$$

where $\vartheta_t = 1$ if and only if subset S_t of scenarios is selected. The constant z_{jt} is equal to one if and only if the j-th scenario is in the t-th subset and so the first set of constraints enforces that every scenario is in exactly one of the chosen subsets. Problem (dmEm) requires to select at most K solutions. Nevertheless, by demanding exactly K subsets instead, which is more convenient for computing the dual program, the optimal solution does not change: if less than K subsets are chosen, any subset can be split into more than one subset without worsening the objective value. The second constraint now guarantees that exactly K subsets are used. Therefore, the first and second constraint combined ensure that the variables ϑ_t describe a partition of the scenarios into K subsets. By taking the optimal solution for each selected subset, we obtain a complete solution to Problem (dmEm), whose cost is given by the sum of the costs of the selected subsets.

Formulation 2. We can derive another formulation for Problem (dmEm) working in the space of solutions. For each scenario i, let X_i denote the set of all feasible solutions, and let $Q = \bigcup_i X_i$. Let r_{ip} be the cost of solution p in scenario i; possibly $r_{ip} = +\infty$ if $p \notin X_i$. For each solution $p \in Q$, introduce a binary variable σ_p, taking value 1 if solution p is among the K chosen solutions, and 0 otherwise. In addition, for each solution p and scenario i, we introduce the variable ρ_{ip}, which is one if scenario i is covered by solution p and 0 otherwise. Thus, Problem (dmEm) can be formulated as follows:

$$
\begin{aligned}
\min \quad & \sum_{i=1}^{l} \sum_{p=1}^{|Q|} r_{ip}\, \rho_{ip} & & \\
\text{s.t.} \quad & \sum_{p=1}^{|Q|} \rho_{ip} = 1 & & \forall\ 1 \leq i \leq l \\
& \sigma_p \geq \rho_{ip} & & \forall\ 1 \leq i \leq l, \forall\ 1 \leq p \leq |Q| \\
& \sum_{p=1}^{|Q|} \sigma_p = K & & \\
& \sigma_p \in \{0,1\} & & \forall\ 1 \leq p \leq |Q| \\
& \rho_{ip} \in \{0,1\} & & \forall\ 1 \leq i \leq l, \forall\ 1 \leq p \leq |Q|
\end{aligned}
\tag{SPP2}
$$

The first set of constraints ensures that every scenario is covered by exactly one solution. The second set of constraints guarantees that only solutions that are selected cover scenarios, while the third set of constraints assures that exactly K solutions are chosen.

Both (SPP1) and (SPP2) require a large amount of enumeration, either of all the feasible subsets of scenarios or of all feasible solutions. In practice, this may be computationally very expensive or even impossible when large-size instances are considered. In such cases, one has to resort to column generation techniques (see Section 4.5.1) and Branch-and-price algorithms. The models can be used for computing heuristic solutions as well, as it will be described in Chapter 5.

We conclude this section showing that the two set partitioning models have the same tightness in terms of continuous relaxation.

Theorem 4.3. *Models* (SPP1) *and* (SPP2) *are equivalent in terms of continuous relaxation.*

Proof. In the following we will denote by $(\text{SPP1})_c$ and $(\text{SPP2})_c$ the continuous relaxations of models (SPP1) and (SPP2), respectively. To prove the statement we show that, given an optimal solution of $(\text{SPP1})_c$, there exists a feasible

solution of $(\text{SPP2})_c$ with the same value or with a better value, and vice versa.

Given an optimal solution ϑ of $(\text{SPP1})_c$, the corresponding solution for $(\text{SPP2})_c$ is defined as follows:

1. **for** $p := 1$ **to** $|Q|$ **do**

   $\sigma_p := 0$;

   for $i := 1$ **to** l **do** $\rho_{ip} := 0$;

2. **for each** $S_t \in \mathcal{F}$ such that $\vartheta_t > 0$ **do**

   let p be the solution induced by subset S_t;

   $\sigma_p := \sigma_p + \vartheta_t$;

   for $i := 1$ **to** l **do** $\rho_{ip} := \rho_{ip} + z_{it}\,\vartheta_t$

Step **1** initializes an empty solution, while the second step defines the correct value for the σ and ρ variables. In this step, only subsets S_t that are actually selected in solution ϑ are taken into account. For each such subset, the associated induced solution p is considered. Every positive value ϑ_t is used to increase the value of a single σ variable, hence

$$K = \sum_{t=1}^{|\mathcal{F}|} \vartheta_t = \sum_{p=1}^{|Q|} \sigma_p.$$

Similarly, for each scenario i, each variable ρ_{ip} is increased by ϑ_t when a selected subset S_t is considered such that p is the associated induced solution and $i \in S_t$ (i.e. $z_{it} = 1$). It follows that

$$1 = \sum_{t=1}^{|\mathcal{F}|} z_{it}\,\vartheta_t = \sum_{p=1}^{|Q|} \rho_{ip},$$

hence solution (σ, ρ) is feasible to $(\text{SPP2})_c$. Finally, consider a selected subset S_t, which contributes with a cost $c_t\vartheta_t$ to the objective function of $(\text{SPP1})_c$. Let p be the associated induced solution and note that, by definition, we have

$$c_t = \sum_{i \in S_t} r_{ip}.$$

Increasing each variable ρ_{ip} $(i \in S_t)$ by ϑ_t produces a cost increase equal to

$$\sum_{i \in S_t} r_{it}\,\vartheta_t = \vartheta_t \sum_{i \in S_t} r_{it} = c_t\,\vartheta_t$$

in the objective function of $(\text{SPP2})_c$. Hence, the two solutions have the same cost.

Assume now that (σ, ρ) is an optimal solution for $(\text{SPP2})_c$. The following procedure defines a solution ϑ for $(\text{SPP1})_c$:

1. **for** $t := 1$ **to** $|S|$ **do** $\vartheta_t := 0$
2. **for** $p := 1$ **to** $|Q|$ **do**

   for $i := 1$ **to** l **do** $\bar{\rho}_{ip} := \rho_{ip}$;

   while there exists a $i \in \{1, \ldots, l\}$ such that $\bar{\rho}_{ip} > 0$ **do**

   $S_t := \{i : \bar{\rho}_{ip} > 0\}$, $min := \min_{i \in S_t} \bar{\rho}_{ip}$, $\vartheta_t := \vartheta_t + min$

   for each $i \in S_t$ **do** $\bar{\rho}_{ip} := \bar{\rho}_{ip} - min$

For every solution p, the sum of all ϑ variables is increased by the maximum value of $\bar{\rho}_{ip}$ over all scenarios i. This value is a lower bound for σ_p and therefore in an optimal solution σ_p will take exactly this value. Therefore, we have

$$K = \sum_{p=1}^{|Q|} \sigma_p = \sum_{t=1}^{|\mathcal{F}|} \vartheta_t.$$

If in the procedure for a subset S_t the $\bar{\rho}$ values are decreased each by an amount of min, the total decrease of the sum of $\bar{\rho}$ variables is $min \cdot |S_t|$. At the same time we increase ϑ_t by min. Note that exactly $|S_t|$ many z_{it} variables have value 1 and therefore the increase of

$$\sum_{t=1}^{|\mathcal{F}|} z_{it}\, \vartheta_t$$

is $min|S_t|$. When all $\bar{\rho}$ variables have value zero, the algorithm stops and no ϑ variables can be increased anymore. Therefore, the total decrease of $\bar{\rho}$ variables is equal to the total increase of

$$\sum_{t=1}^{|\mathcal{F}|} z_{it}\, \vartheta_t.$$

Because we initialized each $\bar{\rho}_{ip}$ with the value of ρ_{ip}, the total decrease of $\bar{\rho}_{ip}$ variables is equal to

$$\sum_{p=1}^{|Q|} \rho_{ip},$$

when the algorithm ends and so we have

$$1 = \sum_{p=1}^{|Q|} \rho_{ip} = \sum_{t=1}^{|\mathcal{F}|} z_{it}\, \vartheta_t.$$

Let us assume without loss of generality that for a solution $\hat{p}$ two $\bar{\rho}$ variables $\bar{\rho}_{i\hat{p}}$ and $\bar{\rho}_{j\hat{p}}$ have a value larger than zero. The algorithm takes the set $S_t := \{i, j\}$ and computes its optimal induced solution p with value c_t and increase ϑ_t by $min := \min_{i \in S_t} \bar{\rho}_{ip}$. The increase of the objective value of (SPP1) is $min\, c_t$, which is smaller than $min(r_{i\hat{p}} + r_{j\hat{p}})$ because p is the optimal solution for S_t and $\hat{p}$ is not necessarily the optimal solution. Because we minimize, a smaller increase leads to a better solution value and therefore the solution value of (SPP1) is at least as good as the solution value of (SPP2).

□

For a better understanding of the transformation the procedure of transferring solutions from (SPP1) to (SPP2) is shown in Example 4.4.

Example 4.4. Given the following instance of Problem (dmEm):

1. $K = 2$, $l = 4$, $n = 3$
2. $X_1 = X_2 = X_3 = X_4 = \{0, 1\}^3$
3. $\xi_1 = (-1, -1, 1)^\top$, $\xi_2 = (1, 2, -2)^\top$, $\xi_3 = (-2, 1, 0)^\top$, $\xi_4 = (2, -2, -1)^\top$

We enumerate all subsets of scenarios and the corresponding costs:

Index i	S_i	c_i	Index i	S_i	c_i
1	$\{1,2,3,4\}$	-2	9	$\{2,3\}$	-3
2	$\{1,2,3\}$	-3	10	$\{2,4\}$	-3
3	$\{1,2,4\}$	-3	11	$\{3,4\}$	-2
4	$\{1,3,4\}$	-3	12	$\{1\}$	-2
5	$\{2,3,4\}$	-3	13	$\{2\}$	-2
6	$\{1,2\}$	-1	14	$\{3\}$	-2
7	$\{1,3\}$	-3	15	$\{4\}$	-3
8	$\{1,4\}$	-3	16	$\{\}$	0

We enumerate all possible solutions and their objective values in the scenarios:

j	p_j	r_{1j}	r_{2j}	r_{3j}	r_{4j}
1	$(0,0,0)^\top$	0	0	0	0
2	$(0,0,1)^\top$	1	-2	0	-1
3	$(0,1,0)^\top$	-1	2	1	-2
4	$(0,1,1)^\top$	0	0	1	-3
5	$(1,0,0)^\top$	-1	1	-2	2
6	$(1,0,1)^\top$	0	-1	-2	1
7	$(1,1,0)^\top$	-2	3	-1	0
8	$(1,1,1)^\top$	-1	1	-1	-1

Consider the following feasible (and optimal) solution of the relaxation of (SPP1) with value -6:

$$\vartheta = (0,0,0,0,0,0,0.5,0.5,0.5,0.5,0,0,0,0,0,0)^\top.$$

We construct a feasible solution of the relaxation of (SPP2) using the algorithm in Theorem 4.3 and skip the parts of the algorithm in which no value changes:

- initialization
- Consider $S_7 = \{1,3\}$
  - one optimal induced solution for S_7 is $p_5 = (1,0,0)^\top$
  - $\sigma_5 = 0.5$, $\rho_{15} = 0.5$, $\rho_{35} = 0.5$
- Consider $S_8 = \{1,4\}$
  - one optimal induced solution for S_8 is $p_3 = (0,1,0)^\top$
  - $\sigma_3 = 0.5$, $\rho_{13} = 0.5$, $\rho_{43} = 0.5$
- Consider $S_9 = \{2,3\}$
  - one optimal induced solution for S_9 is $p_6 = (1,0,1)^\top$
  - $\sigma_6 = 0.5$, $\rho_{26} = 0.5$, $\rho_{36} = 0.5$
- Consider $S_{10} = \{2,4\}$
  - one optimal induced solution for S_{10} is $p_2 = (0,0,1)^\top$
  - $\sigma_2 = 0.5$, $\rho_{22} = 0.5$, $\rho_{42} = 0.5$

The resulting solution is clearly feasible for the relaxation of (SPP2) and has a solution value of -6.

Given the following feasible (but not optimal) solution of the relaxation of (SPP2) with value -5:

- $\sigma_2 = 0.5$, $\rho_{22} = 0.5$, $\rho_{32} = 0$, $\rho_{42} = 0.5$
- $\sigma_6 = 0.5$, $\rho_{16} = 0.25$, $\rho_{26} = 0.5$, $\rho_{36} = 0.75$
- $\sigma_8 = 0.5$, $\rho_{18} = 0.75$, $\rho_{38} = 0.25$, $\rho_{48} = 0.5$

All variables that are not mentioned take the value zero. We construct a feasible solution of the relaxation of (SPP1) using the algorithm in Theorem 4.3 and skip the parts of the algorithm in which no value changes:

- initialization
- $p = 2$:

1. $\bar{\rho}_{22} = 0.5$, $\bar{\rho}_{32} = 0$, $\bar{\rho}_{42} = 0.5$ 2. $S_t = \{2, 4\} = S_{10}$ 3. $\vartheta_{10} = 0.5$	4. $\bar{\rho}_{22} = 0$, $\bar{\rho}_{42} = 0$

- $p = 6$:

1. $\bar{\rho}_{16} = 0.25$, $\bar{\rho}_{26} = 0.5$, $\bar{\rho}_{36} = 0.75$ 2. $S_t = \{1, 2, 3\} = S_2$ 3. $\vartheta_2 = 0.25$	4. $\bar{\rho}_{16} = 0$, $\bar{\rho}_{26} = 0.25$, $\bar{\rho}_{36} = 0.5$ 5. $S_t = \{2, 3\} = S_9$ 6. $\vartheta_9 = 0.25$

7. $\bar{\rho}_{26} = 0$, $\bar{\rho}_{36} = 0.25$
8. $S_t = \{3\} = S_{14}$
9. $\vartheta_{14} = 0.25$

10. $\bar{\rho}_{36} = 0$

- $p = 8$:

1. $\bar{\rho}_{18} = 0.75$, $\bar{\rho}_{38} = 0.25$, $\bar{\rho}_{48} = 0.5$
2. $S_t = \{1, 3, 4\} = S_4$
3. $\vartheta_4 = 0.25$

4. $\bar{\rho}_{18} = 0.5$, $\bar{\rho}_{38} = 0$, $\bar{\rho}_{48} = 0.25$
5. $S_t = \{1, 4\} = S_8$
6. $\vartheta_8 = 0.25$

7. $\bar{\rho}_{18} = 0.25$, $\bar{\rho}_{48} = 0$
8. $S_t = \{1\} = S_{12}$
9. $\vartheta_{12} = 0.25$

10. $\bar{\rho}_{18} = 0$

The resulting solution

$$\vartheta = (0, 0.25, 0, 0.25, 0, 0, 0, 0.25, 0.25, 0.5, 0, 0.25, 0, 0.25, 0, 0)^\top$$

is feasible for the relaxation of (SPP1) and has a solution value of -5.5.

In the example we chose a non-optimal solution because a solution in which corresponding σ and ρ values have different fractions, is more interesting and beneficial for the understanding of the proof of Theorem 4.3. Such a solution occurs only in larger instances as optimal solution.
We want to conclude this section by emphasizing that formulation (SPP1) is used in a Branch-and-price algorithm (see Section 4.5.3) and (SPP2) as a refinement procedure for heuristic solutions (see Chapter 5).

4.5 Branch-and-Price

In this section we present an exact algorithm based on one of the set partitioning formulations introduced in Section 4.4. Both formulations include an exponential number of variables and therefore column generation techniques can be useful for solving the associated linear programming relaxations. The first model is more convenient for this kind of approach because in the second model generating new columns leads to an additional creation of new rows (see the second set of constraints). For this reason and because the models provide the same lower bound, in this section we will concentrate on the first formulation only. The developed algorithm is a Branch-and-price scheme that uses, at each node, the methods described in the next section for computing the column generation lower bound and adopts the branching strategy given later in this section.

4.5.1 Column Generation

By dropping the binary requirements, the domain of the variables of model (SPP1) can be replaced by $\vartheta_t \geq 0$ for all $1 \leq t \leq |\mathcal{F}|$, and hence the dual of the resulting model is:

$$\begin{array}{ll} \max & \sum_{i=1}^{l} \lambda_i + K\mu \\ \text{s.t.} & \sum_{i=1}^{l} z_{it}\lambda_i + \mu \leq c_t \quad \forall\ 1 \leq t \leq |\mathcal{F}| \\ & \mu \geq 0, \end{array}$$

where λ_i for all $1 \leq i \leq l$ and μ are the dual variables associated with the constraints of (SPP1).

Column generation defines a restricted master problem, in which a subset of the ϑ_t variables is used, and solves this continuous problem to optimality. Given the associated dual variables, column generation asks for a variable (*column*) that has a negative reduced cost, i.e., whose associated dual constraint is violated by the current dual solution. For a given subset S of scenarios, the associated dual constraint is violated if

$$\sum_{i \in S_t} \lambda_i + \mu > c_t$$

where c_t is the cost of the solution induced by S_t. This solution must satisfy all constraints of the scenarios in the subset. The problem of determining

the subset S_t whose associated dual constraint is maximally violated can be formulated by introducing, for each scenario i, a binary variable π_i taking value 1 if and only if scenario i belongs to subset S_t. The reduced cost $\bar{c}_t$ of this subset is given by the optimal solution of the following problem

$$\begin{array}{lllll} \min & \sum_{i=1}^{l} ({\xi_i}^\top x - \lambda_i)\pi_i - \mu & & & \\ \text{s.t.} & \pi_i = 1 & \Rightarrow & x \in X_i & \forall\ 1 \leq i \leq l \\ & \pi_i \in \{0,1\} & & & \forall\ 1 \leq i \leq l. \end{array} \tag{PPSPP1}$$

If the optimal solution of the model has a negative value, then the subset of scenarios $S_t = \{i \mid \pi_i = 1\}$ corresponds to a variable with negative reduced cost and should be added on-the-fly to the current restricted master problem.

The model above includes a set of non-linear constraints that can be again either linearized using a BIGM coefficient or formulated as indicator constraints (see Section 4.3). In our experiments with a general-purpose solver, we experienced better performances using the first strategy. Note that the pricing problem is NP-hard, as shown by the following reduction from the Min-E-Min Completion Problem (mEmC) (see Section 7).

Theorem 4.5. *The pricing problem* (PPSPP1) *is equivalent to* (mEmC) *with* $K_f = \bar{K} = 1$.

Proof. For distinguishing the ξ of the pricing problem and the ξ of Problem (mEmC) we denote the latter by $\bar{\xi}$. We can reformulate the objective function of the pricing problem as

$$\min \sum_{i=1}^{l} {\xi_i}^\top x \pi_i + \sum_{i=1}^{l} \lambda_i (1 - \pi_i) - \sum_{j=1}^{l} \lambda_i - \mu$$

where the last two terms are constant and do not depend on the variables x and π.

We now show that Problem (mEmC) can be reduced to the pricing problem. Given an instance of (mEmC), we define an instance of the pricing problem with $n+1$ dimensions and $l+1$ scenarios. Every scenario $i \leq l$ gets the feasible set $(X_i \cup \{\mathbf{0}_n\}) \times \{0,1\}$ and scenario $l+1$ gets the feasible set $X_{l+1} = \{0,1\}^{n+1}$. The objective vector of each scenario $i \leq l$ is

$$(\xi_i)_d := \begin{cases} \bar{\xi}_i & \text{if } d < n+1, \\ \lambda_i & \text{otherwise} \end{cases}$$

and the objective vector of the last scenario is

$$(\xi_{l+1})_d := \begin{cases} 0 & \text{if } d < n+1, \\ \text{BIGM} & \text{otherwise,} \end{cases}$$

where

$$\text{BIGM} > \sum_{\lambda_i < 0} -\lambda_i.$$

Finally, we define the fixed solution

$$\bar{x}_d := \begin{cases} 0 & \text{if } d < n+1, \\ 1 & \text{otherwise.} \end{cases}$$

A scenario i that is covered by the fixed solution contributes to the objective value with λ_i if $i \leq l$, and with BIGM otherwise. Let x_f be the free solution determined by solving Problem (mEmC) with $K_f = \bar{K} = 1$. Because of the choice of BIGM, the free solution must have $(x_f)_{n+1} = 0$, i.e., the solution covers the last scenario with zero cost. For the remaining scenarios $i \in \{1, \ldots, l\}$, a scenario will be covered by the free solution x_f if $x_f \in X_i$ and

$$\sum_{d=1}^{n} (\bar{\xi}_i)_d (x_f)_d < \lambda_i$$

holds and by the fixed solution otherwise. In other words, the scenarios that are covered by the free solution determine the set of π variables and the free solution restricted to the first n dimensions is equal to the x_f vector of the pricing problem. Hence, both problems have the same objective value apart the constant part and the optimal solution of the pricing problem can be computed with the optimal solution of Problem (mEmC).

Now we want to reduce Problem (mEmC) to the pricing problem. To this aim, given a fixed solution $\bar{x}$, we set $\lambda_i = (\bar{\xi}_i)^\top \bar{x}$ for each scenario i, and use the same set of scenarios in both problems. It is easy to see that the objective functions of the two problems are equivalent to each other and the free solution of Problem (mEmC) is equivalent to the x variables of the pricing problem. □

4.5.2 Heuristic Pricing

Having an NP-hard pricing problem, it makes sense to solve it using a heuristic algorithm, resorting to an exact method only in case the former failed in producing a variable with negative reduced cost.

Our heuristic algorithm is described in Figure 4.4. The heuristic pricing algorithm stops if one column with negative reduced costs is found. Of course, it is also possible to go on searching for more columns with negative reduced costs and add the one with the lowest reduced costs or add more columns at the same time. Nevertheless, in our implementation we experienced better results using the strategy to add just the first column with negative reduced costs, because the running time for solving the master problem in our case has no significant effect on the total running time of the column generation algorithm.

Note that the reduced costs of a column (variable) consist of the costs of the induced solutions and of a linear term depending on the dual variables. Computing the first part can be very time consuming depending on the underlying problem, whereas the second part can be computed without a significant loss of time. Observe that only the second part of the reduced costs changes through the column generation process. Therefore, by creating a hash table with sets of scenarios as keys and the corresponding induced solutions and costs as values, one can compute in negligible time the reduced cost of a column involving a subset of the scenarios that occurred before. Every time an induced solution is computed, we store it and its cost in this table.

In the first six lines of the heuristic, we compute the reduced cost for each variable corresponding to a subset of scenarios stored in the hash table until a variable with negative reduced cost is found. If we find such variable, we stop the heuristic and add it to the master problem, which is then re-optimized. Our computational experiments (see Section 6) show that this may have a dramatic impact on the performance of the overall procedure, mainly when the underlying problem is hard to solve. Using hash tables makes the column generation procedure faster and faster the more iterations are executed. As a consequence, the method shows a speedup during the exploration of the enumeration tree in the Branch-and-price algorithm because large parts of these hash tables can be passed on from the parent node to a child node. In order to have a lot of values in this hash table, it is a good improvement to save also the value of all the subsets that are regarded in the exact pricing, no matter which algorithm one uses. For an oracle-based algorithm for the exact pricing, this is trivial. Solving the exact pricing problem with a general-purpose MIQP-

solver, it is possible that it stores solutions and its values in a solution pool that can be accessed without additional efforts after the problem is solved and in this way the hash table can be filled in order to get an additional speedup. The heuristic pricing algorithm has two parameters p_1 and p_2 that can be changed in order to optimize the performance. In line seven of the algorithm, p_1 feasible subsets are sampled at random and for each of them a heuristic cost is computed. The heuristic cost can be computed by using a heuristic instead of the exact algorithm for computing the induced set of solutions and is therefore much faster than the latter. According to this heuristic cost, the best p_2 subsets are stored in the list l_1 for further investigation. In the lines eight to fifteen, the algorithm iterates over the subsets in l_1.
Another improvement of the algorithm is to compute a relaxed reduced cost, a lower bound on the reduced cost, and skip the subset if this is not negative because in this case the real reduced cost cannot be negative. Lower bounds on the reduced costs can be computed in general much faster compared to the exact reduced costs. Therefore it is beneficial to spend some time in computing relaxed reduced costs if it sometimes avoids the computation of the exact reduced cost. If there is no fast method for computing a lower bound on the exact reduced costs, this step can also be skipped. Finally, if a solution with negative exact reduced costs is found, we add it to the master problem (line 11) and the algorithm terminates.

4.5.3 Branching Scheme

As already mentioned, we used a Branch-and-price algorithm for computing an optimal solution of model (SPP1). At the root node, a feasible solution is computed using a heuristic algorithm. At each node, the continuous relaxation of the current subproblem is solved, producing a lower bound on the optimal solution value of the current subproblem. If the solution of the relaxation is integer, the incumbent may not be improved and the node is fathomed. Otherwise, if the lower bound is smaller than the incumbent value, the optimal solution of the continuous relaxation is used to branch, producing two subproblems that are explored according to a depth-first strategy. In particular, let a be a scenario that is included in more than one subset t with $\vartheta_t > 0$ in the current fractional solution. Note that this scenario always exists in a

Heuristic for generating columns

```
1: for each subset s, for which the value of the induced solution is already
   known do
2:   if Reduced costs of s are negative then
3:     Add a new column for s to the master problem
4:     return
5:   end if
6: end for
7: Sample p_1 random, feasible subsets and save the p_2 best of them according
   to a heuristic reduced cost in list l_1
8: for Subset s ∈ l_1 do
9:   if Relaxed reduced costs of s are negative then
10:    if Exact reduced costs of s are negative then
11:      Add a new column for s to the master problem
12:      return
13:    end if
14:  end if
15: end for
```

Figure 4.4: Heuristic pricing algorithm

non-integer solution because of the first constraint of SPP1. Let S_1 and S_2 be two of these subsets, and let b be a scenario that belongs to the symmetric difference of S_1 and S_2. In the first node we impose that the scenarios a and b belong to the same subset, while in the other node we forbid it.
Observe that this branching rule only affects the pricing subproblems at the descendant nodes, whereas the master problem is influenced only implicitly. However, a nice property of this branching scheme is that handling these modifications is very easy both in the heuristic pricing problem, and in the exact pricing problem. Indeed, in the latter case, it is enough to enforce in (PPSPP1) the additional constraints $\rho_a = \rho_b$ for the first node, and $\rho_a + \rho_b \leq 1$ for the second one. The parent node can also inherit parts of the hash tables that store subsets of scenarios and their induced set of solutions and the corresponding costs. Here one has to pay attention that the subsets that violate the new fixation of the child node are not passed on. Also in the sampling of a new subset

the fixation rules can be easily integrated and so the heuristic algorithm will not produce subsets that violate the fixations. In general, we have a number of candidate scenarios for a and b, thus we use the following tie breaking rules. For each scenario i, we define a score

$$\texttt{score}_i^1 = \sum_{t:i\in S_t, \vartheta_t>0} \min\{1-\vartheta_t, \vartheta_t\},$$

and select the scenario a having maximum score. The idea of this score is to find the "most fractional" scenario by balancing two properties: the difference to the closest integer value and the number of occurrences in fractional subsets. For a given a, we assign to each scenario a second score

$$\texttt{score}_i^2 = \min\{|\{t : \vartheta_t > 0, a \in S_t, i \notin S_t\}|, |\{t : \vartheta_t > 0, a \in S_t, i \in S_t\}|\}$$

and take the scenario b that maximizes this value. The second score tries to find a second scenario so that the number of forbidden subsets that were in the solution in both branches is as balanced as possible.
The resulting Branch-and-price algorithm turns out to be very effective for some underlying problems and combinations of the input parameters, as we can see in the computational experiments in Chapter 6.

Chapter 5

Heuristics

Due to the complexity results for Problem (dmEm) presented in Chapter 3, it makes sense to investigate not only exact algorithms for Problem (dmEm) but also algorithms that do not guarantee an optimal solution but are fast in practice. Therefore, in this chapter we present variants of a heuristic algorithm for solving Problem (dmEm). In the first section, we propose a basic heuristic, which we use as a foundation for all the other variants we discuss in this chapter. In Section 5.2 we propose two different approaches for generating starting partitions that the basic heuristic requires. Different ways to improve the heuristic are presented in Section 5.3 and in Section 5.4 we introduce a refinement procedure that combines solutions from different runs of the heuristic using formulation (SPP2) (see Section 4.4).

5.1 Basic Heuristic

Using the concepts of induced partitions and induced solution sets as introduced in Section 2, we obtain a natural heuristic algorithm by applying both constructions alternately, until no changes occur:

Basic Heuristic

1: Choose an arbitrary partition P of $\{1, \ldots, l\}$.
2: **repeat**
3: Set $P' := P$.
4: Compute the solution set $\mathcal{X}_K$ induced by P'.
5: Compute the partition P induced by $\mathcal{X}_K$.
6: **until** $P = P'$
7: **return** $\mathcal{X}_K$

It is clear from the definitions of induced partitions and induced solution sets that the objective values of the solution sets produced by this algorithm are non-increasing. We claim that the algorithm always terminates, i.e., that it cannot get stuck in a cycle. In fact, by the non-increasing objective values, such a cycle can only exist if all involved solution sets have the same objective value. However, if there is more than one optimal solution for a subset of scenarios, we choose the lexicographically first solution with respect to the entries of the vector, by our definition of the induced solution set given in Section 2.6. Therefore a cycle cannot occur.

The running time used by one iteration of the heuristic is dominated by the time consumed by the oracle. We will see later that it runs very quickly in our experiments. However, we cannot guarantee a polynomial number of iterations. In fact, it is possible to construct cases in which the heuristic uses all $S_{l,K}$ scenarios that are considered for the exact algorithm, which enumerates the partitions of the set of scenarios (see Theorem 4.1).

Example 5.1. Let $K = 2$, $l = 4$ and $n = 13$. Consider the scenarios

$$\begin{aligned}
\xi_1 &= (0, 60, 14, 60, 6, 60, 14, 12, 60, 2, 60, 1, 60)^\top \\
\xi_2 &= (28, 30, 12, 60, 21, 20, 60, 16, 24, 25, 0, 60, -2)^\top \\
\xi_3 &= (60, 16, 16, 18, 13, 60, 2, 60, 0, 60, 22, 20, 60)^\top \\
\xi_4 &= (60, 14, 60, 10, 60, 8, 60, 10, 60, 9, 60, 9, 60)^\top
\end{aligned}$$

and let X_i be the set of all basis vectors $\mathbf{e}_1, \ldots, \mathbf{e}_{13} \in \mathbb{Q}^{13}$ in every scenario i. We start with the partition $P = \{\{\xi_1\}, \{\xi_2, \xi_3, \xi_4\}\}$. The steps of the algorithm are illustrated in the following table:

Iteration	Induced/Starting Partition	Induced solution set	Obj. value
0	$\{\xi_1\}, \{\xi_2, \xi_3, \xi_4\}$	$\mathbf{e}_1, \mathbf{e}_2$	60
1	$\{\xi_1, \xi_2\}, \{\xi_3, \xi_4\}$	$\mathbf{e}_3, \mathbf{e}_4$	54
2	$\{\xi_1, \xi_2, \xi_3\}, \{\xi_4\}$	$\mathbf{e}_5, \mathbf{e}_6$	48
3	$\{\xi_1, \xi_3\}, \{\xi_2, \xi_4\}$	$\mathbf{e}_7, \mathbf{e}_8$	42
4	$\{\xi_3\}, \{\xi_1, \xi_2, \xi_4\}$	$\mathbf{e}_9, \mathbf{e}_{10}$	36
5	$\{\xi_2, \xi_3\}, \{\xi_1, \xi_4\}$	$\mathbf{e}_{11}, \mathbf{e}_{12}$	32
6	$\{\xi_2\}, \{\xi_1, \xi_3, \xi_4\}$	$\mathbf{e}_{13}, \mathbf{e}_{12}$	28
7	$\{\xi_2\}, \{\xi_1, \xi_3, \xi_4\}$	$\mathbf{e}_{13}, \mathbf{e}_{12}$	28

We see that all $S_{4,2} = 7$ partitions are enumerated. In this case, it follows in particular that the optimal solution has been found. □

5.2 Computing Initial Partitions

In our experiments, it will turn out that this heuristic takes only a few milliseconds of running time in general. We can thus start it with more than one initial partition independently, and finally choose the best set of solutions obtained. Finding promising starting partitions is crucial for obtaining good results. We want to present two approaches of computing starting partitions with different advantages that might be preferable in different settings. The first algorithm, which requires an integer $x > 0$ as input, computes deterministic partitions:

Basic Algorithm for computing starting partitions

Add all scenarios to the list L of scenarios that have not been assigned.
for $i = 1; i \leq K; i++$ **do**
  Set $P_i = \emptyset$.
  if $i \leq l \mod K$ **then**
    $size = \frac{l}{K} + 1$
  else
    $size = \frac{l}{K}$
  end if
  for $j = 1; j \leq size; j++$ **do**
    $y = (x \cdot j) \mod |L|$.
    Insert the y-th element of L to P_i and remove it from L.
  end for
end for
return $P = \{P_1, \ldots, P_K\}$

In the computed partition, the size of the smallest and the largest subset differ by at most one. The advantage of this property is that the probability that scenarios change the subset is higher the more scenarios are contained in the subset. Numerical tests suggest that using this algorithm for computing partitions is preferable over taking completely random partitions. Instead of using completely random partitions, one can use the following randomized algorithm:

Clustering Algorithm for computing starting partitions

for every scenario i **do**
  Create a vector v_i by appending θ_i to ξ_i.
end for
Cluster the vectors v with the k-means algorithm into K clusters (see Section 2.5)
if v_i is in the j-th cluster **then**
  Place scenario i in the j-th subset of the partition P
end if
return P

In the first step, the scenarios are clustered using the k-means algorithm.

This algorithm itself starts with a random clustering and therefore the resulting algorithm for generating a partition is not deterministic. This is important because using a deterministic clustering algorithm like single-linkage clustering produces every time the same partition, which is not useful if one wants to start the algorithm with different partitions. Intuitively, scenarios that slightly differ from each other are likely to belong to the same subset in a good partition, while very different scenarios should be put in different subsets. In fact, scenarios with ξ vectors that are close to each other are more likely to have a similar optimizer and scenarios with θ vectors that are close to each other are more likely to be feasible for the same solution.

5.3 Improvement

We now describe a further improvement of the heuristic. For this, we assume that the induced set of solutions always consists of exactly K different solutions, which can easily be ensured by replacing any duplicate solution by a random solution not contained in the set yet or a solution that is optimal for a single scenario. The idea of this improvement is to slightly change the induced partition when the heuristic runs into a local optimum. To increase the probability that the next local optimum is better than the old one, it is reasonable to deteriorate the objective value as few as possible when the partition is changed. To achieve this, the change in the partition consists of a shift of z scenarios from their induced subset into their second best subset. Clearly, the role of z is crucial in this algorithm: for larger values of z, the objective value becomes worse and the probability that the new solution is better decreases. Otherwise, if z is chosen too small, the change might be too small to leave the local optimum. In the experimental evaluation described in the following chapter, we use this heuristic with all presented improvements and $z = 5$, because this turned out to yield the best results in our tests.

Improved heuristic

1: Fix a parameter $z \in \mathbb{N}$.
2: Compute a starting partition P of $\{1, \ldots, l\}$.
3: Run the basic heuristic with starting partition P and denote the result by $\mathcal{X}_{K,1}$.
4: **repeat**
5: Compute the partition P induced by $\mathcal{X}_{K,1}$.
6: Set $\mathcal{X}_{K,2} := \mathcal{X}_{K,1}$.
7: For every scenario, save the best and the second best solution from $\mathcal{X}_{K,2}$, and sort the scenarios in ascending order according to the differences between the two corresponding objectives.
8: Set $i := z$.
9: **while** $i > 0$ **do**
10: Let ξ_l be the i-th element of the sorted list.
11: Change P by removing ξ_j from the subset covered by its best solution and adding it to the subset covered by its second best solution.
12: Set $i := i - 1$.
13: **end while**
14: Run the basic heuristic with starting partition P and denote the result by $\mathcal{X}_{K,1}$.
15: **until** Solution value of $\mathcal{X}_{K,2}$ is not worse than the solution value of $\mathcal{X}_{K,1}$.
16: **return** $\mathcal{X}_{K,2}$

The computation of the induced set of solutions is clearly the most expensive part of the algorithm in terms of running time. Since the algorithm may be required to compute the cost of the same subset at different iterations, its computing times can be reduced using hash tables to store the costs and the induced set of solutions of the subsets that have already been evaluated (see Section 4.5.1). This implementation issue has a dramatic effect in the performance of the heuristic, mainly for problems in which the subproblem solved by the oracle is time consuming.

5.4 Refinement

Our refinement procedure is based on a heuristic application of the set partitioning models described in Section 4.4. The idea is to store the solutions of multiple runs of the heuristic in the set $\widetilde{Q}$. Afterwards we solve formulation (SPP2) from Section 4.4, where instead of using the set of all columns Q, we restrict it to $\widetilde{Q}$. It is known that this kind of formulation can be effectively used to derive heuristic solutions for partitioning/covering problems; see, e.g., [24] for the Vehicle Routing Problem, [30] for Two-Dimensional Bin Packing, and [28] for the Vertex Coloring Problem. In all these applications, the main problem is to efficiently compute a large set of alternative solutions, possibly, with high quality. In our case, we take advantage of the fact that the presented heuristic is performed with different starting partitions, which produces a set of solutions $\mathcal{X}_K$ for each starting partition. Executing this algorithm, we may expect that a large set $\widetilde{Q}$ of feasible solutions, where duplicated solutions are removed, is available. Our refinement procedure uses a general-purpose MILP-solver on a restricted formulation that includes solution set $\widetilde{Q}$ only. In particular, our experiments showed that using Formulation 2 of Section 4.4 produces better results than using the first formulation. Note that, by construction, the set of solutions includes the final solution found by the heuristic and for this reason, the solution found by the solver cannot be worse than the solution without refinement and a feasible solution always exists. Actually, our computational experiments (see Chapter 6) show that this approach frequently improves over the basic heuristic and requires a short computing time.

Chapter 6

Experiments

In this chapter, we present the results of an experimental investigation of our exact and heuristic solution approaches for Problem (dmEm). We compare different configurations of the heuristic in Section 6.1, different exact algorithms in Section 6.2 and examine the effect of using an approximation algorithm for computing the induced set of solutions on the objective value in Section 6.3. The described algorithms were implemented in Java version 1.8.0_242 and executed on machines using Intel Xeon processors with 2.6 GHz and a memory limit of 32 GB. For solving all the integer linear and quadratic programming problems we used Cplex version 12.6.3.

In our experiments we considered the following underlying problems, which are described in Section 2.7 :

UCB: Unconstrained Binary Optimization Problem. All values of ξ are independent uniformly distributed rational numbers between -0.5 and 0.5.

ST: Spanning Tree Problem. For these problems, we always used a complete graph as network. For determining the costs of the edges, we computed two-dimensional center coordinates, which are in both dimensions integers between zero and one hundred. The coordinates in every scenario are random gaussians using the center coordinates as mean and having a standard deviation of 5. The cost of an edge in a scenario is the Euclidian distance of the coordinates of its end points.

MFP: Maximum Flow Problem. In our implementation it is in fact a Minimum Flow Problem and therefore the entries of each cost vector ξ were set to -1.0 if the corresponding arc leaves the source and 0.0 otherwise. In particular, the objective function is certain here. For these problems, we always used a quadratic grid graph as network. The capacity of the arcs were generated in each scenario as independent uniformly distributed rational numbers between 0.0 and 1.0. Additionally, for each arc there is a ten percent chance that it fails completely, leading to capacity zero.

1KP: Knapsack Problem. In each scenario, the profits of the items were randomly generated as independent uniformly distributed values between –1.0 and 0.0 and and weights of the items as independent uniformly distributed values between 0.0 and 1.0. The capacity value was always set to $0.75W$, where W denotes the current sum of the weights of the items.

MKP: Multidimensional Knapsack Problem. We considered a variant of 1KP in which there are 5 knapsack constraints. The profit, weight and capacity values for each scenario were generated as in the knapsack case.

TSP: Traveling Salesman Problem. We used the same generation of instances as for the Spanning Tree Problem.

In every underlying problem except for (MFP) the objective function is uncertain. For (MFP), (1KP), and (MKP) the set of constraints is uncertain. For each underlying problem, we considered different settings, which are combinations of different values of the parameters K, l, and n. For each setting we randomly generated ten instances as described above.

6.1 Heuristics

In this section we investigate which heuristic is the most suitable for which underlying problem. We consider all underlying problems presented before except for the Traveling Salesman Problem. In our experiments, we compare the following heuristics:

- Basic: this corresponds to the basic heuristic of Section 5 with deterministic start partitions, no refinement and no hash tables. This approach agrees with the improved heuristic algorithm described in [8].
- Det–PP: obtained from Basic by the addition of hash tables.
- Det+PP: obtained from Det–PP by the addition of the refinement as post processing.
- Clu–PP: this is analogous to Det–PP but the initial partitions are computed using the k-means clustering algorithm.
- Clu+PP: obtained from Clu–PP by the addition of the refinement as post processing.

All algorithms are run with the same time limit, equal to 30 CPU seconds. For algorithms 'Det+PP' and 'Clu+PP', this time was subdivided giving 20 CPU seconds to the first phase and 10 CPU seconds for refinement.

The following Tables 6.1 – 6.5 report the results for the different classes of instances. Each table gives, for each algorithm, the following information:

- value: average gap (over ten instances) to the best solution among the five algorithms for each instance.
- # best: number of instances (out of ten) for which the algorithm found the best solution among the five algorithms (including ties).

The rows that are printed bold present for each value of K the average over all values of l and n for the column 'value' and the total for the column '# best'.

6.1.1 Unconstrained Binary Optimization

The results illustrated in Table 6.1 suggest that for the Unconstrained Binary Problem the use of hash tables is just a small improvement, which is not surprising because solving the underlying problem is very fast. We can see that solving a Clustering Problem to determine starting partitions is beneficial, whereas the refinement procedure does decrease the quality of solutions. This is due to the fact that for a very easy underlying problem, a huge number of solutions are computed. Therefore, solving the Set Partitioning formulation

becomes too hard to solve it in time. Another reason is that for easy underlying problems, the time spent on refinement can be used instead to generate a huge number of other candidate solutions.

			Basic		Det-PP		Det+PP		Clu-PP		Clu+PP	
K	l	n	value	# best	value	# best	value	# best	value	# best	value	# best
20	100	100	0.3 %	2	0.3 %	2	0.5 %	1	0.0 %	7	0.2 %	2
20	100	300	0.1 %	0	0.1 %	0	0.3 %	0	0.1 %	5	0.1 %	5
20	100	500	0.3 %	0	0.3 %	0	0.3 %	0	0.1 %	6	0.1 %	6
20	200	100	0.1 %	4	0.1 %	4	0.3 %	2	0.2 %	4	0.2 %	2
20	200	300	0.2 %	2	0.2 %	2	0.3 %	2	0.1 %	6	0.2 %	2
20	200	500	0.2 %	4	0.2 %	5	0.3 %	1	0.2 %	4	0.3 %	2
20	-	-	**0.2 %**	**12**	**0.2 %**	**13**	**0.3 %**	**6**	**0.1 %**	**32**	**0.2 %**	**19**
30	100	100	0.5 %	0	0.4 %	0	0.6 %	0	0.0 %	8	0.1 %	6
30	100	300	0.3 %	0	0.3 %	0	0.4 %	0	0.1 %	5	0.1 %	7
30	100	500	0.3 %	0	0.3 %	1	0.3 %	0	0.1 %	6	0.1 %	5
30	200	100	0.4 %	1	0.3 %	2	0.4 %	1	0.1 %	5	0.2 %	4
30	200	300	0.3 %	2	0.2 %	4	0.3 %	2	0.1 %	5	0.3 %	1
30	200	500	0.2 %	3	0.2 %	3	0.2 %	2	0.1 %	2	0.2 %	5
30	-	-	**0.3 %**	**6**	**0.3 %**	**10**	**0.4 %**	**5**	**0.1 %**	**31**	**0.2 %**	**28**
40	100	100	0.3 %	1	0.3 %	1	0.5 %	0	0.0 %	9	0.1 %	4
40	100	300	0.3 %	0	0.3 %	0	0.4 %	0	0.0 %	7	0.1 %	5
40	100	500	0.3 %	2	0.3 %	2	0.4 %	2	0.1 %	5	0.1 %	3
40	200	100	0.6 %	1	0.5 %	1	0.7 %	1	0.1 %	8	0.3 %	3
40	200	300	0.3 %	1	0.3 %	2	0.4 %	1	0.1 %	6	0.3 %	3
40	200	500	0.2 %	2	0.2 %	2	0.3 %	1	0.1 %	4	0.1 %	4
40	-	-	**0.4 %**	**7**	**0.3 %**	**8**	**0.5 %**	**5**	**0.1 %**	**39**	**0.2 %**	**22**

Table 6.1: Results for the heuristics for the Unconstrained Binary Problem

6.1.2 The Spanning Tree Problem

For the Spanning Tree Problem (see Table 6.2) the heuristic with deterministic starting partitions without refinement is the best algorithm. Only in the easiest setting with $l = 100$ and $n = 45$, it was beaten by other variants. We can see that for this underlying problem, in most cases the deterministic starting partitions are the better choice. Nevertheless, the solution values of all variants are very close to each other.

				Basic		Det-PP		Det+PP		Clu-PP		Clu+PP	
k	l	n	$\|V\|$	value	# best	value	# best	value	# best	value	# best	value	# best
20	100	45	10	0.0 %	0	0.0 %	1	0.0 %	3	0.0 %	3	0.0 %	6
20	100	190	20	0.0 %	6	0.0 %	10	0.0 %	4	0.1 %	0	0.1 %	0
20	100	435	30	0.0 %	7	0.0 %	9	0.0 %	7	0.2 %	1	0.2 %	1
20	200	45	10	0.0 %	1	0.0 %	4	0.0 %	1	0.0 %	4	0.0 %	3
20	200	190	20	0.0 %	4	0.0 %	7	0.0 %	3	0.1 %	3	0.1 %	2
20	200	435	30	0.0 %	8	0.0 %	10	0.0 %	8	0.2 %	0	0.2 %	0
20	-	-	-	**0.0 %**	**26**	**0.0 %**	**41**	**0.0 %**	**26**	**0.1 %**	**11**	**0.1 %**	**12**
30	100	45	10	0.1 %	0	0.1 %	0	0.0 %	8	0.1 %	2	0.0 %	7
30	100	190	20	0.0 %	4	0.0 %	9	0.0 %	3	0.1 %	1	0.1 %	1
30	100	435	30	0.0 %	3	0.0 %	9	0.0 %	4	0.1 %	1	0.1 %	0
30	200	45	10	0.0 %	4	0.0 %	7	0.0 %	4	0.0 %	1	0.0 %	2
30	200	190	20	0.0 %	8	0.0 %	9	0.0 %	5	0.1 %	1	0.1 %	1
30	200	435	30	0.0 %	5	0.0 %	10	0.0 %	4	0.2 %	0	0.2 %	0
30	-	-	-	**0.0 %**	**24**	**0.0 %**	**44**	**0.0 %**	**28**	**0.1 %**	**6**	**0.1 %**	**11**
40	100	45	10	0.1 %	0	0.1 %	1	0.0 %	9	0.0 %	0	0.0 %	8
40	100	190	20	0.0 %	2	0.0 %	7	0.1 %	2	0.1 %	3	0.1 %	2
40	100	435	30	0.1 %	1	0.0 %	8	0.1 %	4	0.1 %	2	0.1 %	2
40	200	45	10	0.0 %	2	0.0 %	4	0.0 %	4	0.0 %	3	0.0 %	5
40	200	190	20	0.0 %	5	0.0 %	8	0.0 %	2	0.1 %	2	0.1 %	1
40	200	435	30	0.0 %	5	0.0 %	10	0.0 %	4	0.3 %	0	0.3 %	0
40	-	-	-	**0.0 %**	**15**	**0.0 %**	**38**	**0.0 %**	**25**	**0.1 %**	**10**	**0.1 %**	**18**

Table 6.2: Results for the heuristics for the Spanning Tree Problem

6.1.3 The Maximum Flow Problem

Table 6.3 presents the comparison of heuristics for the Maximum Flow Problem. We can see that there is a high variation within the best solution values. We have higher gaps between the different solutions and we do not have an algorithm that is clearly better than all the others. This is due to the fact that in the instances in which an algorithm produces the best solution the entry of the column 'value' is 1.0 no matter how much better the solution is compared to the others. On the other hand, in instances in which the solution value is worse, the gap to the best solution has an impact. This results in comparatively small values in the column 'value'. The best algorithms for this problem are 'Clu+PP', which turns out to be more effective if K is closer

to l, 'Det–PP' and 'Det+PP'. The basic heuristic is dominated by the algorithm with deterministic starting partitions without refinement. In the case of starting partitions obtained with the k-means algorithm, using the refinement procedure as post processing is improving the solution quality.

				Basic		Det-PP		Det+PP		Clu-PP		Clu+PP	
k	l	n	$\|V\|$	value	# best	value	# best	value	# best	value	# best	value	# best
20	100	60	36	11.3 %	1	10.1 %	4	10.7 %	2	20.6 %	2	7.3 %	6
20	100	84	49	17.4 %	1	12.5 %	4	12.5 %	2	26.8 %	2	14.8 %	4
20	100	112	64	18.2 %	2	15.4 %	2	10.3 %	5	22.4 %	1	9.6 %	5
20	200	60	36	5.4 %	7	5.1 %	8	2.9 %	8	24.8 %	0	18.9 %	1
20	200	84	49	9.6 %	2	6.0 %	8	8.4 %	4	27.7 %	0	19.6 %	2
20	200	112	64	16.9 %	3	15.1 %	3	1.5 %	7	24.1 %	2	18.8 %	1
20	-	-	-	**13.1 %**	**16**	**10.7 %**	**29**	**7.7 %**	**28**	**24.4 %**	**7**	**14.8 %**	**19**
30	100	60	36	13.6 %	1	11.0 %	5	8.7 %	4	22.8 %	0	8.7 %	4
30	100	84	49	15.7 %	3	12.2 %	5	11.1 %	4	20.3 %	0	9.2 %	4
30	100	112	64	31.9 %	0	29.6 %	0	14.2 %	5	24.9 %	1	14.7 %	4
30	200	60	36	11.5 %	3	10.3 %	6	11.4 %	3	23.0 %	0	9.7 %	4
30	200	84	49	6.5 %	4	5.0 %	8	5.6 %	7	21.9 %	0	13.6 %	2
30	200	112	64	15.3 %	2	13.1 %	5	11.4 %	4	26.8 %	1	11.7 %	4
30	-	-	-	**15.7 %**	**13**	**13.6 %**	**29**	**10.4 %**	**27**	**23.3 %**	**2**	**11.3 %**	**22**
40	100	60	36	19.4 %	0	17.1 %	1	14.6 %	2	25.1 %	0	4.6 %	8
40	100	84	49	16.9 %	1	13.5 %	4	11.5 %	5	17.3 %	0	4.5 %	5
40	100	112	64	24.3 %	0	21.0 %	2	9.3 %	5	20.6 %	0	5.5 %	4
40	200	60	36	13.9 %	1	13.1 %	4	13.7 %	2	19.8 %	1	4.7 %	6
40	200	84	49	15.8 %	1	14.2 %	5	15.1 %	2	25.4 %	0	6.7 %	5
40	200	112	64	16.1 %	1	15.1 %	4	15.7 %	1	23.2 %	1	5.1 %	6
40	-	-	-	**17.7 %**	**4**	**15.7 %**	**20**	**13.3 %**	**17**	**21.9 %**	**2**	**5.2 %**	**34**

Table 6.3: Results for the heuristics for the Maximum Flow Problem

6.1.4 The Knapsack Problem

In this section we discuss the experiments of the heuristics for the case that the underlying problem is the Knapsack Problem. The results are illustrated in Table 6.4. The underlying problem is harder to solve than for the underlying problems regarded before. Therefore, the refinement becomes more attractive because the resulting IP is easier to solve with less solutions and in the time used for solving it, the other algorithms are producing less solutions with which they can potentially increase their objective value. In our results we do not see a systematic advantage of one method of computing the starting partition over the other. The most successful algorithms for this underlying problem are 'Det+PP' and 'Clu+PP'.

			Basic		Det-PP		Det+PP		Clu-PP		Clu+PP	
K	l	n	value	# best	value	# best	value	# best	value	# best	value	# best
5	50	10	1.6 %	0	1.4 %	0	0.4 %	4	1.1 %	2	0.5 %	4
5	50	15	1.4 %	0	1.3 %	0	0.1 %	7	1.0 %	0	0.3 %	3
5	50	20	0.9 %	0	0.8 %	0	0.3 %	2	0.6 %	0	0.1 %	8
5	100	10	1.2 %	2	1.1 %	2	0.2 %	7	1.2 %	1	0.7 %	3
5	100	15	0.8 %	0	0.6 %	0	0.3 %	4	0.7 %	1	0.2 %	6
5	100	20	0.9 %	0	0.7 %	0	0.3 %	3	0.5 %	0	0.2 %	7
5	200	10	4.3 %	0	2.7 %	2	1.3 %	6	4.1 %	0	2.5 %	2
5	200	15	1.5 %	0	1.3 %	0	1.3 %	1	0.1 %	9	0.4 %	3
5	200	20	1.2 %	0	1.1 %	0	0.4 %	3	0.4 %	4	0.1 %	7
5	-	-	**1.5 %**	**2**	**1.2 %**	**4**	**0.5 %**	**37**	**1.1 %**	**17**	**0.6 %**	**43**
10	50	10	1.0 %	0	0.9 %	0	0.0 %	10	0.9 %	0	0.0 %	10
10	50	15	1.1 %	0	1.0 %	0	0.0 %	5	0.9 %	0	0.1 %	6
10	50	20	1.2 %	0	1.0 %	0	0.0 %	6	1.1 %	0	0.1 %	4
10	100	10	1.9 %	0	1.8 %	0	1.0 %	2	1.1 %	5	0.5 %	6
10	100	15	1.4 %	1	1.4 %	1	0.7 %	4	1.1 %	1	0.4 %	5
10	100	20	0.7 %	1	0.5 %	2	0.5 %	3	0.5 %	4	0.1 %	4
10	200	10	0.1 %	6	0.0 %	8	0.2 %	4	0.5 %	2	0.6 %	2
10	200	15	0.4 %	2	0.3 %	2	0.3 %	4	0.5 %	2	0.5 %	4
10	200	20	1.3 %	0	0.9 %	0	0.3 %	3	1.1 %	1	0.2 %	7
10	-	-	**1.0 %**	**10**	**0.9 %**	**13**	**0.3 %**	**41**	**0.8 %**	**15**	**0.3 %**	**48**
20	50	10	0.3 %	0	0.2 %	0	0.0 %	10	0.1 %	0	0.0 %	9
20	50	15	0.7 %	0	0.5 %	0	0.0 %	10	0.5 %	0	0.0 %	6
20	50	20	0.9 %	0	0.7 %	0	0.0 %	8	0.7 %	0	0.0 %	6
20	100	10	1.1 %	0	0.9 %	0	0.0 %	10	0.8 %	0	0.2 %	8
20	100	15	1.3 %	0	1.1 %	1	0.1 %	6	1.0 %	0	0.5 %	3
20	100	20	0.6 %	2	0.5 %	3	0.2 %	4	0.6 %	2	0.0 %	5
20	200	10	1.4 %	0	1.1 %	0	0.9 %	1	0.2 %	9	0.2 %	5
20	200	15	0.6 %	0	0.5 %	0	0.6 %	0	0.0 %	10	0.1 %	7
20	200	20	0.6 %	0	0.4 %	0	0.6 %	0	0.0 %	10	0.0 %	8
20	-	-	**0.8 %**	**2**	**0.6 %**	**4**	**0.3 %**	**49**	**0.4 %**	**31**	**0.1 %**	**57**

Table 6.4: Results for the heuristics for the Knapsack Problem

6.1.5 The Multidimensional Knapsack Problem

As for the Knapsack Problem, the tests suggest for the Multidimensional Knapsack Problem (see Table 6.5) that the post processing is very effective. Additionally, computing starting partitions by solving a Clustering Problem increases the quality of solutions for this underlying problem. Because the underlying problem is the hardest that we regarded so far, it is not surprising that the impact of using the hash tables is higher than before with improvements up 1.5 percentage points. The best variant of the heuristic for the Multidimensional Knapsack Problem is 'Clu+PP'.

			Basic		Det-PP		Det+PP		Clu-PP		Clu+PP	
K	l	n	value	# best	value	# best	value	# best	value	# best	value	# best
5	50	5	1.0 %	2	0.7 %	4	0.4 %	5	0.2 %	7	0.1 %	8
5	50	10	2.0 %	0	1.7 %	0	0.4 %	6	1.3 %	0	0.2 %	6
5	50	15	1.4 %	0	1.2 %	0	0.3 %	3	1.0 %	0	0.1 %	7
5	100	5	3.1 %	0	1.5 %	3	1.9 %	1	0.7 %	7	1.0 %	5
5	100	10	1.4 %	0	1.3 %	0	0.1 %	7	1.1 %	0	0.3 %	3
5	100	15	3.1 %	0	2.5 %	0	2.1 %	0	2.0 %	0	0.0 %	10
5	200	5	1.0 %	1	0.5 %	4	0.8 %	2	0.3 %	6	0.4 %	4
5	200	10	7.0 %	0	5.5 %	2	4.3 %	1	2.5 %	1	2.2 %	6
5	200	15	1.2 %	0	1.2 %	0	0.7 %	2	0.5 %	2	0.1 %	8
5	-	-	**2.4 %**	**3**	**1.8 %**	**13**	**1.2 %**	**27**	**1.0 %**	**23**	**0.5 %**	**57**
10	50	5	0.1 %	1	0.0 %	3	0.0 %	6	0.0 %	6	0.0 %	7
10	50	10	2.3 %	0	2.0 %	0	0.1 %	5	1.8 %	0	0.0 %	9
10	50	15	1.7 %	0	1.4 %	0	0.1 %	4	1.2 %	0	0.0 %	6
10	100	5	0.6 %	0	0.4 %	1	0.0 %	9	0.2 %	2	0.0 %	9
10	100	10	3.5 %	0	3.3 %	0	1.2 %	0	2.5 %	0	0.0 %	10
10	100	15	1.9 %	0	1.7 %	0	0.4 %	0	1.0 %	0	0.0 %	10
10	200	5	1.7 %	0	0.8 %	2	0.8 %	2	0.5 %	5	0.3 %	5
10	200	10	1.8 %	0	1.7 %	0	0.5 %	4	1.4 %	0	0.4 %	6
10	200	15	3.1 %	0	2.4 %	0	1.9 %	1	1.6 %	0	0.0 %	9
10	-	-	**1.9 %**	**1**	**1.5 %**	**6**	**0.5 %**	**31**	**1.1 %**	**13**	**0.1 %**	**71**
20	50	5	0.0 %	0	0.0 %	8	0.0 %	7	0.0 %	9	0.0 %	7
20	50	10	0.9 %	0	0.6 %	0	0.0 %	10	0.7 %	0	0.0 %	10
20	50	15	1.4 %	0	1.0 %	0	0.0 %	7	1.0 %	0	0.0 %	6
20	100	5	0.0 %	3	0.0 %	4	0.0 %	5	0.0 %	7	0.0 %	7
20	100	10	2.8 %	0	2.7 %	0	0.2 %	2	2.4 %	0	0.0 %	8
20	100	15	2.1 %	0	1.8 %	0	0.2 %	1	1.6 %	0	0.0 %	9
20	200	5	0.0 %	1	0.0 %	2	0.0 %	7	0.0 %	2	0.0 %	8
20	200	10	3.4 %	0	3.3 %	0	0.6 %	6	2.0 %	0	0.9 %	4
20	200	15	1.4 %	0	1.3 %	0	0.6 %	4	0.8 %	1	0.2 %	6
20	-	-	**1.3 %**	**4**	**1.2 %**	**14**	**0.2 %**	**49**	**0.9 %**	**19**	**0.1 %**	**65**

Table 6.5: Results for the heuristics for the Multidimensional Knapsack Problem with five constraints

6.2 Exact Algorithms

In this section we compare different exact algorithms for solving (dmEm) with each other and with the heuristic that turned out to be the most effective for the given underlying problem. We consider again all underlying problems presented before except for the Traveling Salesman Problem. In our experiments, we compare the following algorithms:

- Compact Formulation: direct application of the solver to the compact formulation (see Section 4.3). In this case, we used big M constraints instead of indicator constraints, and to let the solver handle the symmetry reduction on its own. Additionally, we gave ten starting solutions obtained each by a one-time use of the basic heuristic with k-means to the algorithm as so-called 'MIP starts'.

- Set Partitioning: direct application of the solver to the Set Partitioning

formulation (SPP1). In our implementation, we spent at most 90% of the time for subsets enumeration. In case this limit is reached, the solver is applied to a restricted formulation, thus producing a heuristic approach.

- Subsets Enumeration: enumeration of partitions according to the procedure described in Section 4.2.
- Branch-and-price: Branch-and-price algorithm based on column generation, as described in Section 4.5.3.
- Heuristic: heuristic that turned out to be the most effective for the underlying problem.

Before we compare the algorithms to each other, we investigate the role of the parameter 'qtolin' for the algorithms that use Cplex to solve QP's. For this we use the following notation:

- Compact Formulation lin: compact formulation as described before with linearization.
- Compact Formulation no lin: compact formulation as described before without linearization.
- Branch-and-price lin: Branch-and-price algorithm as described before with linearization.
- Branch-and-price no lin: Branch-and-price algorithm as described before without linearization.

After we determined the best way of dealing with quadratic terms for the given underlying problem, we use the best strategy for the Compact Formulation and the Branch-and-price algorithm in the comparison to the other algorithms. All exact algorithms received a time limit of 600 CPU seconds. For the Set Partitioning algorithm the time was subdivided giving 540 CPU seconds for the enumeration of the columns and 60 CPU seconds for solving the resulting IP. The time limit of the heuristic is set to 30 CPU second. If the chosen heuristic uses post processing, this time was subdivided giving 20 CPU seconds to the first phase and 10 CPU seconds for refinement.

The following Tables 6.6 – 6.13 report the results for the different classes of instances. Each table gives, for each algorithm, the following information:

- value: average gap between the solution produced by the algorithm and the best known solution for each instance, where only instances, in which the algorithm finds a feasible solution, are taken into account.

- # opt: number of instances (out of ten) that are solved to proven optimality.

- # best: number of instances (out of ten) with a solution value that is not worse than the solution value of any of the considered methods.

- time: average time, where for the exact algorithms only instances, in which the algorithm finds an optimal solution, are taken into account.

Lines printed in bold represent again averages or total values for each value of K. They can be a good orientation to compare different algorithms, but they have to be read carefully since the different algorithms might have solved different settings. Therefore, an average value of a bad algorithm can be better than the average value of a good one if the first was not able to solve harder instances.

6.2.1 Unconstrained Binary Optimization

In this section, we discuss the results of the tests of the exact algorithms for the Unconstrained Binary Optimization Problem. At first, we investigate the role of the parameter 'qtolin' for the Compact Formulation and the Branch-and-price algorithm. The results for these tests are illustrated by Table 6.6. For the Compact Formulation setting 'qtolin' to zero, which means that Cplex does not linearize the products of variables, is the better choice only for very easy instances. In more than half of the settings, this configuration of the algorithm was not able to find a solution within the time limit. In most of the cases, using linearization was clearly the better choice. For the Branch-and-price algorithm, setting 'qtolin' to zero was in most of the settings slightly faster. For the setting with the highest average running time, linearization was favorable. Therefore, we assume that for instances that become harder, the running time of the variant without linearization is increasing faster than the variant using linearization.

For the main comparison of this section (see Table 6.7) we used for the Compact Formulation the version with linearization and for the Branch-and-price algorithm the version without linearization. For the heuristic the version that computes starting partitions with the k-means algorithm and does not use the refinement procedure as a post processing was used because it was the best method in the comparison of the heuristics.

The only exact algorithm that was able to solve all instances to proven optimality is the Branch-and-price algorithm. For all settings with $l = 25$ and for settings with $l = 20$ and $K > 2$, it was the fastest algorithm. For $l = 15$ and $K > 2$ the Set Partitioning algorithm and for $l = 15$ and $K = 2$ the Subsets Enumeration was the most successful algorithm.

We can see that an increase of K has the highest impact on the Subsets Enumeration algorithm because the number of partitions depends strongly on K. We can also observe a high impact of parameter K on the performance of the Compact Formulation. On the other hand an increase of K only has a small effect on the running time of the Set Partitioning algorithm because it does not affect the expensive enumeration phase but only the phase in which the Set Partitioning Formulation is solved, which is comparably fast. For the Branch-and-price algorithm we cannot see a significant impact on the running time if K increases. If at all, we observe a decrease of the running time. A possible explanation for this is that for higher values of K less subsets have to be regarded and the average size of the subsets is smaller, leading to potentially less produced columns in the column generation method. If the parameter l is increased, we observe the highest impact on the performance for the Set Partitioning algorithm and on the Subsets Enumeration algorithm. This is due to the fact that the parameter l strongly affects the number of partitions and subsets. There is a smaller but still significant effect on the running time of the Compact Formulation and the Branch-and-price algorithm.

On the other hand, an increase of n has a small impact on the Set Partitioning algorithm and on the Subsets Enumeration algorithm because only the running time of the oracle is increased. For the other two exact algorithms, an increase of n has a high impact on the running time. The heuristic was able to solve all instances to optimality. Because it does not use refinement, it always exhausts the 30 seconds given to it as time limit. We conclude that for this

underlying problem the Branch-and-price algorithm is in most of the cases the best choice. For small values of l or very high values of n the Set Partitioning algorithm might be the better choice.

			Compact Formulation lin			Compact Formulation no lin			Branch-and-price lin			Branch-and-price no lin		
K	l	n	value	# opt	time	value	# opt	time	value	# opt	time	value	# opt	time
2	15	10	0.0 %	10	1.1	0.0 %	10	0.3	0.0 %	10	11.3	0.0 %	10	6.1
2	15	15	0.0 %	10	4.0	0.0 %	10	2.4	0.0 %	10	19.5	0.0 %	10	12.6
2	20	10	0.0 %	10	4.5	0.0 %	10	2.6	0.0 %	10	28.3	0.0 %	10	21.9
2	20	15	0.0 %	10	19.3	0.0 %	10	12.7	0.0 %	10	65.0	0.0 %	10	44.2
2	25	10	0.0 %	10	12.1	0.0 %	10	9.0	0.0 %	10	93.7	0.0 %	10	87.9
2	25	15	0.0 %	10	201.1	0.0 %	10	67.8	0.0 %	10	121.8	0.0 %	10	138.0
2	**-**	**-**	**0.0 %**	**60**	**40.4**	**0.0 %**	**60**	**15.8**	**0.0 %**	**60**	**56.6**	**0.0 %**	**60**	**51.8**
3	15	10	0.0 %	10	3.8	0.0 %	7	203.3	0.0 %	10	8.2	0.0 %	10	4.6
3	15	15	0.0 %	10	13.5	0.0 %	2	359.5	0.0 %	10	7.9	0.0 %	10	6.2
3	20	10	0.0 %	10	55.4	0.0 %	0	–	0.0 %	10	12.6	0.0 %	10	9.4
3	20	15	0.0 %	9	382.4	0.0 %	0	–	0.0 %	10	17.8	0.0 %	10	13.3
3	25	10	0.1 %	7	320.7	0.0 %	0	–	0.0 %	10	26.5	0.0 %	10	20.8
3	25	15	1.2 %	0	–	0.0 %	0	–	0.0 %	10	100.2	0.0 %	10	48.4
3	**-**	**-**	**0.2 %**	**46**	**139.4**	**0.0 %**	**9**	**238.0**	**0.0 %**	**60**	**28.9**	**0.0 %**	**60**	**17.1**
4	15	10	0.0 %	10	5.3	0.0 %	0	–	0.0 %	10	3.9	0.0 %	10	2.7
4	15	15	0.0 %	10	56.7	0.0 %	0	–	0.0 %	10	10.7	0.0 %	10	11.3
4	20	10	0.0 %	10	192.1	0.1 %	0	–	0.0 %	10	14.7	0.0 %	10	11.1
4	20	15	0.6 %	0	–	0.2 %	0	–	0.0 %	10	24.3	0.0 %	10	23.7
4	25	10	0.8 %	0	–	0.0 %	0	–	0.0 %	10	13.5	0.0 %	10	13.8
4	25	15	3.3 %	0	–	0.6 %	0	–	0.0 %	10	97.7	0.0 %	10	61.3
4	**-**	**-**	**0.8 %**	**30**	**84.7**	**0.1 %**	**0**	**–**	**0.0 %**	**60**	**27.5**	**0.0 %**	**60**	**20.6**
5	15	10	0.0 %	10	9.3	0.0 %	0	–	0.0 %	10	3.1	0.0 %	10	2.3
5	15	15	0.0 %	10	77.2	0.2 %	0	–	0.0 %	10	5.0	0.0 %	10	4.9
5	20	10	0.0 %	8	220.0	0.0 %	0	–	0.0 %	10	12.5	0.0 %	10	8.6
5	20	15	1.4 %	0	–	0.6 %	0	–	0.0 %	10	15.7	0.0 %	10	18.8
5	25	10	1.8 %	0	–	0.3 %	0	–	0.0 %	10	18.4	0.0 %	10	15.0
5	25	15	2.6 %	0	–	0.4 %	0	–	0.0 %	10	87.4	0.0 %	10	43.0
5	**-**	**-**	**1.0 %**	**28**	**93.8**	**0.2 %**	**0**	**–**	**0.0 %**	**60**	**23.7**	**0.0 %**	**60**	**15.4**

Table 6.6: Investigation of the Cplex parameter 'qtolin' for the exact methods for the Unconstrained Binary Problem

			Compact Formulation			Set Partitioning			Subsets Enumeration			Branch-and-price			Heuristic		
K	l	n	value	# opt	time	value	# opt	time	value	# opt	time	value	# opt	time	value	# best	time
2	15	10	0.0 %	10	1.1	0.0 %	10	1.0	0.0 %	10	0.0	0.0 %	10	6.1	0.0 %	10	30.0
2	15	15	0.0 %	10	4.0	0.0 %	10	1.1	0.0 %	10	0.0	0.0 %	10	12.6	0.0 %	10	30.0
2	20	10	0.0 %	10	4.5	0.0 %	10	41.2	0.0 %	10	2.4	0.0 %	10	21.9	0.0 %	10	30.0
2	20	15	0.0 %	10	19.3	0.0 %	10	43.2	0.0 %	10	2.0	0.0 %	10	44.2	0.0 %	10	30.0
2	25	10	0.0 %	10	12.1	–	0	–	–	0	–	0.0 %	10	87.9	0.0 %	10	30.0
2	25	15	0.0 %	10	201.1	–	0	–	–	0	–	0.0 %	10	138.0	0.0 %	10	30.0
2	**-**	**-**	**0.0 %**	**60**	**40.4**	**0.0 %**	**40**	**21.6**	**0.0 %**	**40**	**1.1**	**0.0 %**	**60**	**56.6**	**0.0 %**	**60**	**30.0**
3	15	10	0.0 %	10	3.8	0.0 %	10	1.6	0.0 %	10	3.4	0.0 %	10	4.6	0.0 %	10	30.0
3	15	15	0.0 %	10	13.5	0.0 %	10	1.2	0.0 %	10	3.7	0.0 %	10	6.2	0.0 %	10	30.0
3	20	10	0.0 %	10	55.4	0.0 %	10	57.3	0.7 %	0	–	0.0 %	10	9.4	0.0 %	10	30.0
3	20	15	0.0 %	9	382.4	0.0 %	10	57.2	0.9 %	0	–	0.0 %	10	13.3	0.0 %	10	30.0
3	25	10	0.1 %	7	320.7	–	0	–	9.5 %	0	–	0.0 %	10	20.8	0.0 %	10	30.0
3	25	15	1.2 %	0	–	–	0	–	7.3 %	0	–	0.0 %	10	48.4	0.0 %	10	30.0
3	**-**	**-**	**0.2 %**	**46**	**139.4**	**0.0 %**	**40**	**29.3**	**3.1 %**	**20**	**3.5**	**0.0 %**	**60**	**28.9**	**0.0 %**	**60**	**30.0**
4	15	10	0.0 %	10	5.3	0.0 %	10	1.0	0.0 %	10	63.3	0.0 %	10	2.7	0.0 %	10	30.0
4	15	15	0.0 %	10	56.7	0.0 %	10	1.7	0.0 %	10	64.7	0.0 %	10	11.3	0.0 %	10	30.0
4	20	10	0.0 %	10	192.1	0.0 %	10	67.9	6.4 %	0	–	0.0 %	10	11.1	0.0 %	10	30.0
4	20	15	0.6 %	0	–	0.0 %	10	63.8	4.8 %	0	–	0.0 %	10	23.7	0.0 %	10	30.0
4	25	10	0.8 %	0	–	–	0	–	16.4 %	0	–	0.0 %	10	13.8	0.0 %	10	30.0
4	25	15	3.3 %	0	–	–	0	–	15.3 %	0	–	0.0 %	10	61.3	0.0 %	10	30.0
4	**-**	**-**	**0.8 %**	**30**	**84.7**	**0.0 %**	**40**	**33.6**	**7.1 %**	**20**	**64.0**	**0.0 %**	**60**	**27.5**	**0.0 %**	**60**	**30.0**
5	15	10	0.0 %	10	9.3	0.0 %	10	1.2	0.0 %	10	365.1	0.0 %	10	2.3	0.0 %	10	30.0
5	15	15	0.0 %	10	77.2	0.0 %	10	1.3	0.0 %	10	365.6	0.0 %	10	4.9	0.0 %	10	30.0
5	20	10	0.0 %	8	220.0	0.0 %	10	63.1	7.7 %	0	–	0.0 %	10	8.6	0.0 %	10	30.0
5	20	15	1.4 %	0	–	0.0 %	10	61.1	7.3 %	0	–	0.0 %	10	18.8	0.0 %	10	30.0
5	25	10	1.8 %	0	–	–	0	–	20.2 %	0	–	0.0 %	10	15.0	0.0 %	10	30.0
5	25	15	2.6 %	0	–	–	0	–	18.3 %	0	–	0.0 %	10	43.0	0.0 %	10	30.0
5	**-**	**-**	**1.0 %**	**28**	**93.8**	**0.0 %**	**40**	**31.7**	**8.9 %**	**20**	**365.4**	**0.0 %**	**60**	**23.7**	**0.0 %**	**60**	**30.0**

Table 6.7: Results for the exact methods for the Unconstrained Binary Problem

6.2.2 The Spanning Tree Problem

For the Spanning Tree Problem, the classical IP formulation has an exponential number of constraints that ensure that the solution does not contain a cycle. In our implementation using the MILP-solver Cplex, we separated these constraints using so-called callbacks. Since the version of Cplex that we used does not allow callbacks combined with quadratic objective functions, we had to provide Cplex with the linearized version of the problem. For this reason an investigation of the parameter 'qtolin' was not necessary for this problem. In our comparison presented in Table 6.8, we chose for the heuristic the version that computes starting partitions deterministically without the refinement procedure. The Compact Formulation was able to solve only a small number of instances because even though the separation of constraints saves a lot of time,

				Compact Formulation			Set Partitioning			Subsets Enumeration			Branch-and-price			Heuristic		
K	l	n	$\|V\|$	value	# opt	time	value	# opt	time	value	# opt	time	value	# opt	time	value	# best	time
2	15	10	5	0.0 %	10	37.2	0.0 %	10	2.0	0.0 %	10	0.0	0.0 %	10	8.2	0.0 %	10	30.0
2	15	45	10	0.0 %	10	190.8	0.0 %	10	2.6	0.0 %	10	1.0	0.0 %	10	203.5	0.0 %	10	30.0
2	20	10	5	0.0 %	2	380.5	0.0 %	10	46.4	0.0 %	10	8.6	0.0 %	10	39.4	0.0 %	10	30.0
2	20	45	10	0.0 %	0	–	0.0 %	10	57.2	0.0 %	10	12.7	0.0 %	0	–	0.0 %	10	30.0
2	25	10	5	0.0 %	2	562.5	–	0	–	0.1 %	0	–	0.0 %	10	138.8	0.0 %	10	30.0
2	25	45	10	0.0 %	0	–	–	0	–	0.1 %	0	–	0.0 %	0	–	0.0 %	10	30.0
2	**-**	**-**	**-**	**0.0 %**	**24**	**173.6**	**0.0 %**	**40**	**27.1**	**0.0 %**	**40**	**5.6**	**0.0 %**	**40**	**97.5**	**0.0 %**	**60**	**30.0**
3	15	10	5	0.0 %	0	–	0.0 %	10	2.1	0.0 %	10	7.5	0.0 %	10	4.6	0.0 %	10	30.0
3	15	45	10	0.0 %	0	–	0.0 %	10	3.1	0.0 %	10	8.2	0.0 %	10	124.1	0.0 %	10	30.0
3	20	10	5	0.0 %	0	–	0.0 %	10	56.2	0.0 %	0	–	0.0 %	10	16.0	0.0 %	10	30.0
3	20	45	10	0.0 %	0	–	0.0 %	10	64.8	0.0 %	0	–	0.0 %	0	–	0.0 %	10	30.0
3	25	10	5	0.0 %	0	–	–	0	–	0.5 %	0	–	0.0 %	10	61.2	0.0 %	10	30.0
3	25	45	10	0.1 %	0	–	–	0	–	0.4 %	0	–	0.0 %	0	–	0.0 %	10	30.0
3	**-**	**-**	**-**	**0.0 %**	**0**	**–**	**0.0 %**	**40**	**31.6**	**0.2 %**	**20**	**7.8**	**0.0 %**	**40**	**51.5**	**0.0 %**	**60**	**30.0**
4	15	10	5	0.0 %	0	–	0.0 %	10	2.5	0.0 %	10	63.2	0.0 %	10	4.2	0.0 %	10	30.0
4	15	45	10	0.1 %	0	–	0.0 %	10	2.8	0.0 %	10	71.6	0.0 %	10	82.0	0.0 %	10	30.0
4	20	10	5	0.0 %	0	–	0.0 %	10	61.5	0.3 %	0	–	0.0 %	10	10.4	0.0 %	10	30.0
4	20	45	10	0.1 %	0	–	0.0 %	10	64.7	0.4 %	0	–	0.0 %	0	–	0.0 %	10	30.0
4	25	10	5	0.0 %	0	–	–	0	–	0.7 %	0	–	0.0 %	10	23.5	0.0 %	10	30.0
4	25	45	10	0.2 %	0	–	–	0	–	0.8 %	0	–	0.0 %	0	–	0.0 %	10	30.0
4	**-**	**-**	**-**	**0.1 %**	**0**	**–**	**0.0 %**	**40**	**32.9**	**0.4 %**	**20**	**67.4**	**0.0 %**	**40**	**30.0**	**0.0 %**	**60**	**30.0**
5	15	10	5	0.0 %	0	–	0.0 %	10	2.6	0.0 %	10	345.3	0.0 %	10	4.1	0.0 %	10	30.0
5	15	45	10	0.1 %	0	–	0.0 %	10	3.3	0.0 %	10	399.9	0.0 %	10	90.6	0.0 %	10	30.0
5	20	10	5	0.0 %	0	–	0.0 %	10	61.0	0.4 %	0	–	0.0 %	10	4.8	0.0 %	10	30.0
5	20	45	10	0.1 %	0	–	0.0 %	10	59.7	0.6 %	0	–	0.0 %	0	–	0.0 %	10	30.0
5	25	10	5	0.0 %	0	–	–	0	–	0.8 %	0	–	0.0 %	10	10.6	0.0 %	10	30.0
5	25	45	10	0.1 %	0	–	–	0	–	0.8 %	0	–	0.0 %	0	–	0.0 %	10	30.0
5	**-**	**-**	**-**	**0.1 %**	**0**	**–**	**0.0 %**	**40**	**31.6**	**0.4 %**	**20**	**372.6**	**0.0 %**	**40**	**27.5**	**0.0 %**	**60**	**30.0**

Table 6.8: Results for the exact methods for the Spanning Tree Problem

it is still a very time consuming procedure. The reason for that is that every time the separation routine finds a new constraint, the LP relaxation has to be solved again. Nevertheless, if the Compact formulation did not find the proven optimal solution, it was able to provide solutions with an objective value close to the optimum or to the best found solution. For $K = 2$ and $l \leq 20$ the Subsets Enumeration algorithm and for $K > 2$ and $l \leq 20$ the Set Partitioning algorithm was the fastest algorithm. The advantage of these two oracle-based algorithms is that the impact on n on the running time is comparably low. The Branch-and-price algorithm suffers from high values of n because it is not purely oracle-based and the exact pricing subroutine also needs callbacks to separate constraints of the Spanning Tree polytope. For $n = 10$, in every setting the Branch-and-price algorithm is either the fastest exact algorithm or slightly slower than the fastest exact algorithm. For every setting the solu-

tion found by the Branch-and-price algorithm or by the heuristic has the same value as the best found solution. We conclude that for our instances of the Spanning Tree Problem, it seems to be comparably easy to find solutions close to the optimum solution. For instances with low values of n or high values of l the Branch-and-price algorithm is a good choice, for the other instances the Set Partitioning algorithm should be used.

6.2.3 The Maximum Flow Problem

Also for the Maximum Flow Problem, Cplex had problems with the product of variables in the objective function. Here the problem was that we have products of binary and continuous variables. Unlike in the Spanning Tree case, here it was sufficient to set 'qtolin' to one and force Cplex to linearize the products to solve the problem. Therefore, we do not have an analysis on the parameter 'qtolin' for the Maximum Flow Problem.
Table 6.9 illustrates the results for the comparison of the exact algorithms for this underlying problem. For the heuristic, we used the variant in which the starting partitions are computed with the k-means algorithm and that uses the refinement procedure. The Set Partitioning algorithm was the fastest exact algorithm only in one setting and the Subsets Enumeration algorithm in no setting. Whenever these algorithms were not able to compute a solution in time, the quality of the provided solutions was poor. For $K < 4$ the Compact Formulation was the fastest exact algorithm, whereas for the other settings the Branch-and-price algorithm was better. The Compact Formulation provided high quality solutions in every setting and the Branch-and-price algorithm in every setting except for one. The results of the heuristic are comparably bad. The gap between the solution found by the heuristic and the best found solution is between 12 and 35 percent for every setting. This confirms the results of the heuristic experiments, in which the gap between the values of different solutions was much higher than for other underlying problems. We conclude that for the Maximum Flow Problem, it is much harder to find solutions that are close to the optimum compared with other underlying problems. Our conjecture is that the reason for this is the fact that the variables are continuous. Another hypothesis that gives the responsibility for this phenomenon to the uncertain constraints, can be rejected after seeing the next sections.

				Compact Formulation			Set Partitioning			Subsets Enumeration			Branch-and-price			Heuristic		
K	l	n	$\|V\|$	value	# opt	time	value	# opt	time	value	# opt	time	value	# opt	time	value	# best	time
2	15	24	16	0.0 %	10	0.0	0.0 %	10	15.9	0.0 %	10	14.5	0.0 %	10	13.8	30.2 %	0	20.0
2	15	40	25	0.0 %	10	0.0	0.0 %	10	15.9	0.0 %	10	17.9	0.0 %	10	14.3	18.1 %	0	20.0
2	20	24	16	0.0 %	10	0.0	0.0 %	10	337.5	0.0 %	10	303.4	0.0 %	10	40.4	22.5 %	1	20.0
2	20	40	25	0.0 %	10	0.7	0.0 %	10	393.9	0.0 %	10	361.0	0.0 %	10	42.1	23.2 %	0	20.0
2	25	24	16	0.0 %	10	0.7	–	0	–	17.4 %	0	–	0.0 %	10	160.3	34.6 %	0	20.0
2	25	40	25	0.0 %	10	2.0	–	0	–	29.5 %	0	–	0.0 %	10	59.1	27.6 %	0	20.0
2	**-**	**-**	**-**	**0.0 %**	**60**	**0.6**	**0.0 %**	**40**	**190.8**	**7.8 %**	**40**	**174.2**	**0.0 %**	**60**	**55.0**	**26.0 %**	**1**	**20.0**
3	15	24	16	0.0 %	10	2.8	0.0 %	10	15.5	0.0 %	10	19.4	0.0 %	10	13.3	27.4 %	0	20.0
3	15	40	25	0.0 %	10	5.4	0.0 %	10	15.3	0.0 %	10	19.1	0.0 %	10	16.3	19.5 %	0	20.0
3	20	24	16	0.0 %	10	32.2	0.0 %	10	348.5	8.1 %	0	–	0.0 %	10	63.3	19.7 %	0	20.0
3	20	40	25	0.0 %	10	37.5	0.0 %	10	404.7	1.3 %	0	–	0.0 %	10	74.5	20.7 %	0	20.0
3	25	24	16	0.0 %	10	137.0	–	0	–	25.9 %	0	–	0.0 %	8	333.0	30.6 %	0	20.0
3	25	40	25	0.0 %	7	272.1	–	0	–	28.1 %	0	–	0.0 %	10	301.3	22.7 %	0	20.0
3	**-**	**-**	**-**	**0.0 %**	**57**	**71.1**	**0.0 %**	**40**	**196.0**	**10.6 %**	**20**	**19.2**	**0.0 %**	**58**	**126.7**	**23.4 %**	**0**	**20.0**
4	15	24	16	0.0 %	10	21.2	0.0 %	10	15.3	0.0 %	10	76.4	0.0 %	10	12.3	17.9 %	0	20.0
4	15	40	25	0.0 %	10	92.2	0.0 %	10	16.3	0.0 %	10	80.5	0.0 %	10	23.2	12.2 %	1	20.0
4	20	24	16	0.6 %	3	285.0	6.3 %	9	371.6	24.6 %	0	–	0.0 %	9	126.0	16.3 %	1	20.6
4	20	40	25	0.2 %	1	93.0	0.0 %	10	408.2	18.4 %	0	–	0.0 %	10	92.4	21.0 %	0	20.4
4	25	24	16	0.0 %	0	–	86.6 %	0	–	33.2 %	0	–	7.6 %	3	487.3	17.2 %	0	22.9
4	25	40	25	1.1 %	0	–	87.8 %	0	–	32.2 %	0	–	0.1 %	7	346.1	19.7 %	0	24.9
4	**-**	**-**	**-**	**0.3 %**	**24**	**86.8**	**29.1 %**	**39**	**198.5**	**18.1 %**	**20**	**78.5**	**0.6 %**	**49**	**128.5**	**17.4 %**	**2**	**21.5**
5	15	24	16	0.0 %	6	45.3	0.0 %	10	16.7	0.0 %	10	367.5	0.0 %	10	12.1	15.6 %	1	20.0
5	15	40	25	0.0 %	3	60.0	0.0 %	10	16.7	0.0 %	10	386.9	0.0 %	10	15.9	11.6 %	1	21.3
5	20	24	16	0.3 %	0	–	0.0 %	10	346.8	27.8 %	0	–	0.0 %	10	97.3	17.6 %	0	24.8
5	20	40	25	0.7 %	0	–	0.0 %	10	402.0	20.5 %	0	–	0.0 %	10	87.2	18.5 %	0	23.4
5	25	24	16	0.5 %	0	–	84.0 %	0	–	35.3 %	0	–	0.0 %	7	360.3	18.0 %	0	23.2
5	25	40	25	0.3 %	0	–	67.1 %	0	–	33.1 %	0	–	0.0 %	2	285.0	16.9 %	0	28.2
5	**-**	**-**	**-**	**0.3 %**	**9**	**50.2**	**25.2 %**	**40**	**195.6**	**19.5 %**	**20**	**377.2**	**0.0 %**	**49**	**106.5**	**16.4 %**	**2**	**23.5**

Table 6.9: Results for the exact methods for the Maximum Flow Problem

6.2.4 The Knapsack Problem

Again we first investigate the impact of linearization. Table 6.10 suggests that for the Compact Formulation and for the Branch-and-price algorithm, linearization is clearly the better choice. That is matching our expectations that for more complex problems the running time of the version without linearization is growing faster.

Table 6.11 displays the results for the case that the underlying problem is the Knapsack Problem. We used the linearized versions of the Compact Formulation and of the Branch-and-price algorithm and the version that computes starting partitions with the k-means algorithm and uses our refinement strategy as post processing for the heuristic. For $K = 2$ the Compact Formulation is clearly the best choice. For $K = 3$ and $l = 10$ it was still better than the

other algorithms but these settings were easy for all algorithms. In all other algorithms the Branch-and-price algorithms are better than the other exact algorithms. It always gave high quality solutions in cases in which it is not able to find the optimum within the time limit and in most of the cases these non-completed solutions are better than the corresponding ones computed by the Compact Formulation. If the time limit is reached, the Set Partitioning algorithm provided high quality solutions if $K > 3$ holds. For no setting the Subsets Enumeration algorithm was faster than all other exact methods. We can explain the high performance loss of the purely oracle-based algorithm compared to the others by the fact that the complexity of the oracle increased significantly. In all settings except for one, the heuristic was able to provide the optimal solution or the best found solution. It did not reach the time limit of 30 CPU seconds because solving the Set Partitioning formulation of the refinement took at most 4 CPU seconds.

6.2.5 The Multidimensional Knapsack Problem

Table 6.12 provides the results of the investigation of the parameter 'qtolin', which controls the handling of products of variables in the MILP-solver Cplex, for the Multidimensional Knapsack Problem with five constraints. We can see that for both algorithms linearization is clearly favorable. For the Compact formulation the algorithm without linearization was only able to solve instances of one setting. For the Branch-and-price algorithm the algorithm using linearization dominates the other. Note that if the algorithm without linearization had a better running time, it is only due to the fact that it solved less settings to proven optimality and therefore the average over the instances solved within the time limit includes less instances - especially some hard ones are missing - than the algorithm using linearization.

In our comparison illustrated by Table 6.13 we used for the Compact Formulation and the Branch-and-price algorithm the version with linearization of products of variables in Cplex. For the heuristic, we used the version that computes starting partitions with the k-means algorithm and uses our refinement strategy as a post processing. The Branch-and-price algorithm is the best algorithm for $K = 5$ and for all other values of K if $l \geq 15$. For $l = 10$ the Set Partitioning algorithm and the Subsets Enumeration algorithm are

			Compact Formulation lin			Compact Formulation no lin			Branch-and-price lin			Branch-and-price no lin		
K	l	n	value	# opt	time	value	# opt	time	value	# opt	time	value	# opt	time
2	10	10	0.0 %	10	0.5	0.0 %	10	27.8	0.0 %	10	10.2	0.0 %	10	11.0
2	10	15	0.0 %	10	2.5	0.0 %	7	357.3	0.0 %	10	24.2	0.0 %	10	44.4
2	15	10	0.0 %	10	21.1	0.0 %	6	350.5	0.0 %	10	110.7	0.0 %	10	159.2
2	15	15	0.0 %	10	71.2	0.0 %	0	–	0.1 %	7	304.0	0.4 %	0	–
2	20	10	0.0 %	8	354.8	0.1 %	0	–	0.4 %	6	345.3	0.7 %	0	–
2	20	15	0.2 %	0	–	0.1 %	0	–	0.7 %	0	–	0.7 %	0	–
2	**-**	**-**	**0.0 %**	**48**	**79.0**	**0.0 %**	**23**	**212.3**	**0.2 %**	**43**	**131.4**	**0.3 %**	**30**	**71.5**
3	10	10	0.0 %	10	2.4	0.1 %	0	–	0.0 %	10	5.6	0.0 %	10	7.6
3	10	15	0.0 %	10	7.4	0.0 %	0	–	0.0 %	10	11.7	0.0 %	10	36.4
3	15	10	0.0 %	10	208.3	0.3 %	0	–	0.0 %	10	83.4	0.0 %	9	108.9
3	15	15	0.3 %	3	379.7	0.5 %	0	–	0.1 %	9	276.7	0.4 %	0	–
3	20	10	0.6 %	0	–	0.5 %	0	–	0.1 %	6	338.3	0.9 %	1	293.0
3	20	15	0.9 %	0	–	0.7 %	0	–	0.5 %	1	243.0	0.9 %	0	–
3	**-**	**-**	**0.3 %**	**33**	**100.6**	**0.3 %**	**0**	**–**	**0.1 %**	**46**	**125.4**	**0.4 %**	**30**	**57.1**
4	10	10	0.0 %	10	4.3	0.1 %	3	0.0	0.0 %	10	3.9	0.0 %	10	6.4
4	10	15	0.0 %	10	18.1	0.1 %	0	–	0.0 %	10	4.4	0.0 %	10	19.0
4	15	10	0.2 %	3	311.0	0.2 %	0	–	0.0 %	10	44.7	0.0 %	10	63.3
4	15	15	0.4 %	0	–	0.8 %	6	1.0	0.0 %	10	141.6	0.1 %	3	272.3
4	20	10	1.4 %	0	–	0.6 %	0	–	0.0 %	10	175.0	0.0 %	7	342.4
4	20	15	1.2 %	0	–	0.6 %	0	–	0.7 %	2	240.5	1.3 %	0	–
4	**-**	**-**	**0.5 %**	**23**	**50.3**	**0.4 %**	**9**	**0.7**	**0.1 %**	**52**	**80.3**	**0.2 %**	**40**	**102.5**
5	10	10	0.0 %	10	6.1	0.4 %	7	0.0	0.0 %	10	2.0	0.0 %	10	2.7
5	10	15	0.0 %	10	26.9	0.4 %	8	0.0	0.0 %	10	2.7	0.0 %	10	18.7
5	15	10	0.3 %	2	428.0	0.1 %	0	–	0.0 %	10	7.9	0.0 %	10	18.0
5	15	15	0.7 %	0	–	0.8 %	5	1.0	0.0 %	10	46.4	0.1 %	8	302.4
5	20	10	1.2 %	0	–	0.4 %	0	–	0.0 %	10	88.0	0.0 %	8	211.4
5	20	15	1.4 %	0	–	0.7 %	0	–	0.1 %	5	330.2	0.6 %	0	–
5	**-**	**-**	**0.6 %**	**22**	**53.9**	**0.4 %**	**20**	**0.2**	**0.0 %**	**55**	**56.7**	**0.1 %**	**46**	**97.9**

Table 6.10: Investigation of the Cplex parameter 'qtolin' for the exact methods for the Knapsack Problem

fast and provide comparable results, but they were not able to solve other settings within the time limit. The Set Partitioning algorithm found solutions with better values in cases in which the time limit was reached, but for $K < 4$ and $l = 20$ it was not able to find solutions at all. The Compact Formulation is faster than all other exact algorithms only for $K = 2$ and $l = 10$, but consider that these instances were not challenging for all tested algorithms. For $K < 4$ the Compact Formulation provides better solutions than the Branch-and-price algorithm, in cases in which they exceeded the time limit. The reverse statement holds for $K > 3$. The gap of the solution provided by the Branch-and-price algorithm to the best found solution is never higher than 1.6% and for the Compact Formulation never higher than 1.7%. The heuristic was able to provide the best solution in all settings except for one, where ties are obviously

			Compact Formulation			Set Partitioning			Subsets Enumeration			Branch-and-price			Heuristic		
K	l	n	value	# opt	time	value	# opt	time	value	# opt	time	value	# opt	time	value	# best	time
2	10	10	0.0 %	10	0.5	0.0 %	10	7.7	0.0 %	10	7.5	0.0 %	10	10.2	0.0 %	10	20.0
2	10	15	0.0 %	10	2.5	0.0 %	10	9.5	0.0 %	10	10.3	0.0 %	10	24.2	0.0 %	10	20.0
2	15	10	0.0 %	10	21.1	0.0 %	10	239.0	0.0 %	10	235.7	0.0 %	10	110.7	0.0 %	10	20.0
2	15	15	0.0 %	10	71.2	0.0 %	10	407.8	0.0 %	10	403.5	0.1 %	7	304.0	0.0 %	10	20.0
2	20	10	0.0 %	8	354.8	–	0	–	2.0 %	0	–	0.4 %	6	345.3	0.0 %	10	20.3
2	20	15	0.2 %	0	–	–	0	–	1.4 %	0	–	0.7 %	0	–	0.0 %	9	20.3
2	**-**	**-**	**0.0 %**	**48**	**79.0**	**0.0 %**	**40**	**166.0**	**0.6 %**	**40**	**164.2**	**0.2 %**	**43**	**131.4**	**0.0 %**	**59**	**20.1**
3	10	10	0.0 %	10	2.4	0.0 %	10	7.6	0.0 %	10	7.8	0.0 %	10	5.6	0.0 %	10	20.0
3	10	15	0.0 %	10	7.4	0.0 %	10	9.4	0.0 %	10	10.6	0.0 %	10	11.7	0.0 %	10	20.0
3	15	10	0.0 %	10	208.3	0.0 %	10	240.4	0.0 %	10	238.2	0.0 %	10	83.4	0.0 %	10	20.7
3	15	15	0.3 %	3	379.7	0.0 %	10	408.3	0.0 %	10	407.6	0.1 %	9	276.7	0.0 %	10	20.2
3	20	10	0.6 %	0	–	0.8 %	0	–	1.4 %	0	–	0.1 %	6	338.3	0.1 %	9	23.9
3	20	15	0.9 %	0	–	–	0	–	1.5 %	0	–	0.5 %	1	243.0	0.0 %	10	22.3
3	**-**	**-**	**0.3 %**	**33**	**100.6**	**0.2 %**	**40**	**166.4**	**0.5 %**	**40**	**166.1**	**0.1 %**	**46**	**125.4**	**0.0 %**	**59**	**21.2**
4	10	10	0.0 %	10	4.3	0.0 %	10	7.2	0.0 %	10	7.3	0.0 %	10	3.9	0.0 %	10	20.0
4	10	15	0.0 %	10	18.1	0.0 %	10	9.2	0.0 %	10	8.8	0.0 %	10	4.4	0.0 %	10	20.0
4	15	10	0.2 %	3	311.0	0.0 %	10	238.3	0.0 %	10	295.6	0.0 %	10	44.7	0.0 %	10	20.3
4	15	15	0.4 %	0	–	0.0 %	10	404.5	0.0 %	10	464.2	0.0 %	10	141.6	0.0 %	10	20.2
4	20	10	1.4 %	0	–	0.1 %	0	–	2.0 %	0	–	0.0 %	10	175.0	0.0 %	10	21.4
4	20	15	1.2 %	0	–	0.1 %	0	–	2.0 %	0	–	0.7 %	2	240.5	0.0 %	9	23.0
4	**-**	**-**	**0.5 %**	**23**	**50.3**	**0.0 %**	**40**	**164.8**	**0.7 %**	**40**	**194.0**	**0.1 %**	**52**	**80.3**	**0.0 %**	**59**	**20.8**
5	10	10	0.0 %	10	6.1	0.0 %	10	5.5	0.0 %	10	5.9	0.0 %	10	2.0	0.0 %	10	20.0
5	10	15	0.0 %	10	26.9	0.0 %	10	7.5	0.0 %	10	7.8	0.0 %	10	2.7	0.0 %	10	20.0
5	15	10	0.3 %	2	428.0	0.0 %	10	231.5	0.0 %	6	565.0	0.0 %	10	7.9	0.0 %	10	20.0
5	15	15	0.7 %	0	–	0.0 %	10	397.2	0.0 %	0	–	0.0 %	10	46.4	0.0 %	10	20.0
5	20	10	1.2 %	0	–	0.0 %	0	–	3.1 %	0	–	0.0 %	10	88.0	0.0 %	10	20.3
5	20	15	1.4 %	0	–	0.0 %	0	–	2.2 %	0	–	0.1 %	5	330.2	0.0 %	10	20.4
5	**-**	**-**	**0.6 %**	**22**	**53.9**	**0.0 %**	**40**	**160.4**	**0.9 %**	**26**	**135.7**	**0.0 %**	**55**	**56.7**	**0.0 %**	**60**	**20.1**

Table 6.11: Results for the exact methods for the Knapsack Problem

included. The refinement procedure never took more than 1 CPU second. We can conclude that for the Multidimensional Knapsack Problem with five constraints the Branch-and-price algorithm is the ideal choice because it has the best running time in most of the settings and in the others it is just slightly slower than the fastest exact algorithm. Additionally, it provides high quality solutions in cases in which the time limit is reached.

6.3 Approximation

The goal of this section is to determine how an approximation factor of an algorithm that is used to compute the induced solution can be conserved while solving Problem (dmEm). In Section 3.1, we proved that the worst case approximation guarantee remains the same. In the following we want to

			Compact Formulation lin			Compact Formulation no lin			Branch-and-price lin			Branch-and-price no lin		
K	l	n	value	# opt	time	value	# opt	time	value	# opt	time	value	# opt	time
2	10	10	0.0 %	10	6.8	0.0 %	10	186.4	0.0 %	10	31.3	0.0 %	10	33.2
2	10	15	0.0 %	10	22.0	0.0 %	0	–	0.0 %	10	49.5	0.0 %	10	126.5
2	15	10	0.0 %	10	396.0	0.0 %	0	–	0.1 %	6	296.2	0.4 %	4	333.5
2	15	15	0.0 %	6	441.8	0.1 %	0	–	0.1 %	6	354.0	0.8 %	0	–
2	20	10	0.2 %	3	413.3	0.2 %	0	–	0.6 %	4	425.8	0.8 %	0	–
2	20	15	0.2 %	0	–	0.6 %	0	–	1.1 %	0	–	1.1 %	0	–
2	**-**	**-**	**0.1 %**	**39**	**208.7**	**0.1 %**	**10**	**186.4**	**0.3 %**	**36**	**178.1**	**0.5 %**	**24**	**122.1**
3	10	10	0.0 %	10	21.3	0.1 %	0	–	0.0 %	10	28.4	0.0 %	10	33.5
3	10	15	0.0 %	10	50.8	0.4 %	0	–	0.0 %	10	52.6	0.0 %	10	197.3
3	15	10	0.7 %	0	–	1.0 %	0	–	0.0 %	7	217.4	0.1 %	7	296.3
3	15	15	0.4 %	0	–	0.5 %	0	–	0.2 %	8	289.6	1.0 %	0	–
3	20	10	1.7 %	0	–	0.7 %	0	–	1.1 %	3	473.0	2.5 %	0	–
3	20	15	0.9 %	0	–	0.5 %	0	–	1.6 %	0	–	1.6 %	0	–
3	**-**	**-**	**0.6 %**	**20**	**36.0**	**0.5 %**	**0**	**–**	**0.5 %**	**38**	**159.7**	**0.9 %**	**27**	**162.3**
4	10	10	0.0 %	10	38.1	0.0 %	0	–	0.0 %	10	23.0	0.0 %	10	25.8
4	10	15	0.0 %	10	108.0	0.4 %	0	–	0.0 %	10	24.0	0.0 %	10	102.3
4	15	10	1.4 %	0	–	0.6 %	0	–	0.0 %	10	218.3	0.2 %	9	198.6
4	15	15	1.2 %	0	–	0.4 %	0	–	0.1 %	9	270.6	1.2 %	0	–
4	20	10	1.6 %	0	–	0.9 %	0	–	0.4 %	6	428.3	2.1 %	1	248.0
4	20	15	0.9 %	0	–	0.5 %	0	–	0.8 %	1	286.0	1.2 %	0	–
4	**-**	**-**	**0.9 %**	**20**	**73.0**	**0.5 %**	**0**	**–**	**0.2 %**	**46**	**172.7**	**0.8 %**	**30**	**110.5**
5	10	10	0.0 %	10	41.9	0.1 %	0	–	0.0 %	10	11.5	0.0 %	10	13.9
5	10	15	0.0 %	10	127.0	0.4 %	0	–	0.0 %	10	12.1	0.0 %	10	61.7
5	15	10	0.7 %	0	–	0.4 %	0	–	0.0 %	10	98.9	0.0 %	10	142.1
5	15	15	1.1 %	0	–	0.5 %	0	–	0.0 %	10	161.4	1.3 %	2	545.0
5	20	10	1.5 %	0	–	0.7 %	0	–	0.2 %	8	273.0	1.1 %	1	150.0
5	20	15	1.3 %	0	–	0.8 %	0	–	0.9 %	1	335.0	1.6 %	0	–
5	**-**	**-**	**0.8 %**	**20**	**84.5**	**0.5 %**	**0**	**–**	**0.2 %**	**49**	**109.3**	**0.7 %**	**33**	**103.5**

Table 6.12: Investigation of the Cplex parameter 'qtolin' for the exact methods for the Multidimensional Knapsack Problem with five constraints

understand the behavior in the average case. For this purpose we examine the Traveling Salesman Problem. We compare two algorithms, both based on the Set Partitioning algorithm. Either the oracle for computing the induced solution is an exact algorithm or the induced solution is computed with the algorithm of Christofides [11]. For every induced solution we computed the gap between the solution of the approximation algorithm and the exact algorithm. We computed the average gap and the maximum gap over all induced solutions and the gap of the solution values of Problem (dmEm). Table 6.14 illustrates the main results in the following columns:

- # subs.: number of subsets computed throughout the algorithm
- # mEm better: number of times (out of 10) the gap of Problem (dmEm)

			Compact Formulation			Set Partitioning			Subsets Enumeration			Branch-and-price			Heuristic		
K	l	n	value	# opt	time	value	# opt	time	value	# opt	time	value	# opt	time	value	# best	time
2	10	10	0.0 %	10	6.8	0.0 %	10	21.1	0.0 %	10	20.4	0.0 %	10	31.3	0.0 %	10	20.0
2	10	15	0.0 %	10	22.0	0.0 %	10	23.4	0.0 %	10	22.4	0.0 %	10	49.5	0.0 %	10	20.0
2	15	10	0.0 %	10	396.0	0.1 %	0	–	0.2 %	0	–	0.1 %	6	296.2	0.0 %	10	20.0
2	15	15	0.0 %	6	441.8	0.2 %	0	–	0.2 %	0	–	0.1 %	6	354.0	0.0 %	10	20.1
2	20	10	0.2 %	3	413.3	–	0	–	2.0 %	0	–	0.6 %	4	425.8	0.0 %	10	20.0
2	20	15	0.2 %	0	–	–	0	–	2.3 %	0	–	1.1 %	0	–	0.0 %	10	20.2
2	**-**	**-**	**0.1 %**	**39**	**208.7**	**0.1 %**	**20**	**22.2**	**0.8 %**	**20**	**21.4**	**0.3 %**	**36**	**178.1**	**0.0 %**	**60**	**20.1**
3	10	10	0.0 %	10	21.3	0.0 %	10	20.6	0.0 %	10	21.6	0.0 %	10	28.4	0.0 %	10	20.0
3	10	15	0.0 %	10	50.8	0.0 %	10	23.1	0.0 %	10	21.6	0.0 %	10	52.6	0.0 %	10	20.0
3	15	10	0.7 %	0	–	0.0 %	0	–	0.3 %	0	–	0.0 %	7	217.4	0.0 %	9	20.2
3	15	15	0.4 %	0	–	0.1 %	0	–	0.2 %	0	–	0.2 %	8	289.6	0.1 %	8	20.1
3	20	10	1.7 %	0	–	–	0	–	3.5 %	0	–	1.1 %	3	473.0	0.0 %	10	20.2
3	20	15	0.9 %	0	–	–	0	–	2.0 %	0	–	1.6 %	0	–	0.0 %	10	20.6
3	**-**	**-**	**0.6 %**	**20**	**36.0**	**0.0 %**	**20**	**21.9**	**1.0 %**	**20**	**21.6**	**0.5 %**	**38**	**159.7**	**0.0 %**	**57**	**20.2**
4	10	10	0.0 %	10	38.1	0.0 %	10	19.6	0.0 %	10	18.8	0.0 %	10	23.0	0.0 %	10	20.0
4	10	15	0.0 %	10	108.0	0.0 %	10	21.7	0.0 %	10	20.8	0.0 %	10	24.0	0.0 %	10	20.0
4	15	10	1.4 %	0	–	0.0 %	0	–	0.1 %	0	–	0.0 %	10	218.3	0.0 %	9	20.0
4	15	15	1.2 %	0	–	0.0 %	0	–	0.1 %	0	–	0.1 %	9	270.6	0.0 %	9	20.0
4	20	10	1.6 %	0	–	0.5 %	0	–	3.3 %	0	–	0.4 %	6	428.3	0.0 %	9	21.0
4	20	15	0.9 %	0	–	0.2 %	0	–	1.8 %	0	–	0.8 %	1	286.0	0.0 %	9	20.4
4	**-**	**-**	**0.9 %**	**20**	**73.0**	**0.1 %**	**20**	**20.6**	**0.9 %**	**20**	**19.8**	**0.2 %**	**46**	**172.7**	**0.0 %**	**56**	**20.2**
5	10	10	0.0 %	10	41.9	0.0 %	10	16.5	0.0 %	10	15.9	0.0 %	10	11.5	0.0 %	10	20.0
5	10	15	0.0 %	10	127.0	0.0 %	10	19.0	0.0 %	10	17.5	0.0 %	10	12.1	0.0 %	10	20.0
5	15	10	0.7 %	0	–	0.0 %	0	–	0.3 %	0	–	0.0 %	10	98.9	0.0 %	10	20.0
5	15	15	1.1 %	0	–	0.0 %	0	–	0.1 %	0	–	0.0 %	10	161.4	0.0 %	10	20.0
5	20	10	1.5 %	0	–	0.1 %	0	–	3.3 %	0	–	0.2 %	8	273.0	0.0 %	8	20.4
5	20	15	1.3 %	0	–	0.1 %	0	–	2.2 %	0	–	0.9 %	1	335.0	0.0 %	8	20.3
5	**-**	**-**	**0.8 %**	**20**	**84.5**	**0.0 %**	**20**	**17.8**	**1.0 %**	**20**	**16.7**	**0.2 %**	**49**	**109.3**	**0.0 %**	**56**	**20.1**

Table 6.13: Results for the exact methods for the Multidimensional Knapsack Problem with five constraints

was lower than the average gap over all induced solutions

- Gap mEm: gap of the solution value of Problem (dmEm) (average over 10 instances)
- aGap oracle: average gap over all induced solutions (average over 10 instances)
- mGap oracle: maximum gap over all induced solutions (average over 10 instances)

The line printed in bold represents again the average or the total value of the column for one K.

In the tests the gap of Problem (dmEm) was lower than the average gap of the oracle in 337 out of 360 instances, and on average it was significantly lower. In 25 out of 36 settings, considering the average over ten instances, the gap of the solutions of Problem (dmEm) was less than half of the average gap over all solutions of the oracle. The highest gap of the solutions of Problem (dmEm) was only 6.2% in all settings and the highest average gap over all solutions of the oracle was 7%. The accuracy of the solution of Problem (dmEm) increases in K: The average gap of the solution of Problem (dmEm) decreases from 2.3% for $K = 2$ to 1.5% for $K = 5$. We conclude that in practice the ratio between the exact solution value and the solution value computed by an approximation

K	l	n	$\|V\|$	# subs.	# mEm better	Gap mEm	aGap oracle	mGap oracle
2	3	10	5	6	6	1.3 %	2.1 %	4.2 %
2	3	45	10	6	8	3.5 %	4.6 %	9.6 %
2	3	105	15	6	7	6.2 %	7.0 %	10.8 %
2	5	10	5	30	9	1.7 %	3.0 %	7.3 %
2	5	45	10	30	10	2.7 %	5.0 %	13.5 %
2	5	105	15	30	10	2.9 %	5.9 %	15.1 %
2	10	10	5	1022	10	0.4 %	1.5 %	10.2 %
2	10	45	10	1022	10	2.5 %	5.5 %	14.2 %
2	10	105	15	1022	10	2.3 %	5.9 %	19.7 %
2	15	10	5	32766	8	0.7 %	2.2 %	11.7 %
2	15	45	10	32766	10	1.5 %	5.6 %	18.8 %
2	15	105	15	32766	10	1.9 %	5.5 %	20.1 %
2	-	-	-	-	**108**	**2.3 %**	**4.5 %**	**12.9 %**
3	5	10	5	25	9	1.5 %	3.1 %	7.3 %
3	5	45	10	25	10	2.5 %	5.1 %	13.5 %
3	5	105	15	25	10	4.0 %	6.2 %	15.1 %
3	10	10	5	1012	10	0.4 %	1.5 %	10.2 %
3	10	45	10	1012	10	2.2 %	5.5 %	14.2 %
3	10	105	15	1012	10	1.9 %	5.9 %	19.7 %
3	15	10	5	32751	8	0.6 %	2.2 %	11.7 %
3	15	45	10	32751	10	1.3 %	5.6 %	18.8 %
3	15	105	15	32751	10	1.8 %	5.5 %	20.1 %
3	-	-	-	-	**87**	**1.8 %**	**4.5 %**	**14.5 %**
4	5	10	5	15	8	2.0 %	3.1 %	7.2 %
4	5	45	10	15	10	3.5 %	5.4 %	13.2 %
4	5	105	15	15	10	5.2 %	6.4 %	14.7 %
4	10	10	5	967	9	0.4 %	1.5 %	10.2 %
4	10	45	10	967	10	2.1 %	5.5 %	14.2 %
4	10	105	15	967	10	2.4 %	6.0 %	19.7 %
4	15	10	5	32646	8	0.5 %	2.2 %	11.7 %
4	15	45	10	32646	10	1.3 %	5.6 %	18.8 %
4	15	105	15	32646	10	1.7 %	5.5 %	20.1 %
4	-	-	-	-	**85**	**2.1 %**	**4.6 %**	**14.4 %**
5	10	10	5	847	9	0.4 %	1.6 %	10.2 %
5	10	45	10	847	10	2.3 %	5.5 %	14.2 %
5	10	105	15	847	10	2.8 %	6.1 %	19.7 %
5	15	10	5	32191	8	0.5 %	2.2 %	11.7 %
5	15	45	10	32191	10	1.4 %	5.6 %	18.8 %
5	15	105	15	32191	10	1.8 %	5.5 %	20.1 %
5	-	-	-	-	**57**	**1.5 %**	**4.4 %**	**15.8 %**

Table 6.14: Comparison of the approximation ratios for the Traveling Salesman Problem

algorithm is much lower for Problem (dmEm) than for the problem solved by the oracle. This is because in case there are many solutions for (dmEm) with the same exact value or exact values that are close to each other, the solutions in which the approximation ratio is lower, are chosen by the Set Partitioning algorithm. Therefore, also for underlying problems that are very hard to solve or for huge instances, good solutions of Problem (dmEm) can be produced by the oracle-based algorithms by applying a good heuristic for the underlying problem.

6.4 Summary

In this section we want to summarize the results of the previous sections and try to generalize them. We can conclude that the Compact Formulation and the Branch-and-price algorithm are in general the best among the exact solution methods that we tested for solving Problem (dmEm). For both algorithms linearizing products of variables is reducing the running time in most of the cases. The version without linearization was better only for very easy instances and the difference was not relevant then.
The Subsets Enumeration algorithm and the Set Partitioning algorithm are better than the latter just in a few instances. Because they are purely oracle-based, they benefit from underlying problems that are easy to solve. For the Compact Formulation and the Branch-and-price algorithm the IP formulation of the underlying problem is important. Formulations that need separation are causing a huge increase of running time. In this case, the Compact Formulation is much more affected than the Branch-and-price algorithm because the latter is just solving the IP in the exact pricing.
The Subsets Enumeration algorithm and the Set Partitioning algorithm are much more affected by an increase of l than the other two exact algorithms. On the other hand, an increase of n has a low effect on the running time of the first two algorithms and a high on the running time of the latter two. The Subsets Enumeration algorithm is better than the Set Partitioning algorithm just for $K = 2$. An increase of K slows down the Subsets Enumeration algorithm and the Compact Formulation significantly, whereas the Set Partitioning algorithm and the Branch-and-price method are nearly unaffected. In some

cases they are even faster for higher values of K.
The underlying problem Maximum Flow leads to solutions that are far away from the optimum if they are not optimal. We assume that the continuous variables in this problem are causing this effect.
The tests suggest that the heuristic is always able to compute solutions that are close to the optimum in a small amount of time. The only exception was the case in which the Maximum Flow Problem was the underlying problem. We observed that the refinement strategy works better if the underlying problem is hard to solve. The tests do not give a clear answer to the question, for which properties of the underlying problem using starting partitions computed by solving a clustering problem is beneficial. This seems to be a feature that has to be individually tested for a new underlying problem.
In the last section, we saw that using approximation algorithms or heuristics for solving the underlying problem can be a good alternative in practice. The tests suggest that the combination of solutions leads in average to a lower gap between the optimal solution and the computed solution for Problem (dmEm) than for the underlying problem.
We can conclude that the presented tests allow us to understand which algorithm is the most suitable based on different features of the problem or the instance. Additionally, we saw that by using heuristics or approximation algorithms even large-scale instances or instances with hard underlying problems can be solved with solutions that are close to the optimum.

Chapter 7

A Related Problem

In this chapter we want to introduce a problem that naturally arises out of Problem (dmEm) and that can be part of future research. In some underlying problems for Problem (dmEm) it is important to establish an infrastructure or to acquire a right of first refusal. For example one can think of a situation, in which rails have to be laid in order to minimize the average time of connection between different stations in different scenarios. In the methodology of (dmEm) after the scenario materialized, one of the K paths that have been prepared is chosen. Because all the rails are already laid and above all paid, it does not make sense to restrict oneself to the K precomputed solutions in this case, when solutions arising from combinations of them can be better. That means in our example that a route can be traveled combining rails from different solutions. Allowing these combinations of solutions, the Stochastic Two-Stage Infrastructure Problem arises.
We answer first questions about complexity and present an ILP formulation for solving it.

7.1 STIP with Continuous Variables

This section covers the Stochastic Two-Stage Infrastructure Problem (STIP), which can be seen as a variant of Problem (dmEm). At the beginning we consider the variant in which all variables are continuous. In practice this problem can model situations in which a right of first refusal has to be bought in advance to have later the possibility to purchase commodities or assets that

are needed in an optimization problem.
Let y be the variable that determines the decision of the implementation of the infrastructure. In the case of continuous variables, this means to determine the amount of each commodity that has to be reserved with the right of first refusal. Let x be the variable that corresponds to the decision taken after the scenario materializes. The vector c represents the cost of reserving the commodity and ξ_i indicates the problem specific cost in scenario i. The different costs can be measured in different units. The unit of the c vectors is in general monetary, whereas the ξ variables can represent totally different costs. We assume that the input is scaled so that it reflects the preferences of the user and that the value of an improvement of one unit in the costs indicated by c is equal to an improvement of the costs indicated by ξ by the same amount. The vector a introduces an additional knapsack constraint on the y variables, which can be seen as a budget constraint. Assuming a discrete set of scenarios leads to the following formulation:

$$\begin{aligned} \min_{y \in Y} \quad & c^\top y + \sum_{i=1}^{l} \min_{\substack{x_i \in X_i \\ x_i \leq y}} \xi_i^\top x_i \\ \text{s.t.} \quad & a^\top y \leq K, \end{aligned} \tag{STIP}$$

where $X_i, Y \subseteq \mathbb{R}^n$ for all $1 \leq i \leq l$. This problem can be modeled with the following LP-formulation:

$$\begin{aligned} \min \quad & c^\top y + \sum_{i=1}^{l} \xi_i^\top x_i \\ \text{s.t.} \quad & a^\top y \leq K \\ & x_i \leq y \qquad \forall\ 1 \leq i \leq l \\ & x_i \in X_i \qquad \forall\ 1 \leq i \leq l \\ & y \in Y \end{aligned} \tag{LPSTIP}$$

The objective value is equivalent to the objective of (STIP). The first constraint is just the knapsack constraint of the original problem and the second set of constraints ensures that in every dimension the values of the y variable is larger than the value of every x variable. Note that in particular, if the underlying problem permits an LP-formulation and because the y variables

are continuous here, we have a polynomial time algorithm for (STIP), which is a difference to Problem (dmEm) (see Remark 3.6). Hence, we see that the freedom of combining the precomputed solutions (that we do not have in many applications) leads to a model that is less challenging from the computational point of view. This observation was also supported by computational experiments for Problem (STIP), in which instances with higher values of l and n could be solved exactly.

7.2 STIP with (Mixed)-Integer Variables

Now we investigate the version of Problem (STIP) in which either the x or the y variables are not continuous anymore. If X_i is restricted to be integer for all scenarios i, one can achieve this implicitly by restricting only the y variables to be integer and drop the integrality constraint in the set X_i because any optimal solution is not affected by this. Naturally, the LP-formulation from the previous chapter can be used, but becomes an IP-formulation. Note that if Y is restricted to be integer, the problem contains a Knapsack Problem and is therefore NP-hard. In the following we want to show that also for convex Y, the problem can be NP-hard.

Theorem 7.1. *Problem* (STIP) *is NP-hard, even if* $Y = [0,1]^n$, $c \equiv 0$ *and* X_i *has polynomial size and is equal in every scenario.*

Proof. We reduce the Set Cover Decision Problem to the decision variant of Problem (STIP). Given an instance of Set Cover (U, S), where U is a set of elements and S is a set of subsets of U. In our construction we set $n := |S|$, X_i to be the set of base vectors of $\mathbb{R}^n$ for every scenario i, $Y := [0,1]^n$, and K to the maximum number of allowed subsets in the Set Cover decision problem, and $a := \mathbf{1}_n$. For every element $u \in U$ we define an objective vector:

$$(\xi_u)_j := \begin{cases} -1 & \text{if } u \in S_j, \\ 0 & \text{otherwise.} \end{cases}$$

The set cover is induced by Y as follows: the subset $S_j \in S$ belongs to the cover if and only if $y_j = 1$. We claim to have a set cover with K or less elements if and only if the objective value of Problem (STIP) is $-|U|$. Let us assume

we have a set cover. For every element u_i there exists a subset containing this element and therefore there exists at least one $x \in X_i$ with $x \leq y$ whose product with ξ_i is -1. Summing up the minimum values of all scenarios we get $-|U|$. Let us assume the solution of (STIP) has a value of $-|U|$. The minimum value for every product of ξ_i and x is -1. Therefore for every ξ_i there exists an x, whose product with ξ_i is -1. Hence every element is covered. □

If the knapsack constraint is dropped, the previous argumentation that Problem (STIP) is hard if Y is restricted to be integer, does not work anymore. Nevertheless, we can show that in this case the problem remains NP-hard.

Theorem 7.2. *If Y is restricted to be integer, Problem* (STIP) *is NP-hard, even if $a \equiv 0$ and X_i has polynomial size and is equal in every scenario.*

Proof. Again we reduce the Set Cover Decision Problem to the decision variant of (STIP). In our construction we set $n := |S|$, X_i to be the set of base vectors of $\mathbb{R}^n$ in every scenario i, $Y = \{0,1\}^n$ and $c \equiv 1$ and we set K to the maximum number of allowed subsets in the Set Cover Decision Problem. For every element $u \in U$ we define an objective vector:

$$(\xi_u)_j := \begin{cases} -K & \text{if } u \in S_j, \\ 0 & \text{otherwise.} \end{cases}$$

The Set Cover is induced by Y as follows: the subset $S_j \in S$ is in the set cover, if $y_j = 1$. We have a set cover with K or less elements if and only if the objective value of (STIP) is lower or equal to $(1-l)K$. Let us assume we have a Set Cover with at most K sets. For every element u there exists a subset containing this element and therefore there exists at least one $x \in X_u$ with $x \leq y$ whose product with ξ_i is $-K$. Therefore our objective value is equal to the sum of $-Kl$ and the number of sets we used in our set cover, which is smaller or equal to K.
Now we consider the case of a solution that is infeasible for the Set Cover Problem. First we investigate the case that the solution of Set Cover does not cover all elements. The minimum objective value in that case corresponds to the subcase, when only one element is not covered and we only use one set. Therefore the minimum objective value is equal to $K(1-l)+1$ and

hence the corresponding solution of (STIP) corresponds to a not accepted instance. Finally we assume that a solution of Set Cover takes more than K sets. The product $\xi_u^\top x$ cannot take a value smaller than $-K$. For this reason the minimum value in this case is $-Kl + K + 1 = K(l-1) + 1$. In summary we have proven that the objective value of Problem (STIP) is smaller or equal to $K(1-l)+1$ if and only if it induces a Set Cover with K or less solutions. □

In the following we want to show that the evaluation of the objective function of Problem (STIP) becomes hard in general. For this, we need to prove some technical results first. Note that the distribution that is used in the following is discrete, but has an exponential number of realizations, which are not explicitly given as input. Therefore, enumerating all realizations is not possible in polynomial time.

Definition 7.3. Let $\mu_d \sim \mathcal{U}\{0,1\}$ for all dimensions $d = 1, \ldots, n$ be i.i.d., i.e., μ is discrete and uncorrelated and each vector in $\{0,1\}^n$ appears with probability 2^{-n}. For given $y \in (\mathbb{Q} \cap [0,1])^n$, $0 < K \in \mathbb{Q}$ let

$$f_K(y) := E_\mu(\max\{\mu^\top x \mid \mathbf{1}^\top x \leq K,\ 0 \leq x \leq y, x \in \mathbb{Q}^n\}).$$

For the case where all upper bounds y are equal to one, we have

Lemma 7.4. *We can compute $f_K(\mathbf{1})$ efficiently.*

Proof.

$$\begin{aligned} f_K(e) &= E_\mu(\max\{\mu^\top x \mid \mathbf{1}^\top x \leq K,\ 0 \leq x \leq 1, x \in \mathbb{Q}^n\}) \\ &= 2^{-n} \sum_{\mu \in \{0,1\}^n} \min\{K, \mathbf{1}^\top \mu\} \\ &= 2^{-n}\Big(\sum_{\substack{\mu \in \{0,1\}^n \\ \mathbf{1}^\top \mu \leq \lfloor K \rfloor}} \mathbf{1}^\top \mu + \sum_{\substack{\mu \in \{0,1\}^n \\ \mathbf{1}^\top \mu \geq \lfloor K \rfloor + 1}} K \Big) \\ &= 2^{-n}\Big(\sum_{j=0}^{\lfloor K \rfloor} \binom{n}{j} j + \sum_{j=\lfloor K \rfloor + 1}^{n} \binom{n}{j} K \Big) . \end{aligned}$$

□

Lemma 7.5. *For all $y \in [0,1]^n$ and $K \geq 1$, we have $f_K(y) = K f_1(\frac{1}{K}y)$.*

Proof.

$$\begin{aligned} f_K(y) &= E_\mu(\max\{\mu^\top x \mid \mathbf{1}^\top x \leq K,\ 0 \leq x \leq y, x \in \mathbb{Q}^n\}) \\ &= E_\mu(\max\{\mu^\top (Kx) \mid \mathbf{1}^\top (Kx) \leq K,\ 0 \leq (Kx) \leq y, x \in \mathbb{Q}^n\}) \\ &= K E_\mu(\max\{\mu^\top x \mid \mathbf{1}^\top x \leq 1,\ 0 \leq x \leq \tfrac{1}{K}y, x \in \mathbb{Q}^n\}) \\ &= K f_1(\tfrac{1}{K}y)\ . \end{aligned}$$

□

Note that for $K = 1$ the polytope $\{x \in [0,1]^n \mid \mathbf{1}^\top x \leq 1\}$ has only $n+1$ vertices. Nevertheless, for general y, even the computation of f_1 is hard:

Lemma 7.6. *It is #P-hard to compute $f_1(y)$ for given y.*

Proof. Let $a \in \mathbb{Z}^n$ and $b \in \mathbb{Z}$ with $0 \leq a_i \leq b$ for all $i = 1, \ldots, n$. We show that, given an oracle for computing $f_1(y)$ efficiently for all y, we can determine

$$\#\{x \in \{0,1\}^n \mid a^\top x \leq b\}\ .$$

Since the latter counting problem is #P-hard [17], the result then follows. Set $y := \frac{1}{b}a$, then $y \in [0,1]^n$. For all K, we have

$$\begin{aligned} f_K(y) &= E_\mu(\max\{\mu^\top x \mid \mathbf{1}^\top x_i \leq K,\ 0 \leq x \leq y, x \in \mathbb{Q}^n\}) \\ &= 2^{-n} \sum_{\mu \in \{0,1\}^n} \min\{K, y^\top \mu\}\ . \end{aligned}$$

Define $\varepsilon := \frac{1}{b}$. For $\mu \in \{0,1\}^n$, we obtain

$$\min\{1+\varepsilon, y^\top \mu\} - \min\{1, y^\top \mu\} = \begin{cases} 0 & \text{if } y^\top \mu \leq 1 \\ \varepsilon & \text{otherwise,} \end{cases}$$

since $y^\top \mu > 1$ implies $y^\top \mu \geq 1 + \varepsilon$ by the choice of ε. Now

$$\begin{aligned} & \#\{\mu \in \{0,1\}^n \mid y^\top \mu \leq 1\} \\ = \ & 2^n - \#\{\mu \in \{0,1\}^n \mid y^\top \mu > 1\} \\ = \ & 2^n - \sum_{\mu \in \{0,1\}^n} \frac{1}{\varepsilon}(\min\{1+\varepsilon, y^\top \mu\} - \min\{1, y^\top \mu\}) \\ = \ & 2^n - 2^n \frac{1}{\varepsilon}(2^{-n} \sum_{\mu \in \{0,1\}^n} \min\{1+\varepsilon, y^\top \mu\} - 2^{-n} \sum_{\mu \in \{0,1\}^n} \min\{1, y^\top \mu\}) \\ = \ & 2^n - 2^n \tfrac{1}{\varepsilon}(f_{1+\varepsilon}(y) - f_1(y)) \\ \overset{*}{=} \ & 2^n(1 - \tfrac{1}{\varepsilon}((1+\epsilon) f_1(\tfrac{1}{1+\varepsilon}y) - f_1(y)))\ , \end{aligned}$$

so that using our oracle for f_1 we can compute

$$\#\{\mu \in \{0,1\}^n \mid y^\top \mu \leq 1\} = \#\{\mu \in \{0,1\}^n \mid a^\top \mu \leq b\} \ .$$

For (*) we used Lemma 7.5. □

Theorem 7.7. *If $\xi \sim \mathcal{U}\{-1,0\}^n$, evaluating the objective function of* (STIP) *for fixed y is #P-hard.*

Proof. For $\mu = -\xi$ and fixed y the function $f_1(y)$ corresponds to the objective function of (STIP), from which the constant term $c^\top y$ is subtracted. Here, the set of feasible solutions is equal to the set of base vectors of $\mathbb{Q}^n$ in every scenario. By using Lemma 7.6, the desired statement follows. □

We showed that Problem (STIP) becomes NP-hard if parts of the variables are restricted to be integer and that evaluating the objective function can become $\#P$-hard when the vector ξ is componentwise discrete distributed. Additionally, we obtained an ILP formulation for solving (STIP). Future research on Problem (STIP) can include approximation algorithms or the proof that these do not exist and exact algorithms that are faster compared to standard ILP-based approaches.

Chapter 8

Conclusion

In this thesis we investigated the Min-E-Min Problem. Among other complexity results we proved that it is NP-hard, W[2]-hard parameterized by K and that the existence of a polynomial time constant factor approximation algorithm implies that P is equal to NP. We showed that the version of the Min-E-Min Problem with uncertain parameters following a continuous distribution is even harder since the evaluation of the objective function becomes already #P-hard. In this case we proposed to discretize the distribution by a set of samples and showed that under some realistic assumptions the solution of the discretized problem converges to the solution of the original problem when the number of samples approaches infinity. We proposed several exact and heuristic solution methods and compared them in an extensive computational study and demonstrated that each algorithm can be useful according to the setting of the problem. Additionally, we examined how the solution value of oracle-based exact algorithms is affected if the underlying problem is solved with an approximation algorithm.

Future research could focus on combinatorial algorithms for solving Problem (mEmC) so that the Branch-and-price method for Problem (dmEm) does not need to use an MILP-solver anymore. Moreover, the complexity of Problem (dmEm) in terms of approximation can be studied for $K = 2$ and the question whether there exists an fpt algorithm for Problem (dmEm) parameterized by $l - K$ can be examined.

Bibliography

[1] A. Ben-Tal, L. El Ghaoui, and A. Nemirovski. *Robust Optimization*. Princeton Series in Applied Mathematics, 2009.

[2] D. Bertsimas and C. Caramanis. Finite adaptability in multistage linear optimization. *IEEE Transactions on Automatic Control*, 55(12):2751–2766, 2010.

[3] D. Bertsimas and M. Sim. The price of robustness. *Operations Research*, 52(1):35–53, 2004.

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

[5] E. Bonnet and V. Th. Paschos. Parameterized (in)approximability of subset problems. *CoRR*, abs/1310.5576, 2013. URL `http://arxiv.org/abs/1310.5576`.

[6] C. Buchheim and J. Kurtz. Min-max-min robust combinatorial optimization. *Mathematical Programming (Series A)*, 163(1–2):1–23, 2017.

[7] C. Buchheim and J. Kurtz. Complexity of min-max-min robustness for combinatorial optimization under discrete uncertainty. *Discrete Optimization*, 28:1–15, 2018.

[8] C. Buchheim and J. Pruente. k-adaptability in stochastic combinatorial optimization under objective uncertainty. *European Journal of Operational Research*, 277:953–963, 2019.

[9] A. Charnes and W. W. Cooper. Chance-constrained programming. *Management Science*, 6(1):73–79, oct 1959.

[10] A. Chassein, M. Goerigk, J. Kurtz, and M. Poss. Min-Max-Min Robustness for Combinatorial Problems with Budgeted Uncertainty. *ArXiv e-prints*, February 2018.

[11] N. Christofides. Worst-case analysis of a new heuristic for the travelling salesman problem. 1976.

[12] S. A. Cook. The complexity of theorem-proving procedures. In *Proceedings of the Third Annual ACM Symposium on Theory of Computing*, pages 151–158, 1971.

[13] R. G. Downey and M. R. Fellows. Fixed-parameter tractability and completeness I: Basic results. *SIAM Journal on Computing*, 24(4):873–921, 1995.

[14] J. Edmonds. Paths, trees, and flowers. *Canadian Journal of Mathematics*, 17:449–467, 1965.

[15] J. Edmonds and R. M. Karp. Theoretical improvements in algorithmic efficiency for network flow problems. *J. ACM*, 19(2):248–264, 1972.

[16] J. Flum and M. Grohe. *Parameterized Complexity Theory*. Springer-Verlag Berlin Heidelberg, 2006.

[17] M. R. Garey and D.S. Johnson. *Computers and Intractability; A Guide to the Theory of NP-Completeness*. W. H. Freeman & Co., USA, 1990.

[18] M. R. Garey, D. S. Johnson, and L. J. Stockemeyer. Some simplified NP-complete graph problems. *Theoretical Computer Science*, 1:237–267, 1976.

[19] M. R. Garey, D. S. Johnson, and H. Witsenhausen. The complexity of the generalized lloyd - max problem (corresp.). *IEEE Transactions on Information Theory*, 28(2):255–256, 1982.

[20] G. A. Hanasusanto, D. Kuhn, and W. Wiesemann. K-adaptability in two-stage robust binary programming. *Operations Research*, 63(4):877–891, 2015.

[21] G. A. Hanasusanto, D. Kuhn, and W. Wiesemann. K-adaptability in two-stage distributionally robust binary programming. *Operations Research Letters*, 44(1):6 – 11, 2016. ISSN 0167-6377.

[22] J. Håstad. Some optimal inapproximability results. *Journal of the ACM*, 48:798–869, 2001.

[23] C. Hierholzer. Ueber die möglichkeit, einen linienzug ohne wiederholung und ohne unterbrechung zu umfahren. *Mathematische Annalen*, 1873.

[24] J.P. Kelly and J. Xu. A set-partitioning-based heuristic for the vehicle routing problem. *INFORMS Journal on Computing*, 11:161–172, 1999.

[25] J. B. Kruskal. On the shortest spanning subtree of a graph and the traveling salesman problem. *Proceedings of the American Mathematical Society*, 7(1):48–50, 1956.

[26] N. Livne. A note on #p-completeness of np-witnessing relations. *Information Processing Letters*, 109(5):259–261, 2009.

[27] S. Lloyd. Least squares quantization in pcm. *IEEE Transactions on Information Theory*, 28(2):129–137, 1982.

[28] E. Malaguti, M. Monaci, and P. Toth. A metaheuristic approach for the vertex coloring problem. *INFORMS Journal on Computing*, 20:302–316, 2008.

[29] E. Malaguti, M. Monaci, and J. Pruente. K-adaptability in stochastic optimization. Technical report, 2019. URL `http://www.optimization-online.org/DB_HTML/2020/04/7759.html`.

[30] M. Monaci and P. Toth. A set-covering-based heuristic approach for bin-packing problems. *INFORMS Journal on Computing*, 18:71–85, 2006.

[31] M. Orlov. Efficient generation of set partitions. Technical report, 2002.

[32] A. Paz and S. Moran. Non deterministic polynomial optimization problems and their approximations. *Theoretical Computer Science*, 15(3):251 – 277, 1981. ISSN 0304-3975.

[33] A. Shapiro, D. Dentcheva, and A. Ruszczyński. *Lectures on stochastic programming: modeling and theory*. SIAM, 2014.

[34] M. Stoer and F. Wagner. A simple min-cut algorithm. *J. ACM*, 44(4): 585–591, 1997.

[35] A. Subramanyam, C. E. Gounaris, and W. Wiesemann. K-Adaptability in Two-Stage Mixed-Integer Robust Optimization. *Mathematical Programming Computation*, 2019.

[36] L.G. Valiant. The complexity of computing the permanent. *Theoretical Computer Science*, 8(2):189 – 201, 1979.