Abstract We address combinatorial optimization problems with uncertain objective functions, given by discrete probability distributions. Within this setting, we investigate the so-called K-adaptability approach: the aim is to calculate a set of $k$ feasible solutions such that the objective value of the best of these solutions, calculated in each scenario independently, is optimal in expectation. We show that this problem is NP-hard even if the underlying certain problem is trivial, and present further complexity results concerning approximability and fixed-parameter tractability with respect to $k$. Moreover, we present exact solution methods as well as a heuristic for this problem and compare them in an extensive experimental evaluation, where the underlying problem is the Unconstrained Binary Optimization Problem, the Shortest Path Problem or the Spanning Tree Problem. It turns out that the performance and the ranking of these approaches strongly depends on the parameter $k$ and on the number of scenarios.

1 Introduction

Consider a general combinatorial optimization problem of the form

$$\min \xi^\top x$$

s.t. $x \in X$, (P)

where $X \subseteq \{0, 1\}^n$ describes the set of feasible solutions and $\xi \in \mathbb{Q}^n$ defines a linear objective function. In practice, the vector $\xi$ in Problem (P) is often

Christoph Buchheim
Fakultät für Mathematik, TU Dortmund, Vogelpothsweg 87, 44227 Dortmund, Germany
E-mail: christoph.buchheim@math.tu-dortmund.de

Jonas Pruente
Fakultät für Mathematik, TU Dortmund, Vogelpothsweg 87, 44227 Dortmund, Germany
E-mail: jonas.pruente@math.tu-dortmund.de
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.

To address such uncertainties, two different approaches have been investigated intensively in the literature: the robust optimization approach [1] and the stochastic optimization approach [4]. In robust optimization, the aim is to find a solution that is worst-case optimal for a given set of possible realizations of \( \xi \). As the strict application of this approach is well-known to lead to very conservative solutions [3], many alternative paradigms for robust optimization have been proposed recently. On of them is the so-called K-adaptability approach, first presented by Bertsimas and Caramanis [2]: instead of one single solution, they choose a set of up to \( k \) feasible solutions such that the best one of them, determined separately in each scenario, is worst-case optimal. The idea is that the user may still choose from a small set of precomputed solutions after the uncertainty is realized. This approach has been further investigated by Hanasusanto et al. [13,14], Buchheim and Kurtz [6,7], Subramanyam et al. [17], and Chassein et al. [9].

In this paper, we consider the same paradigm in the context of stochastic programming, where instead of the worst case one aims at optimizing the expected objective value of a solution. Since the expected value is linear, the standard stochastic approach is pointless for problem (P), having a linear objective and certain constraints. However, the situation changes when considering the K-adaptable approach: we then search for a set of \( k \) feasible solutions such that the expected objective value of the respective best of these solutions is optimal. Formally, this leads to a problem of the form

\[
\min \mathbb{E} \left( \min \{ \xi^\top x_1, \ldots, \xi^\top x_k \} \right)
\]

s.t. \( x_1, \ldots, x_k \in X \).

In the following, we always assume that the random variable \( \xi \) is discrete, i.e., we deal with a finite set of scenarios represented by the corresponding objective vectors \( \xi_1, \ldots, \xi_l \in \mathbb{Q}^n \), appearing with probabilities \( p_1, \ldots, p_l \geq 0 \) such that \( p_1 + \cdots + p_l = 1 \). The problem then becomes

\[
\min \sum_{j=1}^l p_j \min \{ \xi_j^\top x_1, \ldots, \xi_j^\top x_k \}
\]

s.t. \( x_1, \ldots, x_k \in X \). \hspace{1cm} (mEm)

Such an approach is reasonable whenever the computation of an optimal solution from scratch – after the scenario materializes – is too expensive: the problem may be too hard to be solved quickly, or the alternative solutions may have to be prepared in advance. Another reason may be that a human user, e.g., a truck driver, may not be willing to accept an unfamiliar solution every time he has to perform a certain task, but prefers solutions taken from a small set of candidates.

Contribution Our main contribution is an investigation of the complexity of Problem (mEm), where the input consists of the probabilities \( p_1, \ldots, p_l \) and
the corresponding objective vectors \(\xi_1, \ldots, \xi_l \in \mathbb{Q}^n\). Considering \(k\) as part of the input, we show that the problem is not only NP-hard in most situations, even when the underlying problem \((P)\) is trivial for each fixed \(\xi\), we also show that it is not approximable and that an FPT algorithm cannot be expected in parameter \(k\). These results even hold if the feasible set \(X\) has polynomial size. For the case of a fixed \(k \geq 2\), we can still show NP-hardness for \(X = \{0,1\}^n\). In the second part of the paper, we propose different exact and heuristic methods for solving Problem \((mEm)\). The heuristic approach is based on considering \(k\)-partitions of the scenario set, it turns out to yield very good results in many different settings, while all exact methods we present suffer from a large number of scenarios.

Outline The rest of this paper is organized as follows. In the next section, we introduce the main definitions and discuss the connection between Problem \((mEm)\) and partitions of the scenario set. In Section 3, we analyse the complexity of Problem \((mEm)\) in terms of NP-hardness, approximability and fixed-parameter tractability, respectively, and derive hardness results also for min-max-min robust optimization as a by product. In Section 4, we describe different approaches to solve Problem \((mEm)\) exactly, while in Section 5 we develop a fast heuristic method. All algorithms are compared in an extensive test series with different underlying problems in Section 6. Section 7 concludes.

2 Preliminaries

Our aim is to investigate Problem \((mEm)\), where the input consists of the objective vectors \(\xi_1, \ldots, \xi_l \in \mathbb{Q}^n\) with corresponding probabilities \(p_1, \ldots, p_l\). For simplicity, we always integrate the probability \(p_i\) into vector \(\xi_i\) in the following, i.e., we replace \(\xi_i\) by \(p_i \xi_i\) and thus omit the probabilities \(p_i\) in \((mEm)\).

We will consider both the situation where the number \(k\) of solutions is fixed and where it is part of the input. For the set \(X\) of feasible solutions, we make no assumptions except that, for sake of simplicity, we restrict ourselves to the case \(X \subseteq \{0,1\}^n\). The idea is that \(X\) is specified by means of an oracle that can solve the certain problem \((P)\) for any given objective vector \(\xi\). When proving hardness results, we will show that they hold even for specific sets \(X\), in these cases we always consider feasible sets \(X\) where \((P)\) can be solved efficiently. We address the following two variants of Problem \((mEm)\):

Definition 1 The Discrete Min-E-Min Decision Problem is defined as follows: Given \(X \subseteq \{0,1\}^n\), defined implicitly by an oracle for Problem \((P)\), objective vectors \(\xi_1, \ldots, \xi_l \in \mathbb{Q}^n\), as well as \(k, l \in \mathbb{N}\) and \(b \in \mathbb{Q}\), decide whether there exist \(x_1, \ldots, x_k \in X\) such that \(\sum_{j=1}^l \min\{\xi_j^\top x_1, \ldots, \xi_j^\top x_k\} \leq b\). The Discrete Min-E-Min Problem is the optimization version of this problem.

In the resulting solution, each scenario \(\xi_j\) is covered by at least one of the feasible solutions \(x_1, \ldots, x_k \in X\), namely one where \(\min\{\xi_j^\top x_1, \ldots, \xi_j^\top x_k\}\) is attained. This leads to a partition of the set of scenarios into at most \(k\) parts.
In the following, we will call this the \textit{partition induced by} $x_1, \ldots, x_k \in X$. If one scenario is covered by more than one solution, we assign it lexicographically. Conversely, if such a partition $\{1, \ldots, l\} = \bigcup_{i=1}^k I_i$ is given, one can construct an \textit{induced set of solutions} $x_1, \ldots, x_k$ by choosing

$$x_i \in \arg\min_{x \in X} \sum_{j \in I_i} \xi^j x$$

using the oracle for the certain problem (P) with objective function $\sum_{j \in I_i} \xi^j$. 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.

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 (mEm), which is discussed in Section 5 below.

The induced set of solutions is clearly optimal for the given partition and therefore Problem (mEm) can be solved by calling the oracle for each possible $k$-partition of $\{1, \ldots, l\}$. Moreover, it is easy to verify that it suffices to consider partitions without empty subsets. In particular, we obtain

**Theorem 1** The Discrete \textit{Min-E-Min} Problem can be polynomially reduced to the underlying certain problem (P) if either the number of scenarios $l$ or the difference $l - k$ is bounded.

**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 $l - k$. \hfill $\square$

### 3 Complexity Results

For investigating the complexity of the Discrete Min-E-Min Decision Problem, we distinguish between the problem variant where the parameter $k$ is part of the input and the variant where it is fixed, showing NP-hardness (Section 3.1) and inapproximability (Section 3.2) results for both cases, and finally discuss fixed-parameter tractability in terms of $k$. It will turn out that many results even hold when $X$ has polynomial size, in which case the underlying certain problem (P) could even be solved by enumeration.
3.1 NP-hardness

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

**Theorem 2** The Discrete Min-E-Min Decision Problem belongs to NP if membership in $X$ can be tested in polynomial time.

In the following, we distinguish between two variants of the problem: we first consider the number $k$ of solutions 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 has a polynomial number of feasible solutions, i.e., when the set $X$ 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 assignment of scenarios to the $k$ chosen solutions.

**Theorem 3** If $k$ is part of the input, the Discrete Min-E-Min Decision Problem is strongly NP-hard. This remains true even if $|X|$ is polynomial.

**Proof** We use a reduction from the Dominating Set Problem. 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 $k$ \[11\].

For the reduction, let $G = (V, E)$ be the given graph. We choose $X$ as the set of unit vectors in $\mathbb{Q}^V$. For every $v \in V$, we define a scenario $\xi_v \in \mathbb{Q}^V$ by

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

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 Discrete Min-E-Min Problem 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 exist $u \in D$ with $(\xi_v)_u = -1$. Hence $\{e_u \mid u \in D\}$ is a solution set for the Discrete Min-E-Min Problem with

\[
\sum_{v \in V} \min_{u \in D} \{\xi_v^T e_u\} = -|V|.
\]

Conversely, if the Discrete Min-E-Min Problem has a solution $Y \subseteq X$ of size $k$ with value equal to $-|V|$, then for every scenario $\xi_v$ there must exist a unit vector $e_u \in Y$ with $(\xi_v)^T e_u = -1$. Therefore $\{u \in V \mid e_u \in Y\}$ is a dominating set of size $k$ in $G$. \[\square\]

Together with Theorem 2, 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, even if $|X|$ is polynomial.
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:

**Theorem 4** For any fixed $k \geq 2$, the Discrete Min-E-Min Decision Problem is strongly 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. It is NP-complete to decide whether a given graph admits a vertex coloring with $k$ colors when $k \geq 3$ [12], and the problem remains NP-complete even if a vertex is allowed to have more than one color.

Given a graph $G = (V, E)$, we consider the feasible set $X = \{0, 1\}^V$ and define a scenario $\xi_v \in \mathbb{Q}^V$ for each node $v \in V$ by

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

Now for given $x_1, \ldots, x_k \in X$ we have $\sum_{v \in V} \min\{\xi_v^\top x_1, \ldots, \xi_v^\top x_k\} \leq -|V|$ if and only if the sets $V_i := \{v \in V \mid (x_i)_v = 1\}$ for $i \in \{1, \ldots, k\}$ are independent sets covering $V$. This implies the result in the first case.

For the case $k = 2$, we reduce from the NP-complete decision variant of the Maximum Cut Problem [12]. We use the same construction as before, except that we set

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

Then we claim that $G$ has a cut of cardinality at least $b$ if and only if there exist $x_1, x_2 \in \{0, 1\}^V$ with $\sum_{v \in V} \min\{\xi_v^\top x_1, \xi_v^\top x_2\} \leq -2b$. Indeed, if $W \subseteq V$ with $|\delta(W)| \geq b$, 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

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

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

Different from the case of an unbounded $k$, we cannot expect a construction with a polynomially sized feasible set $X$ any more in Theorem 4, since for fixed $k$ the number of solutions $|X|^k$ is polynomial then, so that the problem could be solved efficiently by enumeration.
3.2 Inapproximability

Having shown that the Discrete Min-E-Min Problem 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 \( k \).

**Theorem 5** If \( k \) is part of the input, the Discrete Min-E-Min Problem does not belong to APX unless \( P = NP \), even if \( X \) has polynomial size and the scenarios are restricted to be non-negative.

**Proof** Assume that there exists an \( \alpha \)-approximation algorithm for the Discrete Min-E-Min Problem, 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 \( P = NP \). Given a graph \( G = (V, E) \), let \( X \) be the set of unit vectors in \( \mathbb{Q}^V \) and define a scenario \( \xi_e \geq 0 \) for each \( e \in E \) by

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

Now if \( U \subseteq V \) is a vertex cover of \( G \) with \( |U| = k \), we can consider the \( k \) solutions \( \{e_v \mid v \in U\} \subseteq X \), for which we obtain

\[
\sum_{e \in E} \min_{v \in U} \xi_e^T e_v \leq |E|.
\]

Otherwise, if no such vertex cover exists, we have

\[
\sum_{e \in E} \min_{x_1, \ldots, x_k} \{\xi_e^T x_1, \ldots, \xi_e^T x_k\} \geq (\alpha - 1)|E| + 2 + (|E| - 1) = \alpha|E| + 1
\]

for all \( x_1, \ldots, x_k \in X \). This implies the result. \( \square \)

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 4, we thus used \( X = \{0,1\}^n \). However, when restricting ourselves to non-negative objective functions, the feasibility of the zero vector obviously makes the Discrete Min-E-Min Problem 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 6** For any fixed \( k \geq 3 \), we cannot decide whether the optimal value of the Discrete Min-E-Min Problem is zero, even if all feasible solutions have non-negative objective value and \( X = \{x \in \{0,1\}^n \mid x_n = 1\} \).

**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 $X = \{0, 1\}^V \times \{1\}$ and define a scenario $\xi_v$ 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 the Discrete Min-E-Min Problem for this instance is zero if and only if $G$ admits a vertex coloring with $k$ colors.

**Corollary 1** For fixed $k \geq 3$, the Discrete Min-E-Min Problem does not belong to APX unless $P = NP$, even if all feasible solutions have non-negative objective value and $X = \{x \in \{0, 1\}^n \mid x_n = 1\}$.

For the case $k = 2$, we do not know whether the Discrete Min-E-Min Problem belongs to APX. However, we obtain the following weaker result, which follows from the corresponding well-known result for the Maximum Cut Problem [15] and the construction in the proof of Theorem 4:

**Theorem 7** For $k = 2$, approximating the Discrete Min-E-Min Problem with a factor better than $\frac{17}{16}$ is NP-hard.

### 3.3 Parameterized Complexity

For the complexity results obtained so far, the main distinction was made between the two variants of the Discrete Min-E-Min Problem 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 (fixed-parameter tractable) algorithms. An FPT algorithm with respect to parameter $k$ is an algorithm whose running time is only exponential in $k$. More precisely, it is polynomial in the instance size except for a possible factor $f(k)$, for an arbitrary function $f$. For every fixed $k$, the running time is thus polynomial, and the degree of the polynomial does not depend on $k$.

In Theorem 4, we have shown that the Discrete Min-E-Min Problem 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, 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 8** The Discrete Min-E-Min Decision Problem is W[2]-hard for parameter $k$, even if $|X|$ is polynomial.
Proof Dominating Set is one of the most well-known W[2]-hard problems when considering the size of the set as parameter [11]. The reduction used in the proof to Theorem 3 is a parameterized reduction for $k$ because we use the same $k$ for both problems. This proves the statement.

It follows from Theorem 8 that an FPT algorithm for the Discrete Min-E-Min Problem 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:

Theorem 9 The Discrete Min-E-Min Decision Problem is W[1]-hard for parameter $n-k$, even if $|X|$ is polynomial.

Proof We construct a parameterized reduction from the decision variant of the Set Cover Problem. We are thus given a finite set $U$ and a collection of subsets $s_1,\ldots, s_n$ of $U$ as well as the number $k$, and the question is whether there exist $k$ sets $s_{i_1},\ldots, s_{i_k}$ such that $U = \bigcup_{j=1}^k s_{i_j}$. Now let $X$ consist of all unit vectors in $\mathbb{Q}^n$ and define, for every $u \in U$, a scenario $\xi_u \in \mathbb{Q}^n$ 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 $\{e_{i_1},\ldots, e_{i_k}\} \subseteq X$ with

$$\sum_{u \in U} \min_{j \in \{1,\ldots, k\}} \xi_u^T e_{i_j} = -|U|.$$ 

Conversely, for every solution $e_{i_1},\ldots, e_{i_k}$ 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.

3.4 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 [7]: instead of the expected value, we ask for the worst case, resulting in the problem

$$\min \max_{j \in \{1,\ldots, l\}} \min \{\xi_j^T x_1,\ldots, \xi_j^T x_k\} \quad \text{(mmm)}$$

s.t. $x_1,\ldots, x_k \in X$.

Using the same reduction as in the proof of Theorem 3, we can show that Problem (mmm) is strongly NP-hard when $k$ is part of the input, even if $|X|$ is polynomial. 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 5, we can show that Problem (mmm) is not in APX when $k$ is part the input, even if $|X|$ is polynomial. The only change is that we can simplify the construction to

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

Finally, also the results of Theorem 8 and Theorem 9 can be carried over to Problem (mmm) using the same reductions.

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 Section 6. Heuristic methods are proposed in Section 5.

The methods differ in the way they depend on the underlying problem (P): some of the approaches only need an oracle for (P), others require an ILP formulation for the latter or even a complete enumeration of all elements of $X$. Moreover, as we will see, their performance depends strongly on the parameters $k$, $l$, and $|X|$, and different methods turn out to be preferable in different settings.

We start by describing two types of enumeration algorithms and then propose two approaches based on branch-and-bound.

4.1 Enumeration of Feasible Solutions

The most straightforward method to solve the Discrete Min-E-Min Problem is to compute all subsets of $X$ of size $k$, to calculate their objective values, and to choose the best solution obtained. Clearly, such a complete enumeration is only reasonable when $k$ and $|X|$ are very small, whereas the number of scenarios $l$ only has a minor impact on running time. In fact, we will see later that enumeration may outperform other solution methods for small $|X|$ and $k$ and huge $l$. 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.

4.2 Enumeration of Partitions

Another way of solving the Discrete Min-E-Min Problem exactly is to compute all possible partitions of scenarios, to calculate the induced solutions, and to return the best one of these. As discussed in Section 2, the number of partitions that we need to consider corresponds to the Sterling number of the second kind,
all other partitions are symmetric to one of these and therefore do not need to be considered. An efficient way of enumerating the relevant $S_{l,k}$ partitions is described in [16].

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 cardinality of $X$ does not affect the running time explicitly, but only via a potentially longer solution time for the underlying problem.

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 binary variables: for $j \in \{1, \ldots, k\}$, the vector $x_j \in X$ 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. We obtain the following IQP-formulation of the Discrete Min-E-Min Problem:

$$\min \sum_{i=1}^{l} \sum_{j=1}^{k} y_{ij} \xi_{i}^{\top} x_j$$

$$\text{s.t.} \sum_{j=1}^{k} y_{ij} = 1 \quad \forall i \in \{1, \ldots, l\}$$
$$y_{ij} \in \{0, 1\} \quad \forall i \in \{1, \ldots, l\}, j \in \{1, \ldots, k\}$$
$$x_j \in X \quad \forall j \in \{1, \ldots, k\}$$

(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 $x_1, \ldots, x_k$.

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. Consequently, the branching decisions only need to take $y$-variables into account. When all $y$-variables are fixed, the oracle can be used to compute the $k$ solutions $x_1, \ldots, x_k$: the solution $x_j$ is the optimal one for the objective function

$$\sum_{i=1}^{l} \xi_i .$$

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 \(j > i\). It is also possible to reduce the number of children further such that the total number of leaves of the branch-and-bound tree is exactly \(S_{l,k}\), by adapting the methods of [16]. However, such a branching rule is not easy to implement inside off-the-shelf MIP solvers, so that we restrict ourselves to eliminating variable \(y_{ij}\) for \(j > i\).

### 4.4 Branch-and-bound Based on Separable Underestimators

An oracle-based approach for solving quadratic versions of combinatorial optimization problems has been presented in [8]. The idea is to underestimate the given quadratic objective by a separable quadratic function and then by an equivalent linear function. The resulting problem is solved by an oracle for the linear version of the combinatorial optimization problem, the result yields a dual bound that can be integrated into a branch-and-bound scheme again. For details of the algorithm, see [8].

When applying this approach to the quadratic formulation (IQP), i.e., when replacing the quadratic objective by a linear one, the resulting linear problem decomposes into \(k+1\) subproblems: the first one only concerns the set of \(y\)-variables and can be solved by simply choosing, for each \(i = 1, \ldots, l\), the variable \(y_{ij}\) with highest objective coefficient, setting the resulting variable to one. The remaining \(k\) subproblems ask for optimizing linear functions over \(X\), which is exactly what our oracle for problem (P) can do for us.

It is thus possible to directly apply the method of [8] to our problem formulation, thus obtaining an oracle-based branch-and-bound method for solving (IQP). As described in the previous section, we can reduce the symmetry in the problem by fixing \(y_{ij} = 0\) for \(i > j\).

### 5 Heuristic Algorithms

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:

<table>
<thead>
<tr>
<th>Basic Heuristic</th>
</tr>
</thead>
<tbody>
<tr>
<td>1: Choose an arbitrary partition (P) of ({1, \ldots, l}).</td>
</tr>
<tr>
<td>2: repeat</td>
</tr>
<tr>
<td>3: \texttt{Set } (P' := P).</td>
</tr>
<tr>
<td>4: Compute the solution set (X) induced by (P').</td>
</tr>
<tr>
<td>5: Compute the partition (P) induced by (X).</td>
</tr>
<tr>
<td>6: until (P = P')</td>
</tr>
<tr>
<td>7: return (X)</td>
</tr>
</tbody>
</table>
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-decreasing. We claim that the algorithm always terminates, i.e., that it cannot get stuck in a non-trivial cycle. In fact, by the non-decreasing 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. Therefore a non-trivial 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 $S_{l,k}$ reasonable scenarios.

**Example 1** Let $k = 2, l = 4$ and $n = 13$. Consider the scenarios

$\xi_1 = (0, 60, 14, 60, 6, 60, 14, 12, 60, 2, 60, 1, 60)^\top$

$\xi_2 = (28, 30, 12, 60, 21, 20, 60, 16, 24, 25, 0, 60, -2)^\top$

$\xi_3 = (60, 16, 18, 13, 60, 2, 60, 0, 60, 22, 20, 60)^\top$

$\xi_4 = (60, 14, 60, 10, 60, 8, 60, 10, 60, 9, 60, 9, 60)^\top$

and let $X$ be the set of all unit vectors $e_1, \ldots, e_{13} \in \mathbb{Q}^{13}$. 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:

<table>
<thead>
<tr>
<th>Iteration</th>
<th>Induced/Start Partition</th>
<th>Induced solution set</th>
<th>Obj. value</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>${\xi_1}, {\xi_2, \xi_3, \xi_4}$</td>
<td>$e_1, e_2$</td>
<td>60</td>
</tr>
<tr>
<td>1</td>
<td>${\xi_1, \xi_2}, {\xi_3, \xi_4}$</td>
<td>$e_3, e_4$</td>
<td>54</td>
</tr>
<tr>
<td>2</td>
<td>${\xi_1, \xi_2, \xi_3}, {\xi_4}$</td>
<td>$e_5, e_6$</td>
<td>48</td>
</tr>
<tr>
<td>3</td>
<td>${\xi_1, \xi_2}, {\xi_3, \xi_4}$</td>
<td>$e_7, e_8$</td>
<td>42</td>
</tr>
<tr>
<td>4</td>
<td>${\xi_3}, {\xi_1, \xi_2, \xi_4}$</td>
<td>$e_9, e_{10}$</td>
<td>36</td>
</tr>
<tr>
<td>5</td>
<td>${\xi_2, \xi_3}, {\xi_1, \xi_4}$</td>
<td>$e_{11}, e_{12}$</td>
<td>32</td>
</tr>
<tr>
<td>6</td>
<td>${\xi_2}, {\xi_1, \xi_3, \xi_4}$</td>
<td>$e_{13}, e_{12}$</td>
<td>28</td>
</tr>
<tr>
<td>7</td>
<td>${\xi_2}, {\xi_1, \xi_3, \xi_4}$</td>
<td>$e_{13}, e_{12}$</td>
<td>28</td>
</tr>
</tbody>
</table>

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.

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. In that case, it is preferable to compute the start solutions deterministically instead of randomly, using any algorithm that ensures that the same start partition is not chosen more than once.

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. 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.

**Improved Heuristic**

1. Fix a parameter $z \in \mathbb{N}$.
2. Choose an arbitrary partition $P$ of $\{1, \ldots, l\}$.
3. Run the basic heuristic with start partition $P$. Denote the result by $X_1$.
4. repeat
5. Set $X_2 := X_1$.
6. For every scenario, save the best and the second best solution from $X_2$, and sort the scenarios in ascending order according to the differences between the two corresponding objectives.
7. Set $i := z$.
8. while $i > 0$ do
9.   Set $i := i - 1$.
10. Let $\xi_i$ be the $i$-th element of the sorted list.
11. Change $P$ by removing $\xi_i$ from the subset covered by its best solution and adding it to the subset covered by its second best solution.
12. end while
13. Run the basic heuristic with start partition $P$. Denote the result by $X_1$.
14. Compute the partition $P$ induced by $X_1$.
15. until Solution value of $X_2$ is not worse than the solution value of $X_1$.
16. return $X_2$

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 section, we use this heuristic with all presented improvements and $z = 5$, because this turned out to yield the best results in our tests.

### 6 Experimental Evaluation

In the following, we present an extensive experimental evaluation of the exact and heuristic methods devised in the previous sections. For this, we address three different underlying problems: Unconstrained Binary Optimization, the Shortest Path Problem and the Spanning Tree Problem. For each of these problem classes, we create settings with different combinations of $k$, $l$ and $n$; for every combination, we create 10 random instances. Depending on the problem, we used different subsets of the following seven algorithms:
- **Enumeration**: enumeration of the set of solutions, see Section 4.1;
- **Partition**: enumeration of the partitions, see Section 4.2;
- **CPLEX**: direct application of CPLEX 12.6 [10] to the formulation (IQP), see Section 4.3;
- **CPLEX Plus**: application of CPLEX 12.6 to the IQP formulation with reduced symmetry, using the improved heuristic presented in Section 5 for obtaining an initial upper bound;
- **Underestimator**: the branch-and-bound code from [8] using separable underestimators, see Section 4.4;
- **Heuristic100**: the improved heuristic with 100 different start partitions, see Section 5; and
- **Heuristic10000**: the same improved heuristic with 10000 start partitions.

For all exact algorithms, we used an absolute optimality tolerance of $10^{-6}$. Except for this, standard parameters were used. In the following tables, for every solution method considered, we present the average total cpu time in seconds ($\text{time}$) for all solved instances and the average value of the best solution obtained by this method divided by the best solution obtained by any of the methods on this instance ($\text{value}$). Finally, the column $\text{solved}$ contains two numbers: the number of instances solved to optimality within the time and memory limits, out of the 10 instances considered, and the number of instances for which a feasible solution could be found. The latter limits were set to three cpu hours and 64 GB. For all tests, we used Intel Xeon processors running at 2.60 GHz.

### 6.1 Unconstrained Binary Optimization Problem

We first present results for the Unconstrained Binary Optimization Problem, where $X = \{0, 1\}^n$. We used all presented algorithms except for **Enumeration**, which in this case was far too slow to be competitive. We generated all instances randomly, choosing the entries of $\xi$ to be normally distributed with expected value of zero and variance of one.

The results are summarized in Table 1. It turns out that the heuristic algorithms provide optimal or near-optimal solutions in all settings, while requiring a very short running time. **Heuristic100** always uses less than 0.1 cpu seconds and is never more than 3% away from the best found solution. **Heuristic10000** needs on average at most 11.3 cpu seconds and is never more than 2% away from the best found solution. One reason for the fast running times of the heuristics is the fact that the oracle for the Unconstrained Binary Problem is particularly fast.

Concerning the exact algorithms, **Partition** gives the best results in most cases. For $k = 2$ and $l \leq 20$, it is the fastest algorithm. The running time of **Partition** does not increase significantly in $n$, compared to the other exact solution methods, and therefore especially for $n = 100$ it clearly outperforms the other exact methods. On the other hand, its running time increases significantly in terms of $l$. Comparing **CPLEX** with **CPLEX Plus**, we can see that
the latter is in general faster than the former. In one setting, it only requires 31% of the running time of the former. In some settings, however, CPLEX is faster or CPLEX Plus runs into memory problems. The behaviour of CPLEX and CPLEX Plus is not always easily comparable, as CPLEX contains many implementation tricks that can make its behaviour rather unpredictable. Finally, Underestimator is generally slower than CPLEX and CPLEX Plus for the Unconstrained Binary Problem. However, it never runs into memory problems. Whenever no feasible solution is found by Underestimator, this is because the time needed for the preprocessing exceeds the total time limit.

### 6.2 Shortest Path Problem

We now focus on the case that the underlying problem is a Shortest Path Problem. We used all presented algorithms except for Heuristic10000, as it turned out that the heuristic is slower but more precise for this application, so that using Heuristic100 was enough to obtain high quality solutions.

<table>
<thead>
<tr>
<th>( n \times m )</th>
<th>( k )</th>
<th>( \text{Time} )</th>
<th>( \text{value} )</th>
<th>( \text{Time} )</th>
<th>( \text{value} )</th>
<th>( \text{Time} )</th>
<th>( \text{value} )</th>
<th>( \text{Time} )</th>
<th>( \text{value} )</th>
<th>( \text{Time} )</th>
<th>( \text{value} )</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>5</td>
<td>0.01</td>
<td>0.00</td>
<td>10/10</td>
<td>0.01</td>
<td>0.00</td>
<td>10/10</td>
<td>0.01</td>
<td>0.00</td>
<td>10/10</td>
<td>0.01</td>
</tr>
<tr>
<td>5</td>
<td>60</td>
<td>0.00</td>
<td>1.00</td>
<td>10/10</td>
<td>1.00</td>
<td>1.00</td>
<td>10/10</td>
<td>1.00</td>
<td>1.00</td>
<td>10/10</td>
<td>1.00</td>
</tr>
<tr>
<td>3</td>
<td>10</td>
<td>0.00</td>
<td>1.00</td>
<td>10/10</td>
<td>0.00</td>
<td>1.00</td>
<td>10/10</td>
<td>0.00</td>
<td>1.00</td>
<td>10/10</td>
<td>0.00</td>
</tr>
<tr>
<td>5</td>
<td>60</td>
<td>0.00</td>
<td>1.00</td>
<td>10/10</td>
<td>1.00</td>
<td>1.00</td>
<td>10/10</td>
<td>1.00</td>
<td>1.00</td>
<td>10/10</td>
<td>1.00</td>
</tr>
<tr>
<td>3</td>
<td>10</td>
<td>0.00</td>
<td>1.00</td>
<td>10/10</td>
<td>0.00</td>
<td>1.00</td>
<td>10/10</td>
<td>0.00</td>
<td>1.00</td>
<td>10/10</td>
<td>0.00</td>
</tr>
<tr>
<td>5</td>
<td>60</td>
<td>0.00</td>
<td>1.00</td>
<td>10/10</td>
<td>1.00</td>
<td>1.00</td>
<td>10/10</td>
<td>1.00</td>
<td>1.00</td>
<td>10/10</td>
<td>1.00</td>
</tr>
<tr>
<td>3</td>
<td>10</td>
<td>0.00</td>
<td>1.00</td>
<td>10/10</td>
<td>0.00</td>
<td>1.00</td>
<td>10/10</td>
<td>0.00</td>
<td>1.00</td>
<td>10/10</td>
<td>0.00</td>
</tr>
<tr>
<td>5</td>
<td>60</td>
<td>0.00</td>
<td>1.00</td>
<td>10/10</td>
<td>1.00</td>
<td>1.00</td>
<td>10/10</td>
<td>1.00</td>
<td>1.00</td>
<td>10/10</td>
<td>1.00</td>
</tr>
<tr>
<td>3</td>
<td>10</td>
<td>0.00</td>
<td>1.00</td>
<td>10/10</td>
<td>0.00</td>
<td>1.00</td>
<td>10/10</td>
<td>0.00</td>
<td>1.00</td>
<td>10/10</td>
<td>0.00</td>
</tr>
<tr>
<td>5</td>
<td>60</td>
<td>0.00</td>
<td>1.00</td>
<td>10/10</td>
<td>1.00</td>
<td>1.00</td>
<td>10/10</td>
<td>1.00</td>
<td>1.00</td>
<td>10/10</td>
<td>1.00</td>
</tr>
<tr>
<td>3</td>
<td>10</td>
<td>0.00</td>
<td>1.00</td>
<td>10/10</td>
<td>0.00</td>
<td>1.00</td>
<td>10/10</td>
<td>0.00</td>
<td>1.00</td>
<td>10/10</td>
<td>0.00</td>
</tr>
<tr>
<td>5</td>
<td>60</td>
<td>0.00</td>
<td>1.00</td>
<td>10/10</td>
<td>1.00</td>
<td>1.00</td>
<td>10/10</td>
<td>1.00</td>
<td>1.00</td>
<td>10/10</td>
<td>1.00</td>
</tr>
</tbody>
</table>

Table 1: Results for \( X = \{0, 1\}^n \)
Table 2: Results for the Shortest Path Problem

Enumeration is only used for \( k = 2 \); in all other cases the algorithm reached the time limit in our preliminary tests.

For every instance, we generated a directed quadratic grid graph. The weights of the edges were chosen normally distributed with expected value of one and variance of 0.2, this was enough to avoid any negative values in our instances. Note that the number of variables \( n \) agrees with the number of edges of the graph here. For all approaches except for Underestimator, we used Dijkstra’s algorithm for the oracle. Since for Underestimator also negative edge weights can arise in the oracle calls, we used CPLEX instead for realizing the oracle.

In our tests presented in Table 2, Enumeration is by far the best algorithm for most settings with \( n \leq 112, l \geq 10 \) and \( k = 2 \). In these settings, it is even faster than the heuristic. The reason of its success is the fact that a directed grid graph has a relatively small number of feasible solutions and that these solutions can be enumerated very efficiently. In addition, the value of \( l \) only influences the evaluation of a solution and therefore has a very small influence on the total running time, compared with other methods. Concerning Partition, we can see that the number \( n \) of variables has a larger influence
on the running time than in the unconstrained case, because the time spent for calling the oracle now has a larger share in the total running time. As a result, Partition is in no case strictly better than all other exact methods, and therefore not recommendable for the Shortest Path Problem.

Similar to the unconstrained case, we observe that CPLEX Plus in general outperforms CPLEX. This becomes particularly clear for instances with a larger \( k \), since in these settings the percentage of eliminated variables is higher. Finally, Underestimator is slower than CPLEX (Plus) on easy instances but better in general for harder instances. In particular, it can handle an increase of \( k \) or \( l \) better than CPLEX (Plus). A reason for the bad performance on small instances is that the preprocessing is quite time-consuming, which however pays off for harder instances.

For the Shortest Path Problem, Heuristic100 is clearly slower than in the unconstrained case, due to the fact that the oracle requires much more running time. However, it also produces better solutions, so that in no setting any other solution method could find better solutions within the time and memory limits.

### 6.3 Spanning Tree Problem

We finally investigate the Spanning Tree Problem as underlying problem. We did not consider Enumeration as well as Heuristic10000, for the same reasons as mentioned before. In addition, we did not use CPLEX or CPLEX Plus, since the natural IP formulation for the Spanning Tree Problem has an exponential number of constraints. So either the model is too large or we need a separation algorithm, which however slows down CPLEX significantly.

For our instances, we first computed coordinates for the vertices uniformly at random in the square \([100, 100]\). In every scenario, we added a random value to the coordinates that was chosen normally distributed with expected value of zero and variance of two. The cost vector \( \xi_i \) was then calculated by using the Euclidian distances between all pairs of points. The oracle is given by a simple implementation of Kruskal’s algorithm.

Table 3 contains our results. We observe that Partition is always faster than Underestimator for \( l \leq 10 \) and even faster than Heuristic100 on small instances. There is no setting in which another solution method calculated a better solution even when Partition could not finish due to the time limit. For harder instances, Underestimator is the most successful exact solution method. In every setting, Heuristic100 gives a solution value that is not worse than the value of the other methods. Altogether, we can observe that an increase of \( n \) has a higher influence on the running time of Heuristic100 and Partition than in the unconstrained case. Surprisingly, the influence of increasing \( n \) on the running time of Underestimator is similar to the unconstrained case, although all these methods use a more time consuming oracle than in the unconstrained case.
We have shown that the Discrete Min-E-Min Problem is very hard in terms of complexity, approximation and parameterized complexity. Nevertheless, we presented various solution methods for solving this problem, both exactly and heuristically. In our computational experiments, we could observe that all of these methods may outperform the others depending on the number of solutions required, the number of scenarios, the dimension of the instance, and the type of the underlying problem. The heuristic approach is a good alternative if one is only interested in near-optimal solutions, which it consistently produces in a small amount of time.

Acknowledgements This work has been supported by the German Research Foundation within the Research Training Group 1855.

Table 3: Results for the Spanning Tree Problem

7 Conclusion
References