Generating Hard Instances for Robust Combinatorial Optimization

Marc Goerigk*[1] and Stephen J. Maher[2]

[1]Network and Data Science Management, University of Siegen, Germany
[2]Department of Management Science, Lancaster University, United Kingdom

Abstract

While research in robust optimization has attracted considerable interest over the last decades, its algorithmic development has been hindered by several factors. One of them is a missing set of benchmark instances that make algorithm performance better comparable, and makes reproducing instances unnecessary. Such a benchmark set should contain hard instances in particular, but so far, the standard approach to produce instances has been to sample values randomly from a uniform distribution.

In this paper we introduce a new method to produce hard instances for min-max combinatorial optimization problems, which is based on an optimization model itself. Our approach does not make any assumptions on the problem structure and can thus be applied to any combinatorial problem. Using the Selection and Traveling Salesman problems as examples, we show that it is possible to produce instances which are up to 500 times harder to solve for a mixed-integer programming solver than the current state-of-the-art instances.

Keywords: robustness and sensitivity analysis; robust optimization; problem benchmarking; problem generation; combinatorial optimization

1. Introduction

We consider (nominal) combinatorial optimization problems of the form

$$\min_{\boldsymbol{x}\in\mathcal{X}} \boldsymbol{c}\boldsymbol{x}$$

where $\mathcal{X} \subseteq \{0,1\}^n$ denotes the set of feasible solutions, and $\boldsymbol{c} \in \mathbb{R}^n_+$ is a cost vector. For the case that the cost coefficients $\boldsymbol{c}$ are not known exactly, robust optimization approaches have been developed. In the most basic form, we assume a discrete set $\mathcal{U} = \{\boldsymbol{c}^1, \dots, \boldsymbol{c}^N\}$ of possible costs to be given, the so-called uncertainty set. Depending on the problem application, $\mathcal{U}$ may be found by sampling from a distribution, or by using past observations of data. The robust (min-max) problem is then to solve

$$\min_{\boldsymbol{x}\in\mathcal{X}} \max_{\boldsymbol{c}\in\mathcal{U}} \boldsymbol{c}\boldsymbol{x}$$

This type of problem was first introduced in [KY97], and several surveys are now available, see [ABV09,GS16,KZ16]. The robust problem turns out to be NP-hard for all relevant problems that have been considered so far, even for $N = 2$. This is also the case if the nominal problem

*Corresponding author. Email: marc.goerigk@uni-siegen.de is solvable in polynomial time, for example the Shortest Path or the Assignment problem [KZ16].

However, practical experience tells us that an NP-hard problem can sometimes still be solved sufficiently fast for relevant problem sizes. In fact, where NP-hardness proofs typically rely on constructing problem instances with specific properties, nothing is known about hardness of randomly generated instances, or smoothed analysis, in robust optimization. Where the related min-max regret problem has sparked research into specialized solution algorithms (see, e.g., [CLSN11, PA11, KMZ12]), little such research exists for the min-max problem, as simply using an off-the-shelf mixed-integer programming solver, such as CPLEX, can already lead to satisfactory results.

Faced with a similar situation for nominal knapsack problems, [Pis05] asked: "Where are the hard knapsack problems?" The related aim of this paper is to construct computationally challenging robust optimization problems. To this end, we consider the Selection problem, where $\mathcal{X} = \{\boldsymbol{x} \in \{0,1\}^n : \sum_{k=1}^n x_k = p\}$, and the Traveling Salesman problem (TSP) as examples. The nominal problem of the former can be solved in polynomial time, while it is NP-hard for the latter. However, the proposed methods are general and can be applied to any robust combinatorial problem.

Looking into other fields of optimization problems, instance libraries have been a main driver of algorithm development [MHS10]. Examples include MIPLIB [KAA+11] for mixed-integer programs, road networks from the DIMACS challenge for Shortest Path problems [DGJ09] or the Solomon instances for the vehicle routing problem with time windows [Sol87]. There is a clear gap in robust optimization, where instance generators often need to be re-implemented to reproduce previous results. Our research is intended as a first step towards a library of hard instances to guide future research. Both our benchmark set of instances and the code to generate them are published on a website dedicated to this purpose, `www.robust-optimization.com`.

As there is no free lunch in optimization, we cannot hope to construct instances that are hard for all possible optimization algorithms. We therefore avoid constructing instances that are hard for a particular solution method (e.g., using CPLEX), but rather aim at maximizing hypothetical measures of hardness. Whether or not they actually correspond to harder instances for the solver is then a matter of computational experiments.

Our focus is to find an uncertainty set such that the optimal objective value of the resulting robust problem is as large as possible (Section 2). To solve the resulting optimization problem, Section 3 considers several exact and heuristic solution methods. We briefly discuss our software package for instance generation in Section 4, before comparing solution approaches and the hardness of the resulting instances in Section 5. We find that it is possible to construct instances that are considerably harder to solve than i.i.d. uniformly sampled problems—the current standard approach. Section 6 concludes the paper and points out further avenues for research.

2. An Optimization Model for Maximizing the Robust Objective Value

This paper proposes the use of an optimization problem to construct hard problem instances. Throughout this section the proposed model is presented along with a number of different solution techniques. In the presentation of the model and related discussions, the vectors and matrices are written in bold font, for example $\boldsymbol{x} = (x_1, \dots, x_n)$, and for sets $\{1, \dots, n\}$ the shorthand notation $[n]$ is used.

Let some problem instance with N scenarios be given, represented through the scenario objective coefficient vectors $\tilde{\boldsymbol{c}}^1, \dots, \tilde{\boldsymbol{c}}^N$, with $\tilde{\boldsymbol{c}}^i \in \mathbb{R}^n_+$. From this initial instance, the goal is to modify the inputs in such a way that the resulting robust problem is harder to solve. The approach that is used in this paper is to modify the values of the cost vectors in each of the scenarios. However, the base problem is to be modified, and not completely changed, so a limit on the magnitude of the change for each cost value is imposed.

Consider a scenario $i \in [N]$, which is a vector of cost coefficients denoted by $\tilde{\boldsymbol{c}}^i$. The modification of the problem involves the selection of cost coefficients from the set of all possible candidate values, which is denoted by $\mathcal{U}_i$. In the approach proposed in this paper, the set $\mathcal{U}_i$ is defined as

$$\mathcal{U}_i = \left\{ \boldsymbol{c} \in \mathbb{R}^n_+ : c_k \in [\underline{c}^i_k, \overline{c}^i_k] \ \forall k \in [n], \sum_{k\in[n]} c_k = \sum_{k\in[n]} \tilde{c}^i_k \right\}$$

where $\underline{c}^i_k$ and $\overline{c}^i_k$ denote the lower and upper bounds, respectively, on the cost coefficient k. Additionally, $\mathcal{U}_i$ imposes the constraint that the sum of coefficients for this scenario remains the same, but any feasible sum that respects the upper and lower bounds is permitted as a scenario vector. We will use $\underline{c}^i_k = \max\{\tilde{c}^i_k - b, 0\}$ and $\overline{c}^i_k = \min\{\tilde{c}^i_k + b, C\}$ with a budget parameter b and a global maximum cost coefficient C.

Our approach aims at finding scenarios $\boldsymbol{c}^i \in \mathcal{U}_i$ for all $i \in [N]$, so that the objective value of the optimal solution to the corresponding robust optimization problem is increased. This approach can be formulated as the following optimization problem

$$\max_{\boldsymbol{c}^i \in \mathcal{U}_i \forall i \in [N]} \min_{\boldsymbol{x}\in\mathcal{X}} \max_{j\in[N]} \boldsymbol{c}^j \boldsymbol{x} \tag{MRO}$$

where MRO stands for "maximize robust objective". The intuition behind the proposed optimization problem for generating difficult robust problem instances is the following: For each $\boldsymbol{x} \in \mathcal{X}$, the objective $\max_{j\in[N]} \boldsymbol{c}^j \boldsymbol{x}$ is a piecewise linear, convex function in $\boldsymbol{c}^1, \dots, \boldsymbol{c}^N$. By maximizing the smallest value of the objective over all $\boldsymbol{x}$, we spread out the solution costs, balancing the objective values of the best solutions in $\mathcal{X}$. This way, finding and proving optimality of the best $\boldsymbol{x}$ becomes a more difficult task for an optimization algorithm. Naturally, whether the instances produced using the proposed method are actually more difficult to solve than the original problem $\tilde{\boldsymbol{c}}^1, \dots, \tilde{\boldsymbol{c}}^N$ can only be tested computationally.

As an example, consider a robust variant of the Selection problem where the task is to choose two out of four items such that the maximum costs over two scenarios are as small as possible. The cost vectors for these two scenarios are

	item			
	1	2	3	4
$\boldsymbol{c}^1$	4	1	9	2
$\boldsymbol{c}^2$	4	7	4	4

In this small example there are $\binom{4}{2} = 6$ possible solutions. For this particular instance of the robust selection problem there is only one optimal solution to this problem, which is to choose items 1 and 4 with a robust objective value 8. The sorted vector of the corresponding six robust objective values is

$$(8, 11, 11, 11, 11, 13)$$

Now let us assume that $\boldsymbol{c}^1 \in \mathcal{U}_1$ and $\boldsymbol{c}^2 \in \mathcal{U}_2$ and the budget is given by $b = 1$. Thus, two alternative cost vectors $\hat{\boldsymbol{c}}^1 \in \mathcal{U}_1$ and $\hat{\boldsymbol{c}}^2 \in \mathcal{U}_2$ are

	item			
	1	2	3	4
$\hat{\boldsymbol{c}}^1$	3	2	10	1
$\hat{\boldsymbol{c}}^2$	5	6	3	5

Given these cost vectors, the objective value of the optimal robust solution increases to 10. The optimal solution still remains as the selection of 1 and 4, but the sorted vector of robust objective values has become

$$(10, 11, 11, 11, 12, 13)$$

An important observation is that the difference between the best and second-best solutions has reduced. This can have the effect of increasing the difficulty of proving optimality. As mentioned previously, the difficulty of the instance can only be evaluated computationally. Using CPLEX to solve the min-max robust selection problem given by this small example, the first instance takes 0.013 ticks of the deterministic clock, whereas the second instances is solved in 0.209 deterministic ticks—more than 16 times as long.

3. Solution Approaches

A clear drawback of MRO is that the inner problem is the robust optimization problem that we are attempting to make hard. Therefore, constructing a hard problem is at least as hard as actually solving it. Due to this fact we primarily focus on producing hard, but relatively small instances. This is an alternative to the trivial approach to producing hard instances, which is to produce larger ones. In the last part of this section we shift our focus to the generation of large and hard robust optimization instances. This is achieved through the use of heuristic methods to solve the MRO.

Note that even evaluating the objective value of some fixed scenario variables $\boldsymbol{c}^1, \dots, \boldsymbol{c}^N$ is NP-hard for all commonly considered combinatorial problems (see [KZ16]), as they are equivalent to solving a robust counterpart. Indeed, in Appendix A we show that MRO is Σ_2^p-complete when scenarios can be chose from polyhedral sets.

In the outer maximization problem, we determine N vectors, and choose one of these vectors in the inner maximization problem. Formally, this is similar to the K-adaptability approach in robust optimization (see [BK17]), which uses a min-max-min structure. Whereas their combinatorial part is in the outer minimization, the combinatorial part is in the inner minimization in our problem.

To address the difficulty of MRO, different solution approaches are developed. Each of the solution approaches aim to reduce the difficulty of solving MRO through alternative techniques. These approaches are:

- Iterative method (Section 3.1): an exact approach that exploits the multi-level structure of MRO.
- Alternating heuristic (Section 3.2): a heuristic applied to a reformulation of MRO.
- Replacing the inner subproblem with a heuristic (Section 3.3): a method for larger MRO problems.

A description of each of the solution approaches is presented in the following sections. The experimental results in Section 5 then demonstrate the value of each approach in a comparison with two different methods: column generation and linear decision rules, both explained in Appendices B and C.

3.1. Iterative Solution

Given the multi-level structure, it is difficult to solve MRO directly using general purpose solvers. However, decomposition techniques can be used to exploit this structure and to develop an effective solution approach.

Note that we can write the inner maximization problem for given $\boldsymbol{c}^i$, $i \in [N]$, and $\boldsymbol{x} \in \mathcal{X}$ by introducing a variable vector $\boldsymbol{\lambda}$ representing the choice of scenario:

$$\max_{j\in[N]} \boldsymbol{c}^j\boldsymbol{x} = \max \left\{ \sum_{i\in[N]} \lambda_i \boldsymbol{c}^i\boldsymbol{x} : \sum_{i\in[N]} \lambda_i = 1,\ \lambda_i \in \{0,1\}\ \forall i \in [N] \right\}$$

Let us now assume that some set $\{\boldsymbol{x}^1, \dots, \boldsymbol{x}^K\} \subseteq \mathcal{X}$ of candidate solutions are already known. Then, the restricted MRO problem on this set can be written as

$$\begin{aligned} \max\ & t \\ \text{s.t. } & t \le (\sum_{i\in[N]} \lambda_i^j \boldsymbol{c}^i)\boldsymbol{x}^j && \forall j \in [K] \\ & \sum_{i\in[N]} \lambda_i^j = 1 && \forall j \in [K] \\ & \boldsymbol{c}^i \in \mathcal{U}_i && \forall i \in [N] \\ & \lambda_i^j \in \{0,1\} && \forall i \in [N], j \in [K] \end{aligned} \tag{1}$$

where the variables $\boldsymbol{\lambda}^j$ for each $j \in [K]$ are used to determine the scenario $\boldsymbol{c}$ that is assigned to each candidate $\boldsymbol{x}^j$. We refer to this problem also as the master problem.

Note that problem (1) is nonlinear through the product of $\boldsymbol{\lambda}$ and $\boldsymbol{c}$ variables, which can be linearized using variables $d_{ijk} = \lambda^j_i c^i_k$. The resulting model is then given as

$$
\begin{aligned}
\max\ & t \\
\text{s.t. } & t \le \sum_{i\in[N]} \sum_{k\in[n]} d_{ijk} x^j_k && \forall j \in [K] \\
& \sum_{i\in[N]} \lambda^j_i = 1 && \forall j \in [K] \\
& d_{ijk} \le c^i_k && \forall i \in [N], j \in [K], k \in [n] \\
& d_{ijk} \le \overline{c}^i_k \lambda^j_i && \forall i \in [N], j \in [K], k \in [n] \\
& \boldsymbol{c}^i \in \mathcal{U}_i && \forall i \in [N] \\
& \lambda^j_i \in \{0,1\} && \forall i \in [N], j \in [K]
\end{aligned}
\tag{2}
$$

Once the master problem is solved for a fixed set of candidate solutions, we have determined an upper bound on the MRO problem. By solving the resulting robust optimization problem for $\boldsymbol{x}$, we also construct a lower bound. If both bounds are not equal, we add the current robust solution $\boldsymbol{x}$ to the set of candidate solutions and repeat the process by solving the master problem. This iterative approach will converge after a finite number of steps, as $\mathcal{X}$ contains a finite number of solutions. It is therefore an exact solution approach to MRO.

An interesting question is whether the master problem is solvable in polynomial time. Note that for N scenarios and K solutions, there are N^K possibilities to assign solutions to scenarios. For each assignment, constructing optimal scenarios $\boldsymbol{c}$ can be done in polynomial time by solving a linear program. This means that if K is constant, the master problem can be solved in polynomial time as well.

If K is unbounded, however, the problem becomes hard, as the following theorem shows.

Theorem 1. *The master problem is NP-hard, if K is part of the input.*

Proof. We use a reduction from Hitting Set, see [GJ79]: Given a ground set $[E]$, a collection of sets $S_1, \ldots, S_T \subseteq [E]$, and some integer $L \le E$. Is there a subset $C \subseteq [E]$ with $|C| \le L$ such that $|C| \cap S_i \neq \emptyset$ for all $i \in [T]$?

Let an instance of Hitting Set be given. We set $n = E$, $N = L$ and $K = T$. We further set $b = C = 1$, and $\tilde{c}_k = 1/n$ for each $k \in [n]$ (i.e., we get $\underline{c}_k = 0$ and $\overline{c}_k = 1$ for all $k \in [n]$). Finally, we set $x^i_k = 1$ if $k \in S_i$ and $x^i_k = 0$ otherwise.

We now claim that Hitting Set is a yes-instance if and only if there is a solution to MRO with objective value at least 1.

To prove this claim, let us first assume Hitting Set is a yes-instance. Let $C = \{e_1, \ldots, e_L\}$ be a corresponding subset of $[E]$ (w.l.o.g. we assume that $|C| = L$). Then we build a solution to MRO in the following way. For each $e_\ell \in C$, set $c^\ell_{e_\ell} = 1$ and $c^\ell_k = 0$ for all $k \neq e_\ell$. For each S_i, choose one $e_\ell \in S_i \cap C$ and set $\lambda^i_\ell = 1$ and all other $\lambda^i_k = 0$. Thus we obtain a feasible solution to MRO with objective value at least 1.

We illustrate this process with a small example. Let $E = \{1, \ldots, 7\}$, $S_1 = \{1,2,3\}$, $S_2 = \{3,4,5\}$, $S_3 = \{6,7\}$, and $L = 2$. Our MRO instance has the following values of $\boldsymbol{x}^1$,$\boldsymbol{x}^2$,$\boldsymbol{x}^3$ and $\tilde{\boldsymbol{c}}^1$ and $\tilde{\boldsymbol{c}}^2$:

$\boldsymbol{x}^1$	1	1	1	0	0	0	0
$\boldsymbol{x}^2$	0	0	1	1	1	0	0
$\boldsymbol{x}^3$	0	0	0	0	0	1	1
$\tilde{\boldsymbol{c}}^1$	1/7	1/7	1/7	1/7	1/7	1/7	1/7
$\tilde{\boldsymbol{c}}^2$	1/7	1/7	1/7	1/7	1/7	1/7	1/7
$\boldsymbol{c}^1$	0	0	1	0	0	0	0
$\boldsymbol{c}^2$	0	0	0	0	0	1	0

In the same table, we also show an optimal solution for $\boldsymbol{c}^1$ and $\boldsymbol{c}^2$. The $\boldsymbol{\lambda}$ variables are chosen such that $\boldsymbol{x}^1$ and $\boldsymbol{x}^2$ are assigned to $\boldsymbol{c}^1$, and $\boldsymbol{x}^3$ is assigned to $\boldsymbol{c}^2$.

Now let us assume that for some Hitting Set instance, we construct our MRO problem as detailed above and find an objective value of at least 1. We show that Hitting Set is a yes-instance. To this end, we first show that there exists an optimal solution to MRO where all $\boldsymbol{c}^i_k$-variables are either 0 or 1. Consider any $\boldsymbol{c}^i$, and let $\boldsymbol{x}^{i_1}, \dots, \boldsymbol{x}^{i_p}$ be all $\boldsymbol{x}$-solutions assigned to scenario i. We distinguish two cases:

1. There exists some $s \in [n]$ such that $x^{i_j}_s = 1$ for all $j \in [p]$. In this case, we can set $c^i_s = 1$.
2. There is no such $s \in [n]$, i.e., there are x^{i_k} and x^{i_ℓ} with $k, \ell \in [p]$ that choose disjoint sets of items. As $\sum_{k \in [n]} c^i_k = 1$, at least one of them must have an objective value strictly less than 1, which contradicts our assumptions.

We can thus set C by including all elements $k \in [n]$ for which there is $i \in [N]$ with $c^i_k = 1$. By construction, C is a hitting set with cardinality at most L.

□

While the iterative algorithm is an exact solution approach, there are limitations to its use. Specifically, solving the master problem can become a bottleneck to the solution process as the number of solutions K increases. In each iteration of the algorithm, the addition of a new candidate $\boldsymbol{x}$ results in an additional $2 + 2Nn$ constraints. Computationally, the additional constraints have a significant negative impact between consecutive iterations. Two different solution methods will be presented to address the issue in solving the master problem. We descibe an alternating heuristic in Section 3.2, and discuss a Dantzig-Wolfe decomposition approach in Appendix B.

3.2. Alternating Heuristic

As an alternative to the relaxation and decomposition approach presented in Section B, an alternating heuristic has been developed to solve the master problem (2) of the iterative approach. The alternating heuristic is motivated by the observation that for a given assignment of scenarios $i \in [N]$ to solutions $j \in [K]$, selecting the cost coefficients to maximize the minimum objective becomes a simple task. Similarly, for a fixed set of cost coefficients for each scenario, the difficulty in assigning scenarios to solutions is greatly reduced. As such, the alternating heuristic iterates between fixing either the scenario assignment or the scenario cost coefficients.

To formally present the alternating heuristic, first reconsider the master problem from the iterative method

$$\begin{aligned} \max\ & t \\ \text{s.t. }\ & t \le \Big(\sum_{i \in [N]} \lambda^j_i \boldsymbol{c}^i\Big)\boldsymbol{x}^j && \forall j \in [K] \\ & \sum_{i \in [N]} \lambda^j_i = 1 && \forall j \in [K] \\ & \boldsymbol{c}^i \in \mathcal{U}_i && \forall i \in [N] \\ & \lambda^j_i \in \{0,1\} && \forall i \in [N], j \in [K] \end{aligned}$$

for a subset $\{\boldsymbol{x}^1, \dots, \boldsymbol{x}^K\} \subseteq \mathcal{X}$ of solutions. Let us assume the variables $\boldsymbol{c}^i$ are all fixed. In that case, an optimal solution to the remaining $\boldsymbol{\lambda}$ variables can be found through the following procedure: For each $j \in [K]$, choose one $i \in [N]$ such that $\boldsymbol{c}^i\boldsymbol{x}^j$ is not smaller than $\boldsymbol{c}^\ell\boldsymbol{x}^j$ for all $\ell \neq i$. Then, set $\lambda^j_i = 1$ and all other $\lambda^j_\ell = 0$. To determine which $\boldsymbol{c}^i$ is a maximizer of the objective value for some $\boldsymbol{x}^j$, we can simply calculate all N possible objective values. Thus, finding optimal $\boldsymbol{\lambda}$ values is possible in $\mathcal{O}(nNK)$ time. Now let us assume that all $\boldsymbol{\lambda}$ variables are fixed. In this case, the remaining variables are continuous. Under the assumption that the $\mathcal{U}_i$ are polyhedra, the resulting problem can then be solved in polynomial time as well. This leads to the alternating heuristic described in Algorithm 1.

Algorithm 1 Alternating heuristic to solve (1)

```
Input: set of solutions $\boldsymbol{x}$, set of scenarios $\mathcal{U}_i$, an initial set of cost coefficients $\tilde{\boldsymbol{c}}$
Output: Cost coefficients $\hat{\boldsymbol{c}}_i$ for each scenario $i \in [N]$
1: Let $\hat{\boldsymbol{c}} \leftarrow \tilde{\boldsymbol{c}}$, $t \leftarrow -1$, $z_t \leftarrow 0$
2: repeat
3:     $t \leftarrow t + 1$
4:     fix the value of $\boldsymbol{c}$ to $\hat{\boldsymbol{c}}$ and solve (1) for $\boldsymbol{\lambda}$
5:     set $\hat{\boldsymbol{\lambda}} \leftarrow \boldsymbol{\lambda}$
6:     fix the value of $\boldsymbol{\lambda}$ to $\hat{\boldsymbol{\lambda}}$ and solve (1) for $\boldsymbol{c}$
7:     set $\hat{\boldsymbol{c}} \leftarrow \boldsymbol{c}$
8:     set $z_t$ to the current objective value of (1)
9: until $z_{t-1} \geq z_t$
```

3.3. Heuristics for Large Instances

A limitation of the previously presented approaches is that the exact solution of the resulting robust optimization subproblem is required. Therefore, it is not possible to find instances that take longer to solve than the generation time. This is especially true for the iterative methods, where it is likely that the robust optimization problem will be solved multiple times. While generating hard instances to small robust optimization problems is appropriate for most benchmarking purposes, this limitation prohibits the generation of hard large instances.

Fortunately, with only a small modification the MRO is still possible to generate hard instances to large robust optimization problems. Instead of solving the MRO

$$\max_{\boldsymbol{c}^i \in \mathcal{U}_i \forall i \in [N]} \min_{\boldsymbol{x} \in \mathcal{X}} \max_{j \in [N]} \boldsymbol{c}^j \boldsymbol{x}$$

exactly, we can use any heuristic to solve the robust problem $\min_{\boldsymbol{x} \in \mathcal{X}} \max_{j \in [N]} \boldsymbol{c}^j \boldsymbol{x}$. Given the nature of the MRO and the iterative solution approaches, any approximation of heuristic approach for finding a solution that is a proxy for the optimum—such as solving the linear relaxation of the robust problem and then rounding the solution—can be used. The solution to the robust optimization problem generated by the heuristic algorithm need not belong to $\mathcal{X}$.

Algorithmically, we can use the iterative method, the column generation approach, and the alternating heuristic in combination with the heuristic by just replacing any robust optimization subproblem.

4. Software

The approaches developed in Section 3 have been implemented within the Hiro (Hard Instances for Robust Optimization) C++ software library. This library has been designed to facilitate the generation of hard instances to any robust optimization problem.

The Hiro software library provides a virtual function, `solve_ip()`, within the `HIRO` class so that the user can specify the solution methods of an inner optimization problem when creating a problem specific derived class. In the examples presented in this paper, the inner optimization problems of Selection and TSP have been defined as integer programs that are solved using CPLEX. However, it is also possible to use combinatorial or heuristic algorithms to solve the inner problem. The parameters of `solve_ip()` are the number of elements in the cost vector(such as the number of items in Selection or the number of edges in the TSP), the number of scenarios and a two-dimensional vector containing the cost vectors for each scenario. The provided set of cost vectors define the current hard robust optimization instance. After solving the robust optimization problem, the user must create a `HIROSolution` object that is returned to the Hiro core. The `HIROSolution` object comprises a solution vector and the upper bound of that solution.

The Hiro software library is publicly available and can be found on GitHub[1] in the repository `stephenjmaher/HIRO`. The two optimization problems investigated in this paper, Selection and TSP, are included as examples in the software repository. Extensive instructions for adding new examples are also provided.

5. Experiments

The ultimate goal of this paper is to develop general purpose methods for generating hard instances for min-max robust optimization problems. To this end, we proposed the MRO model and different solution methods. While the theoretical basis of the proposed optimization problem is expected to produce min-max robust optimization instances harder than randomly generated instances, verifying this is only possibly through empirical studies.

In this paper, the hardness of an instance is primarily measured as the run time required to solve it using CPLEX. Due to the empirical nature of this work, it is not immediately obvious what the most effective method for generating hard instances is. Further, it is not possible to determine a priori which of the proposed algorithms will produce the most difficult instance. As such, it is necessary to evaluate the performance of the algorithms presented in Section 3 with respect to instance generation time and generated instance hardness.

5.1. Setup

The approaches presented in Section 3 are general methods that can be applied to the generation of hard instances for any min-max robust optimization problem. The alternative methods that have been developed focus specifically on the computation of cost coefficients for each scenario, which is the master problem in the proposed iterative methods. As such, the inner problem—the subproblem—can be set to any min-max robust optimization problem. To demonstrate the potential of the methods from Section 3, robust variants of the Selection problem and TSP are used as the inner problem. The Selection problem is used for its simplicity, meaning that the impact of the instance generation is more easily observed. We also consider the TSP to demonstrate the potential of using MRO to generate hard instances for a computationally more challenging optimization problem. We used the standard subtour elimination formulation for the TSP, with lazy generation of subtour elimination constraints.

The current state-of-the-art for robust optimization instance generation is to randomly sample scenarios. Thus, the baseline for comparison is a set of instances where the scenario coefficients are sampled randomly uniform i.i.d. with $c^i_k \in \{0, \dots, 100 = C\}$. This method of instance generation will be labeled as RU. In the following results, the proposed methods will be labeled as follows:

- MRO-Ex: The exact method from Section 3.1.
- MRO-Heu: The alternating heuristic from Section 3.2.
- MRO-LSHeu: The approach from Section 3.3, where we round the fractional solution found by solving the linear relaxation of the subproblem (in case of TSP without using subtour elimination constraints).
- MRO-CG: A column generation method that is applied to the relaxation of MRO as described in Appendix B.
- MRO-LDR: A linear decision rules approach explained in Appendix C.

We consider four different computational experiments: In the first, we use relatively small Selection problems, where the robust subproblem is solved to optimality (i.e., the methods from Sections 3.1 to C). Three problem sizes are used: The number of items n is set to 20, 30, and 40. In each case we set the number of items to be selected p to $n/2$ and the total number of scenarios N is set to n. For each problem size, we generate 100 instances using RU. The

[1] `https://github.com/`

scenarios from these random instances are then used as the initial scenarios for the iterative methods described above. To evaluate the effect of the uncertainty set budget b on the run times of the iterative methods and the hardness of the generated instances, budgets of 1, 2 and 5 are used. The total number of randomly generated instances is $3 \cdot 100 = 300$. Since these instances are used as an input to each of the iterative methods, a further $3 \cdot 3 \cdot 4 \cdot 100 = 3600$ *hard* instances are generated. For the second experiment, we considered larger problems with n ranging from 20 to 70, and a budget $b = 20$. In this experiment we only compare RU with MRO-LSHeu.

A maximum run time of 3600 seconds is given to each of the iterative methods. This run time limit is only enforced between the iterations of the algorithm, as such it is possible for this time limit to be exceeded.

Experiments three and four repeat experiments one and two for the TSP. In this setting, n is the number of edges in the complete graph, and hence $\sqrt{n}$ is the number of nodes. We first produce random instances with 100, 144, and 196 edges and in experiment three apply the MRO and associated solution approaches to generate hard instances for these smaller-scale problems. In experiment four, large problem instances are considered with the number of edges, n, ranging from 100 to 400. For both experiments three and four, the number of scenarios is set to $N = \sqrt{n}$.

The hardness of the instances is evaluated by using CPLEX to solve the resulting min-max robust optimization problem. All experiments were conducted using one core of a computer with an Intel Xeon E5-2670 processor, running at 2.60 GHz with 20MB cache, with Ubuntu 12.04 and CPLEX v12.6.

5.2. Hard Instance Generation for Selection

		$n = N$		
Budget	Method	20	30	40
1	MRO-Ex	0.04	0.61	7.83
	MRO-CG	0.04	0.45	6.79
	MRO-Heu	0.04	0.43	6.91
	MRO-LDR	0.03	0.17	2.20
2	MRO-Ex	0.06	0.92	7.49
	MRO-CG	0.05	0.62	9.34
	MRO-Heu	0.06	0.85	15.36
	MRO-LDR	0.03	0.25	3.07
5	MRO-Ex	0.08	0.77	10.59
	MRO-CG	0.06	0.78	11.92
	MRO-Heu	0.08	4.29	22.00
	MRO-LDR	0.04	0.46	8.21
	RU	0.03	0.13	1.43

Table 1: Selection: Average CPU time in seconds when solving the generated problems.

The ability of the MRO to produce robust optimization instances harder than randomly generated instances for Selection is clearly demonstrated in Table 1. Interestingly, as the budget b increases, the dominance of MRO-Heu emerges. In fact, with $b = 5$ and $n = 40$ MRO-Heu produces instances that are more than twice as hard, on average, as those produced by MRO-Ex.

The results from Table 1 are complemented by Table 2, which shows that average and maximum increase in hardness of all instances (i.e., the ratio between the computation time of the

Budget	Method	$n = N$ 20		30		40	
1	MRO-Ex	1.7	(3.7)	4.7	(9.9)	6.9	(15.5)
	MRO-CG	1.7	(3.4)	3.6	(8.1)	5.2	(10.6)
	MRO-Heu	1.7	(5.1)	3.5	(7.4)	5.3	(14.8)
	MRO-LDR	1.1	(2.3)	1.4	(2.4)	1.7	(5.8)
2	MRO-Ex	2.4	(4.2)	8.2	(23.9)	6.7	(40.8)
	MRO-CG	2.0	(5.0)	5.0	(13.6)	7.9	(46.3)
	MRO-Heu	2.2	(4.0)	6.5	(19.4)	12.8	(59.5)
	MRO-LDR	1.2	(2.5)	2.1	(6.8)	2.4	(8.2)
5	MRO-Ex	4.0	(11.3)	7.2	(23.0)	10.0	(49.3)
	MRO-CG	2.4	(5.0)	6.8	(21.7)	10.6	(53.0)
	MRO-Heu	3.6	(13.3)	37.6	(179.4)	23.6	(152.9)
	MRO-LDR	1.5	(3.3)	4.1	(12.2)	7.8	(49.4)

Table 2: Selection: Average (maximum) time increase relative to RU for each set of instances.

hardened and the original instance). We observe that for $n = 40$, method MRO-Heu produces instances that take on average 23 times longer to solve, and up to over 150 times in some cases.

Budget	Method	$n = N$ 20	30	40
1	MRO-Ex	1.7	207.0	3005.8
	MRO-CG	1.0	16.1	253.4
	MRO-Heu	0.2	4.6	151.9
	MRO-LDR	2.1	31.3	232.1
2	MRO-Ex	49.3	3238.1	3796.0
	MRO-CG	2.9	56.3	707.5
	MRO-Heu	0.6	24.2	947.9
	MRO-LDR	2.2	33.8	297.3
5	MRO-Ex	3677.9	4305.1	4081.9
	MRO-CG	11.5	200.1	2035.6
	MRO-Heu	3.1	998.7	3614.6
	MRO-LDR	2.7	40.3	439.7

Table 3: Selection: Average CPU time in seconds to generate instances, using (lenient) 3600 seconds time limit.

The inferior results of MRO-Ex are explained by Table 3, which shows a significant reduction in the run time for all methods compared to MRO-Ex. In most cases, MRO-Ex cannot complete within the lime limit. The best performing approach when $b = 1$ is MRO-Heu, with an average run time that is 5% of that for MRO-Ex when $n = 40$. This remains the case when $b = 5$, where MRO-LDR outperforms all other methods. While the decomposition approach MRO-CG does not dominate any of the other approaches, it exhibits its best performance compared to MRO-Heu as b and n increases.

Comparing Table 1 with the average generation times in Table 3, it is observed that while MRO-LDR is the fastest method for solving MRO, it is the worst method for producing hard robust optimization instances. Also, the approximation methods of MRO-CG and MRO-Heu, while faster than MRO-Ex, are able to produce hard robust optimization instances. This demonstrates a clear advantage to the iterative methods of the MRO, in particular the use of relaxation and heuristic methods.

Considering the iterative methods, MRO-Ex, MRO-CG and MRO-Heu, the exact approach generates the hardest instances when it is capable of solving MRO to optimality. When MRO-Ex can not solve MRO to optimality, then MRO-Heu produces the most difficult instances. Interestingly, when $n = 40$ and $b = 2, 5$ the average run times of the instances generated by MRO-Heu is twice as large as those generated by MRO-Ex. While MRO-CG does not produce instances harder than MRO-Heu, they are harder than those produced by MRO-Ex when MRO is not solved to optimality.

Figure 1 gives a more detailed box plot comparison for the case $n = N = 40$ and $b = 5$, i.e., for the hardest instances. Note the logarithmic vertical axis. The highest observed computation time for RU was 6.59 seconds, while MRO-Heu achieved a maximum of 61.03 seconds. The Wilcoxon signed-rank test confirms that MRO-Heu produces harder instances than RU with $p < 0.001$.

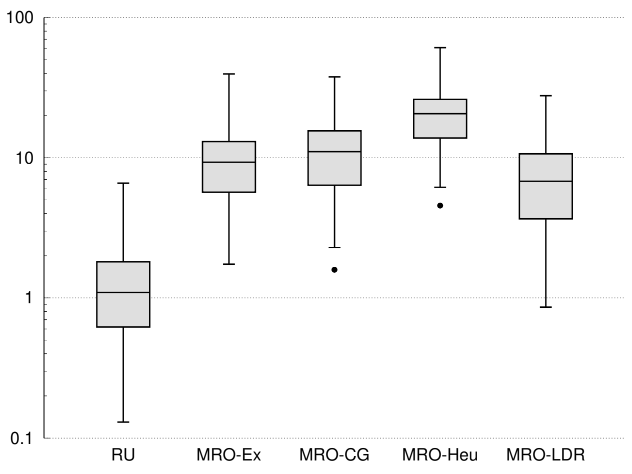

Figure 1: Selection: Box plot of solution times (in seconds) for instances generated with $b = 5$ and $n = N = 40$.

5.3. Large-Scale Instance Generation for Selection

The ability of the MRO, especially the iterative methods, is demonstrated in Section 5.2 to produce instances significantly harder than randomly generated instances. However, Table 3 shows that the generation time is a major limitation to the proposed approaches. Specifically, as the instance size grows, the ability of MRO-Ex and MRO-Heu to solve the MRO decreases. Thus, it is necessary to employ alternative methods when generating hard robust optimization instances for larger problems.

Using the approach proposed in Section 3.3—solving the inner robust optimization problem heuristically—hard instances to larger Selection problems can be generated. In particular, we use a heuristic to solve the robust Selection problem within the alternating heuristic, labeled as MRO-LSHeu. The results from using MRO-LSHeu, compared to RU, for different numbers of items n are presented in Table 4. We show the average time for the instances that were solved to optimality and the number of instances which can be solved within the time limit of one hour.

An initial observation from Table 4 is that MRO-LSHeu is able to produce instances that are an order of magnitude harder than RU, even for the smallest instances. Comparing these results with Section 5.2, it is clear that solving the robust optimization problem heuristically

n	RU		MRO-LSHeu	
20	0.02	(100)	0.06	(100)
30	0.16	(100)	1.21	(100)
40	1.72	(100)	25.45	(100)
50	17.83	(100)	699.46	(100)
60	163.65	(100)	1749.20	(17)
70	1220.19	(86)	-	(0)

Table 4: SELECTION: Average CPU solving time in seconds for instances that were solved to optimality (number of instances that were solved to optimality).

has advantages for the MRO as opposed to solving it exactly in MRO-Heu. Table 4 also demonstrates that as the size of the instances increases, many more of the instances produced by MRO-LSHeu cannot be solved within one hour decreases, compared to those produced by RU. Specifically, when $n = 60$, only 17 of the instances generated by MRO-LSHeu could be solved compared to 100 for the randomly generated instances. This drops to 0 for the MRO-LSHeu instances when n is increased to 70.

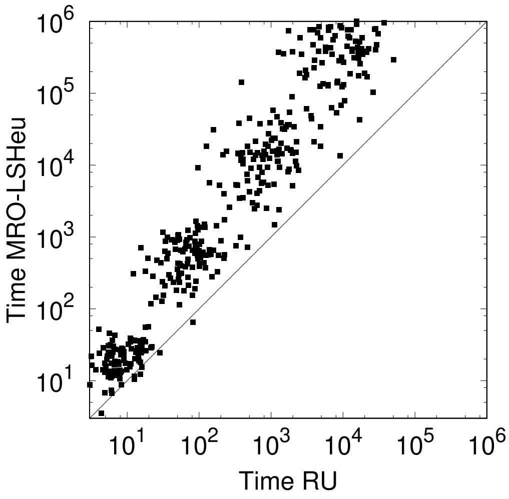

Figure 2: SELECTION: Comparison of solution times per instance for $n = 20$ to $n = 50$.

To better visualize this result, Figure 2 compares the run times required to solve the instances generated by RU (horizontal axis) and MRO-LSHeu (vertical axis). Note that in this figure the axes are scaled logarithmically. Every point corresponds to one instance for $n = 20$ to $n = 50$ (problem sizes where all instances were solved to optimality). A point on the diagonal means that a randomly generated instance remained as hard after applying MRO-LSHeu to increase its difficulty. The clusters of dots are related to the values of n and appear to display a linear relationship as n increases. Since this figure is displayed using double logarithmic axes, this linear relationship represents an exponential increase in the run times after applying MRO-LSHeu to produce hard instances. Thus, MRO-LSHeu is capable of achieving significantly harder instances that RU for large problem sizes of SELECTION.

We complement these observations with Table 5, where the average increase in solution time of MRO-LSHeu instances over the original RU is shown. For $n = 60$ and $n = 70$, the true computation time is not always known. In these cases, we used 3600 seconds, but the actual increase will be higher than the numbers indicate. We see that with increasing instance size, our instance generation approach becomes more powerful, with an increase in hardness by a factor of up to 440.

n	Avg	Max
20	2.8	12.9
30	9.9	45.6
40	23.1	195.9
50	73.8	440.7
60	> 46.1	> 406.5
70	> 4.75	> 44.3

Table 5: Selection: Average and maximum time increase for each instance using MRO-LSHeu. When not solved to optimality, counted as time limit.

Finally, we also present the time to generate these instances in Table 6. On average, it takes less than 20 seconds to generate hard instances for $n = 70$, which is a small fraction of the time needed to solve the resulting problems. This demonstrates that the proposed heuristic is an efficient tool to generate larger and harder instances, overcoming the limitations of the exact approach, and scaling well beyond $n = 70$.

n	20	30	40	50	60	70
Time	0.1	0.3	0.9	2.0	4.4	19.4

Table 6: Selection: Average time to generate instances using MRO-LSHeu.

5.4. Hard Instance Generation for TSP

We turn our attention to the TSP for the next set of experimental results to assess whether the proposed methods can be applied to generate hard instances to more computationally difficult robust problems. Given the dominance of the MRO-Heu approach for the Selection problem, these experiments will only evaluate RU, MRO-Ex and MRO-Heu. Also, for the larger problem sizes, the experiments will compare the difficulty of the instances generated by RU and MRO-LSHeu.

The results of the first experiment, generating hard instances by MRO-Ex and MRO-Heu with different values of $n = N^2$ and b, are presented in Table 7. Similar to the results in Section 5.2, both MRO-Ex and MRO-Heu produce instances that are significantly harder than RU. Also, of the considered algorithms MRO-Heu is shown to generate the hardest instances.

		$n = N^2$		
Budget	Method	100	144	196
2	MRO-Ex	0.4	5.0	43.7
	MRO-Heu	0.5	6.7	78.1
5	MRO-Ex	1.4	10.7	52.2
	MRO-Heu	1.5	14.6	103.0
	RU	0.2	1.1	9.0

Table 7: TSP: Average CPU time in second when solving the generated instances.

We find that with a larger budget, it is possible to increase the hardness considerably. While randomly generated instances for the 14-city TSP take 9 seconds on average to solve, the instances hardened by using MRO-Heu take on average 103 seconds to solve. The increase in hardness is further demonstrated by Table 8, which shows the relative increase in computation time, averaged over all instances. We find the instances generated by MRO-Heu exhibit a factor of 25 for $n = 196$ longer than the randomly generated instances on average. Further, MRO-Heu generates instances that take up to 230 times longer to solve than those originally produced by RU.

		$n = N^2$					
Budget	Method	100		144		196	
2	MRO-Ex	2.8	(5.3)	5.3	(9.6)	6.8	(25.8)
	MRO-Heu	3.0	(6.1)	6.8	(11.7)	11.9	(28.1)
5	MRO-Ex	10.0	(24.1)	14.2	(65.5)	12.1	(169.0)
	MRO-Heu	11.0	(27.2)	22.2	(118.0)	25.0	(232.7)

Table 8: TSP: Average (maximum) time increase relative to RU for each set of instances.

Similar to the results for SELECTION, the generation of hard instances for the TSP is time consuming. Table 9 shows that as budget b and the number of edges n increases, the run time required to generate hard instance also increases. In fact, it is difficult to solve the MRO to optimality for any problems with more than 14 nodes in the underlying graph. While the increase in difficulty for these smaller instances should be sufficient, if hard instances to larger problems are desired, then Table 9 highlights that MRO-Ex and MRO-Heu are not satisfactory algorithms. This shows the need for heuristic approaches for solving the inner robust optimization problem of the MRO, as proposed in Section 3.3.

The ability of MRO-LSHeu to generate hard instances for larger problem sizes of the TSP is shown in Table 10. In these experiments, instances were found for problems formulated with up to 20 nodes. Also, in order to generate harder instances the budget b was increased to 20.

The results presented in Table 10 show that MRO-LSHeu produces instances that are significantly harder than randomly generated instances. When increasing the number of nodes to 16, it is the first time that an instance generated by MRO-LSHeu is unable to be solved within one hour. Further increasing the number of nodes to 18 results in a significant increase in the run times to solve the hard robust optimization instances—only two instances can be solved within one hour. These results further demonstrate the ability of MRO-LSHeu to produce hard instances for large robust optimization problems.

The relative increase in difficulty for the MRO-LSHeu instances over the RU instance is presented in Table 11. While these numbers are lower than for our previous methods for $n \leq 196$ (compare to Table 8), the heuristic approach is shown to scale better for larger instances sizes. It can be observed that MRO-LSHeu can produce instances that are a factor of up to 500 times

		$n = N^2$		
Budget	Method	100	144	196
2	MRO-Ex	6.3	149.1	1327.9
	MRO-Heu	4.6	134.2	2028.4
5	MRO-Ex	954.6	3170.0	3475.6
	MRO-Heu	143.1	2365.1	3507.0

Table 9: TSP: Average CPU time in seconds to generate instances, using (lenient) 3600 seconds time limit.

n	RU		MRO-LSHeu	
100	0.2	(100)	0.4	(100)
144	1.1	(100)	3.5	(100)
196	9.3	(100)	66.0	(100)
256	67.4	(100)	1040.7	(99)
324	442.1	(100)	2512.8	(2)
400	1624.2	(60)	-	(0)

Table 10: TSP: Average CPU solving time in seconds for instances that were solved to optimality (number of instances that were solved to optimality).

more difficult to solve that the original randomly generated instance. Since with $n \geq 256$ many of the generated instances could not be solved to optimality, this decreases the factor increase. Thus, it is expected that MRO-LSHeu is able to produce instances that are more than a factor of 500 times more difficulty than the RU instances as the problem size increases.

n	Avg	Max
100	3.1	11.1
144	4.9	40.7
196	14.8	151.8
256	> 41.8	> 504.3
324	> 15.6	> 267.6
400	> 2.7	> 20.5

Table 11: TSP: Average and maximum time increase for each instance using MRO-LSHeu. When not solved to optimality, counted as time limit.

To finally highlight the potential of the MRO-LSHeu method, the instance generation times are presented in Table 12. For these results, the generation time limit was greatly restricted to 600 seconds. Comparing Tables 9 and 12 it is clear that MRO-LSHeu is capable of generating harder instance than MRO-Heu in much shorter run times. This is a significant benefit to the approach heuristic approach for solving the MRO, since it enables the use of this technique to generate hard instances for difficult robust optimization problems.

n	100	144	196	256	324	400
Time	0.1	24.4	392.7	504.2	552.2	570.2

Table 12: TSP: Average time to generate instances using MRO-LSHeu (600s limit).

6. Conclusions

All relevant min-max robust combinatorial optimization problems are known to be NP-hard. But this theoretical complexity class does not necessarily indicate practical hardness. Indeed, randomly generated instances are usually not too challenging to solve with current off-the-shelf MIP solvers. Furthermore, algorithmic papers need to re-create such random instances every time, resulting not only in additional work for computational experiments, but also in another source of errors and incomparability of results.

The aim of this paper is to address these problems by introducing a way to generate problem instances which are considerably harder to solve than random instances. We first present exact and heuristic approaches for solving an optimization problem to generate hard robust problem instances. While these approaches require much computational effort to find hard instances, they are effective at increasing the difficulty over the randomly generated instances. To address the computational issues in generating hard robust instances, we propose the use of heuristic methods to solve the inner robust optimization problem within iterative solution algorithms. This latter approach is demonstrated to produce the most difficult instances within very short generation times.

Our methods have been illustrated using the Selection problem and TSP as examples, but they are widely applicable to other combinatorial problems. In further work, the foundation that is laid through these instances will be extended to a more comprehensive online problem library for robust optimization. All current instances are already available under `www.robust-optimization.com`. Also, the code used to generate the instances has been made available and can be easily extended to other underlying mathematical programming problems.

Three more approaches were tested, but not described in this paper. The first approach is based on constructing instances where the midpoint heuristic, a cornerstone of exact solution methods, performs badly. While an increase in hardness could be observed, results were not as promising as for MRO. Details on this setting and some computational results are provided in Appendix D. The second approach was based on the linear program developed in [GH18] to construct a solution with small approximation guarantee for a given problem instance. We constructed instances by maximizing this approximation guarantee. The third approach was to train a neural network on a set of problem instance-solution time observations to predict hard instances. However, both approaches were not able to produce instances that were significantly harder than random instances. However, machine learning models seem to be a promising avenue for future research once hard instances have been generated by other models and thus become available for the training data.

Acknowledgments

Stephen J. Maher is supported by the Engineering and Physical Sciences Research Council (EPSRC) grant EP/P003060/1. We thank the anonymous reviewers for the constructive comments that helped improving this paper.

References

[ABV09] H. Aissi, C. Bazgan, and D. Vanderpooten. Min–max and min–max regret versions of combinatorial optimization problems: A survey. *European Journal of Operational Research*, 197(2):427 – 438, 2009.

[BK17] C. Buchheim and J. Kurtz. Min–max–min robust combinatorial optimization. *Mathematical Programming*, 163(1-2):1–23, 2017.

[BTGGN04] A. Ben-Tal, A. Goryashko, E. Guslitzer, and A. Nemirovski. Adjustable robust solutions of uncertain linear programs. *Mathematical Programming*, 99(2):351–376, 2004.

[CG15] A. Chassein and M. Goerigk. A new bound for the midpoint solution in min-max regret optimization with an application to the robust shortest path problem. *European Journal of Operational Research*, 244(3):739–747, 2015.

[CG18] A. Chassein and M. Goerigk. On scenario aggregation to approximate robust combinatorial optimization problems. *Optimization Letters*, 12(7):1523–1533, 2018.

[CLSN11] D. Catanzaro, M. Labbé, and M. Salazar-Neumann. Reduction approaches for robust shortest path problems. *Computers & operations research*, 38(11):1610–1619, 2011.

[DGJ09] C. Demetrescu, A. V. Goldberg, and D. S. Johnson. *The Shortest Path Problem: Ninth DIMACS Implementation Challenge*, volume 74. American Mathematical Soc., 2009.

[GH18] M. Goerigk and M. Hughes. Representative scenario construction and preprocessing for robust combinatorial optimization problems. *Optimization Letters*, Oct 2018. To appear, online first.

[GJ79] M. Garey and D. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness.* Freeman, San Francisco, CA, 1979.

[GS16] M. Goerigk and A. Schöbel. Algorithm engineering in robust optimization. In *Algorithm engineering*, pages 245–279. Springer International Publishing, 2016.

[KAA$^+$11] T. Koch, T. Achterberg, E. Andersen, O. Bastert, T. Berthold, R. E. Bixby, E. Danna, G. Gamrath, A. M. Gleixner, S. Heinz, et al. Miplib 2010. *Mathematical Programming Computation*, 3(2):103, 2011.

[KMZ12] A. Kasperski, M. Makuchowski, and P. Zieliński. A tabu search algorithm for the minmax regret minimum spanning tree problem with interval data. *Journal of Heuristics*, 18(4):593–625, 2012.

[KY97] P. Kouvelis and G. Yu. *Robust Discrete Optimization and Its Applications.* Kluwer Academic Publishers, 1997.

[KZ16] A. Kasperski and P. Zieliński. Robust discrete optimization under discrete and interval uncertainty: A survey. In *Robustness Analysis in Decision Aiding, Optimization, and Analytics*, pages 113–143. Springer, 2016.

[MHS10] M. Müller-Hannemann and S. Schirra. *Algorithm engineering: bridging the gap between algorithm theory and practice*, volume 5971. Springer, 2010.

[PA11] J. Pereira and I. Averbakh. Exact and heuristic algorithms for the interval data robust assignment problem. *Computers & Operations Research*, 38(8):1153–1163, 2011.

[Pis05] D. Pisinger. Where are the hard knapsack problems? *Computers & Operations Research*, 32(9):2271–2284, 2005.

[Sol87] M. M. Solomon. Algorithms for the vehicle routing and scheduling problems with time window constraints. *Operations Research*, 35(2):254–265, 1987.

[Sto76] Larry J Stockmeyer. The polynomial-time hierarchy. *Theoretical Computer Science*, 3(1):1–22, 1976.

A. Complexity of MRO

Theorem 2. *The problem*

$$\max_{(\boldsymbol{c}^1,\dots,\boldsymbol{c}^N)\in\mathcal{U}} \min_{\boldsymbol{x}\in\mathcal{X}} \max_{j\in[N]} \boldsymbol{c}^j\boldsymbol{x} \tag{MRO}$$

with a polyhedron $\mathcal{U}\subseteq\mathbb{R}^{n\times N}$ is Σ_2^p-complete.

Proof. We first note that (MRO) is in class Σ_2^p, as for fixed $(\boldsymbol{c}^1,\dots,\boldsymbol{c}^N)$, the resulting robust optimization problem is in NP. We use a reduction from 2-Quantified 3-Dnf-Sat, which is known to be Σ_2^p-complete [Sto76]: Given a Boolean formula $\phi(\boldsymbol{\alpha},\boldsymbol{\beta})$ in DNF, where every clause consists of exactly three literals, is there a value for $\boldsymbol{\alpha}=(\alpha_1,\dots,\alpha_s)\in\{0,1\}^s$ such that for all $\boldsymbol{\beta}=(\beta_1,\dots,\beta_t)\in\{0,1\}^t$, the formula $\phi(\boldsymbol{\alpha},\boldsymbol{\beta})$ is true?

We build an instance of (MRO) using the Representative Selection problem [KZ16] to define the underlying combinatorial problem. In this problem, we are given a partition of the items $\cup_{\ell\in[k]}T_\ell=[n]$ and a cost vector $\boldsymbol{c}\in\mathbb{R}^n$. The task is to choose exactly one item of each T_ℓ, such that the overall costs are minimized.

Our problem instance is built in the following way. Let $x_i\in\{0,1\}$ be the variable corresponding to variable α_i, and let $y_i^1,y_i^2\in\{0,1\}$, correspond to β_i. Note that $n=s+2t$. We partition the items through sets T_ℓ for each $\ell\in[s]$, where $T_\ell=\{x_\ell\}$ (i.e., each x_i must be equal to one), and T'_ℓ for each $\ell\in[t]$ with $T'_\ell=\{y_\ell^1,y_\ell^2\}$ (i.e, exactly one of y_ℓ^1 and y_ℓ^2 must be equal to one). Let us denote a clause of ϕ by

$$C_i=(a_1^i\alpha_1\wedge a_2^i\alpha_2\wedge\dots\wedge a_s^i\alpha_s\wedge b_1^i\beta_1\wedge b_2^i\beta_i\wedge\dots\wedge b_t^i\beta_t)$$

where a_k^i,b_k^i denote the signs of the variables in $\{-1,0,1\}$ (exactly three signs are non-zero). Let N clauses be given. We build a scenario for each clause. The corresponding polyhedron of possible scenarios is given as:

$$\mathcal{U}=\Big\{(\boldsymbol{c}^1,\dots,\boldsymbol{c}^N)\ :\begin{array}{ll} c^i(x_j)=a_j^i d_j & \forall i\in[N],j\in[s]\\ d_j\in[-1,1] & \forall j\in[n]\\ c^i(y_j^1)=b_j^i & \forall i\in[N],j\in[t]\\ c^i(y_j^2)=-b_j^i & \forall i\in[N],j\in[t]\end{array}\Big\}$$

where

$$(\boldsymbol{c}^1,\dots,\boldsymbol{c}^N)=\begin{pmatrix} c^1(x_1) & c^2(x_1) & \dots & c^N(x_1)\\ c^1(x_2) & c^2(x_2) & \dots & c^N(x_2)\\ \vdots & \vdots & \ddots & \vdots\\ c^1(x_s) & c^2(x_s) & \dots & c^N(x_s)\\ c^1(y_1^1) & c^2(y_1^1) & \dots & c^N(y_1^1)\\ c^1(y_2^1) & c^2(y_2^1) & \dots & c^N(y_2^1)\\ \vdots & \vdots & \ddots & \vdots\\ c^1(y_t^1) & c^2(y_t^2) & \dots & c^N(y_t^2)\\ c^1(y_1^2) & c^2(y_1^2) & \dots & c^N(y_1^2)\\ c^1(y_2^2) & c^2(y_2^2) & \dots & c^N(y_2^2)\\ \vdots & \vdots & \ddots & \vdots\\ c^1(y_t^2) & c^2(y_t^2) & \dots & c^N(y_t^2)\end{pmatrix}$$

Note that the projection of $\mathcal{U}$ onto $\mathbb{R}^{n\times N}$ is indeed a polyhedron. We claim that there exists an optimal solution to our MRO instance with objective value at least 3 if and only if the 2-Quantified 3-Dnf-Sat instance is true. Let us first assume that there exists $\boldsymbol{\alpha}$ such that $\phi(\boldsymbol{\alpha},\boldsymbol{\beta})$ is true for all $\boldsymbol{\beta}$. We construct N scenarios by setting $d_i=1$ if α_i is true, and $d_i=-1$ otherwise. Then the resulting robust problem becomes

$$\min\ z$$

$$
\begin{aligned}
\text{s.t. }\ & z \geq \sum_{j\in[s]} a^i_j d_j + \sum_{j\in[t]} b^i_j (y^1_j - y^2_j) && \forall i \in [N] \\
& y^1_j + y^2_j = 1 && \forall j \in [t] \\
& y^1_j, y^2_j \in \{0,1\} && \forall j \in [t]
\end{aligned}
$$

By construction, it follows that for every feasible solution $(\boldsymbol{y}^1, \boldsymbol{y}^2)$, the optimal value of z is 3. Let us now assume that the objective value of the (MRO) is at least 3. Note that in this case, the objective value is exactly three, and we can assume that $d_j \in \{-1, 1\}$ for all $i \in [s]$. Set α_j as true if and only if $d_j = 1$. Then, it follows that for every possible value $\boldsymbol{\beta}$, the formula ϕ is true, as the robust problem aims at finding values for $(\boldsymbol{y}^1, \boldsymbol{y}^2)$ such that all clauses are false. □

B. Column Generation for MRO

Dantzig-Wolfe reformulation is applied to (2) to decompose the problem into K disjoint subsystems (one for each candidate solution x). A column p corresponds to a feasible assignment of a cost vector $\boldsymbol{c}^i \in \mathcal{U}_i$ to the solution vector $\boldsymbol{x}^j$. For a given column $p \in P^j$, the parameter $\overline{d}_{ikp}$ is the contribution of $c^i_k x^j_k$ to the objective of the inner minimization problem given the assignment of $\boldsymbol{c}^i$ to solution vector $\boldsymbol{x}^j$. The variables α_p equal 1 if the cost vector assignment given by column p is selected and 0 otherwise. Finally, the variables c^i_k are introduced to map the solution of the outer maximization problem to the set of cost vectors for the inner minimization problem.

The formulation of the column generation master problem is given by

$$
\begin{aligned}
& \max\ t \\
\text{s.t. }\ & t \leq \sum_{p\in P^j} \sum_{i\in[N]} \sum_{k\in[n]} \overline{d}_{ikp}\alpha_p && \forall j \in [K]\ (\gamma_j) \\
& \sum_{p\in P^j} \sum_{p\in P^j} \alpha_p = 1 && \forall j \in [K]\ (\delta_j) \\
& \sum_{p\in P^j} \overline{d}_{ikp}\alpha_p \leq c^i_k && \forall i \in [N], j \in [K], k \in [n]\ (\pi_{ijk}) \\
& \boldsymbol{c}^i \in \mathcal{U}_i && \forall i \in [N] \\
& \alpha_p \in \mathbb{Z}_+ && \forall j \in [K], p \in P^j
\end{aligned}
\tag{3}
$$

Initially, the master problem is formulated with only a subset of columns $\bar{P}^j \subseteq P^j$. The corresponding problem is described as the restricted master problem (RMP). For each $j \in [K]$, a single initial column is included in $\bar{P}^j$, which is formed by assigning $\overline{c}_k$ to x^j_k. The variables γ_j, δ_j and π_{ijk} represent the dual variables corresponding to the constraints in (3).

A complicating aspect of the RMP is the set of linking constraints given by the uncertainty sets $\mathcal{U}_i$. This complication arises from the fact that the constraints do not explicitly link the α_p variables, but an implicit linking of the α_p variables is through the third set of constraints in (3). While the uncertainty set linking constraints ensure that exactly one cost vector is selected from each scenario, this requirement could be overly restrictive in our contexts. As such, a relaxation of (2) is formed by replacing $\mathcal{U}_i$ with $\mathcal{U}^j_i$, where $\mathcal{U}^j_i = \mathcal{U}_i, \forall j \in [K]$, so that a different cost vector from scenario i could be selected for each solution $j \in [K]$. Applying this relaxation eliminates the linking constraints from the uncertainty sets $\mathcal{U}_i$ and transfers the additional relaxed constraints to the column generation subproblems.

A column generation subproblem is formed for each solution $j \in [K]$. Given the optimal dual solution to the RMP, each column generation subproblem is solved to find a feasible cost vector assignment that has a positive reduced cost. The dual variables are denoted by γ_j, δ_j and π_{ijk} respectively for the constraints of the RMP. Using an optimal dual solution, the reduced cost of a column for solution j is given by $\overline{d}^j = \sum_{i\in[N]} \sum_{k\in[n]} d_{ijk} x^j_k \overline{\gamma}_j - d_{ijk}\overline{\pi}_{ijk} - \overline{\delta}_j$. A feasible assignment of $\boldsymbol{c}^i \in \mathcal{U}_i^j$ to solution $\boldsymbol{x}^j$ forms an improving column for the RMP if the reduced costs are positive. By solving an optimization subproblem, we find the feasible cost vector assignment that forms a column with the most positive reduced cost. Since only a relaxation of (2) is solved by this approach, the objective function value will be greater than that found by the iterative approach (Section 3.1). However, in the proposed approach for generating hard instances, maximizing the minimum robust objective value is used only as a proxy for hardness. As such, it is expected that even solving the relaxation of (2) will provide instances that are of comparative hardness to the exact approach in Section 3.1.

C. Linear Decision Rules for MRO

A common reformulation of robust optimization problems involves the application of decision rules [BTGGN04]. This approach involves introducing the adjustable variables $\lambda^i : \{0,1\}^n \to [0,1]$ which map solutions $\boldsymbol{x}$ to the worst-case scenario. In the context of MRO, such a mapping would result in setting $\lambda^i(\boldsymbol{x}) = 1$ if scenario $\boldsymbol{c}^i$ is a worst-case scenario for solution $\boldsymbol{x}$, and 0 otherwise.

Considering the MRO, the use of a decision rule results in an equivalent formulation given by

$$\begin{aligned} \max\ & t \\ \text{s.t. } & t \le \sum_{k\in[n]} \sum_{i\in[N]} \lambda^i(\boldsymbol{x}) c_k^i x_k && \forall \boldsymbol{x} \in \mathcal{X} \\ & \sum_{i\in[N]} \lambda^i(\boldsymbol{x}) \le 1 && \forall \boldsymbol{x} \in \mathcal{X} \\ & \lambda^i : \{0,1\}^n \to [0,1] && \forall i \in [N] \\ & \boldsymbol{c}^i \in \mathcal{U}_i && \forall i \in [N] \end{aligned} \tag{4}$$

The optimal decision rule can only be found through the solution to the original robust optimization problem. As such, it is common to apply approximations of the decision rules to find a closed form of the reformulated problem. First-order or linear decision rules involve defining the vector mapping $\lambda^i(\boldsymbol{x})$ as an affine linear function, such as

$$\lambda^i(\boldsymbol{x}) := \lambda_0^i + \sum_{k\in[n]} \lambda_k^i x_k.$$

This introduces the new variables λ_0^i, λ_k^i for all $k \in [n]$. An approximation of MRO is given by substituting the linear function mapping in (4), resulting in the reformulation given by

$$\max\ t \tag{5}$$

$$\text{s.t. } t \le \sum_{k\in[n]} \sum_{i\in[N]} (\lambda_0^i + \sum_{\ell\in[n]} \lambda_\ell^i x_\ell) c_k^i x_k \qquad \forall \boldsymbol{x} \in \mathcal{X} \tag{6}$$

$$\sum_{i\in[N]} (\lambda_0^i + \sum_{k\in[n]} \lambda_k^i x_k) \le 1 \qquad \forall \boldsymbol{x} \in \mathcal{X} \tag{7}$$

$$\lambda_0^i + \sum_{k\in[n]} \lambda_k^i x_k \ge 0 \qquad \forall i \in [N], \boldsymbol{x} \in \mathcal{X} \tag{8}$$

$$\lambda_0^i + \sum_{k\in[n]} \lambda_k^i x_k \le 1 \qquad \forall i \in [N], \boldsymbol{x} \in \mathcal{X} \tag{9}$$

$$\boldsymbol{c}^i \in \mathcal{U}_i \qquad \forall i \in [N] \tag{10}$$

Note that it is possible to remove constraints (9) since they are implied by constraints (7) and (8).

It can be observed that the reformulated problem has an exponential number of constraints, resulting from a set of constraints for each solution contained in $\mathcal{X}$. As such, problem (5)–(10) is intractable in its current form. Using the following linear relaxation assumption, a further reformulation can be performed to address the intractability of problem (5)–(10)

Assumption 3. *There exists a suitable polyhedron*

$$\mathcal{X}' = \{\boldsymbol{x} \in \mathbb{R}^n_+ : \boldsymbol{A}\boldsymbol{x} \leq \boldsymbol{b}, \boldsymbol{x} \leq \boldsymbol{1}\}$$

with $\boldsymbol{A} \in \mathbb{R}^{m\times n}$, $\boldsymbol{b} \in \mathbb{R}^m$, such that for any cost vector $\boldsymbol{c} \in \mathbb{R}^n$, we have

$$\min_{\boldsymbol{x}\in\mathcal{X}} \boldsymbol{c}\boldsymbol{x} = \min_{\boldsymbol{x}\in\mathcal{X}'} \boldsymbol{c}\boldsymbol{x}.$$

To apply Assumption 3, each set of constraints in (6)–(8) are examined in turn to construct a polyhedral description of linear constraints. For each set of constraints, the bounding limit is found by minimizing (maximizing) the activity for greater (less) than constraints. Assumption 3 is given for a wide range of commonly considered robust combinatorial optimization problems, such as Selection, Spanning Tree, and Assignment (see also [KZ16]).

For ease of presentation, we describe the reformulation using Selection as an example. Consider Constraint (6), which is equivalent to

$$t \leq \min_{\boldsymbol{x}\in\mathcal{X}} \sum_{k\in[n]} (\sum_{i\in[N]} \lambda^i_0 c^i_k) x_k + \sum_{k\in[n]} \sum_{\ell\in[n]} (\sum_{i\in[N]} \lambda^i_\ell c^i_k) x_\ell x_k$$

First the product $x_\ell x_k$ is linearized by introducing a new variable y_{kl}. Then the resulting problem can be relaxed to form

$$\begin{aligned}
\min \ & \sum_{k\in[n]} (\sum_{i\in[N]} \lambda^i_0 c^i_k) x_k + \sum_{\ell\in[n]} (\sum_{i\in[N]} \lambda^i_\ell c^i_k) y_{k\ell} & \\
& 2y_{k\ell} \leq x_\ell + x_k & \forall \ell, k \in [n] \\
& x_k + x_\ell \leq y_{k\ell} + 1 & \forall \ell, k \in [n] \\
& y_{k\ell} \geq 0 & \forall \ell, k \in [n] \\
& \sum_{i\in[n]} x_i = p & \\
& x_k \leq 1 & \forall k \in [n] \\
& x_k \geq 0 & \forall k \in [n]
\end{aligned}$$

Note that this will give a conservative approximation to Constraint (6), as the minimum in the right-hand side is underestimated. Also, the right-hand side of (6) is ignored when applying Assumption 3, since it will be enforced in the reformulation of MRO. By dualizing the problem, we find

$$\begin{aligned}
\max \ & -\sum_{k\in[n]} \sum_{\ell\in[n]} \zeta_{k\ell} + p\eta - \sum_{k\in[n]} \theta_k & \\
\text{s.t.} \ & \sum_{\ell\in[n]} (\xi_{k\ell} + \xi_{\ell k} - \zeta_{k\ell} - \zeta_{\ell k}) + \eta - \theta_k \leq \sum_{i\in[N]} \lambda^i_0 c^i_k & \forall k \in [n] \\
& -2\xi_{k\ell} + \zeta_{k\ell} \leq \sum_{i\in[N]} \lambda^i_\ell c^i_k & \forall k, \ell \in [n] \\
& \xi, \zeta, \theta \geq 0 & \\
& \eta \gtrless 0 &
\end{aligned}$$

By strong duality, this model can be substituted for Constraint (6).

By applying this approach also to the other constraints, the linear decision rule approach to MRO is given through the following optimization problem:

$$\max\ p\eta - \sum_{k\in[n]}\sum_{\ell\in[n]} \zeta_{k\ell} - \sum_{k\in[n]} \theta_k \tag{11}$$

$$\text{s.t.}\ \sum_{\ell\in[n]} (\xi_{k\ell} + \xi_{\ell k} - \zeta_{k\ell} - \zeta_{\ell k})$$

$$+ \eta - \theta_k \le \sum_{i\in[N]} \lambda^i_0 c^i_k \qquad \forall k \in [n] \tag{12}$$

$$- 2\xi_{k\ell} + \zeta_{k\ell} \le \sum_{i\in[N]} \lambda^i_\ell c^i_k \qquad \forall k,\ell \in [n] \tag{13}$$

$$\sum_{i\in[N]} \lambda^i_0 + p\pi + \sum_{k\in[n]} \rho_k \le 1 \tag{14}$$

$$\pi + \rho_k \ge \sum_{i\in[N]} \lambda^i_k \qquad \forall k \in [n] \tag{15}$$

$$\lambda^i_0 + p\alpha^i - \sum_{k\in[n]} \beta^i_k \ge 0 \qquad \forall i \in [N] \tag{16}$$

$$\alpha^i - \beta^i_k \le \lambda^i_k \qquad \forall k \in [n], i \in [N] \tag{17}$$

$$\xi, \zeta, \theta \ge 0 \tag{18}$$

$$\eta \gtrless 0 \tag{19}$$

$$\pi \gtrless 0 \tag{20}$$

$$\rho_k \ge 0 \qquad \forall k \in [n] \tag{21}$$

$$\alpha^i \gtrless 0 \qquad \forall i \in [N] \tag{22}$$

$$\beta^i_k \ge 0 \qquad \forall k \in [n], i \in [N] \tag{23}$$

$$\lambda^i_k \gtrless 0 \qquad \forall i \in [N], k \in [n] \cup \{0\} \tag{24}$$

$$\boldsymbol{c}^i \in \mathcal{U}_i \qquad \forall i \in [N] \tag{25}$$

The reformulation of constraint (6) is given by the objective (11) and constraints (12)–(13). For constraint (7), the reformulation is given by (14)–(15). Note that the right-hand side of (7) is the right-hand side of (14). Finally, the reformulation of (8) is given by (16)–(17). Similarly, the left-hand side of (8) is the left-hand side of (16).

There is still a nonlinearity between variables λ^i_ℓ and c^i_k, with λ^i_ℓ being unbounded. We solve the optimization problem heuristically, using an alternating approach similar to Section 3.2. By fixing either variables $\boldsymbol{\lambda}$, π, $\boldsymbol{\rho}$, $\boldsymbol{\alpha}$, $\boldsymbol{\beta}$ or variables $\boldsymbol{c}$, we increase the current objective value in each iteration, until a local optimum has been reached.

Note that while we described the reformulation for the special case of Selection, the same method can be used for any problem with Assumption 3.

D. Maximizing the Midpoint Objective Value

We now explore a different view on problem hardness. Instead of maximizing the objective value of the resulting optimal solution, which, as the discussion in Section 2 has shown, is a complex optimization problem, we use the objective value of the midpoint solution as a proxy. The midpoint method is one of the most popular heuristics for min-max robust combinatorial optimization. It aggregates all scenarios into one average scenario and solves the resulting single-scenario problem, which is possible in polynomial time for some combinatorial problems (see Assumption 3). It is known to give an N-approximation to the robust problem [ABV09], and has been the best known general method until recently [CG18]. Due to its simplicity, it is also a popular submethod for exact branch-and-bound approaches [CG15].

The optimization problem to generate hard instances we consider here is therefore given as

$$\max_{\boldsymbol{c}^1,\ldots,\boldsymbol{c}^N} \max_{i\in[N]} \boldsymbol{c}^i \hat{\boldsymbol{x}} \left(\frac{1}{N} \sum_{\ell\in[N]} \boldsymbol{c}^\ell \right) \tag{MID}$$

where $\hat{\boldsymbol{x}}(\boldsymbol{c})$ denotes an optimal solution to scenario $\boldsymbol{c}$.

To enable the use of general purpose mixed integer programming solvers, a reformulation of problem (MID) is performed. A common reformulation involves applying a linearization if the nominal problem can be written as a linear program under Assumption 3, which is the case for Selection. In the following, we present a reformulation of (MID) when Selection is the nominal robust optimization problem. To apply this linearization, we enforce that $\boldsymbol{x}$ is an optimal solution to the midpoint scenario $\frac{1}{N}\sum_{\ell\in[N]} \boldsymbol{c}^\ell$ by requiring the corresponding primal and dual objective values to be equal. The resulting optimization problem is then

$$\begin{aligned}
\max\ & \sum_{i\in[N]} t_i\lambda_i && && (26)\\
\text{s.t. } & t_i = \sum_{k\in[n]} c^i_k x_k && \forall i\in[N] && (27)\\
& \sum_{i\in[N]} \lambda_i = 1 && && (28)\\
& \sum_{i\in[N]}\sum_{k\in[n]} c^i_k x_k = p\alpha - \sum_{k\in[n]} \beta_k && && (29)\\
& \sum_{k\in[n]} x_k = p && && (30)\\
& \alpha - \beta_k \le \sum_{i\in[N]} c^i_k && \forall k\in[n] && (31)\\
& \lambda_i \in \{0,1\} && \forall i\in[N] && (32)\\
& \boldsymbol{c}^i \in \mathcal{U}^i && \forall i\in[N] && (33)\\
& t_i \ge 0 && \forall i\in[N] && (34)\\
& x_k \in \{0,1\} && \forall k\in[n] && (35)\\
& \alpha \ge 0 && && (36)\\
& \beta_k \ge 0 && \forall k\in[n] && (37)
\end{aligned}$$

Here, t_i denotes the objective value of the midpoint solution in scenario $\boldsymbol{c}^i$ (see Constraint (27)). The optimization problem maximizes the largest t_i by choice variables λ_i (see Objective (26) and Constraint (28)). Constraints (29-31) ensure that $\boldsymbol{x}$ is indeed the midpoint solution by enforcing primal and dual feasibility, and equality of primal and dual objective values.

There are still nonlinearities between $t_i\lambda_i$ and $c^i_k x_k$. We linearize the first product using $q_i = t_i\lambda_i$ with $q_i \le t_i$ and $q_i \le M_i\lambda_i$, where $M_i = \sum_{k\in[n]} \overline{c}^i_k$ suffices. The second product is linearized using $r_{ik} = c^i_k x_k$ with $r_{ik} \le c^i_k$ and $r_{ik} \le \overline{c}^i_k x_k$.

We now compare this approach to MRO-Ex with a similar experimental setup as before. For ease of exposition, the evaluation of the efficacy of the exact solution methods will be performed using Selection only. We refer to the results using the midpoint method as Mid.

The average run times of the instances generated from MRO-Ex and Mid are presented in Table 13. For comparison, the average run times to solve the randomly generated instances is also presented.

The results presented in Table 13 show that while Mid produces instances that are more difficult than random instances, the increase in difficulty is less than that achieved by MRO-Ex. Given that Mid is a more complex algorithm for generating problem instances than a random generator, the results presented in Table 13 suggest that Mid is not a satisfactory

		$n = N$		
Budget	Method	20	30	40
1	MRO-Ex	0.04	0.61	7.83
	Mid	0.03	0.15	1.62
2	MRO-Ex	0.06	0.92	7.49
	Mid	0.03	0.16	1.74
5	MRO-Ex	0.08	0.77	10.59
	Mid	0.03	0.23	2.90
	RU	0.03	0.13	1.43

Table 13: Average CPU time in seconds when solving the random instances and the instances generated using the exact iterative method and the midpoint method.

method for instance generation. This is further highlighted by the average computation times of Mid presented in Table 14. These results demonstrate that maximizing the minimum solution objective is a better proxy for instance *hardness* than maximizing the midpoint objective.

		$n = N$		
Budget	Method	20	30	40
1	MRO-Ex	1.7	207.0	3005.8
	Mid	1.1	15.2	49.2
2	MRO-Ex	49.3	3238.1	3796.0
	Mid	2.2	28.3	148.5
5	MRO-Ex	3677.9	4305.1	4081.9
	Mid	6.0	1047.4	1807.2

Table 14: Average CPU time in seconds to produce instances using the exact iterative method and the midpoint method. A (lenient) time limit of 3600 seconds was used.