Multistage Robust Combinatorial Optimization via Quantified Integer Programming

Marc Goerigk and Michael Hartisch

Network and Data Science Management, University of Siegen, Germany
Technology Management, University of Siegen, Germany

Abstract

Decision making needs to take an uncertain environment into account. Over the last decades, robust optimization has emerged as a preeminent method to produce solutions that are immunized against uncertainty. The main focus in robust combinatorial optimization has been on the analysis and solution of one- or two-stage problems, where the decision maker has limited options in reacting to additional knowledge gained after parts of the solution have been fixed. Due to its computational difficulty, multistage problems beyond two stages have received less attention.

In this paper we argue that multistage robust combinatorial problems can be seen through the lens of quantified integer programs, where powerful tools to reduce the search tree size have been developed. By formulating both integer and quantified integer programming formulations, it is possible to compare the performance of state-of-the-art solvers from both worlds. Using selection and assignment problems as a testbed, we show that problems with up to nine stages can be solved in reasonable time.

Keywords: robust optimization; multistage optimization; quantified integer programming; combinatorial optimization

1 Introduction

Uncertainty affects most aspects of decision making, and thus needs to be taken into account preemptively. Different methodologies have been developed for this purpose, such as stochastic programming [44] or robust optimization [7], which is the focus of this paper. We consider combinatorial optimization problems of the form

\[ \min_{\mathbf{c} \in \mathbf{c}_U} \]

where \( \mathbf{c} \subseteq \{0,1\}^n \) is the set of feasible solutions, and \( \mathbf{c} \) is an uncertain cost vector. In the following, vectors are always written in bold font and the transpose sign for the scalar product between vectors is dropped for ease of notation. The field of robust optimization incorporates a diverse set of approaches to formulate robust counterparts for such problems [38]. In the most basic model we assume that a set of possible cost scenarios \( \mathbf{U} \) is given,

*Corresponding author. Email: michael.hartisch@uni-siegen.de
the so-called uncertainty set. The (min-max, one-stage) robust counterpart is then to find a solution \( x \in X \) that performs best under its worst-case outcome, i.e., to solve

\[
\min_{x \in X} \max_{c \in U} cx
\]

This approach does not take a dynamic decision making process into account, where it is possible to react to new information when it becomes revealed. For this reason, two-stage approaches have been introduced [67], which usually increase both the theoretical and practical complexity of solving robust problems. In many cases, it is possible to formulate such problems as mixed-integer programs (MIPs).

Different to the world of stochastic programming, where multistage problems beyond two stages are well-established, only few such approaches have been considered in the robust world. An intuitive reason is that the worst-case performance of a solution may depend on a single outcome of the scenario tree, and thus approximation methods such as sampling approaches are not as powerful as in the stochastic world.

In this paper we show that multistage robust combinatorial problems are actually well within the reach of current computational prowess. In particular, it is possible to see multistage robust problems through the lens of quantified integer programming (QIP), which is an extension of integer programming where the variables are ordered explicitly and some variables are existentially and others are universally quantified. QIPs are known to be \( \text{PSPACE}\)-complete [52] and also can be interpreted as two-person zero-sum games between an existential and a universal (or adversarial) player.

The structure of this paper is as follows. In Section 2 we review related literature. We formally introduce QIPs in Section 3 before discussing the multistage selection and assignment problems in Section 4. We introduce two QIP-based models, and an extended MIP formulation for the robust counterpart. In Section 5 we discuss extensive computational experiments that compare CPLEX as a state-of-the-art general MIP solver with Yasol, a solver developed for QIPs. The paper is concluded in Section 6.

2 Related Literature

This section provides an overview of recent work in the areas mainly related to this paper: robust multistage optimization (Section 2.1) and quantified programming (Section 2.2). For more general surveys on optimization under uncertainty, we refer to [46, 59, 4].

2.1 Robust Optimization

Robust optimization problems are mathematical optimization problems with uncertain data, where a valid solution is sought for any (anticipated) realization of that data as represented by the uncertainty set \( U \) [9]. Solving the robust counterpart ensures performance of the solution regarding \( U \), but may result in a high price of robustness [14], i.e. the solution is often too conservative. Different concepts were developed to overcome this problem, e.g. the concepts of light robustness [33], soft robustness [5], adjustable robustness [67] and recoverable robustness [49]. In [18] the authors discuss which approach is suitable for the problem at hand. A compact overview of prevailing uncertainty sets and robustness concepts can be found in [39, 38] where the latter focuses on the algorithm engineering methodology with
regard to robust optimization. Furthermore, in [45] a survey on robust discrete optimization is presented.

Two-stage models, e.g. adjustable robust optimization and recoverable robust optimization, are often challenging to solve as even for simple cases the problem is \textit{NP}-hard [8]. Nevertheless, within the last few years several results regarding multistage models were obtained (e.g. [10, 7]) and a tutorial-like survey on robust multistage decision-making was conducted in [24]. A discussion of multistage optimization can be found in [7, pp. 408–410] where the authors acknowledge its “extreme applied importance” but point out the computational problems that arise and question the usefulness of most approximation techniques. Besides frequently used variants of lot sizing problems (e.g. [22, 15]) robust multistage optimization has been applied to the daily operation of power systems [51], resource allocation problems [50], as well as planning and scheduling problems [48, 58, 57]. An alternative approach in multistage combinatorial optimization is to construct a solution in the first stage, and modify it in further stages. Here one aims at finding stable solutions, which require little modification to remain near optimal for a changing cost function. Examples for this setting include matroids and matchings [10], the facility location problem [32], and the knapsack problem [3]. Dynamic programming techniques can be used to solve multistage problems under uncertainty [61, 55], but often suffer from the curse of dimensionality. Other solution methods include variations of Bender’s decomposition [60], column-and-constraint generation [68], and Fourier–Motzkin elimination [69]. Additionally, iterative splitting of the uncertainty set is used to solve robust multistage problems in [60] and a partition-and-bound algorithm is presented in [11]. By considering specific robust counterparts a solution can be approximated and sometimes even guaranteed [8, 21]. Furthermore, several approximation schemes based on (affine) decision rules can be found in the literature (e.g. [12, 34]).

2.2 Quantified Programming

Quantified programs have been studied since at least 1995, when a first polynomial time algorithm for a restricted class of quantified linear programs based on quantifier elimination techniques was introduced in [35]. The term \textit{quantified linear programming} (QLP) was coined in [64] and extended to quantified integer programming (QIP) in [65]. In [28] an objective function was introduced to the framework of quantified programming. Furthermore, a geometric analysis for QLPs [52] and quantified programming models for games [30, 53] and combinatorial problems [29] have been presented. Algorithms for general QLPs were developed and implemented: an alpha–beta nested Bender’s decomposition was proposed to solve the quantified linear optimization problem and tested in a computational study [54]. A polyhedral uncertainty set was introduced for QIPs [42], which is closely related to the quantified linear implication problem [31]. Furthermore, an open-source solver for quantified programs was introduced in [27] and QIP specific pruning techniques were presented in [43].

Related to the concept of quantified programming is the quantified Boolean formula problem (QBF), which can be viewed as the satisfiability problem of a Boolean QIP where each constraint is a clause. Beside being the prototypical \textit{PSPACE}-complete problem [63] QBF allows a very compact problem description and thus several areas of application arise [62].
3 Quantified Integer Programming

In the following, we formally introduce quantified integer programming. An extended version of this paper \[\text{[11]}\] can be consulted for a more detailed discussion.

3.1 Definition and Notation

Let \( n \in \mathbb{N} \) be the number of variables and \( \mathbf{x} \in \mathbb{Z}^n \) a vector of integer variables. We use the notation \([n] := \{1, \ldots, n\}\) to denote index sets. For each variable \( x_j \) its domain \( \mathcal{L}_j \) with \( l_j, u_j \in \mathbb{Z}, \; l_j \leq u_j, \; j \in [n] \), is given by \( \mathcal{L}_j = \{y \in \mathbb{Z} : l_j \leq y \leq u_j\} \neq \emptyset \). The domain of the variable vector is \( \mathcal{L} = \{y \in \mathbb{Z}^n : \forall j \in [n] : y_j \in \mathcal{L}_j\} \). Let \( Q \in \{\exists, \forall\}^n \) denote a vector of quantifiers. We call each maximal consecutive subsequence in \( Q \) consisting of identical quantifiers a quantifier block. The quantifier corresponding to the \( i \)-th quantifier block is given by \( Q(i) \in \{\exists, \forall\} \) and the corresponding \( i \)-th variable block is given by the (ordered) index set \( B_i \subseteq [n] \). Let \( \beta \in [n] \) denote the number of variable blocks and thus \( \beta - 1 \) is the number of quantifier changes. Note that \( B_1 \cup B_2 \cup \ldots \cup B_\beta = [n] \) with \( B_i \cap B_{i'} = \emptyset \) for \( i \neq i' \).

With \( \mathcal{L}(i) \) we denote the corresponding domain of the \( i \)-th variable block as in \( \mathcal{L} \).

**Definition 3.1 (Quantified Integer Linear Program (QIP)).**

Let \( A^3 \in \mathbb{Q}^{m_2 \times n} \) and \( b^3 \in \mathbb{Q}^{m_3} \) for \( m_3 \in \mathbb{N} \). Let \( \mathcal{L} \) and \( Q \) be given with \( Q^{(1)} = Q^{(\beta)} = \exists \). Let \( \mathbf{c} \in \mathbb{Q}^n \) be the vector of objective coefficients, for which \( \mathbf{c}^{(i)} \) denotes the vector of coefficients belonging to variable block \( B_i \). The term \( Q \circ \mathbf{x} \in \mathcal{L} \) with the component-wise binding operator \( \circ \) denotes the quantification sequence \( Q^{(1)}\mathbf{x}^{(1)} \in \mathcal{L}^{(1)} \; \ldots \; Q^{(\beta)}\mathbf{x}^{(\beta)} \in \mathcal{L}^{(\beta)} \), such that every quantifier \( Q^{(i)} \) binds the variables \( \mathbf{x}^{(i)} \) of block \( i \) ranging in their domain \( \mathcal{L}^{(i)} \).

We call \((A^3, b^3, \mathbf{c}, \mathcal{L}, Q)\) with

\[
\begin{align*}
  z &= \min_{\mathbf{x}^{(i)} \in \mathcal{L}^{(i)}} \left( c^{(1)}(\mathbf{x}^{(1)}) + \max_{\mathbf{x}^{(2)} \in \mathcal{L}^{(2)}} \left( c^{(2)}(\mathbf{x}^{(2)}) + \min_{\mathbf{x}^{(3)} \in \mathcal{L}^{(3)}} \left( c^{(3)}(\mathbf{x}^{(3)}) + \ldots \min_{\mathbf{x}^{(\beta)} \in \mathcal{L}^{(\beta)}} c^{(\beta)}(\mathbf{x}^{(\beta)}) \right) \right) \right) \\
  \text{s.t. } Q \circ \mathbf{x} \in \mathcal{L} : A^3 \mathbf{x} \leq b^3 
\end{align*}
\]

(1)

a quantified integer linear program (QIP) with objective function.

We call \( A^3 \mathbf{x} \leq b^3 \) the (existential) constraint system and \( \mathcal{E} = \{i \in [\beta] \mid Q^{(i)} = \exists\} \) the set of existential variable blocks and \( \mathcal{A} = \{i \in [\beta] \mid Q^{(i)} = \forall\} \) the set of universal variable blocks. Further, we call variable \( x_j \) an existential (universal) variable if the corresponding quantifier \( Q_j \) is \( \exists \) (\( \forall \)).

A QIP can be interpreted as a two-person zero-sum game between an existential player setting the existentially quantified variables and a universal player setting the universally quantified variables with payoff \( z \). The variables are set in consecutive order according to the variable sequence \( x_1, \ldots, x_n \). We say that a player makes the move \( \mathbf{x}^{(i)} = \mathbf{y} \), if she fixes the variable vector \( \mathbf{x}^{(i)} \) of block \( i \) to \( \mathbf{y} \in \mathcal{L}^{(i)} \). At each such move, the corresponding player knows the settings of \( \mathbf{x}^{(1)}, \ldots, \mathbf{x}^{(i-1)} \) before taking her decision \( \mathbf{x}^{(i)} \).

Each fixed vector \( \mathbf{x} \in \mathcal{L} \), that is, when the existential player has fixed the existential variables and the universal player has fixed the universal variables, is called a play. If \( \mathbf{x} \) satisfies the linear constraint system \( A^3 \mathbf{x} \leq b^3 \), the existential player pays \( z = c \mathbf{x} \) to the universal player. If \( \mathbf{x} \) does not satisfy \( A^3 \mathbf{x} \leq b^3 \), we say the existential player loses and the payoff is \(+\infty\). Therefore, it is the existential player’s primary goal to ensure the fulfillment of the constraint system, while the universal player tries to violate some constraints. If
A second linear constraint system

\[ A^\circ x \in L. \]

Let desirable or too conservative, i.e. the price of robustness framework provided by QIPs the modeler must ensure that the modeled worst case is not unselected cautiously as their influence might become too powerful. In the worst-case optimization In optimization under uncertainty, uncertain events and variables must be modeled and selected cautiously as their influence might become too powerful. In the worst-case optimization framework provided by QIPs the modeler must ensure that the modeled worst case is not undesirable or too conservative, i.e. the price of robustness must be appropriate [14]. In [42] a second linear constraint system \( A^\circ x \leq b^\circ \) was introduced that restricts the universal variables to a polytope resulting in the QIP with polyhedral uncertainty.

**Definition 3.2 (QIP with Polyhedral Uncertainty (QIP\(_{PU}\)).**

Let \( m_\nu \in \mathbb{N}_0, b^\nu \in \mathbb{Q}^{m_\nu} \) and \( A^\nu \in \mathbb{Q}^{m_\nu \times n} \) with

\[
A^\nu_{k,j} = 0 \quad \forall \, k \in [m_\nu], \, j \in [n] : Q_j = \emptyset. \tag{2}
\]

Let \( D = \{ x \in L \mid A^\nu x \leq b^\nu \} \neq \emptyset \). The term \( Q \circ x \in D \) with the component-wise binding operator \( \circ \) denotes the quantification sequence \( Q^{(1)} x^{(1)} \in D^{(1)} Q^{(2)} x^{(2)} \in D^{(2)}(x^{(1)}) \ldots Q^{(\beta)} x^{(\beta)} \in D^{(\beta)}(x^{(1)}, \ldots, x^{(\beta-1)}) \) such that every quantifier \( Q^{(i)} \) binds the variables \( x^{(i)} \) of block \( i \) ranging in their domain \( D^{(i)}(x^{(1)}, \ldots, x^{(i-1)}) \), with

\[
D^{(i)}(\tilde{x}^{(1)}, \ldots, \tilde{x}^{(i-1)}) = \begin{cases} L^{(i)} & \text{if } i \in \mathcal{E} \\ \{ \mathbf{y} \in L^{(i)} \mid \exists \mathbf{x} = (\tilde{x}^{(1)}, \ldots, \tilde{x}^{(i-1)}, \mathbf{y}, x^{(i+1)}, \ldots, x^{(\beta)}) \in D \} & \text{if } i \in \mathcal{A}. \end{cases}
\]

We call

\[
z = \min_{x^{(1)} \in D^{(1)}} \left( c^{(1)} x^{(1)} + \max_{x^{(2)} \in D^{(2)}} \left( c^{(2)} x^{(2)} + \min_{x^{(3)} \in D^{(3)}} \left( c^{(3)} x^{(3)} + \ldots \min_{x^{(\beta)} \in D^{(\beta)}} (c^{(\beta)} x^{(\beta)}) \right) \right) \right)
\]

s.t. \( Q \circ x \in D : A^3 x \leq b^3 \) \tag{3}

a QIP with polyhedral uncertainty (QIP\(_{PU}\)) given by the tuple \( (A^3, A^\nu, b^3, b^\nu, c, L, Q) \).

With Condition \([3]\) each entry of \( A^\nu \) belonging to an existential variable is zero. Therefore, the universal constraint system \( A^\nu x \leq b^\nu \) restricts universal variables in such way that their range only depends on previous universal variables: when assigning a value to the universal variable \( x_i \), there must exist a series of future assignments for \( x_{i+1}, \ldots, x_n \) such that the resulting vector \( x \) fulfills \( A^\nu x \leq b^\nu \). This means that a universal variable assignment must not make it impossible to satisfy the system \( A^\nu x \leq b^\nu \). With \( D \neq \emptyset \) at least one fixation of universal variables fulfills the universal constraint system and therefore a universal strategy to fulfill \( A^\nu x \leq b^\nu \) exists. This is similar to demanding a non-empty polyhedral uncertainty set, which is a common condition in robust optimization.
3.3 The Open-Source QIP Solver Yasol

The open-source solver Yasol\(^1\) is a search-based solver for QIPs [27]. There are no other general QIP solvers that we know of. The heart of the search algorithm is an arithmetic linear constraint database together with an alpha-beta algorithm, which has been successfully used in gaming programs, e.g. chess programs for many years [47, 26]. In order to realize fast backjumps—as typically performed in SAT- and QBF-solvers (e.g. [36, 20])—the alpha-beta algorithm was extended as outlined in [27]. Yasol deals with constraint learning on the so-called primal side as known from SAT- and QBF-solving (e.g. [56, 37]), as well as with constraint learning on the dual side known from MIP (e.g. [17]). Several other techniques from various research fields are implemented, e.g. the killer heuristic [3], restart strategies [16] and strong branching [1]. Yasol is currently able to solve multistage quantified mixed integer programs with the following properties: a) The basic structure must be a quantified program, i.e. linear (existential and universal) constraints and objective function, existentially or universally quantified variables, all variables are bounded from below and above and \(Q_1 = Q_n = 3\). b) Integer variables are allowed in all existential and universal variable blocks. c) Continuous variables are allowed only in the last closing stage. Due to the exponential size of a solution (strategy) the output of a feasible instance is the optimal assignment of the first (existential) variable block, as well as the optimal worst-case outcome, i.e. the value of the optimal strategy.

Yasol makes intensive use of a linear programming solver in order to assess the quality of a branching variable (e.g. [1]), the satisfiability of the existential constraint system in the current subtree or for the generation of bounds. These tools are black-box used, while not exploiting the possible integer solving abilities of the foreign solver.

4 Robust Multistage Problems

4.1 Robust Multistage Selection

4.1.1 Problem Formulations

In this section we examine the selection problem

\[
\min_{\mathbf{x} \in \mathcal{X}} \sum_{i \in [n]} c_i x_i
\]

with \(\mathcal{X} = \{ \mathbf{x} \in \{0, 1\}^n : \sum_{i \in [n]} x_i = p \} \), where \(p\) out of \(n\) items must be selected, such that the costs are minimized. We first recall the one-stage robust counterpart to this problem. We assume that uncertainty is only present in the objective, and a discrete list of \(N \in \mathbb{N}\) potential cost vectors is given. A thorough overview of the one-stage robust selection problem can be found in [19]. It is given by

\[
\min \max_{\mathbf{c} \in \mathcal{U}} \mathbf{c} \mathbf{x}
\]

with \(\mathcal{U} = \{ \mathbf{c}_1, \ldots, \mathbf{c}_N \}\) being the set of anticipated scenarios. Let \(c_{i,k} \in \mathbb{R}_+\) be the cost for item \(i \in [n]\) in scenario \(k \in [N]\) and let \(x_i\) be the variable indicating the selection of item \(i\). A mixed-integer program for the robust counterpart is then given as follows:

\[
\min z
\]

\(^1\)Sources and further information regarding the solver can be found on http://www.q-mip.org (accessed May 3, 2020).
\[ \sum_{i \in [n]} c_{i,k} x_i \leq z \quad \forall k \in [N] \]

\[ \sum_{i \in [n]} x_i = p \]

\[ x_i \in \{0,1\} \quad \forall i \in [n] \]

This can be formulated as a QIP\textsuperscript{PU} by introducing universal variables \( q_k \) that indicate whether cost scenario \( k \) is selected. As only one scenario can occur, the universal constraint \( \sum_{k \in [N]} q_k = 1 \) is used. Therefore, the universal variable domain \( D \) is given by

\[ D = \left\{ q \in \{0,1\}^N \mid \sum_{k \in [N]} q_k = 1 \right\}. \]

A first straightforward attempt to model the objective function results in the nonlinear expression \( \sum_{i \in [n]} \sum_{k \in [N]} q_k (c_{i,k} x_i) \). This nonlinearity is avoided by using the auxiliary variable \( z \), which bundles the costs, and Constraint (4c), which connects the selected scenario to the resulting costs using the Big-M method. The entire QIP\textsuperscript{PU} model for the robust selection problem is given as follows:

\[
\begin{align*}
\text{min} & \quad z \\
\text{s.t.} & \quad \exists \ x \in \{0,1\}^{\mathbb{N}} \quad \forall q \in D \quad \exists z \in \mathbb{R}_+: \\
& \quad \sum_{i \in [n]} x_i = p \\
& \quad \sum_{i \in [n]} c_{i,k} x_i \leq z + M_k (1 - q_k) \quad \forall k \in [N]
\end{align*}
\]

If \( M_k \) is selected appropriately for each potential scenario \( k \) (e.g. \( M_k \geq \sum_{i \in [n]} c_{i,k} \)), all but one of the Constraints (4c) are trivially fulfilled for a realization of \( q \in D \): if scenario \( k \) is selected by the universal player \( (q_k = 1) \) the corresponding costs \( c_{.,k} \) are decisive for the cost calculation.

The presented selection problem can be adapted to a multistage decision problem as follows. In the first (existential) decision stage a set of items can be selected for fixed costs \( c^0 \). Then, in a universal decision stage, a cost scenario is revealed and in the subsequent existential decision stage further items can be selected. Those two stages can be repeated iteratively several times. If an item is selected, it cannot be selected again in a later stage and the goal remains to select \( p \) items such that the resulting costs are minimized.

Note that adversary stages are not usually counted in robust optimization; this is different to quantified programming. What we consider a one-stage robust problem has \( \beta = 3 \) stages as a QIP (see the notation in Section 3). Let \( S \in \mathbb{N} \) be the number of universal decision stages. We refer to \( S \) as the number of iterations. The universal domain for each iteration \( s \in [S] \) is given by

\[ D_s = \left\{ q^s \in \{0,1\}^N \mid \sum_{k \in [N]} q^s_k = 1 \right\}. \]

and \( q^s \in D_s \) is the vector indicating the selected scenario. As before, let \( N \) be the number of scenarios per iteration, i.e. at each iteration, one of \( N \) scenarios is revealed. The cost of
item $i$ in scenario $k$ of iteration $s$ are given by $c_{i,k}^s$. This multistage selection problem under uncertainty can be modeled as a quantified program with a polyhedral uncertainty set as follows:

$$\text{(SELQ}^{\text{PU}}\text{)} \quad \min \sum_{i \in [n]} c_i^0 x_i^0 + \sum_{s \in [S]} z_s$$

s.t. $\exists x^0 \in \{0,1\}^n \quad \forall q^1 \in D_1 \quad \exists x^1 \in \{0,1\}^n \quad \forall q^2 \in D_2 \quad \cdots \quad \forall q^S \in D_S \quad \exists x^S \in \{0,1\}^n \exists z \in \mathbb{R}^S$: 

$$\sum_{i \in [n]} \sum_{s=0}^S x_i^s = p$$

$$\sum_{s=0}^S x_i^s \leq 1 \quad \forall i \in [n]$$

$$\sum_{i \in [n]} c_{i,k}^s x_i^s \leq z_s + M_k^s (1 - q_k^s) \quad \forall k \in [N], s \in [S]$$

The Objective (5a) consists of the expenses from the first stage with invariable costs and the expenses of subsequent iterations in which the cost for each item depends on the selected scenario. Note that we omit the min/max alternation in the objective and only specify the optimization orientation for the existential variables. The first Constraint (5b) demands that overall exactly $p$ items must be selected. Constraint (5c) prevents that an item is selected more than once. Constraint (5d) enforces the link between the selected scenario, selected items and resulting costs in each iteration $s$. As Yasol can only deal with (existential) continuous variables in the last variable block, we put the $z$ variables at the end of the quantification sequence. Here, however, the cost variables $z_s$ also could be placed immediately after the corresponding selection in iteration $s$. Additionally, when explicitly stating the model, one has to specify an upper bound on $z_s$, which can be easily computed by taking the cost vectors of the corresponding scenarios into account.

In order to build an equivalent QIP, i.e., a model without constraints on the universal variables, we use an integer variable $\ell_s \in [N]$ in order to select one of $N$ scenarios in each iteration. This integer can then be transformed into existential indicator variables, resulting in the following problem:

$$\text{(SELQ)} \quad \min \sum_{i \in [n]} c_i^0 x_i^0 + \sum_{s \in [S]} z_s$$

s.t. $\exists x^0 \in \{0,1\}^n \quad \forall \ell_1 \in [N] \quad \exists q^1 \in \{0,1\}^N \quad \exists x^1 \in \{0,1\}^n \quad \cdots \quad \forall \ell_S \in [N] \quad \exists q^S \in \{0,1\}^N \quad \exists x^S \in \{0,1\}^n \exists z \in \mathbb{R}^S$: 

$$\sum_{i \in [n]} \sum_{s=0}^S x_i^s = p$$

$$\sum_{s=0}^S x_i^s \leq 1 \quad \forall i \in [n]$$

$$\sum_{i \in [n]} c_{i,k}^s x_i^s \leq z_s + M_k^s (1 - q_k^s) \quad \forall k \in [N], s \in [S]$$
\[
\sum_{k \in [N]} q^*_k = 1 \quad \forall s \in [S] \tag{6e}
\]
\[
\sum_{k \in [N]} k \cdot q^*_k = \ell_s \quad \forall s \in [S] \tag{6f}
\]

The variables \( q^* \), which were universal variables in \( \text{SELQ}^\text{PU} \), are now used as existential variables indicating the selected scenario. Constraints (6e) and (6f) ensure that the scenario number \( \ell_s \) selected by the universal player is transformed into a corresponding assignment of \( q^* \). Thus, the number of variables and constraints in \( \text{SELQ} \) increased compared to \( \text{SELQ}^\text{PU} \).

We also provide an equivalent deterministic program (DEP), i.e. an equivalent MIP, for which each possible scenario sequence must be listed explicitly. The set containing all possible sequences of scenarios is \( \mathcal{R} = [N]^S \). For one such sequence \( r \in \mathcal{R} \) the scenario in iteration \( s \) is \( r_s \). The entire sub-sequence up to iteration \( s \) is denoted by \( r(s) \in [N]^s \). For each item \( i \in [n] \), iteration \( s \in [S] \) and sequence \( r \in \mathcal{R} \), the variable \( x^r_i \) indicates the decision of selecting item \( i \) in iteration \( s \) after the sub-sequence \( r(s) \) of \( r \) occurred. This ensures the nonanticipativity property: even for different scenario sequences the selection decisions must be the same, as long as the sub-sequences are identical.

**Example 4.1.** For \( N = 4 \) and \( S = 6 \) a possible sequence of scenarios is \( r = (1, 4, 2, 3, 1, 1) \). The sub-sequence until iteration \( s = 4 \) is \( r(4) = (1, 4, 2, 3) \). The variable indicating whether item \( i \) is selected after 4 iterations and the occurrence of this particular sub-sequence is denoted \( x^r_i(4) = x^r_i(1,4,2,3) \) and the scenario in iteration \( s = 4 \) for this sequence is \( r_4 = 3 \). The cost of item \( i \) in iteration \( s = 4 \) does not depend on the entire sequence, but only on the occurred scenario and is given by \( c^r_i = c^r_i \). For the sequence of scenarios \( \hat{r} = (1, 4, 2, 3, 2, 4) \) it holds \( r(4) = \hat{r}(4) \) and therefore, the variables \( x^r_i(4) \) and \( x^\hat{r}_i(4) \) are the same.

The DEP of \( \text{SELQ}^\text{PU} \) represents the robust counterpart to the problem and is given as follows.

\[
\text{(SELRC)} \quad \min \sum_{i \in [n]} c^0_i x_i^0 + z \tag{7a}
\]
\[
\text{s.t.} \quad z \geq \sum_{i \in [n]} \sum_{s \in [S]} c^s_i r_s x_i^r(s) \quad \forall r \in \mathcal{R} \tag{7b}
\]
\[
\sum_{i \in [n]} \left( x_i^0 + \sum_{s \in [S]} x_i^r(s) \right) = p \quad \forall r \in \mathcal{R} \tag{7c}
\]
\[
x_i^0 + \sum_{s \in [S]} x_i^r(s) \leq 1 \quad \forall i \in [n], r \in \mathcal{R} \tag{7d}
\]
\[
x_i^0 \in \{0, 1\} \quad \forall i \in [n] \tag{7e}
\]
\[
x_i^r(s) \in \{0, 1\} \quad \forall i \in [n], r \in \mathcal{R}, s \in [S] \tag{7f}
\]
\[
z \in \mathbb{R} \tag{7g}
\]

Constraint (7b) ensures that the expenses from the worst-case scenario sequence appear in the objective function. Constraint (7c) ensures for each scenario sequence that exactly \( p \) items are selected in the end, whereas Constraint (7d) ensures that each item is selected at most once.
4.1.2 Heuristics

The question arises to what extent the exact solution of robust multistage problems is superior to applying heuristics and whether there are simple online decision strategies that come close to an optimal solution. Therefore, we present three online decision strategies as comparator methods in order to be able to grasp the relevance of the optimization model.

Strategy 1: Buy All Now. The easiest strategy is to neglect any knowledge of future events and buy the $p$ cheapest items right away. The resulting costs of this strategy in terms of the presented models are

$$\min \left\{ \sum_{i \in [n]} c_i^0 x_i^0 \mid \sum_{i \in [n]} x_i^0 = p, \ x^0 \in \{0,1\}^n \right\}.$$ 

Since knowledge of future iterations and scenarios is not taken into account, this trivial strategy almost always leads to significantly sub-optimal results.

Strategy 2: Buy Now, If Never Cheaper in Worst Case. In this decision strategy partial knowledge of future scenarios is incorporated: Items are sorted according to their lowest guaranteed costs incurred in the current or in future iterations. Starting with the cheapest, an item is bought if its best price is the current price. Let $\mathcal{P}$ be the set of already bought items. Let $s \in \{0,\ldots,S\}$ be the current iteration and $k \in [N]$ the current scenario (if $s > 0$). In such a situation we propose to look at the $p - |\mathcal{P}|$ cheapest items according to their best worst-case price and buy such an item now, if there is no future iteration in which this item is cheaper in the worst case as shown in Algorithm 1.

By applying this strategy, obviously detrimental purchases are prevented, i.e. if it is guaranteed that the same item is available later for a cheaper price. However, other items are not considered.

Strategy 3: Don’t Buy, If Others Will Be Cheaper. Similar to Strategy 2, a buying decision depends on the worst-case costs in future scenarios, but incorporates prices of all remaining items. Let $\langle a \rangle_b$ be the function returning the value of the $b$-smallest element of vector $a$. Let $a(N) \in \mathbb{R}^{|N|}$ be the entries of vector $a \in \mathbb{R}^p$ corresponding to indices in set $N \subseteq [n]$. As before, let $s \in \{0,\ldots,S\}$ be the current iteration, $k \in [N]$ the current scenario (if $s > 0$) and $\mathcal{P}$ the set of already bought items. In such a situation we propose to look at the $p - |\mathcal{P}|$ cheapest items according to $c^*_k$ and buy the cheapest item as long as no future iteration is found in which $p - |\mathcal{P}|$ items are cheaper, even in the worst case. In Algorithm 2 this strategy is presented. This way it is prevented that by buying an item now an obviously cheaper selection in the future is no longer possible. In particular, if only a single item remains to be bought it is checked whether there is a guaranteed cheaper item in a later iteration.

Example 4.2. Let $n = 6$, $p = 3$, $S = 2$ and $N = 2$. The costs in the initial stage and each scenario is given in Table 1.

a) Strategy 1: “Buy All Now”

Buying the three cheapest items in the first stage yields costs of 90.
Algorithm 1: Selection strategy 2: “Buy Now, If Never Cheaper in Worst Case”.

Data: target \( p \), current iteration \( s \), current scenario \( k \), costs \( c \), set of bought items \( P \)

1: \hspace{1em} for each \( i \in [n] \) do
2: \hspace{2em} \( b_i = c_{i,k}^s \) // best price of item \( i \) initialized to current price
3: \hspace{2em} \( d_i = “Buy Now” \) // initialize decision for item \( i \)
4: \hspace{1em} if \( i \in P \) then
5: \hspace{2em} \( b_i = \infty \)
6: \hspace{2em} \( d_i = “Already bought” \)
7: \hspace{2em} continue loop with \( i = i + 1 \)
8: \hspace{1em} end if
9: \hspace{1em} for each \( s > s \) do
10: \hspace{2em} if \( b_i \geq \max_{k \in [N]} c_{i,k}^{s/|P|} \) then
11: \hspace{3em} \( b_i = \max_{k \in [N]} c_{i,k}^{s/|P|} \)
12: \hspace{3em} \( d_i = “Buy Later” \)
13: \hspace{2em} end if
14: \hspace{1em} end for
15: \hspace{1em} end for
16: \hspace{1em} Sort \( b \) and \( d \) in ascending order according to the values in \( b \)
17: \hspace{1em} for \( i = 1 \) to \( p - |P| \) do
18: \hspace{2em} if \( d_i = “Buy Now” \) then
19: \hspace{3em} \( P = P \cup \{i\} \) // the cheapest items according to \( b \)
20: \hspace{2em} end if
21: \hspace{1em} end for
22: \hspace{1em} return \( P \)

Algorithm 2: Selection strategy 3: “Don’t Buy, If Others Will Be Cheaper”.

Data: target \( p \), current iteration \( s \), current scenario \( k \), costs \( c \), set of bought items \( P \)

1: \hspace{1em} Remove the items in \( P \) from each cost vector
2: \hspace{1em} Resort the not yet selected items according to the current costs \( c_{s,k}^s \)
3: \hspace{1em} for \( i = 1 \) to \( p - |P| \) do
4: \hspace{2em} \( N = [n] \setminus P \)
5: \hspace{2em} if \( c_{i,k}^s < \min_{s \leq s \leq s \in [N]} \max_{s \in [N]} c_{i,k}^s \) then
6: \hspace{3em} \( P = P \cup \{i\} \) // the \( p - |P| \) cheapest items according to \( c_{s,k}^s \)
7: \hspace{2em} else
8: \hspace{3em} return \( P \)
9: \hspace{2em} end if
10: \hspace{2em} end for

b) Strategy 2: “Buy Now, If Never Cheaper in Worst Case”

In the first decision stage the vector \( b \)—containing the best worst-case costs of each item—is filled with the values \((45, 14, 29, 32, 31, 45)\). The three smallest values are examined resulting in the decision of buying items 2 and 5 now. Item 3 is not bought in this iteration, as a better price is guaranteed later on. If in iteration 1 scenario 1 occurs, the vector \( b \) holds the values \((40, \infty, 29, 32, \infty, 50)\). Since only one item must be bought
Table 1: Cost scenarios for an instance of the multistage selection problem.

| i | 1  | 2  | 3  | 4  | 5  | 6  |
|---|----|----|----|----|----|----|
| $c_i^0$ | 84 | 14 | 76 | 61 | 31 | 45 |
| $c_{i,1}^1$ | 40 | 24 | 29 | 41 | 90 | 71 |
| $c_{i,2}^1$ | 45 | 30 | 15 | 18 | 44 | 44 |
| $c_{i,1}^2$ | 13 | 25 | 12 | 11 | 75 | 50 |
| $c_{i,2}^2$ | 80 | 10 | 29 | 32 | 64 | 30 |

To reach $p = 3$ only the cheapest item is considered, but again item 3 is not bought, as later on the same price is ensured. If scenario 2 occurs in iteration 1, item 3 would be bought. The overall worst-case costs when this strategy is applied is 74.

c) Strategy 3: “Don’t Buy, If Others Will Be Cheaper”

In the first decision stage items 2, 5 and 6 are considered. Item 2 costs 14. Note that if the first scenario of iteration 2 occurs, buying item 2 right away would be bad as there would be three cheaper items. However, this is the best cast scenario. Hence, the question is whether buying item 2 now eliminates the option of buying three cheaper items in a single future iteration, even in the worst case. Therefore, the worst-case third cheapest cost of each iteration is calculated, which are 40 in iteration 1 and 30 in iteration 2. Since both values are larger than 14 item 2 is bought in the first stage. For item 5 this procedure is repeated, but now the second cheapest of the remaining items are considered. Those are 40 in iteration 1 (since item 2 is excluded) and 30 in iteration 2. Therefore, item 5 is not bought, as it is ensured, that in another future scenario two cheaper items exist. If in iteration 1 scenario 1 occurs, items 3 and 1 are considered and only item 3 is bought. If scenario 2 occurs in iteration 1, both items 3 and 4 are bought. The overall worst-case costs when this strategy is applied is 73.

In the optimal strategy no item is bought in the first stage and the overall worst-case costs are 69. The explicit strategies of the three heuristics, as well as the optimal solution can be found in Appendix A.

4.2 Robust Multistage Assignment

Another frequently considered combinatorial problem is the assignment problem: Given a complete bipartite graph $G = (V, E)$ with $V = A \cup B$, $n = |A| = |B|$. A cost value $c_{i,j} \in \mathbb{R}_+$ is associated with each edge $(i, j) \in E$. The assignment problem consists of determining a perfect matching of minimum costs. Research on the min-max and min-max regret assignment problems can be found in [2] and further complexity results are obtained in [23]. We want to adapt the robust approach to a multistage setting.

In a first decision stage the existential player can select edges with known costs. Then, iteratively, new costs of the edges are presented (by the universal player) which then can be selected (by the existential player). Similar to the preceding subsection, the costs selected by the universal player come from a predefined scenario pool. Let $N$ be the number of scenarios and $S$ the number of iterations. We use the universal variable $q^s_k$ to indicate whether cost scenario $k$ is selected in iteration $s$. As only one scenario can occur at each iteration the
universal constraint $\sum_{k \in [N]} q_k^s = 1$ must be fulfilled and thus at each iteration $s \in [S]$ the universal variables have to obey the domain

$$D_s = \left\{ q^s \in \{0,1\}^N \mid \sum_{k \in [N]} q_k^s = 1 \right\}.$$ 

The cost for edge $(i,j) \in E$ in scenario $k$ and iteration $s$ is given by $c_{i,j,k}^s \in \mathbb{R}_+$. As before, auxiliary variables $z_s$ are used to bundle the costs incurred in iteration $s$ and to avoid a nonlinear term in the objective function. The QIP$^\text{PU}$ model for this multistage assignment problem is given below.

\[
\begin{align*}
\text{min} \quad & \sum_{i \in [n]} \sum_{j \in [n]} c_{i,j}^0 x_{i,j}^0 + \sum_{s \in [S]} z_s \quad \text{(ASSQ$^\text{PU}$)} \\
\text{s.t.} \quad & \exists x^0 \in \{0,1\}^{n \times n} \quad \forall q^1 \in D_1, \quad \exists x^1 \in \{0,1\}^{n \times n} \ldots \quad \forall q^S \in D_S, \quad \exists x^S \in \{0,1\}^{n \times n} \quad \exists z \in \mathbb{R}_+^S : \\
& \sum_{j \in [n]} x_{i,j}^s = 1 \quad \forall i \in [n] \quad \text{(8b)} \\
& \sum_{i \in [n]} x_{i,j}^s = 1 \quad \forall j \in [n] \quad \text{(8c)} \\
& \sum_{i \in [n]} \sum_{j \in [n]} c_{i,j,k}^s x_{i,j}^s \leq z_s + M_k^s (1 - q_k^s) \quad \forall k \in [N], \ s \in [S] \quad \text{(8d)}
\end{align*}
\]

The Objective (8a) consists of the expenses from the first stage with fixed costs and each iteration with uncertain costs. Constraints (8b) and (8c) ensure that the found solution is indeed a perfect matching. Constraint (8d) linearizes the dependence between selected scenario and incurred costs. Similar to the multistage selection problem, in order to build an equivalent QIP we represent the universal player’s decision as an integer variable $\ell_s \in [N]$ and then convert it into existential indicator variables $q^s \in \{0,1\}^N$. We then define the following problem.

\[
\begin{align*}
\text{min} \quad & \sum_{i \in [n]} \sum_{j \in [n]} c_{i,j}^0 x_{i,j}^0 + \sum_{s \in [S]} z_s \quad \text{(9a)} \\
\text{s.t.} \quad & \exists x^0 \in \{0,1\}^{n \times n} \quad \forall \ell_1 \in [N], \quad \exists q^1 \in \{0,1\}^N \quad \exists x^1 \in \{0,1\}^{n \times n} \ldots \quad \forall \ell_S \in [N], \quad \exists q^S \in \{0,1\}^N \quad \exists z \in \mathbb{R}_+^S : \\
& \sum_{j \in [n]} x_{i,j}^s = 1 \quad \forall i \in [n] \quad \text{(9b)} \\
& \sum_{i \in [n]} x_{i,j}^s = 1 \quad \forall j \in [n] \quad \text{(9c)} \\
& \sum_{i \in [n]} \sum_{j \in [n]} c_{i,j,k}^s x_{i,j}^s \leq z_s + M_k^s (1 - q_k^s) \quad \forall k \in [N], \ s \in [S] \quad \text{(9d)} \\
& \sum_{k \in [N]} q_k^s = 1 \quad \forall s \in [S] \quad \text{(9e)}
\end{align*}
\]
\[
\sum_{k \in [N]} k \cdot q^s_k = \ell_s \quad \forall s \in [S] \tag{9f}
\]

As before, we are interested in a robust counterpart that can be solved using standard MIP solvers. Similar to the notation used in the previous subsection, let \( \mathcal{R} \) denote the set of all possible sequences of scenarios and let \( \mathbf{r}(s) \) denote the sub-sequence of the scenario sequence \( \mathbf{r} \) up to iteration \( s \). The variable \( x^\mathbf{r}(s)_{i,j} \) indicates the decision of selecting edge \((i, j)\) in iteration \( s \) after the sub-sequence \( \mathbf{r}(s) \) of \( \mathbf{r} \) occurred. The robust counterpart of ASSQ\textsubscript{PU} is given below.

\[
\text{(ASSRC)} \quad \min \sum_{i \in [n]} \sum_{j \in [n]} c^0_{i,j} x^0_{i,j} + z 
\]

s.t. 
\[
z \geq \sum_{i \in [n]} \sum_{j \in [n]} \sum_{s \in [S]} c^s_{i,j,r} x^\mathbf{r}(s)_{i,j} \quad \forall \mathbf{r} \in \mathcal{R} \tag{10b}
\]

\[
\sum_{j \in [n]} \sum_{s=0}^{S} x^\mathbf{r}(s)_{i,j} = 1 \quad \forall i \in [n], \mathbf{r} \in \mathcal{R} \tag{10c}
\]

\[
\sum_{i \in [n]} \sum_{s=0}^{S} x^\mathbf{r}(s)_{i,j} = 1 \quad \forall j \in [n], \mathbf{r} \in \mathcal{R} \tag{10d}
\]

\[
x^0_{i,j} \in \{0, 1\} \quad \forall i, j \in [n] \tag{10e}
\]

\[
x^\mathbf{r}(s)_{i,j} \in \{0, 1\} \quad \forall i, j \in [n], \mathbf{r} \in \mathcal{R}, s \in [S] \tag{10f}
\]

\[
z \in \mathbb{R} \tag{10g}
\]

5 Experiments on Robust Multistage Selection

We conduct experiments on robust multistage selection and robust multistage assignment problems. The aim of these experiments is to answer the questions: To what size can the approaches presented in this paper solve multistage problems to optimality? Which of the exact approaches performs best? And how do the heuristic policies compare to exact solutions? We present a detailed discussion of robust multistage selection problems in the following. Results on robust multistage assignment are provided in Appendix B.

5.1 Setup

We investigate the multistage selection problem as introduced in Section 4.1 in which \( p \) out of \( n \) items must be selected. We compare the performance of Yasol on the quantified models SELQ\textsubscript{PU} and SELQ with the performance of CPLEX on the robust counterpart SELRC. An instance is given by the number of available items \( n \), the number of items \( p \) to be selected, the number of iterations \( S \) and the number of scenarios \( N \) per iteration. Recall that \( S \) denotes the number of max min blocks after the first min. Hence, \( S = 1 \) corresponds to two-stage robust optimization. We limit experiments to instances with \( n = 2p \) and thus, the value of \( p \) is omitted from now on. The remaining parameters of an instance are the costs \( c^s_{i,k} \) of each each item \( i \) in scenario \( k \) of iteration \( s \), which are randomly selected from the range \( 0, 1, \ldots, 99 \). Those values are created using the C++ function \texttt{rand()} from the standard general utilities library and the modulo operator.
We use CPLEX (12.9.0) as MIP solver in order to solve the robust counterpart. Yasol uses CPLEX (12.6.1) as its black-box LP solver. Since Yasol currently only uses a single thread we also restricted CPLEX to a single thread in order to obtain a balanced comparison (CPLEX turned out to be even faster if restricted to a single thread). All experiments were run on an Intel(R) Core(TM) i7-4790 with 3.60 GHz and 32 GB RAM.

5.2 Results

5.2.1 Memory Usage

When creating multistage selection instances, the difference in the file sizes of the three presented models is noticeable. For example, for an instance with \( n = 10, S = 5 \) and \( N = 32 \), model SELQ\textsuperscript{PU} requires about 28 KB, model SELQ requires 36 KB and the CPLEX LP file for the robust counterpart SELRC requires more than 91 GB. Thus, solving the robust counterpart of such instances may already fail when trying to import the model file into the solver.

In addition to the compact problem description of the SELQ\textsuperscript{PU} and SELQ models, Yasol does not excessively make use of memory during the search process. The same cannot be said about CPLEX when solving SELRC. This is partially due to the high number of variables and constraints in the robust counterpart, but also due to the fact that Yasol does not explicitly store the search tree. In Figure 1 the average maximum RAM used during the entire search process is depicted. Similar to the increasing size of the instance itself, the memory usage of CPLEX solving the robust counterpart increases dramatically with the number of iterations and scenarios (note the logarithmic vertical axis), while the memory usage of Yasol only slightly increases. In the plot, lines for different values of \( n \) are not distinguishable when using Yasol. The memory usage when solving SELQ\textsuperscript{PU} is slightly higher compared to the solution process for SELQ. This is partially due to the overhead of having to maintain the universal constraint system \( A^T x \leq b^T \).

![Figure 1: Average of the maximum RAM required during the solution process (time limit 1800 seconds) of the different models for N = 4. 50 instances for each S ∈ \{3, \ldots, 8\} and n ∈ \{10, 20, \ldots, 50\}.

---

15
5.2.2 Computation Times for Fixed $N$

We now fix the number of scenarios in each iteration to $N = 4$ and vary the number of items $n \in \{10, 20, 30, 40, 50\}$ and the number of iterations $S \in \{1, \ldots, 8\}$. For each setting 50 instances of each of the three models SELQ$^{PU}$, SELQ and SELRC are created. The quantified programs are solved using Yasol and the robust counterpart is solved using CPLEX with a maximum time limit of 1800 seconds. We are interested in the number of instances solved within the time limit and the runtimes. In Table 2 the number of solved instances for each setting is displayed. As expected, the number of solved instances within the time limit tends to decrease for increasing $n$ and $S$. With the exception of $S = 6$, $n \geq 40$ we can observe that a) the number of solved robust counterparts is never higher than the number of solved quantified programs and b) the number of solved instances with universal constraints (SELQ$^{PU}$) is always the highest. The number of solved robust counterparts in particular tends to decrease significantly faster for an increasing $S$. In most cases all 50 instances are solved and thus the average runtimes of those instances for which all three models are solved are shown in Table 3.

Table 2: Number of solved multistage selection instances for fixed number of scenarios $N = 4$ and various $n$ and $S$ within 1800 seconds.

| model       | $S = 1$ | $S = 2$ | $S = 3$ | $S = 4$ | $S = 5$ | $S = 6$ | $S = 7$ | $S = 8$ |
|-------------|---------|---------|---------|---------|---------|---------|---------|---------|
| $n = 10$    |         |         |         |         |         |         |         |         |
| SELRC       | 50      | 50      | 50      | 50      | 50      | 50      | 49      | 30      |
| SELQ        | 50      | 50      | 50      | 50      | 50      | 50      | 50      | 50      |
| SELQ$^{PU}$ | 50      | 50      | 50      | 50      | 50      | 50      | 50      | 50      |
| $n = 20$    |         |         |         |         |         |         |         |         |
| SELRC       | 50      | 50      | 50      | 50      | 50      | 50      | 32      | 0       |
| SELQ        | 50      | 50      | 50      | 50      | 50      | 50      | 50      | 50      |
| SELQ$^{PU}$ | 50      | 50      | 50      | 50      | 50      | 50      | 50      | 50      |
| $n = 30$    |         |         |         |         |         |         |         |         |
| SELRC       | 50      | 50      | 50      | 50      | 50      | 48      | 21      | 0       |
| SELQ        | 50      | 50      | 50      | 50      | 50      | 50      | 50      | 49      |
| SELQ$^{PU}$ | 50      | 50      | 50      | 50      | 50      | 50      | 50      | 50      |
| $n = 40$    |         |         |         |         |         |         |         |         |
| SELRC       | 50      | 50      | 50      | 50      | 50      | 50      | 15      | 0       |
| SELQ        | 50      | 50      | 50      | 50      | 50      | 50      | 50      | 49      |
| SELQ$^{PU}$ | 50      | 50      | 50      | 50      | 50      | 49      | 49      | 41      |
| $n = 50$    |         |         |         |         |         |         |         |         |
| SELRC       | 50      | 50      | 50      | 50      | 50      | 49      | 7       | 0       |
| SELQ        | 50      | 50      | 50      | 50      | 50      | 50      | 46      | 31      |
| SELQ$^{PU}$ | 50      | 50      | 50      | 50      | 48      | 33      | 20      |         |

The values marked with an asterisk (*) are less significant as they are based on less than 20 instances for which all three models were solved to optimality. All runtimes have been rounded to seconds before taking the average. For most settings the average runtime of the quantified program with universal constraints SELQ$^{PU}$ is lower than for the pure QIP SELQ. For fixed $S$ and increasing number of items $n$ the runtime of the robust counterpart does not grow as quick as the runtime of the quantified programs. In particular, for $S = 6$ SELQ and SELQ$^{PU}$ are solved faster on average up to $n = 40$. For $n = 50$ CPLEX can display its strength and the (average) runtime even decreases compared to the average runtime for instances with $n = 40$. This decrease, however, is owed to the only 44 instances for which all three models with $n = 50$ are solved. Note that for $S = 6$ and $n = 50$ the SELRC model already has more than 250,000 variables and 200,000 constraints, while SELQ$^{PU}$ needs 380
Table 3: Average runtime (in seconds) of multistage selection instances with $N = 4$ for which each model was solved. Values marked with an asterisk resulted from less than 20 solved instances. A hyphen indicates that for no instance all three models were solved.

| model   | $S = 1$ | $S = 2$ | $S = 3$ | $S = 4$ | $S = 5$ | $S = 6$ | $S = 7$ | $S = 8$ |
|---------|---------|---------|---------|---------|---------|---------|---------|---------|
| n = 10  |         |         |         |         |         |         |         |         |
| SELRC   | 0.0     | 0.0     | 0.0     | 0.0     | 1.5     | 13.3    | 186.6   | 1016.7  |
| SELQ    | 0.1     | 0.1     | 0.2     | 0.3     | 0.7     | 2.6     | 7.5     | 32.1    |
| SELQ$^{PU}$ | 0.1 | 0.1 | 0.1 | 0.2 | 0.6 | 1.2 | 4.2 | 9.7 |
| n = 20  |         |         |         |         |         |         |         |         |
| SELRC   | 0.0     | 0.0     | 0.0     | 0.7     | 4.3     | 54.7    | 587.5   | -       |
| SELQ    | 0.1     | 0.2     | 0.7     | 1.4     | 5.5     | 12.1    | 40.9    | -       |
| SELQ$^{PU}$ | 0.1 | 0.2 | 0.5 | 0.9 | 2.6 | 7.5 | 26.2 | -       |
| n = 30  |         |         |         |         |         |         |         |         |
| SELRC   | 0.0     | 0.0     | 0.0     | 1.1     | 8.6     | 101.0   | 859.8   | -       |
| SELQ    | 0.2     | 0.5     | 2.7     | 7.9     | 15.5    | 38.4    | 88.5    | -       |
| SELQ$^{PU}$ | 0.2 | 0.4 | 1.7 | 8.1 | 14.3 | 36.5 | 65.9 | -       |
| n = 40  |         |         |         |         |         |         |         |         |
| SELRC   | 0.0     | 0.0     | 0.0     | 1.6     | 11.6    | 232.3   | 1029.2* | -       |
| SELQ    | 1.3     | 1.8     | 5.6     | 28.3    | 35.1    | 136.5   | 177.6*  | -       |
| SELQ$^{PU}$ | 1.3 | 1.9 | 3.7 | 14.3 | 46.9 | 113.5 | 162.9* | -       |
| n = 50  |         |         |         |         |         |         |         |         |
| SELRC   | 0.0     | 0.0     | 0.0     | 2.1     | 17.0    | 206.4   | 1538.2* | -       |
| SELQ    | 1.4     | 3.0     | 13.9    | 55.8    | 112.6   | 350.3   | 290.6*  | -       |
| SELQ$^{PU}$ | 1.4 | 2.6 | 8.0 | 35.1 | 94.6 | 281.6 | 462.2* | -       |

variables and 81 constraints to represent the same instance. For fixed $n$ and increasing $S$, however, there always exists a threshold for which the quantified programs outperform the robust counterpart: the vertical lines indicate in which area the robust counterpart is solved faster (on average) than SELQ$^{PU}$. The conjecture that this dominance remains true for even larger $S$ is strongly supported by the growth of the instance itself and the resulting difficulty of CPLEX to manage the needed RAM or even load the model file. In summary, CPLEX is able to solve the robust counterpart faster for a large number of items and Yasol can better handle a large number of iterations in the quantified programs. For a better comparison of the performances of Yasol on SELQ$^{PU}$ and SELQ we refer to Table 4, which shows the average runtimes of those instances for which both quantified models were solved. The main observation remains, that instances with universal constraints are solved faster on average than the pure QIPs.

Furthermore, for each model we consider the number of instances for which this model was solved the fastest. Figure 2 shows the percentage of instances where the other models were slower. Note that the numbers do not necessarily add up to 100% due to instances with two or more models with the same runtime and due to unsolved instances.

These graphs further support the claim that the higher the number of items, the better CPLEX performs compared to Yasol on the same instance. On the other hand, the more iterations are considered within an instance, the better Yasol performs compared to CPLEX.

We also present performance profiles [25] and therefore briefly recall this concept: Let $S$ be the set of considered solvers, $P$ the set of instances and $t_{p,s}$ the runtime of solver $s$ on instance $p$. We assume $t_{p,s}$ is set to infinity (or large enough) if solver $s$ does not solve instance $p$ within the time limit. The percentage of instances for which the performance ratio...
Table 4: Average runtime (in seconds) of multistage selection instances with $N = 4$ for which both quantified model were solved. An asterisk indicates less than 20 solved instances.

| $n$  | model  | $S = 1$ | $S = 2$ | $S = 3$ | $S = 4$ | $S = 5$ | $S = 6$ | $S = 7$ | $S = 8$ |
|------|--------|---------|---------|---------|---------|---------|---------|---------|---------|
| 10   | SELQ   | 0.1     | 0.1     | 0.2     | 0.3     | 0.7     | 2.6     | 7.5     | 35.4    |
|      | SELQ$^P_U$ | 0.1     | 0.1     | 0.1     | 0.2     | 0.6     | 1.2     | 4.2     | 10.8    |
| 20   | SELQ   | 0.1     | 0.2     | 0.7     | 1.4     | 5.5     | 12.1    | 43.1    | 149.7   |
|      | SELQ$^P_U$ | 0.1     | 0.2     | 0.5     | 0.9     | 2.6     | 7.5     | 28.8    | 86.0    |
| 30   | SELQ   | 0.2     | 0.5     | 2.7     | 7.9     | 15.5    | 43.7    | 159.7   | 448.9   |
|      | SELQ$^P_U$ | 0.2     | 0.4     | 1.7     | 8.1     | 14.3    | 41.6    | 97.9    | 368.1   |
| 40   | SELQ   | 1.3     | 1.8     | 5.6     | 28.3    | 35.1    | 136.5   | 332.2   | 783.1   |
|      | SELQ$^P_U$ | 1.3     | 1.9     | 3.7     | 14.3    | 46.9    | 113.5   | 304.0   | 643.4   |
| 50   | SELQ   | 1.4     | 3.0     | 13.9    | 55.8    | 112.6   | 377.5   | 643.5   | 1021.0* |
|      | SELQ$^P_U$ | 1.4     | 2.6     | 8.0     | 35.1    | 94.6    | 281.7   | 601.8   | 831.7*   |

Figure 2: Percentage of instances per model where it was solved the fastest, i.e. if both other models took more time to solve. Plot for each number of items $n$ (a) and each number of iterations $S$ (b).

of solver $s$ is within a factor $\tau \geq 1$ of the best ratio of all solvers is given by

$$p_s(\tau) = \frac{1}{|P|} \left| \left\{ p \in P \mid \frac{t_{p,s}}{\min_{s \in S} t_{p,s}} \leq \tau \right\} \right|.$$  

Hence, the function $p_s(\tau)$ can be viewed as the distribution function for the performance ratio, which is plotted in a performance profile for each solver. For each performance profile in this paper we calculate $p_s(\tau)$ for $\tau = 1 + 0.5t$ with $t \in \mathbb{N}_0$ and $t$ large enough until $p_s(\tau)$ is constant. In Figure 3 we provide the performance profile for $N = 4$. Note that for each instance with (rounded) runtime of 0 seconds we used the runtime of 1 second in order to be able to generate useful performance profiles. The robust counterpart as well as the QIP with universal constraints is solved fastest on about 60% of all instances. Furthermore, the performance profile for SELQ$^P_U$ remains above the other two profiles and Yasol is able to
Figure 3: Performance profile for all examined multistage selection instances with $N = 4$ and each modeling variant.

solve more than 87% of the SELQ$^{PU}$ instances within a factor of 4 compared to the fastest solved instance.

We provide more detailed performance profiles in Figures 4 and 5 where we differentiate by $n$ and by $S$, respectively. In Figure 4 we see that for increasing $n$ CPLEX is more often the fastest method (solving SELRC) but Yasol is able to solve more instances using the quantified models. In Figure 5 we see that for large $S$ CPLEX cannot keep up with Yasol on SELQ$^{PU}$ and SELQ instances. For fewer iterations ($S \leq 5$) Yasol cannot outperform CPLEX but always solves all instances within the time limit.

5.2.3 Objective Values for Fixed $N$

To determine the benefit of solving the robust multistage optimization problem to optimality, we now compare the performance of the three heuristics presented in Section 4.1.2 with the respective optimal solutions. Recall that in strategy 1 the $p$ cheapest items in the initial stage are bought. In strategy 2 the lowest guaranteed future price for an item is compared to its current price. In strategy 3 an item is bought only if there exists no future iteration in which the remaining number of items can be bought for a cheaper price.

We use the same instances as in Section 5.2.2 with fixed number of $N = 4$ scenarios and various constellations of $n \in \{10, 20, 30, 40, 50\}$ and $S \in \{1, \ldots, 8\}$ with 50 instances per constellation resulting in 2000 instances. In order to detect the worst-case outcome when using one of the presented heuristics, a tree search is implemented in Python. For any instance it took only seconds to determine the worst-case outcome of any strategy, which is the obvious advantage of a heuristic. Determining the worst-case outcome of strategy 1 was the fastest ($\ll 1$ second per instance), followed by strategy 2 ($< 1$ second per instance) and strategy 3 ($\approx 1$ second per instance).

Note that for 45 of the 2000 instances, no optimal solution was found or ensured by either model in Section 5.2.2 within the time limit. All three heuristic strategies managed to outperform the best known value for 7 of these instances, in which cases only a trivial starting solution was found during the optimization process of each of the three models SELQ$^{PU}$, SELQ and SELRC. From now on we disregard the 45 instances of which we do not know the
optimal solution. Strategy 1 never resulted in the optimal value, while strategy 2 reached the optimal value in 19 cases and strategy 3 in 143 cases. Those 143 optimally solved instances by using strategy 3, however, are either instances with a single iteration, or with at most 20 items. In general we can say that the more items and the more iterations are considered, the larger is the relative deviation from the optimum for all three heuristics. In Table 5 the average relative deviation from the optimum are shown, e.g., a value of 0.50 means that on average, a heuristic had an objective value that was 50% larger than optimal. Strategy 3 is always closest to the optimal value on average and the average relative deviation almost always increases for increasing number of items and iterations. We expect similar behavior for a growing number of scenarios. Note that even though strategy 3 is the best on average there are 7 instances in which strategy 2 results in a better worst-case outcome. Obviously, further
Figure 5: Performance profiles for multistage selection models for various $S$. 
improvements are possible, and other online strategies could lead to even smaller deviations from the optimal value, but with potentially higher computing time. Due to the negligible computational effort and the good results, strategy 3 may be suitable as a generator of good starting solutions for a domain-specific solver for the multistage selection problem.

5.2.4 Computation Times for Fixed $n$

So far the number of scenarios was fixed to $N = 4$, but we also want to examine how the different approaches deal with instances with various numbers of scenarios. It is expected that with an increasing number of scenarios the approach of solving the quantified program is superior to solving the robust counterpart. We fix the number of items to $n = 10$, which is quite small but necessary in order to allow large values of $S$ and $N$ and still be able to find the optimal solution for many instances in reasonable time. For various $S \in \{1, \ldots, 8\}$ and $N \in \{2^1, 2^2, \ldots, 2^8\}$ again 50 instances are created for most constellations. Note that not all constellations are considered, since the prospects of finding the optimal solution within the time limit of 1800 seconds is very small, if both $S$ and $N$ are large. Table 6 shows the number of instances solved. For cells marked with a hyphen no experiments were conducted as it is expected that none (or very few) instances would be solved in the given time limit. The zeros in brackets indicate that the robust counterparts could not be created due to their size exceeding 200GB. As expected, for increasing $N$ and $S$ the number of quantified programs solved by Yasol tends to be larger than the number of robust counterpart solved by CPLEX. For each configuration, the number of solved SELQ$^{PU}$ models is highest and at least one SELQ$^{PU}$ instance is always solved. On 243 of the 303 instances where no model was solved to optimality, SELQ$^{PU}$ resulted in the best incumbent solution. On 175 of those instances SELQ$^{PU}$ was the only model for which any solution was found at all. On additional 23 instances the optimal solution of SELQ$^{PU}$ was found while for the other models not even an
Table 6: Number of solved multistage selection instances for fixed number of items $n = 10$ and various $N$ and $S$ within 1800 seconds.

| model | $S = 1$ | $S = 2$ | $S = 3$ | $S = 4$ | $S = 5$ | $S = 6$ | $S = 7$ | $S = 8$ |
|-------|---------|---------|---------|---------|---------|---------|---------|---------|
| $N = 2^4$ | SELRC | 50 | 50 | 50 | 50 | 50 | 50 | 50 |
| | SELQ | 50 | 50 | 50 | 50 | 50 | 50 | 50 |
| | SELQPU | 50 | 50 | 50 | 50 | 50 | 50 | 50 |
| $N = 2^2$ | SELRC | 50 | 50 | 50 | 50 | 50 | 49 | 26 |
| | SELQ | 50 | 50 | 50 | 50 | 50 | 50 | 50 |
| | SELQPU | 50 | 50 | 50 | 50 | 50 | 50 | 50 |
| $N = 2^3$ | SELRC | 50 | 50 | 50 | 50 | 48 | 4 | 0 | (0) |
| | SELQ | 50 | 50 | 50 | 50 | 50 | 50 | 47 |
| | SELQPU | 50 | 50 | 50 | 50 | 50 | 50 | 38 |
| $N = 2^4$ | SELRC | 50 | 50 | 50 | 50 | 44 | 0 | (0) | (0) |
| | SELQ | 50 | 50 | 50 | 50 | 50 | 0 | 0 |
| | SELQPU | 50 | 50 | 50 | 50 | 50 | 50 | 36 |
| $N = 2^5$ | SELRC | 50 | 50 | 50 | 50 | 3 | (0) | (0) | - |
| | SELQ | 50 | 50 | 50 | 50 | 50 | 0 | 0 |
| | SELQPU | 50 | 50 | 50 | 50 | 50 | 23 | 1 |
| $N = 2^6$ | SELRC | 50 | 50 | 50 | 32 | (0) | (0) | - | - |
| | SELQ | 50 | 50 | 50 | 50 | 0 | 0 | - | - |
| | SELQPU | 50 | 50 | 50 | 50 | 27 | 2 | - | - |
| $N = 2^7$ | SELRC | 50 | 50 | 50 | 0 | (0) | - | - | - |
| | SELQ | 50 | 50 | 50 | 14 | 0 | - | - | - |
| | SELQPU | 50 | 50 | 50 | 48 | 3 | - | - | - |
| $N = 2^8$ | SELRC | 50 | 48 | (0) | - | - | - | - | - |
| | SELQ | 50 | 50 | 0 | - | - | - | - | - |
| | SELQPU | 50 | 50 | 11 | - | - | - | - | - |

The incumbent solution was found.

Furthermore, we are interested in the runtimes of CPLEX and Yasol on the robust counterpart and the quantified programs, respectively. In Table 7 we present the average runtimes on instances of which all models were solved. For configurations with too few or no instances solved for all models, we use the average runtime of all solved instances of that model type. Those values are marked with a dagger ($\dagger$). We highlight the modest increase of the runtime of SELQ\textsuperscript{PU} models for increasing $S$ and $N$, even compared to SELQ. These two tables show that a) for instances with many iterations and scenarios the use of quantified programs is superior to solving the robust counterpart, and b) utilizing universal constraints rather than standard QIPs is of advantage in this setting.

6 Conclusion

Solving multistage robust problems is a formidable challenge, as the scenario tree grows exponentially. At the same time, from a practical perspective, there is little reason to only consider problems with at most two stages. Hence, new methods to solve multistage robust problems are required.

In this paper we argue that multistage robust combinatorial problems can be naturally
Table 7: Average runtime (in seconds) of multistage selection instances with \( n = 10 \) for which each model type was solved. Values marked with an asterisk show the average runtime on all solved instances of that model type.

| Model | \( S = 1 \) | \( S = 2 \) | \( S = 3 \) | \( S = 4 \) | \( S = 5 \) | \( S = 6 \) | \( S = 7 \) | \( S = 8 \) |
|-------|----------------|----------------|----------------|----------------|----------------|----------------|----------------|----------------|
| \( N = 2^1 \) | SELRC | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.1 |
| | SELQ | 0.1 | 0.1 | 0.1 | 0.2 | 0.2 | 0.3 | 0.4 | 0.4 |
| | SELQ\( ^\text{PU} \) | 0.1 | 0.1 | 0.1 | 0.1 | 0.0 | 0.2 | 0.3 | 0.5 |
| \( N = 2^2 \) | SELRC | 0.0 | 0.0 | 0.0 | 0.1 | 1.5 | 10.8 | 191.1 | 1023.7 |
| | SELQ | 0.1 | 0.1 | 0.2 | 0.2 | 0.7 | 2.2 | 8.7 | 28.6 |
| | SELQ\( ^\text{PU} \) | 0.0 | 0.1 | 0.2 | 0.2 | 0.5 | 1.1 | 4.1 | 6.3 |
| \( N = 2^3 \) | SELRC | 0.0 | 0.0 | 0.3 | 7.3 | 306.3 | 1261\( ^\text{†} \) | - | - |
| | SELQ | 0.2 | 0.2 | 0.4 | 2.0 | 13.9 | 103.9\( ^\text{†} \) | 848.2\( ^\text{†} \) | - |
| | SELQ\( ^\text{PU} \) | 0.1 | 0.1 | 0.4 | 1.0 | 3.9 | 26.2\( ^\text{†} \) | 146.8\( ^\text{†} \) | 678.8\( ^\text{†} \) |
| \( N = 2^4 \) | SELRC | 0.0 | 0.6 | 6.3 | 396.3 | - | - | - | - |
| | SELQ | 0.1 | 0.2 | 2.2 | 30.8 | 557.5\( ^\text{†} \) | - | - | - |
| | SELQ\( ^\text{PU} \) | 0.0 | 0.3 | 1.1 | 7.3 | 111.7\( ^\text{†} \) | 816.9\( ^\text{†} \) | 895.0\( ^\text{†} \) | - |
| \( N = 2^5 \) | SELRC | 0.0 | 0.8 | 102.4 | 1607.7\( ^\text{†} \) | - | - | - | - |
| | SELQ | 0.1 | 0.8 | 19.7 | 600.0\( ^\text{†} \) | - | - | - | - |
| | SELQ\( ^\text{PU} \) | 0.2 | 0.5 | 5.3 | 129.3\( ^\text{†} \) | 644.6\( ^\text{†} \) | 970.0\( ^\text{†} \) | - | - |
| \( N = 2^6 \) | SELRC | 0.0 | 4.2 | 807.5 | - | - | - | - | - |
| | SELQ | 0.3 | 3.6 | 222.7 | - | - | - | - | - |
| | SELQ\( ^\text{PU} \) | 0.2 | 1.4 | 38.7 | 800.9\( ^\text{†} \) | 1490.5\( ^\text{†} \) | - | - | - |
| \( N = 2^7 \) | SELRC | 0.0 | 25.4 | - | - | - | - | - | - |
| | SELQ | 0.4 | 20.0 | 1221.9\( ^\text{†} \) | - | - | - | - | - |
| | SELQ\( ^\text{PU} \) | 0.4 | 6.8 | 358.3\( ^\text{†} \) | 398.7\( ^\text{†} \) | - | - | - | - |
| \( N = 2^8 \) | SELRC | 0.0 | 187.6 | - | - | - | - | - | - |
| | SELQ | 1.0 | 147.5 | - | - | - | - | - | - |
| | SELQ\( ^\text{PU} \) | 1.0 | 45.0 | 621.1\( ^\text{†} \) | - | - | - | - | - |

phrased as quantified integer programs. This allows us to use tools developed for QIPs, in particular the solver Yasol, to solve robust problems. Two variants of QIP models were presented, which differ regarding the use of a constraint system for the universal player. In experiments using selection and assignment instances, we show that while CPLEX as an MIP solver for the equivalent deterministic problem performs better for few stages, Yasol has a clear advantage as the number of stages or the number of scenarios per stage increases. Additionally, the quantified models explicitly containing constraints on universal variables are solved faster than the corresponding QIP without universal constraint system.

There are several interesting further research questions that arise from this work. So far, we have only considered random data. How does the QIP approach perform when using real-world data, e.g., when deriving scenarios using approaches from data-drive robust optimization? Furthermore, our experiments are restricted to discrete scenario sets. It is an open question how methods perform on polyhedral uncertainty sets, such as budgeted uncertainty sets [13]. Finally, is it possible to develop techniques that are specific to QIPs with the structure considered here, as opposed to the more generic capabilities of Yasol? Our encouraging results show that such an approach may lead to an additional performance boost in the solution of multistage robust combinatorial problems.
Acknowledgements

This research was partially funded by the Deutsche Forschungsgemeinschaft (DFG, German Research Foundation) - 399489083.

References

[1] T. Achterberg, T. Koch, and A. Martin. Branching rules revisited. *Operations Research Letters*, 33(1):42–54, 2005.

[2] H. Aissi, C. Bazgan, and D. Vanderpooten. Complexity of the min–max and min–max regret assignment problems. *Operations Research Letters*, 33(6):634–640, 2005.

[3] S. Akl and M. Newborn. The principal continuation and the killer heuristic. In *Proceedings of the 1977 annual conference, ACM ’77, Seattle, Washington, USA*, pages 466–473, 1977.

[4] H. Bakker, F. Dunke, and S. Nickel. A structuring review on multi-stage optimization under uncertainty: Aligning concepts from theory and practice. *Omega*, 96:102080, 2020.

[5] E. Bampis, B. Escoffier, and A. Teiller. Multistage knapsack. *arXiv preprint arXiv:1901.11260*, 2019.

[6] A. Ben-Tal, D. Bertsimas, and D. B. Brown. A soft robust model for optimization under ambiguity. *Operations Research*, 58(4):1220–1234, 2010.

[7] A. Ben-Tal, L. E. Ghaoui, and A. Nemirovski. *Robust Optimization*. Princeton University Press, 2009.

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

[9] A. Ben-Tal and A. Nemirovski. Robust optimization - methodology and applications. *Mathematical Programming*, 92(3):453–480, 2002.

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

[11] D. Bertsimas and I. Dunning. Multistage robust mixed-integer optimization with adaptive partitions. *Operations Research*, 64(4):980–998, 2016.

[12] D. Bertsimas and A. Georgiou. Design of near optimal decision rules in multistage adaptive mixed-integer optimization. *Operations Research*, 63(3):610–627, 2015.

[13] D. Bertsimas and M. Sim. Robust discrete optimization and network flows. *Mathematical Programming*, 98(1-3):49–71, 2003.

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

[15] D. Bertsimas, M. Sim, and M. Zhang. Adaptive distributionally robust optimization. *Management Science*, 65(2):604–618, 2019.
[16] A. Biere. Adaptive restart strategies for conflict driven SAT solvers. In *International Conference on Theory and Applications of Satisfiability Testing*, pages 28–33. Springer Berlin Heidelberg, 2008.

[17] S. Ceria, C. Cordier, H. Marchand, and L. A. Wolsey. Cutting planes for integer programs with general integer variables. *Mathematical programming*, 81(2):201–214, 1998.

[18] A. Chassein and M. Goerigk. Performance analysis in robust optimization. In *Robustness Analysis in Decision Aiding, Optimization, and Analytics*, pages 145–170. Springer International Publishing, Cham, 2016.

[19] A. Chassein, M. Goerigk, A. Kasperski, and P. Zieliński. On recoverable and two-stage robust selection problems with budgeted uncertainty. *European Journal of Operational Research*, 265(2):423–436, 2018.

[20] X. Chen and P. Van Beek. Conflict-directed backjumping revisited. *Journal of Artificial Intelligence Research*, 14:53–81, 2001.

[21] X. Chen and Y. Zhang. Uncertain linear programs: Extended affinely adjustable robust counterparts. *Operations Research*, 57(6):1469–1482, 2009.

[22] F. J. de Ruiter, A. Ben-Tal, R. C. Brekelmans, and D. den Hertog. Robust optimization of uncertain multistage inventory systems with inexact data in decision rules. *Computational Management Science*, 14(1):45–66, 2017.

[23] V. G. Deı, G. J. Woeginger, et al. On the robust assignment problem under a fixed number of cost scenarios. *Operations Research Letters*, 34(2):175–179, 2006.

[24] E. Delage and D. A. Iancu. Robust multistage decision making. In *The Operations Research Revolution*, pages 20–46. INFORMS, 2015.

[25] E. D. Dolan and J. J. Moré. Benchmarking optimization software with performance profiles. *Mathematical Programming*, 91(2):201–213, 2002.

[26] C. Donninger and U. Lorenz. The chess monster Hydra. In *Field Programmable Logic and Application*, pages 927–932. Springer Berlin Heidelberg, 2004.

[27] T. Ederer, M. Hartisch, U. Lorenz, T. Opfer, and J. Wolf. Yasol: An open source solver for quantified mixed integer programs. In *15th International Conference on Advances in Computer Games, ACG 2017*, pages 224–233. Springer International Publishing, Cham, 2017.

[28] T. Ederer, U. Lorenz, A. Martin, and J. Wolf. Quantified linear programs: A computational study. In *Proceedings of the 19th European Conference on Algorithms*, ESA’11, pages 203–214. Springer Berlin Heidelberg, 2011.

[29] T. Ederer, U. Lorenz, and T. Opfer. Quantified combinatorial optimization. In *Operations Research Proceedings 2013*, pages 121–128. Springer International Publishing, Cham, 2014.

[30] T. Ederer, U. Lorenz, T. Opfer, and J. Wolf. Modeling games with the help of quantified integer linear programs. In *13th International Conference on Advances in Computer Games, ACG 2011*, pages 270–281. Springer Berlin Heidelberg, 2011.
[31] P. Eirinakis, S. Ruggieri, K. Subramani, and P. Wojciechowski. Computational complexities of inclusion queries over polyhedral sets. In *International Symposium on Artificial Intelligence and Mathematics (ISAIM)*, 2012.

[32] D. Eisenstat, C. Mathieu, and N. Schabanel. Facility location in evolving metrics. In *International Colloquium on Automata, Languages, and Programming*, pages 459–470. Springer, 2014.

[33] M. Fischetti and M. Monaci. Light robustness. In *Robust and Online Large-Scale Optimization*, pages 61–84. Springer Berlin Heidelberg, 2009.

[34] A. Georgiou, D. Kuhn, and W. Wiesemann. The decision rule approach to optimization under uncertainty: methodology and applications. *Computational Management Science*, 16(4):545–576, 2019.

[35] R. Gerber, W. Pugh, and M. Saksena. Parametric dispatching of hard real-time tasks. *IEEE Trans. Computers*, 44(3):471–479, 1995.

[36] E. Giunchiglia, M. Narizzano, and A. Tacchella. Backjumping for Quantified Boolean Logic satisfiability. *Artificial Intelligence*, 145(1):99–120, 2003.

[37] E. Giunchiglia, M. Narizzano, A. Tacchella, et al. Learning for quantified Boolean logic satisfiability. In *Proceedings of the AAAI Conference on Artificial Intelligence*, pages 649–654, 2002.

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

[39] B. L. Gorissen, İ. Yanköğlu, and D. den Hertog. A practical guide to robust optimization. *Omega*, 53:124–137, 2015.

[40] A. Gupta, K. Talwar, and U. Wieder. Changing bases: Multistage optimization for matroids and matchings. In J. Esparza, P. Fraigniaud, T. Husfeldt, and E. Koutsoupias, editors, *Automata, Languages, and Programming*, pages 563–575, Berlin, Heidelberg, 2014. Springer Berlin Heidelberg.

[41] M. Hartisch. *Quantified Integer Programming with Polyhedral and Decision-Dependent Uncertainty*. PhD thesis, University of Siegen, Germany, accessible via: [http://dx.doi.org/10.25819/ubs/4841](http://dx.doi.org/10.25819/ubs/4841), 2020.

[42] M. Hartisch, T. Ederer, U. Lorenz, and J. Wolf. Quantified integer programs with polyhedral uncertainty set. In *Computers and Games - 9th International Conference, CG 2016*, pages 156–166. Springer International Publishing, Cham, 2016.

[43] M. Hartisch and U. Lorenz. Game tree search in a robust multistage optimization framework: Exploiting pruning mechanisms. *arXiv preprint arXiv:1811.12146*. To appear in: *16th International Conference on Advances in Computer Games, ACG 2019*, 2019.

[44] P. Kall and J. Mayer. *Stochastic linear programming. Models, theory and computation*. Springer, 2005.
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 International Publishing, Cham, 2016.

A. J. Keith and D. K. Ahner. A survey of decision making and optimization under uncertainty. *Annals of Operations Research*, pages 1–35, 2019.

D. Knuth and R. Moore. An analysis of alpha-beta pruning. *Artificial Intelligence*, 6(4):293–326, 1975.

N. Lappas and C. Gounaris. Multi-stage adjustable robust optimization for process scheduling under uncertainty. *AIChE Journal*, 62(5):1646–1667, 2016.

C. Liebchen, M. Lübbeke, R. Möhring, and S. Stiller. The concept of recoverable robustness, linear programming recovery, and railway applications. In *Robust and online large-scale optimization*, pages 1–27. Springer Berlin Heidelberg, 2009.

C.-M. Lin and M. Gen. Multi-criteria human resource allocation for solving multistage combinatorial optimization problems using multiobjective hybrid genetic algorithm. *Expert Systems with Applications*, 34(4):2480–2490, 2008.

Á. Lorca, X. A. Sun, E. Litvinov, and T. Zheng. Multistage adaptive robust optimization for the unit commitment problem. *Operations Research*, 64(1):32–51, 2016.

U. Lorenz, A. Martin, and J. Wolf. Polyhedral and algorithmic properties of quantified linear programs. In *Proceedings of the 18th Annual European Conference on Algorithms: Part I, ESA’10*, pages 512–523. Springer Berlin Heidelberg, 2010.

U. Lorenz, T. Opfer, and J. Wolf. Solution techniques for quantified linear programs and the links to gaming. In *Computers and Games - 8th International Conference, CG 2013*, pages 110–124. Springer International Publishing, Cham, 2013.

U. Lorenz and J. Wolf. Solving multistage quantified linear optimization problems with the alpha–beta nested Benders decomposition. *EURO Journal on Computational Optimization*, 3(4):349–370, 2015.

M. Mankowski and M. Moshkov. Extensions of dynamic programming for multi-stage combinatorial optimization. *Theoretical Computer Science*, 2020.

J. Marques-Silva, I. Lynce, and S. Malik. Conflict-driven clause learning SAT solvers. In *Handbook of Satisfiability*, pages 131–153. Ios Press, 2009.

F. Motamed Nasab and Z. Li. Multistage adaptive optimization using hybrid scenario and decision rule formulation. *AIChE Journal*, 65(12):e16764, 2019.

C. Ning and F. You. A data-driven multistage adaptive robust optimization framework for planning and scheduling under uncertainty. *AIChE Journal*, 63(10):4343–4369, 2017.

C. Ning and F. You. Optimization under uncertainty in the era of big data and deep learning: When machine learning meets mathematical programming. *Computers & Chemical Engineering*, 125:434–448, 2019.
The three heuristic strategies and the optimal strategy with their worst-case outcome explored in Example 4.2 are presented. The investigated problem is a multistage selection problem with \( n = 6 \) items of which \( p = 3 \) have to be selected. There are \( S = 2 \) iterations, each with \( N = 2 \) scenarios. We repeat the costs in the initial stage and each scenario in Table 8.

Tables 9-12 describe the constructed strategies. Values in parentheses are the item’s cost, when bought in the proposed iteration and scenario. Gray cells indicate that this cell represents the second scenario of this iteration. The tables should be read from left to right, and can be interpreted as following the paths in the scenario tree. Cells with a hyphen indicate that nothing should be done. The costs in the worst-case scenario are printed in bold.

The optimal solution as shown in Table 12 is to select no items in the initial stage and wait for the scenario in iteration 1. This is surprising, as buying item 2 in the initial stage seems inevitable due to the low cost of 14. In this example the costs after each realization of the scenarios in each iteration are smaller when applying the optimal winning strategy compared to the other heuristic strategies. Note that this does not have to be the case, as the sole aim of the optimization is minimize the worst-case costs.
Table 8: Cost scenarios for an instance of the multistage selection problem.

|   | 1  | 2  | 3  | 4  | 5  | 6  |
|---|----|----|----|----|----|----|
| $c_i^0$ | 84 | 14 | 76 | 61 | 31 | 45 |
| $c_{i,1}^1$ | 40 | 24 | 29 | 41 | 90 | 71 |
| $c_{i,2}^1$ | 45 | 30 | 15 | 18 | 44 | 44 |
| $c_{i,1}^2$ | 13 | 25 | 12 | 11 | 75 | 50 |
| $c_{i,2}^2$ | 80 | 10 | 29 | 32 | 64 | 30 |

Table 9: Selection strategy according to “Buy All Now”.

| iteration 0 | iteration 1 | iteration 2 | costs |
|-------------|-------------|-------------|-------|
| select item 2(14) | - | - | 90 |
| select item 5(31) | - | - | 90 |
| select item 6(45) | - | - | 90 |

Table 10: Selection strategy according to “Buy Now, If Never Cheaper in Worst Case”.

| iteration 0 | iteration 1 | iteration 2 | costs |
|-------------|-------------|-------------|-------|
| select item 2(14) | select item 5(31) | select item 4(11) | 56 |
| select item 3(29) | select item 4(11) | 74 |
| select item 3(15) | 56 |
| - | 60 |
| - | 60 |

Table 11: Selection strategy according to “Buy Now, If Few are Cheaper”.

| iteration 0 | iteration 1 | iteration 2 | costs |
|-------------|-------------|-------------|-------|
| select item 2(14) | select item 3(29) | select item 4(11) | 54 |
| select item 6(30) | select item 4(11) | 73 |
| select item 3(15) | select item 4(18) | 47 |
| - | 47 |

**B Robust Multistage Assignment**

In this appendix we provide experimental results on the multistage assignment problem as introduced in Subsection 4.2 in which a perfect matching in a bipartite graph with minimal costs has to be determined. We compare the performance of Yasol on the quantified models ASSQ_{PU} and ASSQ with the performance of CPLEX on the robust counterpart ASSRC.

Each instance is given by the size $n$ of each partition, the number of iterations $S$ and the number of scenarios $N$ per iteration. The remaining parameters of an instance are the cost
Table 12: Optimal selection strategy.

| iteration 0 | iteration 1 | iteration 2 | costs |
|-------------|-------------|-------------|-------|
|             |             | 4(11), 3(12) and 1(13) | 36   |
|             |             | 2(10), 3(29) and 6(30) | 69   |
| select item 3(15) | select item 1(13) | select item 2(10) | 46   |
| select item 4(18) |             |             | 43   |

$c_{i,j,k}$ for each edge $(i, j)$ in scenario $k$ of iteration $s$, which are randomly selected from the range $0, 1, \ldots, 99$.

For each $n \in \{4, \ldots, 10\}$, $S \in \{1, \ldots, 4\}$ and $N \in \{2, 4, 8\}$ we create 50 instances for each model type ASSQ$^{PU}$, ASSQ and ASSRC. Each solver has a time limit of 1800 seconds. We examine how the realization of $n$, $S$ and $N$ affects the runtime and which model-solver combination is best suited for the different instances.

In Tables 13, 14 and 15 the number of solved instances as well as their average runtime is presented for instances with $N = 2$, $N = 4$ and $N = 8$ scenarios, respectively. Runtimes are averaged over instances that were solved to optimality by the respective method.

Table 13: Number of solved multistage assignment instances (opt) with $N = 2$ and the respective average runtime for each method (time).

| model | $S = 1$ | $S = 2$ | $S = 3$ | $S = 4$ |
|-------|---------|---------|---------|---------|
|       | opt time| opt time| opt time| opt time|
| $n = 4$ | ASSRC  | 50 0.0  | 50 0.0  | 50 0.0  | 50 0.0  |
|       | ASSQ   | 50 0.0  | 50 0.1  | 50 0.2  | 50 0.2  |
|       | ASSQ$^{PU}$ | 50 0.1 | 50 0.1  | 50 0.2  | 50 0.2  |
| $n = 5$ | ASSRC  | 50 0.0  | 50 0.0  | 50 0.0  | 50 0.0  |
|       | ASSQ   | 50 0.1  | 50 0.2  | 50 0.3  | 50 0.7  |
|       | ASSQ$^{PU}$ | 50 0.1 | 50 0.1  | 50 0.2  | 50 0.4  |
| $n = 6$ | ASSRC  | 50 0.0  | 50 0.0  | 50 0.0  | 50 0.0  |
|       | ASSQ   | 50 0.1  | 50 0.3  | 50 0.7  | 50 1.7  |
|       | ASSQ$^{PU}$ | 50 0.1 | 50 0.2  | 50 0.5  | 50 0.8  |
| $n = 7$ | ASSRC  | 50 0.0  | 50 0.0  | 50 0.0  | 50 0.0  |
|       | ASSQ   | 50 0.2  | 50 0.7  | 50 2.7  | 50 7.6  |
|       | ASSQ$^{PU}$ | 50 0.1 | 50 0.4  | 50 0.9  | 50 2.1  |
| $n = 8$ | ASSRC  | 50 0.0  | 50 0.0  | 50 0.0  | 50 0.5  |
|       | ASSQ   | 50 0.2  | 50 2.0  | 50 11.6 | 50 20.1 |
|       | ASSQ$^{PU}$ | 50 0.2 | 50 0.6  | 50 2.6  | 50 5.1  |
| $n = 9$ | ASSRC  | 50 0.0  | 50 0.0  | 50 0.0  | 50 1.1  |
|       | ASSQ   | 50 0.2  | 50 5.1  | 50 22.6 | 50 116.3|
|       | ASSQ$^{PU}$ | 50 0.2 | 50 1.7  | 50 7.7  | 50 21.4 |
| $n = 10$ | ASSRC | 50 0.0  | 50 0.0  | 50 0.0  | 50 2.1  |
|       | ASSQ   | 50 0.6  | 50 10.0 | 50 98.0 | 50 351.6|
|       | ASSQ$^{PU}$ | 50 0.3 | 50 4.6  | 50 22.9 | 50 84.3 |

For $N = 2$ all instances for each model and configuration are solved and the strength of
Table 14: Number of solved multistage assignment instances (opt) with $N = 4$ and the respective average runtime for each method (time).

| $S$ | $n = 4$ | $n = 5$ | $n = 6$ | $n = 7$ | $n = 8$ | $n = 9$ | $n = 10$ |
|-----|---------|---------|---------|---------|---------|---------|---------|
|     | ASSRC   | ASSQ    | ASSQPU  | ASSRC   | ASSQ    | ASSQ    | ASSRC   |
| 1   | 50 0.0  | 50 0.0  | 50 0.0  | 50 0.0  | 50 0.0  | 50 0.0  | 50 0.0  |
| 2   | 50 0.1  | 50 0.1  | 50 0.1  | 50 0.1  | 50 0.1  | 50 0.1  | 50 0.1  |
| 3   | 50 0.2  | 50 0.2  | 50 0.2  | 50 0.2  | 50 0.2  | 50 0.2  | 50 0.2  |
| 4   | 50 0.5  | 50 0.5  | 50 0.5  | 50 0.5  | 50 0.5  | 50 0.5  | 50 0.5  |

CPLEX is clearly visible: the runtime barely increases for increasing $n$. For the quantified models the runtime increases significantly faster, whereas ASSQPU models are solved considerably faster than ASSQ models. For increasing $S$ and $N$, the ASSQPU model tends to yield the best results but CPLEX (on ASSRC) remains highly competitive and is able to catch up with increasing $n$. In Figure 6 the performance profile on all 4200 instances is given. It can

![Figure 6: Performance profile for all examined assignment instances and each model.](image-url)
Table 15: Number of solved multistage assignment instances (opt) with \( N = 8 \) and the respective average runtime for each method (time).

|          | \( S = 1 \) | \( S = 2 \) | \( S = 3 \) | \( S = 4 \) |
|----------|-------------|-------------|-------------|-------------|
|          | opt         | time        | opt         | time        |
| \( n = 4 \) |             |             |             |             |
| ASSRC    | 50          | 0.0         | 50          | 0.0         | 50          | 3.2         | 50          | 118.6       |
| ASSQ     | 50          | 0.1         | 50          | 0.2         | 50          | 0.8         | 50          | 4.0         |
| ASSQ\text{PU} | 50          | 0.1         | 50          | 0.2         | 50          | 1.3         | 50          | 5.8         |
| \( n = 5 \) |             |             |             |             |
| ASSRC    | 50          | 0.0         | 50          | 0.2         | 50          | 11.8        | 43          | 349.0       |
| ASSQ     | 50          | 0.1         | 50          | 0.6         | 50          | 2.5         | 50          | 13.9        |
| ASSQ\text{PU} | 50          | 0.2         | 50          | 0.4         | 50          | 2.4         | 50          | 18.4        |
| \( n = 6 \) |             |             |             |             |
| ASSRC    | 50          | 0.0         | 50          | 0.9         | 50          | 70.0        | 25          | 662.8       |
| ASSQ     | 50          | 0.3         | 50          | 2.5         | 50          | 13.6        | 50          | 63.6        |
| ASSQ\text{PU} | 50          | 0.2         | 50          | 1.7         | 50          | 8.4         | 50          | 70.0        |
| \( n = 7 \) |             |             |             |             |
| ASSRC    | 50          | 0.0         | 50          | 1.4         | 49          | 215.2       | 7           | 762.7       |
| ASSQ     | 50          | 0.4         | 50          | 11.3        | 50          | 58.8        | 49          | 310.7       |
| ASSQ\text{PU} | 50          | 0.3         | 50          | 4.1         | 50          | 33.7        | 50          | 248.0       |
| \( n = 8 \) |             |             |             |             |
| ASSRC    | 50          | 0.0         | 50          | 4.8         | 41          | 445.8       | 2           | 496.5       |
| ASSQ     | 50          | 1.2         | 50          | 47.4        | 50          | 251.1       | 39          | 854.5       |
| ASSQ\text{PU} | 50          | 0.5         | 50          | 4.1         | 50          | 33.7        | 44          | 869.4       |
| \( n = 9 \) |             |             |             |             |
| ASSRC    | 50          | 0.0         | 50          | 21.9        | 24          | 470.5       | 3           | 991.3       |
| ASSQ     | 50          | 2.9         | 50          | 228.8       | 31          | 1117.9      | 7           | 898.4       |
| ASSQ\text{PU} | 50          | 1.5         | 50          | 45.8        | 50          | 598.5       | 4           | 1131.0      |
| \( n = 10 \) |            |             |             |             |
| ASSRC    | 50          | 0.0         | 50          | 56.4        | 15          | 517.3       | 1           | 388.0       |
| ASSQ     | 50          | 8.1         | 43          | 633.9       | 7           | 675.9       | 1           | 173.0       |
| ASSQ\text{PU} | 50          | 3.0         | 50          | 266.5       | 8           | 1056.0      | 1           | 1154.0      |

It can be seen that CPLEX is significantly faster on most ASSRC instances and is fastest on more than 75% of the instances. However, overall more of the ASSQ\text{PU} and ASSQ instances are solved. In comparison with Figure 3, the benefit of using universal constraints instead of the standard QIP seems to be even greater for this assignment problem than for the selection problem on the tested instances, as the blue and black curves approach each other much more slowly.