Solution Techniques for Quantified Linear Programs and the links to Gaming*

Ulf Lorenz, Thomas Opfer, and Jan Wolf

Institute of Mathematics, Technische Universität Darmstadt, Germany

Abstract. Quantified linear programs (QLPs) are linear programs (LPs) with variables being either existentially or universally quantified. QLPs are two-person zero-sum games between an existential and a universal player on the one side, and convex multistage decision problems on the other side. Solutions of feasible QLPs are so called winning strategies for the existential player that specify how to react on moves – certain fixations of universally quantified variables – of the universal player to certainly win the game. To find a certain best one among different winning strategies, we propose the extension of the QLP decision problem by an objective function. To solve the resulting QLP optimization problem, we exploit the problem's hybrid nature and combine linear programming techniques with solution techniques from game-tree search. As a result, we present an extension of the Nested Benders Decomposition algorithm by the $\alpha\beta$-heuristic and move-ordering, two techniques that are used in game-tree search to solve minimax trees. The principle applicability of our method to both QLPs and QIP models of PSPACE-complete games like Connect6 is examined in an experimental evaluation.

1 Introduction

In the last 40 years, algorithmic achievements like the introduction of the Alpha-Beta-Algorithm strengthened game-tree search remarkably [3, 6, 13]. Nowadays it is almost impossible for human chess masters to beat high-level chess computers. Proof Number Search has shown to be an effective algorithm for solving games [10, 14, 16]. Interestingly, there is a strong relation between these Artificial Intelligence techniques and effective treatment of uncertainty in Operations Research applications.

For many decades, a large amount of practical problems have been modeled as linear or mixed-integer linear programs (MIPs), which are well understood and can be solved quite effectively. However, there is a need for planning and deciding under uncertainty, as 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.

* Research partially supported by German Research Foundation (DFG) funded SFB 805 and by the DFG project LO 1396/2–1.

Uncertainty, often pushes the complexity of problems that are in P or NP, to the complexity class PSPACE. Also many optimization problems under uncertainty are PSPACE-complete [8] and therefore, NP-complete integer programs are not suitable to model these problems anymore. 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, 12]. While it is known that quantified linear integer programs (QIPs) are PSPACE-complete, the exact complexity class of their QLP-Relaxations is unknown in general. It turned out that quantified mixed-integer programming is a suitable modeling language for both optimization under uncertainty and games [4, 5]. The idea of our research is to explore the abilities of linear programming techniques when being applied to PSPACE-complete problems and their combination with techniques from other fields.

In this paper we show how the problem's hybrid nature of being both a two-person zero-sum game on the one hand, and being a convex multistage decision problem on the other hand, can be utilized to combine linear programming techniques with techniques from game-tree search. To the best of our knowledge, a combination of techniques from these two fields has not been done before. Solutions of feasible QLPs are so called winning strategies for the existential player that specify how to react on moves - certain fixations of universally quantified variables - of the universal player to certainly win the game. However, if there are several winning strategies, one might wish to find a certain (the best) one with respect to some kind of measure. We therefore propose an extension of the QLP decision problem by the addition of a linear objective function, which tries to minimize the objective function with respect to the maximum possible loss that can result from the universal player's possible decisions. To solve the resulting QLP optimization problem, we propose an extension of the Nested Benders Decomposition algorithm as presented in [4]. Solving a QLP with this algorithm can be interpreted as solving a tree of linear programs by passing information among nodes of the tree, and for the special case of QLPs with an objective function, the way this information are passed is similar to the minimax principle as it is known from game-tree search. This allows the integration of $\alpha\beta$-*cuts* in combination with *move-ordering*, as used in the $\alpha\beta$-Algorithm. The applicability of this approach is examined in an experimental evaluation, where we solve a set of QLPs that were generated from the well-known Netlib test set. Revealing that quantified programming is not only itself a game but can also be used to model and solve conventional games, we also model the Connect6 game as quantified program.

The rest of this paper is organized as follows. In Section 2, we formally describe the QLP optimization problem, followed by an explanation of the Nested Benders Decomposition approach in Section 3. Section 4 introduces the concepts of game-tree search and afterwards, in Section 5, we show how these techniques can be embedded into our existing algorithmic framework. We proceed with an experimental evaluation in Section 6, give an outlook how to apply our method to PSPACE complete games in section 7 and end up with a conclusion in Section 8.

2 The Problem Statement: Quantified Linear Programs (QLPs)

Within this paper, we intend to concentrate on quantified linear programs (QLPs), as they were introduced in [11, 12], and in-depth analyzed in [4, 7]. In contrast to traditional linear programs where all variables are implicitly existentially quantified, QLPs are linear programs with the variables being either existentially or universally quantified.

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).

We denote the quantification vector $\mathcal{Q} \circ x \in [l, u]$ as quantification sequence $\mathcal{Q}_1 x_1 \in [l_1, u_1] \ \ldots \ \mathcal{Q}_n x_n \in [l_n, u_n]$. In a similar manner, we denote $\mathcal{Q}$ as a quantifier sequence $\mathcal{Q}_1 \ \ldots \ \mathcal{Q}_n$ and x as a variable sequence $x_1 \ \ldots \ x_n$. 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 total number of blocks less one is the number of *quantifier changes*. In contrast to traditional linear programs, the order or the variables in the quantification vector is of particular importance.

Each QLP instance is 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 *the universal player wins*. The variables are set in consecutive order according to the quantification 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 the context of answering the question whether the existential player can certainly win the game, we use the term *policy*.

Definition 2 (Policy) *Given a QLP $(\mathcal{Q}, l, u, A, b)$ with $\mathcal{Q} \circ x \in [l, u] : Ax \leq b$. An algorithm that fixes all existential variables x_i with the knowledge, how $x_1, \ldots, x_{i-1}$ have been set before, is called a* policy.

A policy can be represented as a set of computable functions of the form $x_i = f_i(x_1, \ldots, x_{i-1})$ for all existentially quantified variables x_i. A policy is called a *winning policy* if these functions ensure that the existential player wins all games that can result from this policy, independently of the universal player's moves.

Definition 3 (QLP Decision Problems) *Given a QLP, the decision problem "Is there a winning policy for the existential player?" is called the QLP Decision Problem.*

It has been shown that the QLP problem with only one quantifier change is either in P (when the quantification begins with existential quantifiers and ends with universal ones) or coNP-complete (when the quantification begins with universal quantifiers and ends with existential ones) [12]. In [7] 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. 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 (cf. [12] and with a completely different proof in [7]).

Example 1 *The QLP*

$$\exists x_1 \in [1,6]\ \forall x_2 \in [1,2] : x_1 + x_2 \leq 6\ \wedge\ x_2 - x_1 \leq 0$$

has the following graphical representation of bounding box (dashed lines) and constraints (solid lines).

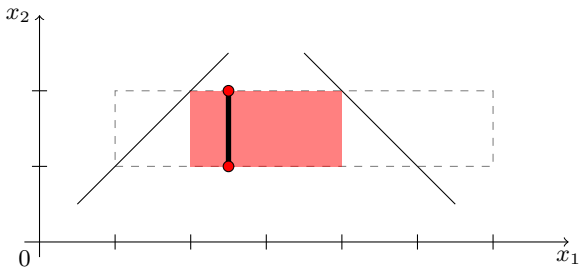


Fig. 1. Polyhedral QLP solution space

We say a solution to this problem is a move for the existential player such that he wins the game regardless of the universal player's reaction, or rather, the set of games $(x_1, x_2)^T$ which can result from the existential player's decision (black line segment in the figure above), e.g. the output of a winning policy. The set of all solutions, i.e. the set of 'existential sticks' fitting in the specified trapezoid, is called the solution space *(filled rectangle). In this example, we see that it indeed suffices to analyze discrete points (filled dots) to find a solution. Note that the order in the quantification sequence is crucial.*

The fact that it suffices to check the bounding values of the universally quantified variables in order to answer the question whether the existential player can certainly win the game, can be exploited in terms of asking whether a *winning strategy* for the existential player does exist.

Definition 4 (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 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 [9].

Example 2 *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 2 shows a visualization of the constraint polyhedron restricted to the unit cube. Since this example is rather small, we can guess a winning strategy for the existential player from the picture: 'Choose $x_1 \in [0, \frac{1}{2}]$, then choose x_3 appropriate 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 3 shows a winning-strategy for the existential player.

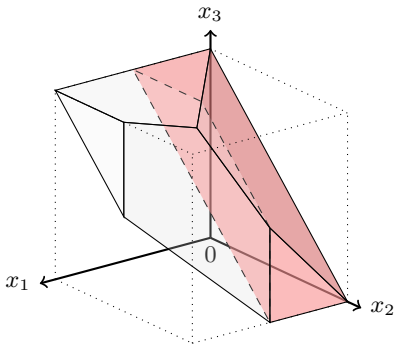


Fig. 2. Solution space of Example 2

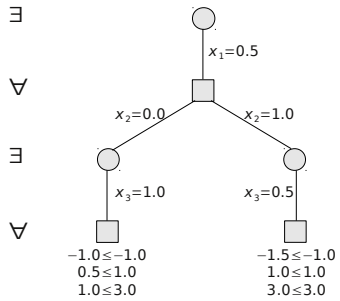


Fig. 3. Winning Strategy for Example 2

If there is more than one winning strategy for the existential player, it can be reasonable to search for a certain (the 'best') one. We can 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 5 (QLPs 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*

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

a QLP 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, it is a dynamic multistage decision process, similar as it is also known from multistage stochastic programming [2]. However, whereas in the latter an expected value is minimized, in our case we try to minimize the possible worst case (maximum loss) scenario that can result from the universal player's decisions. In the following we use the abbreviation $\min c^T x$ for the objective function and denote by

$$\min\{c^T x \mid \mathcal{Q} \circ x \in [l,u] : Ax \leq b\}$$

a *quantified linear program with objective function.*

Definition 6 (QLP Optimization Problems) *Given a QLP with objective function, the problem "Is it feasible? If yes, what is the best objective value of the existential player's winning policies?" is called* QLP Optimization Problem.

Note, that to find the existential player's optimal objective value, it is not enough to fix the universal players variables to their worst-case values regarding the objective function. For clarification, consider the following program:

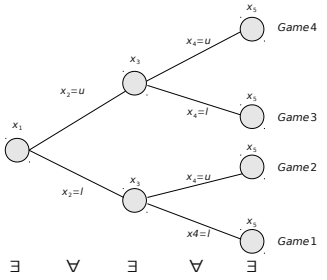
$$\min\{-x_1 - 2x_2 \mid \forall\, x_1 \in [0,1]\ \exists\, x_2 \in [0,1] : x_1 + x_2 \leq 1\}$$

Judging from the objective function the universal player should fix $x_1 = 0$, but this results in a better objective value for the existential player than forcing the existential player to fix $x_2 = 0$ in the constraint system.

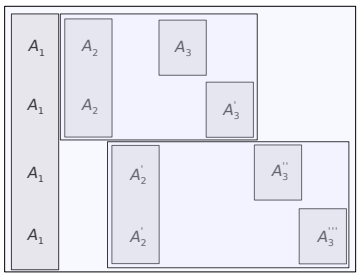
3 Nested Benders Decomposition (NBD)

The algorithm we extend in this paper was first proposed in [4] and uses decomposition techniques to solve an implicit reformulation of a QLP, which we call a *deterministic equivalent problem* (DEP). The concept is similar to the notion of deterministic equivalence as it is known from stochastic programming (cf. [2]) in the context of multistage stochastic linear programs (MSSLPs). Using the assumption of a finite time horizon and a discrete probability space, the resulting scenario tree is encoded into a DEP by replicating the LP for each possible scenario (possible path of events). Additionally, it is required that decisions must not depend on future events (*nonanticipativity*). The DEP of a QLP instance can be constructed in a similar way, however, instead of encoding the scenario

tree of randomly arising scenarios, we encode the decision tree of the universal player, which results from the series of all possible upper and lower bound combinations of the universal variables as determined by the quantification sequence. Nodes at stage t are decision points where the existential player has to fix variables, e.g. by solving a linear program, with respect to all previous moves $(x^1, \ldots, x^{t-1})$. Arcs of the tree represent moves of the universal player when he fixes his variables to the corresponding lower and upper bounds. Figure 4 a) shows the universal player's decision tree for a QLP with quantification sequence $\exists\, x_1 \in [l_1, u_1]\ \forall\, x_2 \in [l_2, u_2]\ \exists\, x_3 \in [l_3, u_3]\ \forall\, x_4 \in [l_4, u_4]\ \exists\, x_5 \in [l_5, u_5]$. The tree is similar to the strategy of a QLP where the moves of the existential player have not been fixed.



a) Universal Player Decision Tree



b) DEP Matrix Structure

Fig. 4. Deterministic Equivalent Linear Program

Figure 4 b) shows the resulting DEP matrix structure using compact variable formulation, which implicitly satisfies the nonanticipativity property because all nodes in the tree that share a common history also have the same set of decision variables up to that point. The resulting DEP grows exponentially with the number of universally quantified variables of the corresponding QLP, but the special block structure of the matrix can be exploited by the Nested Benders Decomposition (NBD) algorithm. The NBD algorithm is a recursive application of the well-known Benders Decomposition principle [1] and is widely used in the Stochastic Programming community to solve MSSLPs [2].

To illustrate how the Benders Decomposition algorithm works to solve QLPs, we consider w.l.o.g. the DEP that results from a QLP with quantification sequence $\exists x_1 \in [0, u_1] \forall x_2 \in [1, 1] \exists x_3 \in [0, u_3]$ and an objective function $\min c_1^T x_1 + c_3^T x_3$. Let the constraint system $A_1 x_1 + A_2 x_2 + A_3 x_3 \leq b$ contain the upper bound u_1 of x_1 and u_3 of x_3. Since the $\forall$-variable $x_2 \in [1, 1]$ is a fixed variable, the DEP consists of a single game where the corresponding right hand side b' results from $b' = b - A_2 \overline{x}_2$ with $\overline{x}_2 = 1$.

The resulting DEP looks as follows:

$$\begin{aligned} Z = \min\ & c_1^T x_1 + c_3^T x_3 \\ s.t.\quad & A_1 x_1 + A_3 x_3 \leq b' \\ & x_1 \geq 0, x_3 \geq 0 \end{aligned}$$

Applying Benders Decomposition, the decision variables of the DEP are stage-wise partitioned and then decomposed into a *restricted master problem*

(RMP) that contains the first-stage variable x_1, and one *subproblem (SP)* that contains the second-stage variable x_3. The corresponding *dual SP (DSP)* has the property that the solution space no longer depends on the value of x_1, regardless whether it is feasible or infeasible for the SP. The SP and its DSP can be written as follows:

$$\begin{array}{lll} SP(x_1) = \min c_3^T x_3 & & DSP(x_1) = \max \pi^T(b' - A_1 x_1) \\ s.t. & A_3 x_3 \leq b' - A_1 x_1 & s.t. \quad A_3^T \pi \leq c_3 \\ & x_3 \geq 0 & \pi \leq 0 \end{array}$$

For a non-optimal $\overline{x}_1$ obtained by solving the RMP, which can be empty at the beginning, the following two cases can happen. If the SP is feasible, the solution of the DSP is bounded and located at an extreme point of its solution space. If the SP is infeasible, the solution of the DSP is unbounded, which corresponds to an extreme ray of its solution space. Using this dual information, two different types of cutting planes - called *Benders cuts* - can be added to the RMP to cutoff the last $\overline{x}_1$ in the next solution of the RMP.

1. *feasibility cut*: $(\pi_r^j)(b' - A_1 x_1) \leq 0$, if the DSP($\overline{x}_1$) is unbounded, where π_r^j is the vector that corresponds to the extreme ray j.
2. *optimality cut*: $(\pi_p^i)(b' - A_1 x_1) \leq q$, if the DSP($\overline{x}_1$) is bounded, where π_p^i is the vector that corresponds to the extreme point i.

Since the DSP can only have finitely many extreme points and extreme rays, the RMP can be written as follows, where q is an auxiliary variable used to represent the objective function value of the SP:

$$\begin{array}{ll} RMP = \min c_1^T x_1 + q & \\ s.t. & (\pi_r^j)(d - A_1 x_1) \leq 0 \ \forall j \in J \\ & (\pi_p^i)(d - A_1 x_1) \leq q \ \forall i \in I \\ & x_1 \geq 0 \end{array}$$

This reformulation is equivalent to the initial DEP. However, there can be exponentially many extreme rays and extreme points and not all of them are needed to find the optimal solution. Therefore, the algorithm starts with I and J being empty, and computes cuts in an iterative process until an optimal solution is found or infeasibility is detected. In the latter case also the DEP and the corresponding QLP are infeasible. The optimal solution is found, if for a given candidate optimal solution (x_1^*, q^*), called proposal, also the SP(x_1^*) has an optimal solution with value $q(x_1^*)$ and the optimality condition $q(x_1^*) = q^*$ is satisfied. If this is the case, the algorithm stops. Otherwise a feasibility or optimality cut is added to the RMP, which is then re-solved again to obtain a new proposal. In each iteration where the SP is feasible, $c^T x_1^* + q^*$ yields a lower bound for the initial problem, while $c^T x_1^* + q(x_1^*)$ yields an upper bound. The difference between these bounds gets smaller, and if it becomes less than a predefined ϵ, the algorithm terminates.

If there are k universally quantified variables, then there are 2^k games and therefore 2^k subproblems are solved in each iteration, each yielding a cut that is

added to the RMP. The *min-max* property of the objective function is achieved, because all optimality cuts that result from the subproblems restrict the same auxiliary variable q. For the computation of the upper bound, the maximum over all subproblems from the last iteration is used. For multistage QLPs resulting from a quantifier string $\exists\forall\exists\forall...\forall\exists$, Benders Decomposition can be recursively applied, which is known as Nested Benders Decomposition. Solving the DEP of a multistage QLP can be illustrated as solving a tree of linear programs that are attached to the nodes of the decision tree of the universal player. The tree is traversed forwards and backwards multiple times, with information being passed between adjoined nodes of the tree. A node at stage t passes proposals for the variables from the root up to stage t to its immediate descendants at stage $t+1$ and cuts to its immediate ancestor at stage $t-1$.

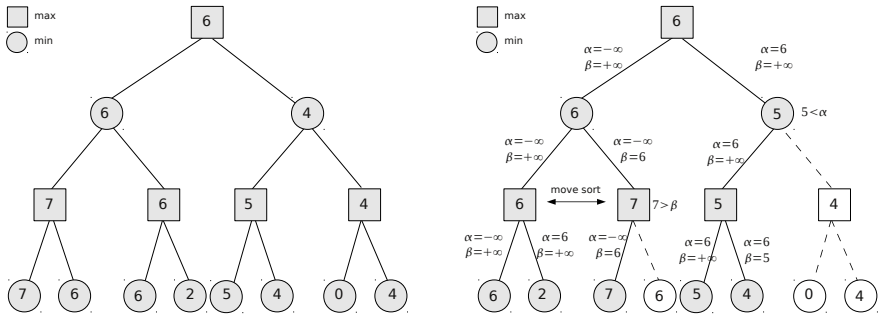
The algorithm has been implemented and tested in a detailed computational study with instances that were generated from existing LP and IP test sets [4].

4 Game-Tree-Search and the $\alpha\beta$-Algorithm

The term *minimax tree* describes one of the most important data structures that allows computers to play two-person zero-sum games such as checkers, chess, and go. Nodes of the tree are decision points for the players and are therefore subdivided in min and max nodes. Nodes from different stages are connected with branches, leaf nodes are end positions of the game and can be evaluated as a win, lose, or draw using the rules of the game. Often, a specific score from the max player's point of view is computed with the help of a weighting function and assigned to a leaf to represent how good or bad the sequence of moves from the root to the leaf is. With a complete game-tree, it is possible to solve the game with the *MiniMax*-Algorithm, which fills the inner node values of the tree bottom-up starting with the evaluated values at the leafs. Nodes that belong to the max player get the maximum values of their successors, while nodes for the min player get the minimum. Figure 5a) illustrates this behavior.

While the MiniMax-Algorithm must evaluate the entire game-tree to compute the root value, the *$\alpha\beta$-Algorithm* [9] prunes away branches that cannot influence the final result. Therefore, it maintains two values, α and β, which represent the minimum score that the max player is sure to gain at least until that point in the tree, and the maximum score of the min player respectively. If the evaluation of a position where the min player has to move becomes less than α, the move need not to be further explored, since a better move has already been found. The same holds, if at a position where the max player has to choose his move, the evaluation provides a value that is greater than β. Figure 5b) illustrates this behavior, the dashed subtrees were not visited. The left one due to a *β-cutoff*, the subtree on the right hand due to an *α-cutoff*.

While the order in which the nodes of the tree are evaluated does not matter for the MiniMax-Algorithm, it is essential for the performance of the $\alpha\beta$-Algorithm.



a) MiniMax Game-Tree b) $\alpha\beta$-Algorithm

Fig. 5. MiniMax Game-Tree and $\alpha\beta$-Algorithm

The *best moves* need to be evaluated first in order to find strong α and β values as soon as possible. Figure 5b) illustrates this, without swapping the subtrees under the first successor of the root on the left side, the β-cutoff would not have occurred. If the best moves are searched first, the runtime of the $\alpha\beta$-Algorithm is only $O(\sqrt{b^d})$ where d is the depth of the tree and b is the number of possible moves at each node. The MiniMax-Algorithm has a runtime of $O(b^d)$.

5 The $\alpha\beta$-Nested Benders Decomposition ($\alpha\beta$-NBD)

In this Section we describe how the $\alpha\beta$-heuristic in combination with move-ordering can be integrated into the Nested Benders Decomposition (NBD) algorithm. Let us recall that solving a multistage QLP with the NBD algorithm can be illustrated as solving a tree of linear programs that are attached to the nodes of the universal player's decision tree. The tree is therefore traversed multiple times and information in the form of proposals and cuts are passed between nodes of the tree. If a node v_i at stage $t \in \{0, \ldots, T\}$ receives a new proposal $\overline{x}^{t-1}$ from its direct successor at stage $t-1$, the subtree rooted at node v_i is solved to optimality, or until the nodal linear program attached to v_i becomes infeasible. After feasibility of the subtree is established, the upper and lower bounds of v_i converge and for the node's optimal objective function value z_i holds $L_i \leq z_i \leq U_i$ until the values coincide at the end. Then node v_i passes $\overline{z}_i$ and the corresponding optimality cut to its direct ancestor at stage $t-1$. After an iteration where node v_i passed its current proposal $\overline{x}^t$ to its direct successors $v_j \in J$ at stage $t+1$, and all of them were feasible with $\overline{z}_j$ denoting the corresponding optimal objective function values, v_i's upper bound computes as $U_i = c^T\overline{x}^t + \max_{j \in J} \overline{z}_j$. This is equal to the minimax principle as mentioned in Section 4. The existential player tries to minimize the value of a nodal linear program, with respect to the worst-case move of the universal player, which is the corresponding subproblem with the *maximal* objective function value. When using a depth first-search to traverse the tree as it is done in the $\alpha\beta$-Algorithm, the knowledge of the current maximal value $\alpha_i = \max_{k \in K} \overline{z}_k$ of some successors

$v_k \in K \subset J$ of node v_i can be used in a similar manner as α in the $\alpha\beta$-Algorithm. In the current iteration, it denotes the minimum value, the maximizing player (the universal one) will at least obtain at node v_i. When α_i is passed to the remaining nodes from the set of successors $J \backslash K$ each node v_l from this set can stop computing its exact optimal objective function value after it determines feasibility with respect to the current proposal $\overline{x}^t$, and detects that its current upper bound U_l is less than or equal to α_i. We can also integrate a value analogously to β that depicts the maximum value the minimizing player will gain for sure at a specific node v_i at stage t. In terms of the NBD algorithm this is the upper bound U_i from the previous iteration. In the next iteration, this value can be passed to all successors $v_j \in J$ at stage $t+1$ together with the new proposal $\overline{x}^t$. If a successor v_j determines feasibility with respect to the current proposal $\overline{x}^t$ and detects that its lower bound $L_i = c^T \overline{x}^{t+1} + \overline{q}^{t+1}$ is greater than this value, it can stop computing its exact optimal objective function value because a better solution has already been found in the previous iteration. In the following, we will therefore also use the abbreviations α and β for these values.

As in the case of the $\alpha\beta$-Algorithm, the order in which the nodes of the tree are solved is an important issue in the $\alpha\beta$-NBD algorithm. Whereas the $\alpha\beta$-Algorithm uses heuristics, our algorithm organizes the order in which nodes are visited based on information from previous iterations. To obtain strong bounds as soon as possible, the successor of a node v_i that provided the worst-case sub solution in the previous iteration, is visited first in the next iteration, speculating that it will again provide a strong α-bound. Also the other successors are arranged in descending order by their solution values from the last iteration. However, many other sorting criterions are possible.

The algorithmic framework has been implemented in C++ using the LP Solver CPLEX 12.4 to solve nodal linear programs.

6 Computational Results

In the following we present the results of our experimental evaluation. All tests were run on a quad-core processor AMD Phenom II X4 945 with 8GB RAM. For our tests we took LP instances with a maximum number of 500 variables and constraints and generated QLPs with 10, 15, and 18 universally quantified variables. For each new universally quantified variable $x_i \in [0, 1]$, we randomly added matrix coefficients from the interval $[-1, 1]$ with a density of 25%. We furthermore varied the number of $\forall$-quantifier blocks to 1, 2, and 5 and distributed them equally in the QLP. This results in twostage and multistage QLP instances and nine different test sets. A similar test set was used in [4].

Table 1 shows the summed up results solving each of the test sets with the standard NBD algorithm, the $\alpha\beta$-Nbd algorithm, and when solving the corresponding DEPs with CPLEX. Column 1 contains the number of universally quantified variables followed by the number of blocks of universally quantified variables in column 2. Column 3 contains the solution times when the corresponding DEPs are solved with CPLEX running with standard settings and its

preprocessor enabled. Columns 4 and 5 show the solution times and the number of LPs that were solved using the standard NBD algorithm. Column 6 and 7 show the same numbers when $\alpha\beta$-cuts and the move sort heuristic are used.

		CPLEX	NBD		NBD ($\alpha\beta$ + move-ordering)	
∀-Vars	∀-Blocks	Time (s)	Time (s)	Subproblems	Time (s)	Subproblems
10	1	572.84	36.79	123844	21.72	82832
10	2	225.22	69.48	300410	32.51	137550
10	5	109.65	101.20	567935	53.96	334402
15	1	>172800.00	1375.60	5923805	1010.70	4117858
15	2	>172800.00	1755.77	7592959	1101.67	4578949
15	5	130934.78	2281.55	13021052	1113.95	6724840
18	1	>172800.00	10044.65	38954373	7047.60	24542092
18	2	>172800.00	15174.01	75414983	10059.71	48808515
18	5	>172800.00	27007.41	123214174	14313.93	66535654

Table 1. Computational Results

The results show that the even the standard NBD algorithm implementation is clearly faster than solving the DEP in most cases, especially with an increasing number of universally quantified variables. This is due to the exponential growth of the DEP with an increasing number of universally quantified variables in the corresponding QLP. When we additionally use the $\alpha\beta$-heuristic and move sort, we observe notable time savings up to about 50% compared to the standard implementation as we can e.g. see in the last row of Table 1. The extended algorithm was able to halve the number of subproblems that had to be solved from 123214174 to 66535654, resulting in a reduction of the solution time from 450 minutes to 238 minutes, a difference of 47%. The effect becomes stronger with an increasing number of stages but even in the twostage case, move-ordering alone leads to a performance gain of $25\% - 40\%$. These results show the high potential of combining techniques from game-tree search with the Nested Benders Decomposition approach and motivate a further research in this direction.

7 Outlook: Application to other Games

Apart from the fact that QLPs taken by themselves can be interpreted as two-person zero-sum games, they provide a adequate modeling tool for many purposes. In [5] a QIP model of the two-person game Gomoku was proposed. We adopted this model to a very similar PSPACE-complete game: Connect6. Here two players playing on a Go board try to achieve a connected row of six stones. At the beginning, black places one stone, then white and black take turns placing two stones. The player, who has the first connected row of six stones, wins.

While the former model was never practically solved, we use only a logarithmic number of universally quantified variables in our current model to reduce the computational burden. This is necessary because the effort to solve a QIP with both DEP and $\alpha\beta$-NBD grows exponentially with the number of universally quantified variables. To model a set of n binary variables x_i where exactly one equals 1 while all others are 0 (similar to a so called SOS1-constraint in

mathematical optimization) with a logarithmic amount of binary variables y_i, one can use the following transformation between binary and unary encoding[1]:

$$\sum_{i=1}^{\log_2(n)} 2^{i-1} \cdot y_i = \sum_{i=1}^{n} i \cdot x_i - 1$$

$$\sum_{i=1}^{n} x_i = 1$$

Still the DEP gets too large to solve directly. Thus we used a variant of the proposed $\alpha\beta$-NBD algorithm to decide whether black can win in n moves starting from an arbitrary situation. However, as the model contains binary variables, an additional type of cut, called Combinatorical Benders Cut [15], had to be added to the algorithm. Given a node v_i at depth t a proposal $\overline{x}^{t-1}$ that turns out to be invalid, at least the corresponding part of the solution space of the master problem is cut off by the following cut:

$$\sum_{j \in \Lambda : \overline{x}_j^{t-1} = 0} \overline{x}_j^{t-1} + \sum_{j \in \Lambda : \overline{x}_j^{t-1} = 1} (1 - \overline{x}_j^{t-1}) \geq 1$$

Here Λ depicts the set of existentially quantified variables from stage 0 to stage $t-1$. Our plan is to improve this cut by methods of conflict analysis in MIPs.

In our preliminary tests, we could solve instances of this problem for $n \leq 10$ on a board of size 8×8.

8 Summary

In the course of this paper we considered QLPs with objective functions and showed how their hybrid nature of being a two-person zero-sum game on the one side, and being a convex multistage decision problem on the other side, can be used to combine linear programming techniques with solution techniques from game-tree search. We therefore extended the Nested Benders Decomposition algorithm by the $\alpha\beta$-heuristic in combination with move-ordering, two techniques that are used in the $\alpha\beta$-Algorithm to evaluate minimax trees. We showed the applicability in an experimental evaluation, where we solved QLPs that were generated from the well-known Netlib test set. The results showed a speedup of up to 50% compared to the standard Nested Benders Decomposition implementation without techniques from game-tree search.

[1] Heed that this transformation is only valid if n is a power of 2. It can be easily adopted to the general case.

Bibliography

[1] J. F. Benders. Partitioning procedures for solving mixed-variables programming problems. *Numerische Mathematik*, 4(1):238–252, Dec. 1962.

[2] J. R. Birge and F. Louveaux. *Introduction to Stochastic Programming*. Springer Series in Operations Research and Financial Engineering. Springer, July 1997.

[3] C. Donninger and U. Lorenz. The hydra project. *Xcell Journal*, (53), 2005.

[4] T. Ederer, U. Lorenz, A. Martin, and J. Wolf. Quantified linear programs: A computational study. In *Proceedings of the 18th annual European conference on Algorithms: Part I*, ESA'11, pages 203–214, Berlin, Heidelberg, 2011. Springer-Verlag.

[5] T. Ederer, U. Lorenz, T. Opfer, and J. Wolf. Modelling games with the help of quantifieded integer linear programs. In *ACG 13*. Springer, 2011.

[6] F.-H. Hsu. Ibm's deep blue chess grandmaster chips. *IEEE Micro*, 18(2):70–80, 1999.

[7] 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, Berlin, Heidelberg, 2010. Springer-Verlag.

[8] C. Papadimitriou. Games against nature. *J. of Comp. and Sys. Sc.*, pages 288–301, 1985.

[9] W. Pijls and A. de Bruin. Game tree algorithms and solution trees. *Theor. Comput. Sci.*, 252(1-2):197–215, 2001.

[10] A. Plaat, J. Schaeffer, W. Pijls, and A. De Bruin. Best-first fixed-depth game-tree search in practice. In *Proceedings of the 14th international joint conference on Artificial intelligence - Volume 1*, pages 273–279, San Francisco, CA, USA, 1995. Morgan Kaufmann Publishers Inc.

[11] K. Subramani. Analyzing selected quantified integer programs. *Springer, LNAI 3097*, pages 342–356, 2004.

[12] K. Subramani. On a decision procedure for quantified linear programs. *Annals of Mathematics and Artificial Intelligence*, 51(1):55–77, 2007.

[13] H. van den Herik, J. Nunn, and D. Levy. Adams outclassed by hydra. *ICGA Journal*, 28(2):107–110, 2005.

[14] H. van den Herik, J. Uiterwijk, and J. van Rijswijk. Games solved: Now and in the future. *Artificial Intelligence*, 134:277–312, 2002.

[15] F. Vanderbeck and L. Wolsey. Reformulation and decomposition of integer programs. CORE Discussion Papers 2009016, Université catholique de Louvain, Center for Operations Research and Econometrics (CORE), 2009.

[16] M. H. Winands, J. W. Uiterwijk, and J. van den Herik. Pds-pn: A new proof-number search algorithm. In *Computers and Games (CG)*, pages 61–74, 2002.