An algorithmic framework for 0/1-QIP solvers

(extended abstract) ⋆

T. Ederer[2], U. Lorenz[1], T. Opfer[2], J. Wolf[2]
{ederer,opfer,wolf}@mathematik.tu-darmstadt.de,
ulf.lorenz@fst.tu-darmstadt.de

[1]Fluid Systems Technology, Technische Universität Darmstadt, Germany
[2]Institute of Mathematics, Technische Universität Darmstadt, Germany

Abstract. Quantified linear programs (QLPs) are linear programs with variables being either existentially or universally quantified. The integer variant (QIP) is PSPACE-complete, and the problem is similar to games like chess, where an existential and a universal player have to play a two-person-zero-sum game. We present the algorithmic concepts of our solver Yasol which solves multistage 0/1-QIPs to optimality with up to several hundred existential and two dozens of universal variables. Being a mixture of a QSAT solver and a branch&cut IP-solver, Yasols opens a new category of quantifier-extended IP solvers.

1 Introduction

Mixed-integer linear programming (MIP) [1] is the state-of-the art technique for computer aided optimization of real world problems. Nowadays, we are able to solve large MIPs of practical size, but companies observe an increasing danger of disruptions, which prevent them from acting as planned. One reason is that input data for a given problem is often assumed to be deterministic and exactly known when decisions have to be made, but in reality they are often afflicted with some kinds of uncertainties. Examples are flight and travel times, throughput-time, or arrival times of externally produced goods. Thus, there is a need for planning and deciding under uncertainty. Uncertainty, however, often pushes the complexity of problems that are in P or NP, to the complexity class PSPACE [2]. Therefore, NP-complete integer programs are not suitable to model these problems anymore. Prominent solution paradigms for optimization under uncertainty are Stochastic Programming [3], Robust Optimization [4,5], Dynamic Programming [6], Sampling [7] and others, e.g. LP-based approximation techniques [8,9,10].

Relatively unexplored are the abilities of linear programming extensions for PSPACE-complete problems. In this context, Subramani introduced the notion of quantified linear programs (QLPs) [11]. Quantified Linear (Integer) Programming (QIP) obviously gives the opportunity to combine traditional linear programming formulations with some uncertainty bits. From our point of view it

⋆ Research partially supported by German Research Foundation (DFG), Lo 1396/2-1 and SFB 805

is therefore a highly interesting object for algorithmic research. The expressive power of QIPs allows to model many existing PSPACE complete problems as e.g. QSAT. A model for the two person parlor games Gomoku [12] and connect-6 (QIP) were built, as well as the optimal outline of a booster station (QMIP, quantified mixed integer program)[1], and an optimization problem in airline industry (QIP) is in preparation. However, modeling is not the topic of this paper.

The currently best practical way to solve e.g. the small booster example is to build a so called deterministic equivalent program [13] (DEP) and to solve the resulting MIP. We investigate the limitations of the deterministic equivalents which motivate us to build an integrated solver that does not build or examine a deterministic equivalent program. Consequently, we care for the following research questions concerning the multistage QIP optimization problem.

Is it possible to compete with solvers like Cplex, Gurobi or Scip when they solve the DEPs of QIPs? There are several rationales for it. Firstly, building the deterministic equivalent program takes exponential time because there are exponentially many scenarios in the number of universal variables. If we solve LP-relaxed subproblems with the dual simplex algorithm, the total process might take double exponential time in its worst case. From the viewpoint of complexity, this sounds inadequate for a PSPACE-complete problem. Secondly, all current MIP-solvers walk through a B&B tree and examine millions of search nodes when the instance is large enough. Why should it be impossible to go through the corresponding AND/OR search graph of QIPs with a similar amount of time? However, the situation is far less simple. The success of MIP modeling bases on the fact that the hardness of the problem does not come up in many practice models. Thus it can be doubted that the theoretically possible double exponential running time will be observable. When we additionally aim at 20 to 30 universal variables (i.e. 2^{20} to 2^{30} scenarios), the hardness also does not necessarily show up in the DEP. Moreover, for fundamental reasons, the traditional cut&branch techniques cannot be directly transferred to QIP solution algorithms. And also for the branch&cut mechanism, it is not clear, which of the known cutting planes for IPs can be transferred into the quantified field. They are mainly designed to cut away some pieces of the LP relaxation, approximating the convex hull of all integer points. However, they do possibly not even touch the set of points which belong to solution spaces of QIPs. *The second question then is how such a search procedure can be organized.*

For ease of computation, we only consider 0/1 QIPs in the remainder of this paper. Nevertheless, all the techniques described can be applied to any QIP containing 0/1 variables. The rest of this paper is organized as follows. In section 2.1 we recapture some important results from QLP theory. Section 2.2 then deals with the three major ingredients of our solution algorithm: the reasoning mechanism known for SAT-solvers, the Alphabeta game tree search

[1] for reviewer: we provide supplementary material on this webpage: http://www3.mathematik.tu-darmstadt.de/index.php?id=2001. E.g. the booster example.

algorithm, and dual information of ordinary LPs. Section 3 is the experimental result section and at the end, we give an outlook for further investigations.

2 Quantified Linear Programming (QLP), and Quantified Linear Integer Programming (QIP)

In this subsection, we recapture the major definitions within the context of Quantified Linear Programs, beginning with the continuous problem. The latter gives several polyhedral insights which are important to note also for the engineering of integer algorithms. The most important points are a) the QLP can be written in DEP form and can therefore be decomposed and solved with Bender's decomposition, b) the solution space is polyhedral and all points in that polyhedron may be geometrically far away from the optimal LP or IP point. Moreover, we introduce the additional feature of a objective function for QLPs and QIPs, in analogy to conventional mathematical programming.

2.1 The continuous QLP problem

Definition 1 (Quantified Linear Program) *Let there be a vector of n variables $x = (x_1, \ldots, x_n)^T \in \mathbb{Q}^n$, lower and upper bounds $l \in \mathbb{Z}^n$ and $u \in \mathbb{Z}^n$ with $l_i \leq x_i \leq u_i$, a coefficient matrix $A \in \mathbb{Q}^{m \times n}$, a right-hand side vector $b \in \mathbb{Q}^m$ and a vector of quantifiers $\mathcal{Q} = (\mathcal{Q}_1, \ldots, \mathcal{Q}_n)^T \in \{\forall, \exists\}^n$, let the term $\mathcal{Q} \circ x \in [l, u]$ with the component wise binding operator $\circ$ denote the* quantification vector $(\mathcal{Q}_1 x_1 \in [l_1, u_1], \ldots, \mathcal{Q}_n x_n \in [l_n, u_n])^T$ *such that every quantifier $\mathcal{Q}_i$ binds the variable x_i ranging over the interval $[l_i, u_i]$. We call $(\mathcal{Q}, l, u, A, b)$ with*

$$\mathcal{Q} \circ x \in [l, u] : Ax \leq b \tag{QLP}$$

a quantified linear program (QLP).

Each maximal consecutive subsequence of $\mathcal{Q}$ consisting of identical quantifiers is called a *quantifier block* – the corresponding subsequence of x is called a *variable block*. The i-th quantifier block is denoted by $\mathcal{Q}^i \in \{\forall, \exists\}$, likewise x^i denotes the corresponding variable block.The total number of blocks less one is the number of *quantifier changes*.
A QLP instance is interpreted as a two-person zero-sum game between an *existential player* setting the $\exists$-variables and a *universal player* setting the $\forall$-variables. Each fixed vector $x \in [l, u]$, that is, when the existential player has fixed the existential variables and the universal player has fixed the universal variables, is called *a game*. If x satisfies the linear program $Ax \leq b$, we say *the existential player wins*, otherwise *he loses* and *the universal player wins*. The variables are set in consecutive order according to the variable sequence. Consequently, we say that a player makes the move $x_k = z$, if he fixes the variable x_k to the value z. At each such move, the corresponding player knows the settings of $x_1, \ldots, x_{k-1}$ before taking his decision x_k.

In [14] it was shown that the solution space of a QLP with n variables forms a polytope in $\mathbb{R}^n$, which is included in the polytope induced by the constraint set $Ax \leq b$ as shown in Figure 1. This insight implies that, in general, a cut&branch approach is not target leading. E.g., in Fig. 1, it can be observed that all points belonging to any solutions (dark region) are far away from an optimal LP relaxation solution, when we assume that the optimization aim is maximizing x_1. It was furthermore shown that it suffices to inspect the bounds of the universal quantified variables in order to check whether a *winning policy* does exist [14]). Thus, in order to answer the question whether the existential player can certainly win the game, it is sufficient to analyze so called strategies, and as a consequence, QLPs can be solved with the help of DEPs.

Definition 2 (Strategy) *A* strategy $S = (V, E, c)$ *is an edge-labeled finite arborescence with a set of nodes* $V = V_\exists \dot\cup V_\forall$*, a set of edges* E *and a vector of edge labels* $c \in \mathbb{Q}^{|E|}$*. Each level of the tree consists either of only nodes from* $V_\exists$ *or only of nodes from* $V_\forall$*, with the root node at level 0 being from* $V_\exists$*. The* i*-th variable of the QLP is represented by the inner nodes at depth* $i-1$*. Each edge connects a node in some level* i *to a node in level* $i+1$*. Outgoing edges represent moves of the player at the current node, the corresponding edge labels encode the variable allocations of the move. Each node* $v_\exists \in V_\exists$ *has exactly one child, and each node* $v_\forall \in V_\forall$ *has as two children, with the edge labels being the corresponding upper lower and upper bounds.*

A path from the root to a leaf represents a game of the QLP and the sequence of edge labels encodes its moves. A strategy is called a *winning strategy* if all paths from the root node to a leaf represent a vector x such that $Ax \leq b$. This terminology is also very similarly used in game-tree search [15].

Example 1 *The QLP*

$$\exists\, x_1 \in [0,1]\, \forall\, x_2 \in [0,1]\, \exists\, x_3 \in [0,1] :$$

$$\begin{pmatrix} 0 & -1 & -1 \\ -1 & 1 & 1 \\ 2 & 2 & 0 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix} \leq \begin{pmatrix} -1 \\ 1 \\ 3 \end{pmatrix}$$

has two quantifier changes. Figure 1 shows a visualization of the constraint polyhedron restricted to the unit cube. A simple rule-based description of the winning strategy is as follows. 'Choose $x_1 \in [0, \frac{1}{2}]$*, then choose* x_3 *apt to* x_2*, e.g.* $x_3 = 1 - x_2$*.' The highlighted solution space visualizes the set of all games with a definite winning outcome for the existential player. Figure 2 shows a winning-strategy for the existential player.*

Further details can be taken from [14] [2]. If there is more than one winning strategy for the existential player, it can be reasonable to search for a certain

[2] or, corrected and better presented in a working paper, submitted to journal. Please, find a link at http://www.mathematik.tu-darmstadt.de/preprint.php?id=2650

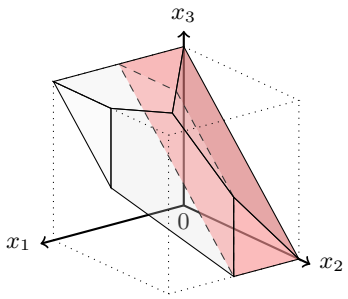

Fig. 1: Constraint polyhedron

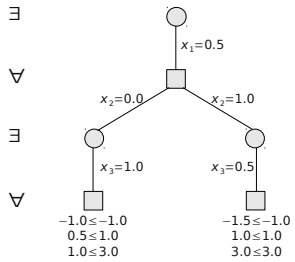

Fig. 2: Winning Strategy

(the 'best') one. We therefore modify the problem to include a linear objective function as shown in the following (where we note that transposes are suppressed when they are clear from the context to avoid excessive notation).

Definition 3 (QLPs/QIPs with Objective Function) *Let $\mathcal{Q} \circ x \in [l, u] : Ax \leq b$ be given as in Definition 1 with the variable blocks being denoted by B_i. Let there also be a vector of objective coefficients $c \in \mathbb{Q}^n$. We call*

$$z = \min_{B_1}(c^1x^1 + \max_{B_2}(c^2x^2 + \min_{B_3}(c^3x^3 + \max_{B_4}(\ \ldots \min_{B_m} c^m x^m))))$$
$$\mathcal{Q} \circ x \in [l, u] : Ax \leq b$$

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

Note that the variable vectors $x^1, ..., x^i$ are fixed when a player minimizes or maximizes over variable block B_{i+1}. Consequently, we deal with a dynamic multistage decision process, similar as it is also known from multistage stochastic programming [3] [3].

2.2 Quantified Integer Programming

As with linear programs and integer programs, we can restrict the variables of a QLP to a discrete domain. To the best of out knowledge, no practically useful solver for Quantified Mixed Integer Linear Programming is available at the moment, not even when we restrict the domains of all integer variables to 0 and 1 and allow continuous variables only in the last stage with existential variables. Up to now, the only practical way to solve e.g. our small booster station example was to build a so called deterministic equivalent program [16,13] and to solve the resulting MIP, even when all variables are binary.

We investigate the limits of this technique in the field of quantified linear programs, and our aim is to construct an integrated algorithm which does not need to examine a deterministic equivalent program. Yasol, the ideas of which are presented in this paper, proceeds in two phases as described in subsection 2.2 in order to find optimal solutions of 0/1-QIP instances.

[3] for reviewer: further details can be found in a working paper at http://www.mathematik.tu-darmstadt.de/preprint.php?id=2660

– Phase 1: Determine the instance's feasibility, i.e. whether the instance has any solution at all. If it has any solution, present it. During this phase, Yasol acts like a QBF solver [17] with some extra abilities.
– Phase 2: Go through the solution space and find the provable optimal solution.

Algorithm 1: Function alphabeta(int d, int a, int b) // A basic alphabeta algorithm with LP relaxation based bounding

```
 1 if all variables are fixed then return objective value ; // leaf reached
 2 val[0] := 0; val[1] := 1; // standard polarity
 3 x_i := getNextBranchingVariable(); // phase dependent
 4 if instance has objective and a dual bound is desired then
 5 |   sample the universal variables;
 6 |   adjust the LP, solve it, and extract a branching variable or a cut;
   end
 7 probe(...) ; // cf. [18]
 8 if x_i is an existential variable then  score := −∞;
 9 else  score := +∞;
10 for val_ix from 0 to 1 do // search actually begins ...
11 |   if level_finished(t) then // leave marked recursion levels
12 |   |   if x_i is an extistential variable then  return score ;
13 |   |   else  return −∞ ;
   |   end
14 |   assign(x_i, val[val_ix], ... );
15 |   v := alphabeta(d-1, fmax(a, score), b);
16 |   unassign(x_i);
17 |   if x_i is an existential variable then
18 |   |   if v > score then  score := v; // existential player maximizes
19 |   |   if score ≥ b then  return score ;
   |   else
20 |   |   if v < score then  score := v;
21 |   |   if score ≤ a then
22 |   |   |   Explore the Quantified LP relaxation;
23 |   |   |   return score;
   |   |   end
   |   end
   end
24 return score;
```

In the following, we describe a rudimentary 0/1-QIP-solver that is based on an alphabeta algorithm with boolean constraint propagation, non-chronologic backtracking and restarts as known from QSAT [17] solving on the one hand, and on the generation of cutting planes with the help of the LP-dual on the other hand. We start with the description of the second phase. The alphabeta algorithm replaces the DPLL algorithm because the latter is less suited for optimization problems with an objective function. It walks through the search space recursively and fixes variables to 0 or 1 when going into depth or relaxes the fixations again when returning from a subtree.

A QIP allows at least two natural relaxations. One could relax the integrality or the universal property of some variables. After all, relaxing both results in an ordinary LP which gives us the opportunity to cut off parts of the search tree with the help of dual information, i.e. dual bounds or cutting planes for the original program. Algorithm 1 shows a basic alphabeta algorithm with the ability of non-chronologic back-jumping with the help of dual information, i.e. by solving an LP-relaxation, cf. lines 4-6 and 11-13 of Algorithm 1. The idea is quite old and goes back to Benders and has been described already in the seventies [19]. For a long time it has been thought that these cuts are too weak for driving the search. However, since in [18] the technique has been improved in combination with an implication graph, the cuts can often be made quite sparse.

Algorithm 2: The local repetition loop which extends the basic algorithm with conflict analysis and learning; replaces line 15 in Algorithm 1

```
1  repeat
2      if level_finished[t] then // leave the recursion level
3          unassign(x_i);
4          if x_i is an existential variable then return score;
           else return −∞ ;
       end
5      if propagate(confl, confl_var, confl_partner, false) // unit prop. [17]
       then
6          if x_i is an existential variable then
7              v = alphabeta(t+1, lsd-1, fmax(a, score), b);
8              if v > score then score := v;
9              if score ≥ b then break;
           else
10             v := alphabeta(t+1, lsd-1, a, fmin(score, b));
11             if v < score then score := v;
12             if score ≤ a then break;
           end
       else
13         add reason, i.e. a constraint, to the database ; // [17]
14         if limit of conflicts reached then initiate restart ; // cf. [20]
15         returnUntil(out_target_dec_level) ; // set level_finished(...)
16         if x_i is an existential variable then return score;
17         else return −∞ ;
       end
   until there is a backward implied variable;
```

In principle, this works as follows. Let $max\{c_1^t x_1 + c_2^t x_2 | A_1 x_1 + A_2 x_2 \leq b; x_1, x_2 \in \{0,1\}\}$ be the optimization problem and let x_1 be the part of the x-vector that has temporarily been fixed by the alphabeta depth first search. Then, free universal variables are sampled and the variable bounds of them are set to 0 or 1 in the LP. The sampling is useful because every choice of the universal variables leads to a relaxation of the original subproblem. The LP-relaxation with some fixed variables gives us an upper bound on the objective value or, if the LP is already infeasible, a dual ray that is used to construct a new

constraint for the original system of constraints. The dual of the LP-relaxation has the form $min\{\pi^t(b - A_1x_1|A_2^t\pi \geq c_2; \pi \leq 0)\}$ and the Farkas lemma leads to the feasibility cut $(\pi^t)(b - A_1x_1) \leq 0$. This feasibility cut describes a subset of the fixed variables one of which must be changed from 1 to 0 or vice versa. The cut is added to the database of cuts on the integer side. No cuts are added to the LP. The new cut possibly indicates that the search process can be directly continued several recursion levels above the current level. In this case, the superfluous levels are marked as finished - in Algorithm 1 that is indicated by level_finished(t) - and the alphabeta procedure breaks out of all loops and walks up the search tree with the help of the lines 11 to 13. A further important design decision for every solver is which branching variables is selected and whether the zero or the one assignment of this variable is preferred first. Concerning this issue during the second phase, we refer to [18].

Still missing, however, is the opportunity to propagate implied variables and to organize backward implications as SAT solvers or modern IP solvers like Scip [18] perform them. The non-chronologic backtracking is realized with the help of Algorithm 2, which replaces line 15 of Algorithm 1. In fact, there is a loop around the alphabeta procedure call which is walked through as long as backward implied variables occur deeper in the search tree. The procedure `propagate(.)` performs the implication of variables. In the context of boolean formulae, this process works identically with unit propagation. In more general constraints, further implications or bound propagations can be implied. The method `addReasonCut(.)` implements the learning algorithm as used by SAT solvers and modern IP-solvers. With the help of the original and the learnt constraints, the algorithm implicitly maintains an implication graph as described in [18]. The implication graph is additionally used to strengthen the new constraints learnt in Algorithm 1, ll. 11-13. Because it is more convenient to deal with feasibility cuts than with objective cuts, an artificial constraint is contained in the IP-constraint database as well as in the LP relaxation, i.e. the objective with the best known solution value at right hand side.

There are two differences to ordinary IP-solving which are worth to be considered. The given procedure can only deliver a globally valid constraint while in some parts of the search tree, a locally valid constraint being derived from the α bound, is desired. This is the first difference.

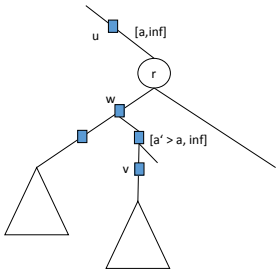

Fig. 3: Evolution of α and β

Thus, when a search node v is entered there are three different bounds which can be used for cutting off the search subtree. Firstly, there is the value of the current best solution, which gives a global bound, let us call it z. Additionally, there are α and β. If we find a solution better then β, the search can leave v. If the LP-relaxation optimal value $z_r \leq z$, a new global cut can be derived and added to the constraint database. Last but not least, if $z \leq z_r < \alpha$, a local cut can be generated

that is not supplied to the database. However, and this is the second difference to ordinary IP-solving, it can be used to compute a back-jump level in the tree, where it must be considered not to cross a universal node.

Fig. 3 shows an example. Starting from the existential node u with alphabeta window $[a, \infty]$, the universal node r is reached. After the left branch of the existential node w is completed, node v can be reached with the window $[a', \infty]$ with $a' > a$. A conflict analysis at v may result with the insight 'in order to increase the value at the root node, the parent of v must branch to the right and this is implied already in w'. In this case, a back jump to w is possible. If the conflict analysis results in the insight '... and this is implied already in node u', the search can proceed in node r, but not in u, because the right branch of r might not be irrelevant. The fact that the solver cannot cross the node u is relevant, as we could observe that the solver sometimes is stuck at universal nodes. In contrast to an IP-solver that computes the DEP, our solver has LP-information for only one scenario at a time. Therefore, there is line 22 in Algorithm 1. The QLP-relaxation attenuates the described effect.

During the first phase, the algorithm does not use the LP relaxations and proceeds more like a QBF solver as described in [17]. The alphabeta search routine is embedded in an outer loop and restarts are controlled by the number of conflicts and the Luby-Sequence [20], where we use 128 as the Luby-unit. Especially SAT solvers tend to an increasing rate of restarts [21]. The determination of the next branching variable is determined with the help of two statistical values, i.e. $p\_\ activity_x$ and $n\_\ activity_x$. These two parameters describe how often a variable x has been involved into a conflict when it was temporarily fixed to 0 or 1. Whether the chosen variable is set to 0 or to 1 first, is also determined with the help of these statistical values plus some randomization which hinders the search process from being stuck in a local area of the search tree.

3 Computational results

In this section, we report on computational experiments undertaken in order to show the effectivity of the algorithm described in the previous section. Depending on the type of test instances, i.e. QFB or 0/1-QIP with objective, we competed with Cplex solving a DEP and with Depqbf, one of the leading QBF-solvers. In the context of 0/1-QIPs, we passed on the QBF-solver and took into additional consideration Gurobi and Scip. Our own prototype is called Yasol, which currently utilizes the Cplex LP-solver as a subroutine. In order to keep a certain fairness, also Scip has been compiled with the Cplex LP solver. Therefore, the underlying LP solver should not influence the results. All experiments were done on PCs with Intel i7 (3.6 GHz) processors and 64 GB RAM. All computations were limited to one thread, all other parameters were left at default.

3.1 Results for QBF

At the current state, our solver Yasol can solve QSAT instances reasonable well. This can be taken from the tables 1 and 2. Quaffle, Cube and Depqbf are high end QBF solvers that were downloaded from the Internet. Cplex (V 12.5.0.1) and Gurobi (V 5.5) are two of the leading MIP solvers and DEP Cplex and

DEP Gurobi operate with the corresponding MIP solvers on the deterministic equivalents of the QBF instances which have been translated from boolean formulae format to MIP format. Scip (V 3.0.1) is the best open-source MIP solver at the current time. The first test collection, cf. Table 1, consists of 389 instances with only a small number of universal variables. We see that the DEP approach works surprisingly well. Cplex could solve 307 of the instances, not less then Depqbf, one of the best QBF solvers. Yasol, our own solver solves 10 percent less instances, but at least as many as Scip. Each program got a maximum of 5 minutes solution time for each instance.

	DEP cplex	DEP Gurobi	DEP Scip	Yasol	Quaffle	Cube	Depqbf
Time	29596s	31960s	41704s	36091s	28053s	29877s	26764s
#solved	307/389	300/389	271/389	277/389	303/389	299/389	307/389

Table 1: Computational Results on QBF

The presented time is taken as the sum of all solution times for the instances punishing a non-solved instance with 300 seconds.

The ranking changes when we increase the test set and examine test instances with an increasing number of universal variables. On a test collection of 797 instances, taken from the qbflib.org, Depqbf solves 674 instances, Yasol 584, but the DEP approach collapses. Cplex can solve only 478. Each program got a maximum of 10 minutes solution time for each instance. Again, the solution time consists of the sum of all individual solution times for a specific program. Not solving the instance was punished with 600 seconds. Table 2 shows the

# univ. var.		Depqbf	DEP cplex	Yasol
1-5	Time	44243s	42286s	69275
	#solved	306/373	313/373	264/373
6-10	Time	1810s	1314s	3125s
	#solved	38/41	39/41	36/41
11-15	Time	8242s	22862s	15083s
	#solved	84/97	65/97	79/97
16-20	Time	2757s	79301s	12922s
	#solved	166/170	61/170	156/170
21+	Time	26858s	50017s[a]	49523s
	#solved	80/116	0/116	49/116
Σ	Time	83910s	195780s	149928s
Σ	#solved	674/797	478/797	584/797

Table 2: Computational Results

	Gurobi	Cplex	Yasol	Scip
air04	25	11	117	33
air05	14	13	2698	11
mitre	1	1	2	5
p0033	0	0	0	0
p0201	0	0	6	0
p0282	0	0	1	0
cap6000	1	1	3600	2
harp2	37	19	3600	262
lseu	0	0	16	0
p0548	0	0	605	0
nw04	2	110	3600	23
mod008	0	0	151	0

Table 3: 0/1-IPs, taken from miplib.zib.de

[a] the small number is due to aborting, Cplex could not solve the instances with 64GB memory

number of solved instances grouped by the number of universal variables. We see that the DEP approach becomes more and more unalluring, the more universal variables come into play. Experimental results draw a similar picture when we examine 0/1-QIPs with objective. In order to examine the effects, we picked out

the 0/1 IPs from the miplib test set. Compared with the MIP solvers Cplex, Scip and Gurobi, Yasol performs poor as can be taken from table 3.

We generated quantified instances with the help of the lseu and the p0282 instances as follows. Firstly, a certain amount of the IP-variables was converted to universal ones. Then, at randomly chosen positions, the universal variable blocks were incorporated, such that the universal blocks do not overlap. One additional instance, an encoding of a game position of the connect-6 game could not be solved by any of the solvers. We also applied this procedure to the other 0/1-IP instances, but they were trivially infeasible in the sense that our QLP-solver could detect this unfeasibility in short time.

3.2 Results for QIP

Finally, table 4 shows the results on modified instances, where up to 16 of the variables are converted to universal ones. In the first column, we see the name of the IP base instance, the number of universal variable blocks with their lengths, and the number of instances of this kind in brackets. E.g. 'lseu 7x2A (5)' describes 5 instances with 14 universal variables, grouped into 7 pieces with 2 universal variables in each block. The other cells describe, which solver has solved how many of the instances within 600 seconds and, in brackets, the average number (i.e. the sum divided by the number of instances) of used seconds on the solved instances. E.g. Yasol has solved three of the four 'lseu 10x1A' instance and it spent 494 seconds on the four solved ones, and of course 600s on the missed one. We interpret the table in such a way that an increasing number of universal variables increases the difficulties of DEP based solvers for the solution process. Yasol seems to have by far less difficulties with the universal variables, however, has huge improvement potential concerning its existential variable handling.

	Gurobi	Cplex	Yasol	Scip
p0282 2x7A (5)	2 (42)	2 (28)	2 (0)	2 (52)
p0282 2x8A (10)	2 (162)	2 (113)	2 (1)	2 (379)
lseu 10x1A (4)	4 (5)	4 (6)	3 (165)	4 (163)
lseu 5x2A (5)	5 (6)	5 (5)	3(145)	4 (104)
lseu 1x10A (5)	5 (22)	5 (8)	4 (158)	4 (293)
lseu 11x1A (4)	4 (13)	4 (12)	3 (43)	4 (333)
lseu 1x11A (7)	7 (63)	7 (21)	4 (124)	4 (337)
lseu 12x1A (3)	3 (80)	3 (31)	2 (53)	0 (-)
lseu 6x2A (5)	5 (41)	5 (26)	5 (74)	2 (307)
lseu 4x3A (1)	1 (407)	1 (40)	0 (-)	0 (-)
lseu 3x4A (7)	7 (84)	7 (36)	7 (99)	2 (565)
lseu 7x2A (5)	5 (249)	5 (185)	5 (122)	0 (-)
connect6	-	-	-	-

Table 4: 0/1-QIPs, generated from lseu and p0282

4 Conclusion

In this paper, we investigated for the domain of 0/1-QIPs with objective, in how far a direct search approach can compete with building a DEP and solving it with state-of-the-art MIP solvers. On some of instances, the prototypical implementation in the solver Yasol could solve 0/1-QIPs up to optimality, being even faster than the best commercial solvers on the corresponding DEPs. The next step is to integrate further IP-programming techniques and to utilize conflict graphs and cliques in the sense of [22]. Moreover, it is to be examined, in how far well known cutting planes (e.g. [23]) can be incorporated into the search process and proven to be useful.

References

1. Schrijver, A.: Theory of linear and integer programming. John Wiley & Sons, Inc., New York, NY, USA (1986)
2. Papadimitriou, C.: Games against nature. J. of Comp. and Sys. Sc. (1985) 288–301
3. Birge, J.R., Louveaux, F.: Introduction to Stochastic Programming. Springer Series in Operations Research and Financial Engineering. Springer (July 1997)
4. Ben-Tal, A., Ghaoui, L.E., Nemirovski, A.: Robust Optimization. Princeton University Press (2009)
5. Liebchen, C., Lübbecke, M., Möhring, R., Stiller, S.: The concept of recoverable robustness, linear programming recovery, and railway applications. Robust and online large-scale optimization (2009) 1–27
6. Bellmann, R.: Dynamic programming. Princeton University Press (1957)
7. Kleywegt, A., Shapiro, A., Homem-De-Mello, T.: The sample average approximation method for stochastic discrete optimization. SIAM Jour. of Opt. (2001) 479–502
8. König, F., Lübbecke, M., Möhring, R., Schäfer, G., Spenke, I.: Solutions to real-world instances of pspace-complete stacking. Proc. ESA'07 729–740
9. Megow, N., Vredeveld, T.: Approximation results for preemtive stochastic online scheduling. ESA 2006 14th Annual European Symposium on Algorithms (2006)
10. Möhring, R., Schulz, A., Uetz, M.: Approximation in stochastic scheduling: The power of lp-based priority schedules. Journal of ACM **46**(6) (1999) 924–942
11. Subramani, K.: On a decision procedure for quantified linear programs. Annals of Mathematics and Artificial Intelligence **51**(1) (2007) 55–77
12. Ederer, T., Lorenz, U., Opfer, T., Wolf, J.: Modeling games with the help of quantified integer linear programs. In Herik, H., Plaat, A., eds.: Advances in Computer Games. Volume 7168 of Lecture Notes in Computer Science. Springer Berlin Heidelberg (2012) 270–281
13. Ederer, T., Lorenz, U., Martin, A., Wolf, J.: Quantified linear programs: A computational study. In: Part I. ESA'11, Springer (2011) 203–214
14. Lorenz, U., Martin, A., Wolf, J.: Polyhedral and algorithmic properties of quantified linear programs. In: Part I. ESA'10, Springer (2010) 512–523
15. Pijls, W., de Bruin, A.: Game tree algorithms and solution trees. Theor. Comput. Sci. **252**(1-2) (2001) 197–215
16. Birge, J., Louveaux, F.: Intro. to Stochastic Programming. Springer ('97)
17. Zhang, L.: Searching for truth: techniques for satisfiability of boolean formulas. PhD thesis, Princeton, NJ, USA (2003)
18. Achterberg, T.: Constraint Integer Programming. PhD thesis, Berlin (2007)
19. Johnson, E., Suhl, U.: Experiments in integer programming. Discrete Applied Mathematics **2**(1) (1980) 39 – 55
20. Audemard, G., Simon, L.: Refining restarts strategies for sat and unsat. In: Proceedings of the 18th international conference on Principles and Practice of Constraint Programming. CP'12, Berlin, Heidelberg, Springer (2012) 118–126
21. Haim, S., Heule, M.: Towards ultra rapid restarts. Technical report, UNSW and TU Delft (2010)
22. Atamtürk, A., Nemhauser, G.L., Savelsbergh, M.W.P.: Conflict graphs in solving integer programming problems. European Journal of Operational Research **121**(1) (2000) 40–55
23. Andreello, G., Caprara, A., Fischetti, M.: Embedding 0, 1/2-cuts in a branch-and-cut framework: A computational study. INFORMS J. on Computing **19**(2) (2007) 229–238