A Novel Application for Game Tree Search - Exploiting Pruning Mechanisms for Quantified Integer Programs

Michael Hartisch(✉) and Ulf Lorenz

Chair of Technology Management, University of Siegen, Siegen, Germany
{michael.hartisch,ulf.lorenz}@uni-siegen.de

Abstract. We investigate pruning in search trees of so-called quantified integer (linear) programs (QIPs). QIPs consist of a set of linear inequalities and a minimax objective function, where some variables are existentially and others are universally quantified. A good way to solve a QIP is to apply game tree search, enhanced with non-chronological back-jumping. We develop and theoretically substantiate tree pruning techniques based upon algebraic properties. The presented Strategic Copy-Pruning mechanism allows to *implicitly* deduce the existence of a strategy in linear time (by static examination of the QIP-matrix) without explicitly traversing the strategy itself. We show that the implementation of our findings can massively speed up the search process.

1 Introduction

Prominent solution paradigms for optimization under uncertainty are Stochastic Programming [5], Robust Optimization [3], Dynamic Programming [2], Sampling [11] and POMDP [21]. Relatively unexplored are the abilities of linear programming extensions for PSPACE-complete problems. In the early 2000s the idea of universally quantified variables, as they are used in quantified constraint satisfaction problems [10], was picked up again [27], coining the term quantified integer program (QIP). Quantified integer programming is a direct, very formal extension of integer linear programming (IP), making QIPs applicable in a very natural way. They allow robust multistage optimization extending the two-stage approach of Robust Optimization [3]. Multistage models - in contrast to two-stage models - allow more precise planning strategies as uncertain events typically do not occur all at the same time (delay in timetables, changed cost estimate for edges in a graph, alternating moves in games).

Let us start with the following illustrative application. There are b runways at your airport and all arriving airplanes must be assigned to exactly one time slot for the landing (therefore a natural *worst-case* optimization problem). Further,

This research is partially supported by the German Research Foundation (DFG) project "Advanced algorithms and heuristics for solving quantified mixed - integer linear programs".

the airplanes are expected to arrive within some time window and hence the assigned time slot must adhere to those time windows. Finding an initial matching, even an optimal one considering some objective function, can be modeled and solved using mixed integer programming techniques [14]. However, the time windows are uncertain due to adjusted airspeed (due to weather) or operational problems and an initial schedule might become invalid (see for example Fig. 1).

Thus, one is interested in a robust initial plan that can be adapted cheaply,

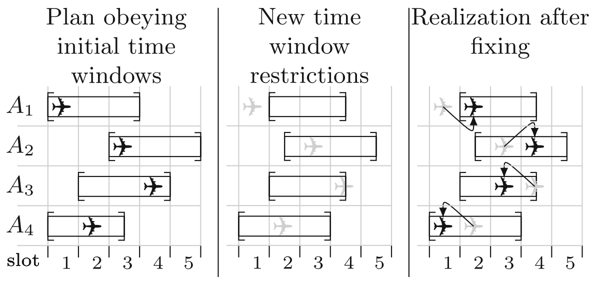
Plan obeying initial time windows
New time window restrictions
Realization after fixing
A_1
A_2
A_3
A_4
slot
1
2
3
4
5

Fig. 1. Process of runway scheduling: A schedule for the initial time windows is made (left). If the predicted time windows differ from the actually occurring time windows (middle), the initial plan becomes invalid and a new scheduling must be found (right).

e.g. the initial and adapted time slot of each airplane should not be too far apart from each other [13]. These uncertain events, however, do not uncover all at the same time: final time slots must be assigned to some airplanes while for other airplanes the actual time window is still unknown. This problem is literally crying out to be modeled as a QIP.

A solution of a QIP is a strategy – in the game tree search sense [22] – for assigning existentially quantified variables such that some linear constraint system is fulfilled. By adding a minimax objective function the aim is to find the best strategy [19]. As not unusual in the context of optimization under uncertainty [3,4] a polyhedral uncertainty set can be used [12]. There are two different ways known how to tackle a QIP: On the one hand the so-called deterministic equivalent program can be built, similar to the ones known from stochastic programming [5], and solved using standard integer programming solvers. On the other hand the more natural approach is to conduct a game tree search [18,26]. We are interested in utilizing game solving techniques [25,28] in combination with linear programming techniques as well as pruning and backjumping techniques from QBF [6]. Recently our solver for quantified mixed integer programs was made available as open source. This solver combines techniques known from game tree search, linear programming and QBF [9].

An optimization task is often split up into two parts: finding the optimal solution itself and proving that no better solution can exist. For the latter, it turned out that applying backjumping techniques as utilized by QBF-solvers [29] and cutting planes as commonly used in integer programming [20] are also highly beneficial for QIPs in order to assess that no (better) strategy can exist in certain subtrees. For the first task, however, it seems that the exponential number of leaves belonging to a strategy must be traversed explicitly. This is certainly true in the worst-case. However, typically there are "difficult" parts of a game tree where a very deliberated substrategy must be found but also other parts where a less sophisticated substrategy suffices. In this paper we present a procedure, called strategic copy-pruning (SCP), that is capable of recognizing such subtrees making it possible to *implicitly* deduce the existence of a winning strategy therein. In contrast to similar ideas in QBF, as e.g. counterexample guided abstraction refinement [16], an optimization process over a minimax objective must be considered. Further, our SCP draws its power not from memory-intensive learning, but from deep findings in the search tree. This perspective has led to remarkable achievements in the past [7,15]. For game tree search there are already several algorithms trying to rapidly show the existence of winning strategies such as Kawano's simulation [17], MTD(f) [23] and (nega)scout [24]. They, however, always have to traverse an exponential number of leafs. In our experiments, SCP often allows to conclude the existence of an winning strategy with a linear number of algebraic operations and in particular, in those cases it is not necessary to examine an exponential number of leaves resulting in a significant performance improvement both in time (about a factor 4) and number of solved instances. The effect of SCP is reinforced if the sequence of variable assignments predicted as optimal by minimax for both sides, called the principal variation [8], is traversed in an early stage of the tree search. Detecting and verifying this particular variable assignment is essential in order to obtain the objective value. Thus having reasonable knowledge of which universal variable assignments are particularly vicious can massively boost the search process. Several heuristics exist to analyze and find such promising moves in a game tree search environment [1,23,25]. The paper is organized as follows: First basic definitions and notations regarding QIPs are presented. Then two pruning techniques for the QIP game tree search are introduced: First, the well known monotonicity [6] of variables is recaptured. Second, as our main result, we derive from already found strategies the existence of winning strategies in other branches. This happens in a way such that these branches do not need to be investigated explicitly. Finally the conducted experiments are presented.

2 Preliminaries: Basics of Quantified Integer Programming

Let $n \in \mathbb{N}$ be the number of variables and $x = (x_1, \ldots, x_n)^\top \in \mathbb{Z}^n$ a vector of variables.[1] For each variable x_j its domain $\mathcal{L}_j$ with $l_j, u_j \in \mathbb{Z}$, $l_j \leq u_j$, $1 \leq j \leq n$, is given by $\mathcal{L}_j = \{y \in \mathbb{Z} \mid l_j \leq y \leq u_j\}$. The domain of the entire variable vector is described by $\mathcal{L} = \{y \in \mathbb{Z}^n \mid \forall j \in \{1, \ldots, n\} : y_j \in \mathcal{L}_j\}$,

[1] $\mathbb{Z}$, $\mathbb{N}$ and $\mathbb{Q}$ denote the set of integers, natural numbers, and rational numbers, respectively.

i.e. each variable must obey its domain. Let $Q \in \{\exists, \forall\}^n$ denote the vector of quantifiers. We call $\mathcal{E} = \{j \in \{1,\dots,n\} \mid Q_j = \exists\}$ the set of existential variables and $\mathcal{A} = \{j \in \{1,\dots,n\} \mid Q_j = \forall\}$ the set of universal variables. Further, each maximal consecutive subsequence in Q consisting of identical quantifiers is called *quantifier block* with $B_i \subseteq \{1,\dots,n\}$ denoting the i-th block. Let $\beta \in \mathbb{N}$, $\beta \leq n$, denote the number of blocks and thus $\beta - 1$ is the number of quantifier changes. The variable vector of variable block B_i will be referred to as $x^{(i)}$.

Definition 1 (Quantified Integer Linear Program (QIP)). *Let $A \in \mathbb{Q}^{m \times n}$ and $b \in \mathbb{Q}^m$ for $m \in \mathbb{N}$ and let $\mathcal{L}$ and Q be given as described above. Let $c \in \mathbb{Q}^n$ be the vector of objective coefficients and let $c^{(i)}$ denote the vector of coefficients belonging to block B_i. Let the term $Q \circ x \in \mathcal{L}$ with the component wise binding operator $\circ$ denote the* quantification vector $(Q_1 x_1 \in \mathcal{L}_1, \dots, Q_n x_n \in \mathcal{L}_n)$ *such that every quantifier Q_j binds the variables x_j to its domain $\mathcal{L}_j$. We call $(A, b, c, \mathcal{L}, Q)$ with*

$$\begin{aligned} & z = \min_{B_1} \left(c^{(1)} x^{(1)} + \max_{B_2} \left(c^{(2)} x^{(2)} + \dots \min_{B_\beta} c^{(\beta)} x^{(\beta)} \right) \right) \\ & s.t.\ Q \circ x \in \mathcal{L} :\ Ax \leq b \qquad (\star) \end{aligned}$$

a QIP with objective function (for a minimizing existential player).

For simplicity's sake, and since it goes well with the example in Fig. 1, we will consider only binary QIPs, i.e. $l_j = 0$ and $u_j = 1$ for all $j \in \{1,\dots,n\}$. However, note that our results and in particular Theorem 1 can easily be adapted to be valid for general integer variables.

A QIP instance 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. Consequently, we say that a player makes the move $x_k = y$ if she fixes the variable x_k to $y \in \mathcal{L}_k$. At each such move, the corresponding player knows the settings of $x_1, \dots, x_{k-1}$ before taking her decision x_k. If the completely assigned vector $x \in \mathcal{L}$ satisfies the linear constraint system $Ax \leq b$, the existential player pays $z = c^\top x$ to the universal player. If x does not satisfy $Ax \leq b$, we say *the existential player loses* and the payoff will be $+\infty$. This is a small deviation from conventional zero-sum games but using[2] $\infty + (-\infty) = 0$ also fits for zero-sum games. The chronological order of the variable blocks given by Q can be represented using a game tree $G = (V, E, c)$ with $V = V_\exists \cup V_\forall \cup V_L$ consisting of existential, universal and leaf nodes [9]. Thus, a path from the root to a leaf represents a play of the QIP and the sequence of edge labels encodes its moves, i.e. the corresponding variable assignments. Solutions of a QIP are strategies [9]. In the following, the word *strategy* will always refer to an *existential* strategy. A strategy is called a *winning strategy* if all paths from the root node to a leaf represent a vector x such that $Ax \leq b$. A QIP is called *feasible* if $(\star)$ is true (see Definition 1), i.e. if a

[2] This is only a matter of interpretation and consequences are not discussed further.

winning strategy exists. If there is more than one winning strategy, the objective function aims for a certain (the "best") one. The value of a strategy is given by its minimax value which is the maximum value at its leaves [22]. Note that a leaf not fulfilling $Ax \leq b$ can be represented by the value $+\infty$. The objective value of a feasible QIP is the minimax value at the root, i.e. the minimax value of the optimal winning strategy, defined by the *principal variation* (PV) [8]: the sequence of variable assignments being chosen during optimal play. For any $v \in V$ we call $f(v)$ the outcome of optimal play by both players starting at v.

Example. Let us consider a QIP with $n = 4$ binary variables:

$$\begin{array}{rccccl}
\min(& 2x_1 \quad \max(& -2x_2 \quad \min(& -3x_3 \quad \max(& -2x_4)))) & \\
\text{s.t.} & \exists x_1 \in \{0,1\} & \forall x_2 \in \{0,1\} & \exists x_3 \in \{0,1\} & \forall x_4 \in \{0,1\} & : \\
 & x_1 & +x_2 & +x_3 & & \leq 2 \\
 & -x_1 & & +x_3 & -x_4 & \leq 0 \\
 & & -x_2 & +x_3 & -x_4 & \leq 0 \\
 & -x_1 & +x_2 & -x_3 & +x_4 & \leq 1
\end{array}$$

The minimax value of the root node (for the minimizing starting player) of the game tree is 2 and the principal variation is given by $x_1 = 1$, $x_2 = 0$, $x_3 = 0$ and $x_4 = 0$. The inner node at level 1 resulting from setting $x_1 = 0$ has the minimax value $+\infty$, i.e. after setting $x_1 = 0$ there exists no winning strategy.

3 Pruning in QIP Search Trees

3.1 Theoretical Analysis

In a natural way, a quantified integer program can be solved via game tree search. During such a tree search we are interested in quickly evaluating or estimating the minimax value of nodes, i.e. we want to examine the optimal (existential) strategy of the corresponding subtree. In order to speed up the search process, limiting the number of subtrees that need to be explored is extremely beneficial. Such pruning operations are applied in many search based algorithms, e.g. the alpha-beta algorithm [18], branch-and-bound [20] and DPLL [29]. In the following, we will present two approaches that allow pruning in a QIP game tree search, and thus in a strategic optimization task.

In case of QIPs certain variable assignments never need to be checked as they are worse than their counterparts. The concept of monotone variables is already well known for quantified boolean formulas [6] and integer programming [20].

Definition 2 (Monotone Variable)
A variable x_k of a QIP is called monotone if it occurs with only positive or only negative sign in the matrix and objective, i.e. if the entries of A and c belonging to x_k are either all non-negative or all non-positive.

Using this easily verifiable monotonicity allows us to omit certain subtrees a priori since solving the subtree of its sibling is guaranteed to yield the desired minimax value.

In contrast to this usage of prior knowledge we also want to gather *deep knowledge* during the search process: found strategies in certain subtrees can be useful in order to assess the minimax value of related subtrees rapidly. The idea is based upon the observation that typically in only a rather small part of the game tree a distinct and crafty strategy is required in order to ensure the fulfillment of the constraint system: in the right-hand side subtree of Fig. 2 it suffices to find a fulfilling existential variable assignment for only one scenario (universal variable assignment) and reuse it in the other branches.

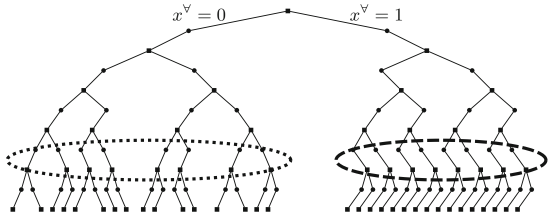

Fig. 2. Illustrative strategy for which the universal assignment $x^\forall = 1$ entails a simple winning strategy: Regardless of future universal decisions existential variables can be set in a certain simple way, e.g. the existential decisions in the dashed ellipse are all the same. $x^\forall = 0$ on the other hand compels a more clever strategy, e.g. the existential decisions in the dotted ellipse differ depending on previous universal decisions.

Theorem 1. *[Strategic Copy-Pruning (SCP)]*
Let $k \in \mathcal{A}$ and let $(\tilde{x}_1, \ldots, \tilde{x}_{k-1}) \in \{0,1\}^{k-1}$ be a fixed variable assignment of the variables $x_1, \ldots, x_{k-1}$. Let $v \in V_\forall$ be the corresponding universal node in the game tree. Let $\tilde{w} \in V$ and $\hat{w} \in V$ be the two children of v corresponding to the variable assignment $\tilde{x}_k$ and $\hat{x}_k = 1 - \tilde{x}_k$ of the universal variable x_k, respectively. Let there be an optimal winning strategy for the subtree below $\tilde{w}$ with minimax value $f(\tilde{w}) = \tilde{z}$ defined by the variable assignment $\tilde{x} = (\tilde{x}_1, \ldots, \tilde{x}_n) \in \{0,1\}^n$, i.e. $\tilde{z} = c^\top \tilde{x}$. If the minimax value of the copied strategy for the subtree below $\hat{w}$ - obtained by adoption of future[3] existential variable assignments as in $\tilde{x}$ - is not larger than $\tilde{z}$ and if this copied strategy constitutes a winning strategy then $f(v) = \tilde{z}$. Formally: If both

$$c_k(\hat{x}_k - \tilde{x}_k) + \sum_{\substack{j \in \mathcal{A},\ j>k \\ \text{and } c_j \geq 0}} c_j(1 - \tilde{x}_j) - \sum_{\substack{j \in \mathcal{A},\ j>k \\ \text{and } c_j < 0}} c_j \tilde{x}_j \leq 0 \tag{1}$$

and

$$\sum_{\substack{j \in \mathcal{E} \\ \text{or } j<k}} A_{i,j}\tilde{x}_j + A_{i,k}\hat{x}_k + \sum_{\substack{j \in \mathcal{A},\ j>k \\ \text{and } A_{i,j}>0}} A_{i,j} \leq b_i \tag{2}$$

[3] *Future* means variable blocks with index $\geq k$.

for all constraints $i \in \{1, \dots, m\}$ *then* $f(v) = \tilde{z}$.

For clarification note that Condition (1) ensures that the change in the minimax value of the copied strategy, resulting from flipping x_k and using the worst case assignment of the remaining future universal variables, is not positive, i.e. that its minimax value is still smaller than or equal to $\tilde{z}$. Condition (2) verifies that every constraint is satisfied in each leaf of the copied strategy by ensuring the fulfillment of each constraint in its specific worst case scenario.

Proof. If (2) is satisfied there automatically exists a winning strategy for the subtree of v corresponding to $x_k = \hat{x}_k$ with root node $\hat{w}$, since for any future universal variable assignment the assignment of upcoming existential variables as in $\tilde{x}$ fulfills the constraint system. Further, the minimax value $\hat{z}$ of this strategy is smaller than or equal to $\tilde{z}$ due to Condition (1):

$$\begin{aligned}\hat{z} &= \sum_{\substack{j \in \mathcal{E} \\ \text{or } j<k}} c_j \tilde{x}_j + c_k \hat{x}_k + \sum_{\substack{j \in \mathcal{A},\ j>k \\ \text{and } c_j \geq 0}} c_j \\ &\overset{(1)}{\leq} \sum_{\substack{j \in \mathcal{E} \\ \text{or } j<k}} c_j \tilde{x}_j + c_k \tilde{x}_k + \sum_{j \in \mathcal{A},\ j>k} c_j \tilde{x}_j \quad = \tilde{z}\end{aligned}$$

Hence, the (still unknown) optimal strategy for the subtree below $\hat{w}$ has a minimax value smaller than or equal to $\tilde{z}$, i.e. $f(\hat{w}) \leq \hat{z} \leq \tilde{z} = f(\tilde{w})$. Therefore, $f(v) = f(\tilde{w}) = \tilde{z}$.

Note that, since $A\tilde{x} \leq b$, Condition (2) is trivially fulfilled for any constraint $i \in \{1, \dots, m\}$ with $A_{i,j} = 0$ for all $j \in \mathcal{A}, j \geq k$, i.e. constraints that are not influenced by future universal variables do not need to be examined. Hence, only a limited number of constraints need to be checked in case of a sparse matrix. Further, note that (1) is fulfilled if $c_j = 0$ for all $j \in \mathcal{A}, j \geq k$, i.e. if the future universal variables have no direct effect on the objective value. In particular, if $c = 0$, i.e. it is a satisfiabilty problem rather than an optimization problem, Condition (1) can be neglected as it is always fulfilled.

3.2 SCP Implementation Details

As soon as a leaf v is found during the tree search with the corresponding x_v being a potentially new PV for this subtree the following mechanism is invoked: the two Conditions (1) and (2) of Theorem 1 are checked at each universal node starting from this leaf towards the root (Line 5). While both conditions are fulfilled the corresponding universal nodes are marked as potentially finished. If one of the conditions is not satisfied the remaining universal nodes above are marked as unfinished. If a level is closed during the tree search and the above universal node is marked as potentially finished this level also can be closed immediately as a strategy is guaranteed in the other branch with worse objective value (from the universal player's point of view). The unmarking of universal nodes (Line 8) is necessary since Theorem 1 demands x_v to be the actual PV of this subtree and hence previous markings where made based on a false assumption.

Algorithm 1. Marking of potentially finished universal nodes

Input: leaf node v

```
 1: useSCP=true;
 2: repeat
 3:    $v$=parent($v$);
 4:    if $v \in V_\forall$ then
 5:       if useSCP and $v$ fulfills Conditions (1) and (2) then
 6:          mark $v$ as potentially finished;
 7:       else
 8:          useSCP=false; mark $v$ as unfinished;
 9:       end if
10:    end if
11: until $v$ is root node
```

3.3 Example

Consider the following binary QIP (The min/max alternation in the objective and the binary variable domains are omitted):

$$\begin{array}{rrrrrrl} \min & 2x_1 + & 3x_2 - & 2x_3 - & 2x_4 + & x_5 & \\ \text{s.t.} & \exists x_1 & \forall x_2 & \exists x_3 & \forall x_4 & \exists x_5 & : \\ & x_1 - & x_2 + & x_3 + & 3x_4 - & x_5 & \leq 2 \\ & 3x_1 + & 2x_2 + & 3x_3 + & x_4 - & 2x_5 & \leq 1 \end{array}$$

Starting at the root node of the corresponding game tree we can immediately omit the subtree corresponding to $x_1 = 1$ due to the monotonicity of x_1. Keep in mind that the result of Theorem 1 is particularly beneficial if the search process of a QIP solver first examines the principal variation, i.e. the variable assignment defining the actual minimax value. Assume the search process follows the path drawn thick in Fig. 3 to node v_8, i.e. the path corresponding to the variable assignment $x_1 = 0$, $x_2 = 1$, $x_3 = 0$ and $x_4 = 0$. Setting $x_5 = 1$ is optimal in this case, as $x_5 = 0$ would violate the second constraint. Hence, the minimax value of v_8 is 4. On the way up in the search tree we then want to determine $f(v_5)$. As (1) and (2) are fulfilled for $k = 4$, $\tilde{z} = 4$ and $\tilde{x} = (0, 1, 0, 0, 1)$ we know that $f(v_5) = 4$. That means we have (easily) verified a winning strategy starting from v_9 with minimax value smaller than or equal to 4. In node v_3 setting $x_3 = 1$ is obviously to the detriment of the existential player, as the second constraint would become unfulfillable. Hence, $f(v_3) = f(v_5) = 4$. In node v_1 we once again try to apply Theorem 1 by copying the existential decisions of x_3 and x_5 in the thick path to the not yet investigated subtree associated with $x_2 = 0$. As (1) and (2) are fulfilled for $k = 2$, $\tilde{z} = 4$ and $\tilde{x} = (0, 1, 0, 0, 1)$ this attempt is successful and $f(v_1) = 4$. Note that by applying Theorem 1 the minimax value of the subtrees below v_2 and v_9 are not known exactly: in particular we only obtain $f(v_2) \leq \hat{z} = 1$, whereas a better strategy exists resulting in $f(v_2) = 0$ (Setting $x_5 = 0$ in node v_6).

Hence, by finding the principal variation first (thick path), exploiting monotonicity of x_1 at node v_0, Theorem 1 at node v_1 and v_5 and some further reasoning

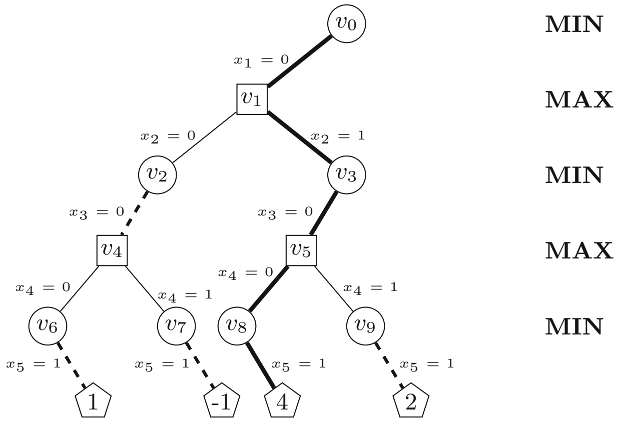

Fig. 3. Optimal winning strategy for the QIP. Circular nodes are existential decision nodes, rectangular nodes are universal decision nodes and pentagonal nodes are leaves. The values given in the leaves constitute the objective value corresponding to the variable assignment along the path from the root to this leaf. The dashed lines indicate that those existential decisions where simply copied from the path drawn thicker.

from linear programming at node v_3 and v_8 the minimax value at the root node v_0 was found to be 4 with optimal first stage solution $x_1 = 0$.

Theorem 1 can particularly come into effect if the branching decisions at universal nodes result in rather vicious scenarios, i.e. in variable assignments restricting the constraint system and maximizing the objective value. Hence, the applicability of the presented results largely depends on the implemented diving and sorting heuristic.

4 Solver, Experiments and Results

We use our open source[4] solver Yasol [9] to analyze the theoretical findings. The solver basically performs an enhanced alpha-beta-search and proceeds in two phases: a *feasibility phase*, where it is checked whether the instance has any solution at all, and an *optimization phase* for finding the provable optimal solution. We enhanced this solver in two different ways:

1. The detection and exploitation of monotone variables.
2. The adoption of existing winning strategies from one branch of a universal node to another (SCP).

The SCP-enhancement can be switched on and off in both phases separately.

The instances used to study the effect of the presented results are runway scheduling problems under uncertainty as motivated in the introduction.

[4] Sources are available at http://www.q-mip.org.

They were created following the ideas presented in [13]. The task is to find a b-matching: all airplanes must be assigned to exactly one time slot, while one time slot can take in at most b airplanes. Furthermore, the airplanes must land within an uncertain time window. Hence, we are interested in an initial matching plan that can be fixed cheaply if the mandatory time windows for some airplanes do not contain the initially scheduled time slot. The testset contains 29 instances[5], varying in the number of airplanes, the number of time slots, the type of allowed disturbances, the number of universal blocks and the cost function. In terms of the sizes of the (solved feasible) instances this results in between 100–300 existential variables, 10–30 universal variables and 50–100 constraints.

Table 1. Number of solved instances dependend on the solver setting: exploitation of monotone variables and SCP in different phases.

monotonicity	SCP	solved instances
off	both phases	24
	only optimization phase	21
	only feasibility phase	16
	off	14
on	both phases	25
	only optimization phase	25
	only feasibility phase	24
	off	23

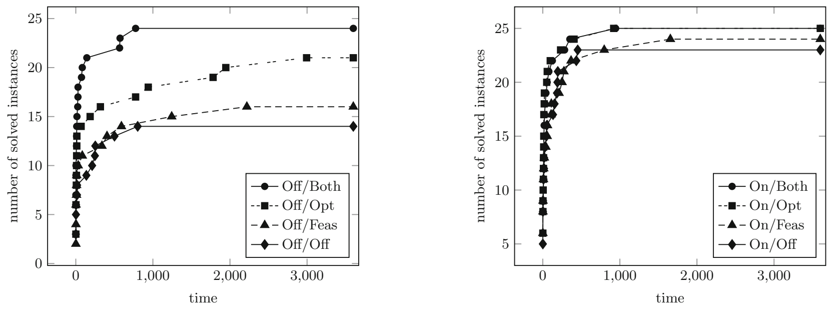

Fig. 4. Comparison of the effect of SCP in different phases of the solver without exploiting monotonicity (left) and when also exploiting monotonicity (right).

In Table 1 and Fig. 4 the number of solved instances and the cumulative solution diagram is displayed for different settings. For each instance a maximum

[5] The studied benchmark instances and a brief explanation can be found at http://www.q-mip.org/index.php?id=41.

Table 2. Average time needed for the 23 instances solved in all four settings while exploiting monotonicity.

SCP setting	off	only feas	only opt	both
average runtime	84s	102s	25s	32s

of one hour solution time was provided. All experiments were executed on a PC with an Intel i7-4790 (3.6 GHz) processor and 32 GB RAM. If neither of the presented procedures is used 14 out of 29 instances are solved. Without taking advantage of the monotonicity SCP can be beneficial in either solution phase regarding the number of solved instances. If applied in both phases the number of solved instances is increased up to 24. When also exploiting the monotonicity the number of solved instances increases to 25. However, SCP turns out to be somewhat disadvantageous in the feasibility phase. Even though an additional instance is solved (24) compared to the setting with SCP turned off (23) the average solution time increases: in Table 2 the average time needed for the 23 instances solved by all versions with turned on monotonicity is displayed. Additionally using SCP in the feasibility phase slightly increases the average solution time. Our conjecture is that this is due to biasing effects. Four instances with more than 100 universal variables and 10000 existential variables were not solved at all. However, there also are infeasible instances of the same magnitude that are solved within seconds. In order to assess the performance results, we also built the deterministic equivalent program of each instance and tried to solve the resulting integer program using CPLEX 12.6.1.0, a standard MIP solver. Only six of the 29 instances where solved this way, given the same amount of time, while for 14 instances not even the construction of the corresponding DEP could be finished, some of them because of the limited memory of 32 GB RAM. Experiments conducted on a QBF test collection of 797 instances, taken from www.qbflib.org, also show positive effects for the SCP version. When only exploiting monotonicity 644 instances are solved. If additionally SCP is turned on 674 instances can be solved. Further, the solution time decreases by 15%.

5 Conclusion

We introduced the concept of strategic copy-pruning (SCP) during tree search for quantified integer programs. SCP makes it possible to omit certain subtrees during the search process by implicitly verifying the existence of a strategy in linear time: finding a single leaf and applying SCP can be sufficient to guarantee an optimal strategy in an entire subtree. This is standing in contrast to existing algorithms in which the existence of a strategy is proven by traversing it explicitly. In addition, we presented how those findings can be applied in a search environment. Experiments showed that utilizing our approach resulted in a massive boost in both the number of solved instances and the solution time (about 4 times faster) on a particular testset. The achievement opens the door to solving larger and more complex real-world problems.

References

1. Akl, S., Newborn, M.: The principal continuation and the killer heuristic. In: ACM 1977, pp. 466–473 (1977)
2. Bellman, R.: Dynamic Programming. Dover Publications Incorporated, Mineola (2003)
3. Ben-Tal, A., Ghaoui, L.E., Nemirovski, A.: Robust Optimization. Princeton University Press, Princeton (2009)
4. Bertsimas, D., Brown, D., Caramanis, C.: Theory and applications of robust optimization. SIAM Rev. **53**(3), 464–501 (2011)
5. Birge, J.R., Louveaux, F.: Introduction to Stochastic Programming. Springer, New York (2011). https://doi.org/10.1007/978-1-4614-0237-4
6. Cadoli, M., Schaerf, M., Giovanardi, A., Giovanardi, M.: An algorithm to evaluate quantified Boolean formulae and its experimental evaluation. J. Autom. Reasoning **28**(2), 101–142 (2002)
7. Campbell, M., Hoane, A., Hsu, F.H.: Search control methods in deep blue. In: AAAI Spring Symposium on Search Techniques for Problem Solving under Uncertainty and Incomplete Information, pp. 19–23 (1999)
8. Campbell, M., Marsland, T.: A comparison of minimax tree search algorithms. Artif. Intell. **20**(4), 347–367 (1983)
9. Ederer, T., Hartisch, M., Lorenz, U., Opfer, T., Wolf, J.: Yasol: an open source solver for quantified mixed integer programs. In: Winands, M.H.M., van den Herik, H.J., Kosters, W.A. (eds.) ACG 2017. LNCS, vol. 10664, pp. 224–233. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-71649-7_19
10. Gerber, R., Pugh, W., Saksena, M.: Parametric dispatching of hard real-time tasks. IEEE Trans. Comput. **44**(3), 471–479 (1995)
11. Gupta, A., Pál, M., Ravi, R., Sinha, A.: Boosted sampling: approximation algorithms for stochastic optimization. In: ACM 2004, pp. 417–426. ACM (2004)
12. Hartisch, M., Ederer, T., Lorenz, U., Wolf, J.: Quantified integer programs with polyhedral uncertainty set. In: Plaat, A., Kosters, W., van den Herik, J. (eds.) CG 2016. LNCS, vol. 10068, pp. 156–166. Springer, Cham (2016). https://doi.org/10.1007/978-3-319-50935-8_15
13. Heidt, A., Helmke, H., Kapolke, M., Liers, F., Martin, A.: Robust runway scheduling under uncertain conditions. JATM **56**, 28–37 (2016)
14. Helmke, H.: Scheduling algorithms for ATM applications–tools and toys. In: 2011 IEEE/AIAA 30th Digital Avionics Systems Conference, p. 3C2-1. IEEE (2011)
15. van den Herik, H., Nunn, J., Levy, D.: Adams outclassed by hydra. ICGA J. **28**(2), 107–110 (2005)
16. Janota, M., Klieber, W., Marques-Silva, J., Clarke, E.: Solving QBF with counterexample guided refinement. Artif. Intell. **234**, 1–25 (2016)
17. Kawano, Y.: Using similar positions to search game trees. Games No Chance **29**, 193–202 (1996)
18. Knuth, D., Moore, R.: An analysis of alpha-beta pruning. Artif. Intell. **6**(4), 293–326 (1975)
19. Lorenz, U., Wolf, J.: Solving multistage quantified linear optimization problems with the alpha-beta nested benders decomposition. EURO J. Comput. Optim. **3**(4), 349–370 (2015)
20. Nemhauser, G., Wolsey, L.: Integer and Combinatorial Optimization. Wiley-Interscience, New York (1988)

21. Nguyen, D., Kumar, A., Lau, H.: Collective multiagent sequential decision making under uncertainty. In: AAAI 2017. AAAI Press (2017)
22. Pijls, W., de Bruin, A.: Game tree algorithms and solution trees. Theoret. Comput. Sci. **252**(1), 197–215 (2001)
23. Plaat, A., Schaeffer, J., Pijls, W., de Bruin, A.: Best-first fixed-depth minimax algorithms. Artif. Intell. **87**(1–2), 255–293 (1996)
24. Reinefeld, A.: An improvement to the scout tree search algorithm. ICGA J. **6**(4), 4–14 (1983)
25. Schaeffer, J.: The history heuristic and alpha-beta search enhancements in practice. IEEE Trans. Pattern Anal. Mach. Intell. **11**(11), 1203–1212 (1989)
26. Silver, D., et al.: Mastering the game of go with deep neural networks and tree search. Nature **529**, 484–503 (2016)
27. Subramani, K.: Analyzing selected quantified integer programs. In: Basin, D., Rusinowitch, M. (eds.) IJCAR 2004. LNCS (LNAI), vol. 3097, pp. 342–356. Springer, Heidelberg (2004). https://doi.org/10.1007/978-3-540-25984-8_26
28. Winands, M., van den Herik, H., Uiterwijk, J., van der Werf, E.: Enhanced forward pruning. Inf. Sci. **175**(4), 315–329 (2005)
29. Zhang, L.: Searching for truth: techniques for satisfiability of Boolean formulas. Ph.D. thesis, Princeton, USA (2003)