An Expansion-Based Approach for Quantified Integer Programming

Michael Hartisch
University of Passau, Germany
FAU Erlangen-Nürnberg, Germany

Leroy Chew
TU Wien, Austria

Abstract

Quantified Integer Programming (QIP) bridges multiple domains by extending Quantified Boolean Formulas (QBF) to incorporate general integer variables and linear constraints while also generalizing Integer Programming through variable quantification. As a special case of Quantified Constraint Satisfaction Problems (QCSP), QIP provides a versatile framework for addressing complex decision-making scenarios. Additionally, the inclusion of a linear objective function enables QIP to effectively model multistage robust discrete linear optimization problems, making it a powerful tool for tackling uncertainty in optimization.

While two primary solution paradigms exist for QBF – search-based and expansion-based approaches – only search-based methods have been explored for QIP and QCSP. We introduce an expansion-based approach for QIP using Counterexample-Guided Abstraction Refinement (CEGAR), adapting techniques from QBF. We extend this methodology to tackle multistage robust discrete optimization problems with linear constraints and further embed it in an optimization framework, enhancing its applicability. Our experimental results highlight the advantages of this approach, demonstrating superior performance over existing search-based solvers for QIP in specific instances. Furthermore, the ability to model problems using linear constraints enables notable performance gains over state-of-the-art expansion-based solvers for QBF.

2012 ACM Subject Classification Mathematics of computing → Discrete mathematics; Mathematics of computing → Solvers; Theory of computation → Discrete optimization; Theory of computation → Constraint and logic programming; Theory of computation → Algorithm design techniques; Applied computing → Operations research; Theory of computation → Abstraction

Keywords and phrases Quantified Integer Programming, Quantified Constraint Satisfaction, Robust Discrete Optimization, Expansion, CEGAR

Digital Object Identifier 10.4230/LIPIcs.CP.2025.12

Related Version *Full Version*: `https://doi.org/10.48550/arXiv.2506.04452`

Supplementary Material
Software (Source Code): `https://github.com/MichaelHartisch/EQuIPS` [36]
archived at `swh:1:dir:29f0d2d80f69841007fd3a315a47c2865669f3a6`

Funding Furthermore, we acknowledge that this research was partially funded by Deutsche Forschungsgemeinschaft (DFG, German Research Foundation) – 534441421 and Austrian Science Fund FWF Project ESP197.
Michael Hartisch: Partially funded by Deutsche Forschungsgemeinschaft (DFG, German Research Foundation) – 534441421.
Leroy Chew: funded by Austrian Science Fund FWF Project ESP197.

Acknowledgements We thank the three anonymous reviewers for their helpful comments.

1 Introduction

1.1 Motivation

Solving Quantified Boolean Formulas (QBF) typically relies on two complementary approaches: Quantified Conflict-Driven Clause Learning (QCDCL) and expansion-based solving. QCDCL extends classical SAT solving techniques by incorporating learning and backjumping in a search-based framework – explicitly traversing the assignment tree respecting the quantifier order – making it particularly effective for QBF instances with deep quantifier alternations. In contrast, expansion-based solving employs a controlled form of Shannon expansion, duplicating the formula to eliminate quantifiers, at the cost of a larger formula with more variables. While expansion methods excel on formulas with few quantifier alternations, they struggle as the number of alternations increases. QCDCL solvers exhibit the opposite behavior: they handle deeply nested quantifications well but can be inefficient for problems where expansion-based approaches are advantageous.

Quantified Integer Programming generalizes QBF by introducing integer variables and linear constraints, placing it within the broader domain of Quantified Constraint Satisfaction Problems (QCSP). Existing QIP solvers follow a QCDCL-inspired approach. However, no expansion-based QIP solver exists, leaving a gap in the landscape of QIP solution methods. In this work, we address this gap by introducing and investigating a new expansion-based QIP solver, enabling efficient resolution of problems where search-based methods struggle.

Expansion-based methods are refutationally complete: given that QIP requires bounded variable domains, a full expansion reduces satisfiability to a finite Integer linear Programming (IP) problem. However, this naive approach is computationally prohibitive, as the expansion tree grows exponentially with the number of variables. In practice, proving unsatisfiability does not require full expansion – an unsatisfiable core often involves only a fraction of the search space. Inspired by techniques from QBF solving, such as those employed in `RAReQS` and `QFUN`, we leverage Counterexample Guided Abstraction Refinement (CEGAR) to construct a more targeted expansion. Our method iteratively refines the search space by alternating between satisfiability and unsatisfiability checks, incorporating counterexamples (countermoves) to guide the exploration process efficiently.

By bridging the gap between expansion-based solving and QIP, our approach broadens the applicability of QIP solvers and introduces new avenues for tackling complex quantified constraint problems.

1.2 Related Work

The study of quantified integer variables dates back at least to the 1990s [27], with the term QIP being coined in [62]. Early research focused on the complexity of QIP [62, 19, 53] and treated QIP as a satisfiability problem until optimization aspects were incorporated [47].

Building on these foundations, a search-based approach for solving QIP was proposed [23, 40], followed by several algorithmic enhancements, including expansion-related relaxation techniques [35] and pruning heuristics [39]. Extensions enabled restricting universal variable assignments, confining them beyond their default bounds [37, 38], similar to robust optimization under polyhedral uncertainty [31], budgeted uncertainty [7, 32], or decision-dependent uncertainty [56, 55]. Furthermore, the link to multistage robust discrete optimization has been established, demonstrating practical applicability [30].

In multistage robust discrete optimization, approaches such as scenario generation [31] and row-and-column generation (e.g. [1]) iteratively refine relaxed formulations, intrinsically incorporating optimization. Unlike these, in order to optimize, we repeatedly solve QIP

feasibility problems in a binary search environment to determine the optimal objective value. For general multistage optimization approaches, most available methods rely on approximation techniques, typically using either sampling methods [5, 48] or decision rules [64, 57, 6], which restrict the solution space to strategies with a predefined (often linear) structure.

Our approach builds upon the expansion-based methods developed for QBF [42, 43, 44], contrasting with traditional search-based approaches [66, 45], though efforts have been made to combine both ideas [12]. Theoretical [9] and practical [46] results suggest that expansion-based solving is advantageous when there are few quantifier alternations and several limitations of search-based QBF methods have been documented [10, 8, 14, 20].

In QCSP, numerous search-based algorithms exist [49, 26, 65, 54, 15], with only a few approaches employing ideas related to expansion – one repair-based [61] and one relaxation-based [25] approach. Early attempts to also incorporate optimization date back to the early 2000s [16]. These efforts primarily relied on search-based methods [4] and were later extended to more general problem formulations [51]. Similar developments occurred in QBF, where optimization versions have also been explored [41].

Finally, several extensions of satisfiability modulo theories (SMT) have been developed to address quantified reasoning [3, 13, 58, 2]. These techniques differ from our QIP-specific approach, which leverages the linear structures for greater efficiency. A recent approach in [63] seems related, given the very similar title, but differs in scope, focusing on semi-infinite problems – optimization problems with finitely many (existentially quantified) variables and infinitely many (quantified) constraints.

2 Preliminaries

2.1 Quantified Integer Programming

A Quantified Integer Program is a natural extension of an integer linear program in which each variable is subject to either existential or universal quantification. Consider n integer variables $x_1, x_2, \dots, x_n$ arranged in order such that if $i < i'$, variable x_i, is said to lie to the left of $x_{i'}$. Each variable x_i, $i \in [n]$[1], takes values in a bounded domain $D_i = \{x_i \in \mathbb{Z} \mid l_i \leq x_i \leq u_i\}$, where l_i and u_i are the lower and upper integer bounds, respectively. In addition, every variable x_i is associated with a quantifier $Q_i \in \{\exists, \forall\}$. The *quantification level* of a variable is defined as the number of alternations of quantifiers to its left plus one. If there are $k \leq n$ such levels, then all variables sharing the same quantification level are grouped together. For each such level j, the common quantifier $\mathcal{Q}_j \in \{\exists, \forall\}$ applies to the vector of variables $\boldsymbol{X}_j$, which collectively range over the domain $\mathcal{D}_j$ (the Cartesian product of their individual domains). We sometimes omit the level index and simply write $\mathcal{D}$, which implies that the variables must remain integral and adhere to their prescribed bounds.

A *QIP feasibility problem* can be written in the compact form $\mathcal{Q}_1\boldsymbol{X}_1 \in \mathcal{D}_1\ \mathcal{Q}_2\boldsymbol{X}_2 \in \mathcal{D}_2\ \cdots\ \mathcal{Q}_k\boldsymbol{X}_k \in \mathcal{D}_k : A^\exists \boldsymbol{x} \leq \boldsymbol{b}^\exists$, where the *existential constraint system* is given by matrix $A^\exists \in \mathbb{Q}^{m \times n}$ and right-hand side vector $\boldsymbol{b}^\exists \in \mathbb{Q}^m$, with $\boldsymbol{x} = (\boldsymbol{X}_1, \dots, \boldsymbol{X}_k)$. Note, that throughout the paper we use bold letters, to indicate vectors. We sometimes write $\mathcal{Q}\boldsymbol{X} \in \mathcal{D} : \Phi$, decomposing the problem into the problem of finding an assignment for the first level variables $\boldsymbol{X}$ and the remaining QIP Φ that either starts with quantifier $\bar{Q}$ or only consists of a constraint system. For QIP Φ and variables $\boldsymbol{X}$ of quantification level $j \in [k]$,

[1] We use $[n] = \{1, \dots, n\}$ to denote index sets.

we write $\Phi[\boldsymbol{\tau}]$ to denote the modified QIP obtained by removing $\boldsymbol{X}$ from the quantification sequence and assigning $\boldsymbol{X}$ to $\boldsymbol{\tau} \in \mathcal{D}_j$ in the constraint system of Φ leading to a change in the right-hand side vector. Then, the new constraint system is described by $A^{\exists}_{(-\boldsymbol{X})}\boldsymbol{x}_{(-\boldsymbol{X})} \leq \boldsymbol{b}^{\exists}_{\boldsymbol{X}=\boldsymbol{\tau}}$, where $A^{\exists}_{(-\boldsymbol{X})}$ contains all columns as $A^{\exists}$ without the columns corresponding to $\boldsymbol{X}$, $\boldsymbol{x}_{(-\boldsymbol{X})}$ corresponds to the variable vector $\boldsymbol{x}$ without variables $\boldsymbol{X}$, and $\boldsymbol{b}^{\exists}_{\boldsymbol{X}=\boldsymbol{\tau}} = \boldsymbol{b}^{\exists} - A^{\exists}_{(\boldsymbol{X})}\boldsymbol{\tau}$, where $A^{\exists}_{(\boldsymbol{X})}$ is the matrix comprising of the columns of $A^{\exists}$ corresponding to $\boldsymbol{X}$.

2.2 QIP Game Semantics

We can think of a QIP as a game between the *universal* ($\forall$) and *existential* ($\exists$) player, where in move j, player $\mathcal{Q}_j$ assigns variables $\boldsymbol{X}_j$ of level j with values from domain $\mathcal{D}_j$. The existential player wins, if the existential constraint system is satisfied after all variables have been assigned. The universal player wins, if at least one constraint is violated. More formally, a *play* is an assignment of all variables $\boldsymbol{X}_1, \ldots, \boldsymbol{X}_k$. A strategy for the assignment of variables $\boldsymbol{X}_j$ of level j, is a function $s_j : \mathcal{D}_1 \times \cdots \times \mathcal{D}_{j-1} \to \mathcal{D}_j$. A strategy $S = (s_i)_{i\in[k],\mathcal{Q}_i=Q}$ for player Q consists of a strategy for all variables associated to her. For a given QIP, S is a *winning strategy* for the existential player, if

$$\forall \boldsymbol{X}_i \in \mathcal{D}_i, i \in [k], Q_i = \forall : \; A^{\exists}\boldsymbol{x} \leq \boldsymbol{b}^{\exists} \bigwedge_{\substack{i\in[k]:\\ \mathcal{Q}_i=\exists}} \boldsymbol{X}_i = s_i(\boldsymbol{X}_1, \ldots, \boldsymbol{X}_{i-1})$$

is true, i.e., for every assignment of the universally quantified variables, the strategy results in a satisfied constraint system. Conversely, S is a winning strategy for the universal player, if

$$\forall \boldsymbol{X}_i \in \mathcal{D}_i, i \in [k], Q_i = \exists : \; A^{\exists}\boldsymbol{x} \not\leq \boldsymbol{b}^{\exists} \bigwedge_{\substack{i\in[k]:\\ \mathcal{Q}_i=\forall}} \boldsymbol{X}_i = s_i(\boldsymbol{X}_1, \ldots, \boldsymbol{X}_{i-1})$$

is true, i.e., if regardless of the assignment of the existentially quantified variables, the strategy of the universal player leads to a violation of the constraint system.

For QIP $\mathcal{Q}\boldsymbol{X} \in \mathcal{D} : \Phi$, a strategy s for assigning $\boldsymbol{X}$ (which is just an assignment $\boldsymbol{\tau} \in \mathcal{D}$), is called a *winning move*, if there exists a winning strategy S containing s. Furthermore, for $\mathcal{Q}\boldsymbol{X}_1 \in \mathcal{D}_1 \bar{\mathcal{Q}}\boldsymbol{X}_2 \in \mathcal{D}_2 : \Phi$ and assignment $\boldsymbol{\tau} \in \mathcal{D}_1$ to $\boldsymbol{X}_1$, the assignment $\boldsymbol{\mu} \in \mathcal{D}_2$ is called a *countermove* to $\boldsymbol{\tau}$, if $\boldsymbol{\mu}$ is a winning move for the QIP $\bar{\mathcal{Q}}\boldsymbol{X}_2 \in \mathcal{D}_2 : \; \Phi[\boldsymbol{\tau}]$. We also adopt the notion of a multi-game as used in [44]. A QIP *multi-game* is given by $\mathcal{Q}\boldsymbol{X} \in \mathcal{D} : \{\Phi_1 \ldots \Phi_\ell\}$, where each Φ_i, referred to as a *subgame*, is a QIP beginning with $\bar{Q}$ or that solely contains a constraint system. A multi-game is won by player $\mathcal{Q}$, if there exists a move $\boldsymbol{\tau} \in \mathcal{D}$ such that $\mathcal{Q}$ has a winning strategy for all subgames $\Phi_i[\boldsymbol{\tau}]$.

2.3 Additional Constraints

To further enhance the modeling power of a QIP, we explicitly enable the restriction of universally quantified variables. This has been introduced in [37] for QIP and has a similar notion as QCSP$^+$ [65] and Quantified Linear Implication [24]. In robust optimization, this corresponds to introducing an uncertainty set [31], while in the context of QBF it is related to including cubes in the initial formula. To this end, let $A^{\forall}\boldsymbol{x} \leq \boldsymbol{b}^{\forall}$ be the *universal constraint system* with $A^{\forall} \in \mathbb{Q}^{\bar{m}\times n}$ and $\boldsymbol{b}^{\forall} \in \mathbb{Q}^{\bar{m}}$, $\bar{m} \in \mathbb{Z}_{\geq 0}$. To ensure that the universal player's constraints remain independent of the existential variables, we require that $A^{\forall}_{\ell,i} = 0$ for every $i \in [n]$ with $Q_i = \exists$ and for all $\ell \in [\bar{m}]$. We also assume that $\{\boldsymbol{x} \in \mathcal{D} \mid A^{\forall}\boldsymbol{x} \leq \boldsymbol{b}^{\forall}\} \neq \emptyset$, i.e., the

"*uncertainty set*" is nonempty. In presence of a universal constraint system we speak of a *QIP with polyhedral uncertainty*. To this end, we redefine the domains of universally quantified variables, by making them depended on assignments $\tilde{\boldsymbol{X}}_1, \dots, \tilde{\boldsymbol{X}}_{j-1}$ of the preceding levels. Specifically, for $j \in [k]$ with $\mathcal{Q}_j = \forall$, we define (with a slight abuse of notation)

$$\mathcal{D}_j(\tilde{\boldsymbol{X}}_1, \dots, \tilde{\boldsymbol{X}}_{j-1}) = \left\{ \boldsymbol{Y} \in \mathcal{D}_j \;\middle|\; \begin{array}{l} \exists\, \boldsymbol{X}_{j+1} \in \mathcal{D}_{j+1}, \dots, \boldsymbol{X}_k \in \mathcal{D}_k \text{ such that } A^{\forall}\boldsymbol{x} \leq \boldsymbol{b}^{\forall}, \\ \text{with } \boldsymbol{x} = (\tilde{\boldsymbol{X}}_1, \dots, \tilde{\boldsymbol{X}}_{j-1}, \boldsymbol{Y}, \boldsymbol{X}_{j+1}, \dots, \boldsymbol{X}_k) \end{array} \right\},$$

while for existentially quantified variables, the domain remains unchanged. Note that, due to the structure of the universal constraint system, only universal variables from preceding levels can influence this domain. This definition guarantees that when the universal player selects an assignment $\boldsymbol{Y}$ at level j, it obeys the lower and upper bounds and there exists an extension in later levels that results in the satisfaction of the universal constraint system. This way, no play can result in a violation of the universal constraint system. If $\bar{m} = 0$, i.e., if there are no universal constraints, using the domain described above, boils down to the bounded domain and we obtain a standard QIP. Hence, for the remainder of the paper, whenever we write $\mathcal{D}$ for a universal domain, it may also be subject to a universal constraint system and we omit stating the dependence on variables of previous levels.

We also allow for a linear objective function $\boldsymbol{c}^\top \boldsymbol{x}$, $\boldsymbol{c} \in \mathbb{Z}^n$, which changes the nature of the game for the existential player, who now has to both satisfy the constraint system and minimize the objective value. Suppose $\mathcal{Q}_1 = \exists$ and $\mathcal{Q}_k = \forall$. Then, the *QIP optimization problem* can be stated as

$$\min_{\boldsymbol{X}_1 \in \mathcal{D}_1} \max_{\boldsymbol{X}_2 \in \mathcal{D}_2} \cdots \min_{\boldsymbol{X}_{k-1} \in \mathcal{D}_{k-1}} \max_{\boldsymbol{X}_k \in \mathcal{D}_k} \boldsymbol{c}^\top \boldsymbol{x} \quad : \quad A^{\exists}\boldsymbol{x} \leq \boldsymbol{b}^{\exists}.$$

In particular, this optimization problem is feasible with optimal objective value $z^\star \in \mathbb{Z}$, if and only if the following holds: the QIP feasibility problem $\mathcal{Q}_1 \boldsymbol{X}_1 \in \mathcal{D}_1\ \mathcal{Q}_2 \boldsymbol{X}_2 \in \mathcal{D}_2\ \cdots\ \mathcal{Q}_k \boldsymbol{X}_k \in \mathcal{D}_k : \ A^{\exists}\boldsymbol{x} \leq \boldsymbol{b}^{\exists} \wedge \boldsymbol{c}^\top \boldsymbol{x} \leq z^\star$ can be won by the existential player, while for the QIP $\mathcal{Q}_1 \boldsymbol{X}_1 \in \mathcal{D}_1\ \mathcal{Q}_2 \boldsymbol{X}_2 \in \mathcal{D}_2\ \cdots\ \mathcal{Q}_k \boldsymbol{X}_k \in \mathcal{D}_k : \ A^{\exists}\boldsymbol{x} \leq \boldsymbol{b}^{\exists} \wedge \boldsymbol{c}^\top \boldsymbol{x} \leq z^\star - 1$ no winning strategy for the existential player exists. The notion of a winning strategy remains the same and we only add the term *optimal winning move* for the existential player, which is the winning move that attains the best worst-case objective.

▶ **Example 1.** Let us consider an example with $n = 5$ variables, with domains $D_1 = \{0, 1, 2\}$ and $D_i = \{0, 1\}$, for $i \in \{2, 3, 4, 5\}$. The quantification sequence (without the bounding domains for sake of presentation) is given by $\exists x_1 \forall x_2 \exists x_3 \forall x_4 \exists x_5$ and we have a universal constraint system consisting of a single constraint $x_2 + x_4 \leq 1$. Hence, $\mathcal{D}_2 = \{0, 1\}$, $\mathcal{D}_4(x_2 = 0) = \{0, 1\}$, and $\mathcal{D}_4(x_2 = 1) = \{0\}$. Let the QIP optimization problem be given by

$$\min_{x_1 \in \mathcal{D}_1} \max_{x_2 \in \mathcal{D}_2} \min_{x_3 \in \mathcal{D}_3} \max_{x_4 \in \mathcal{D}_4} \min_{x_5 \in \mathcal{D}_5} -x_1 + 2x_2 - 3x_3 + x_4 + 2x_5 : \begin{array}{rrrrrl} 2x_1 & & +x_3 & & -x_5 & \leq 4 \\ x_1 & -x_2 & +x_3 & & -x_5 & = 1 \\ & +x_2 & +x_3 & -x_4 & -x_5 & \leq 2 \\ x_1 & +x_2 & +x_3 & +x_4 & & \leq 3. \end{array}$$

Here, $x_1 = 0$ is not a winning move, as there is a countermove $x_2 = 1$ which renders the second constraint violated. There are two winning strategies $S = (s_1, s_3, s_5)$ and $T = (t_1, t_3, t_5)$ for the existential player: $s_1 = 1$, $s_3 = 1$, and $s_5 = 1 - x_2$ as well as $t_1 = 2$, $t_3 = 0$, and $t_5 = 1 - x_2$. Note, that if the universal constraint was not present, neither of them would be a winning strategy, as the fourth existential constraint is always violated in case of universal strategy $x_2 = x_4 = 1$.

The worst-case objective value of T is equal to 1, stemming from the worst case scenario $x_2 = 0$, $x_4 = 1$. The same universal assignment also defines the worst-case for S, resulting in an objective value of -1. Hence, S is a better strategy than T and in fact $x_1 = 1$ is the optimal winning move. To further clarify the used notation, $A^{\exists}_{(-x_1)}\boldsymbol{x}_{(-x_1)} \leq \boldsymbol{b}^{\exists}_{x_1=1}$ is

$$\begin{array}{cccccc} & +x_3 & & -x_5 & \leq 2 \\ -x_2 & +x_3 & & -x_5 & = 0 \\ +x_2 & +x_3 & -x_4 & +x_5 & \leq 2 \\ +x_2 & +x_3 & +x_4 & & \leq 2, \end{array}$$

which is the constraint system of the remaining QIP $\Phi[\tau]$ after making the optimal winning move $\tau = x_1 = 1$.

3 Expansion-Based Quantified Integer Programming Solver

3.1 The Framework

In this section we present our novel solution approach – the *Expansion-based Quantified Integer Programming Solver* (`EQuIPS`) – that is able to solve QIP problems with polyhedral uncertainty. `EQuIPS` is based on an algorithm that iteratively solves abstractions until convergence is achieved. The idea is essentially the same as in the solvers `RAReQS` and `QFUN` and our pseudo-code is based on the one shown in [44]. We adapt their QBF approach to general integer domains and linear functions by accounting for semantic and technical differences, introducing an expansion rule tailored to QIP, and defining a refinement step specific to QIP. The solution process starts with an empty abstraction of the full QIP – a trivial problem where only compliance with variable domains (possibly including universal constraints) have to be followed – and recursively refines the abstraction by adding found countermoves. This way, it partially expands the inner quantifiers until it is sufficient to solve the original QIP. The pseudocode in Algorithm 1 shows the main function that calls itself in a nested fashion. We refer to Appendix A.1 for a formal proof on correctness and insights regarding the underlying proof system.

Algorithm 1 `EQuIPS`$(Q\boldsymbol{X} \in \mathcal{D} : \{\Phi_1 \dots \Phi_\ell\})$ – Find Winning Move for Multi-Game.

Input: multi-game $\mathcal{Q}\boldsymbol{X} \in \mathcal{D} : \{\Phi_1 \dots \Phi_\ell\}$
Output: assignment of $\boldsymbol{X}$ that wins the multi-game or $\perp$ if no winning move exists

1: **if** each Φ_l is quantifier free **then return** `wins1`$(\mathcal{Q}\boldsymbol{X} \in \mathcal{D} : \{\Phi_1 \dots \Phi_\ell\})$
2: $\alpha \leftarrow \mathcal{Q}\boldsymbol{X} \in \mathcal{D} : \varnothing$ ▷ start with empty abstraction
3: **while** True **do**
4: $\quad\boldsymbol{\tau}' \leftarrow$ `EQuIPS`(α) ▷ find winning move for abstraction
5: $\quad$**if** $\boldsymbol{\tau}' = \perp$ **then return** $\perp$ ▷ no move for abstraction ⇒ no move for multi-game
6: $\quad\boldsymbol{\tau} \leftarrow$ `extract`$(\boldsymbol{\tau}', \boldsymbol{X})$
7: $\quad\lambda \leftarrow -1$
8: $\quad$**for** $l \in [\ell]$ **do**
9: $\quad\quad\boldsymbol{\mu} \leftarrow$ `EQuIPS` $(\Phi_l[\boldsymbol{\tau}])$ ▷ find countermove to $\boldsymbol{\tau}$
10: $\quad\quad$**if** $\boldsymbol{\mu} \neq \perp$ **then** $\lambda \leftarrow l$
11: $\quad$**if** $\lambda = -1$ **then return** $\boldsymbol{\tau}$ **else** $\alpha \leftarrow$`refine` $(\alpha, \Phi_l, \boldsymbol{\mu})$ ▷ refine abstraction

The initial input of Algorithm 1 is the QIP $\mathcal{Q}\boldsymbol{X} \in \mathcal{D} : \Phi$, i.e., a multi-game with a single subgame. In Line 1, we deal with the case where each subgame of the multi-game is quantifier free. It is noteworthy, that the `wins1` function call significantly differs from the

one used in [44]. The main reason is that in case of QBF and $\mathcal{Q} = \forall$, a winning move for the universal player can be found by solving a SAT problem consisting of the conjunction of the negated Boolean formulas of each subgame. Recall, that the goal for the existential player is to identify a move that ensures each constraint system of every subgame to be violated. In the case of QIP, it is not immediately clear what the counterpart of a negated Boolean formula would be for a system of linear constraints. We discuss the `wins1` function in more detail in Section 3.2.

In Line 2 of Algorithm 1 we initialize the abstraction, containing no subgames. A move that wins the current abstraction is found in (Line 4). If α is the empty abstraction, this call to `EQuIPS` will immediately invoke `wins1`, returning any assignment that satisfies the domain $\mathcal{D}$. When $\mathcal{Q} = \exists$, any assignment from the bounded domain may be returned. However, when $\mathcal{Q} = \forall$, compliance with the universal constraint system must also be ensured.

A technical detail to note is that the abstraction may include copies of later-stage variables (due to the subsequent `refine` call), necessitating the extraction of only those assignments corresponding to the variables of interest, $\boldsymbol{X}$, in Line 6. After obtaining the corresponding move $\boldsymbol{\tau}$, in case a countermove $\boldsymbol{\mu}$ exists, the abstraction is refined by adding the subgame that corresponds to $\boldsymbol{\mu}$ (Lines 8–11). We will provide more insights into the `refine` function in Section 3.3. If we are not able to find a move that wins the abstraction, we know by construction, that there cannot exist a winning move for the initial multi-game, and return $\perp$ in Line 5. Similarly, if no countermove to $\boldsymbol{\tau}$ can be found, this means that $\boldsymbol{\tau}$ is not only a winning move for the abstraction but also for the entire multi-game. In this case, we return $\boldsymbol{\tau}$ (see Line 11).

3.2 `wins1` on Integer Linear Programs

One crucial aspect of our algorithm is the call `wins1`$(\mathcal{Q}\boldsymbol{X} \in \mathcal{D} : \{\Phi_1 \dots \Phi_\ell\})$, where all subgames $\Phi_1 \dots \Phi_\ell$ are quantifier free, i.e., they each only contain a constraint system with variables $\boldsymbol{x} = \boldsymbol{X}$. Simply speaking, this call tries to answer the question, whether player $\mathcal{Q}$ can find an assignment of $\boldsymbol{X}$, which is a winning move for all subgames $\Phi_1, \dots, \Phi_\ell$. The notation of `wins1` is borrowed from `QFUN`, and in `QFUN`, it is more or less a call to a SAT solver. Our subroutine of `wins1`, however requires some non trivial modification. In the case of a QBF, this problem can be stated in a straight-forward manner as if $\mathcal{Q} = \exists$ one has to find an assignment satisfying the conjunction of the ℓ copies of the Boolean function, while if $\mathcal{Q} = \forall$ a solution to the conjunction of all negated Boolean formulas must be found. But in particular regarding the latter case, there is no counterpart in integer programming: there is no notion of a "negated constraint system".

For $l \in [\ell]$, let $A_l^\exists \boldsymbol{x} \leq \boldsymbol{b}_l^\exists$ be the constraint system corresponding to subgame Φ_l. Recall, that if $\mathcal{Q} = \exists$, a winning move ensures that the constraint system of each subgame is satisfied. Hence, in order to find a winning move, we need to find a solution to Problem (1):

$$A_l^\exists \boldsymbol{x} \leq \boldsymbol{b}_l^\exists \quad \forall l \in [\ell] \tag{1a}$$

$$\boldsymbol{x} \in \mathcal{D} \tag{1b}$$

This is a standard integer program and can be solved using any standard solver.

If $\mathcal{Q} = \forall$, we need to find an assignment of $\boldsymbol{x} \in \mathcal{D}$, such that all constraint systems are violated, while obeying the own domain, i.e., the uncertainty set. Violating a constraint system means that at least one of its constraints is not satisfied. To the end of modeling this as an integer linear program, let $\boldsymbol{L}^l \in \mathbb{Q}^m$ be a vector for each $l \in [\ell]$, where

$$L_j^l \leq \min_{\boldsymbol{x}\in\mathcal{D}}(A_l^\exists)_{j,\star}\boldsymbol{x} = \sum_{\substack{i\in[n]\\(A_l^\exists)_{j,i}<0}} (A_l^\exists)_{j,i}u_i + \sum_{\substack{i\in[n]\\(A_l^\exists)_{j,i}\geq 0}} (A_l^\exists)_{j,i}l_i$$

for every $j \in [m]$. In other words, the j-th entry of $\boldsymbol{L}^l$ provides a lower bound that is not larger than the minimum possible value of the left-hand side of row j in constraint system l. As a result, the inequality $\boldsymbol{L}^l \leq A_l^\exists \boldsymbol{x}$ is trivially fulfilled for any $\boldsymbol{x} \in \mathcal{D}$.

These lower bounds only need to be computed once at the start of our solver since the constraint system remains (basically) the same across all subgames, only differing in the values of already assigned variables. One also could refine these bounds dynamically for each subgame by considering already assigned variables, aiming to accelerate the IP solution process through potentially improved relaxations. However, this approach comes at the cost of additional computational effort, as the bounds must be recomputed at each invocation of `wins1`. In our implementation, we opted for the weaker bounds that only need to be calculated once.

Furthermore, we need to be able to certify a violation of a constraint. To this end, we need the following lemma.

▶ **Lemma 2.** *Given a linear constraint $\sum_{i\in[n]} a_i x_i \leq b$ with rational coefficients $a_i, b \in \mathbb{Q}$ and integer variables x_i. Then, for any integer assignment $\tilde{\boldsymbol{x}}$, it holds that $\sum_{i\in[n]} a_i\tilde{x}_i \not\leq b \Leftrightarrow \sum_{i\in[n]} a_i\tilde{x}_i \geq b + r$, where $r = \frac{1}{d}$, for $d = lcd\{a_1, \ldots, a_n, b\}$, being the reciprocal of the lowest common denominators of the a_i and b.*

Proof. Let d be the lowest common denominator of the $n+1$ parameters. Then the constraint can be rewritten as $\sum_{i\in[n]} \frac{\tilde{a}_i}{d} x_i \leq \frac{\tilde{b}}{d}$, with integers $\tilde{a}_i$ and $\tilde{b}$. For any assignment $\tilde{\boldsymbol{x}}$ with $\sum_{i\in[n]} a_i\tilde{x}_i > b$ the gap between the right-hand side and the left-hand side can be stated as $\left|\frac{\tilde{b}-\sum_{i\in[n]}\tilde{a}_i x_i}{d}\right| > 0$, where the numerator is integer. Hence, a lower bound for this gap is attained if the numerator is equal to one, i.e., $\frac{1}{d}$ is a lower bound on the violation of the constraint, which proves the claim. ◀

This lemma shows the need for integrality of universally quantified variables in our approach, as otherwise, we would not be able to specify a value of minimal violation, as a continuous variable could violate the right-hand side by an arbitrarily small value. For existentially quantified variables this in principle is not necessary in our current setting. Now, let $\boldsymbol{r}^l \in \mathbb{Q}^m$ be a vector with positive entries less than or equal to the reciprocals of the lowest common denominators of the rows of $A_l^\exists$ and $\boldsymbol{b}_l^\exists$, ensuring for any row $j \in [m]$ and any $\boldsymbol{x} \in \mathcal{D}$ that $(A_l^\exists)_{j,\star}\boldsymbol{x} \not\leq (\boldsymbol{b}_l^\exists)_k \Leftrightarrow (A_l^\exists)_{j,\star}\boldsymbol{x} \geq (\boldsymbol{b}_l^\exists)_k + r_j^l$. Note that the lowest common denominator of the $n+1$ rational numbers can be computed in polynomial time, using the Euclidean algorithm, assuming that the denominators of the coefficients are know. But also note that smaller values to bound the gap between right-hand side and left-hand side are allowed. E.g. for the constraint $0.5x_1 + 2x_2 <= 4$ the designated value would be 0.5, as the lowest common denominator of 0.5, 2 and 4 is 2 with a reciprocal of 0.5. On the other hand, finding the number with the most decimal places also is valid. Let p be this number. Then setting $r_j^l = 10^{-p}$ also suffices. In the above example this equates to setting $r_j^l = 0.1$. The latter case is what we implemented. Further note, that that if all coefficients are integers, this value always can be set to one.

Now, consider the following integer linear program (2):

$$-A_l^{\exists}\boldsymbol{x} - (\boldsymbol{L}^l - \boldsymbol{b}_l^{\exists} - \boldsymbol{r}^l)\boldsymbol{y}^l \leq -\boldsymbol{L}^l \qquad \forall l \in [\ell] \tag{2a}$$

$$v_l \leq \sum_{j\in[m]} y_j^l \qquad \forall l \in [\ell] \tag{2b}$$

$$v \leq v_l \qquad \forall l \in [\ell] \tag{2c}$$

$$v \geq 1 \tag{2d}$$

$$A^{\forall}\boldsymbol{x} \leq \boldsymbol{b}^{\forall} \tag{2e}$$

$$\boldsymbol{x} \in \mathcal{D} \tag{2f}$$

$$v, v_1, \ldots, v_\ell \in \{0,1\} \tag{2g}$$

$$\boldsymbol{y}^l \in \{0,1\}^m \qquad \forall l \in [\ell] \tag{2h}$$

The idea is that the indicator variable v only can be set to 1, if all ℓ systems are violated by an assignment of $\boldsymbol{x}$. We can immediately see, that in order to set v to 1, all v_l must be set to 1. Hence, for any $l \in [\ell]$ there has to exist at least one constraint $j \in [m]$ for which $y_j^l = 1$. So let us consider the setting of y_j^l. Setting $y_j^l = 0$ can never result in a violation of the respective constraint, as $A_l^{\exists}\boldsymbol{x} \geq \boldsymbol{L}^l$ is always true. On the other hand, it is only feasible to set y_j^l to 1, if $(A_l^{\exists})_{k,*}\boldsymbol{x} \geq (b_l^{\exists})_j + r_j^l$ holds, which is only the case, if $\boldsymbol{x}$ violates constraint j of system l. Consequently, v_l is an indicator whether constraint system l is violated. Only if we find some $\boldsymbol{x}$ that violates all constraint systems while at the same time obeys the uncertainty set given by $A^{\forall}\boldsymbol{x} \leq \boldsymbol{b}^{\forall}$, we can also set $v = 1$. In other words: If and only if there exists an assignment of $\boldsymbol{x} \in \mathcal{D}$ with $A_l^{\exists}\boldsymbol{x} \not\leq \boldsymbol{b}_l^{\exists}$ for all $l \in [\ell]$, Problem (2) has a feasible solution.

▶ **Example 3.** Consider the constraint $0.5x_1 + 2x_2 <= 4$ and variable bounds $-2 \leq x_1 \leq 2$ and $0 \leq x_2 \leq 3$. The lower bound L for the left-hand side is -1. We set $r = 0.5$ to indicate the minimal violation of this constraint. Consequently, the corresponding Constraint (2a) is given by $-0.5x_1 - 2x_2 + 5.5y \leq 1$ and in particular, in case of assigning $y = 1$, $0.5x_1 + 2x_2 \geq 4.5$ must be true, indicating the violation of the original constraint.

The pseudocode of the `wins1` function is presented in Algorithm 2. In our implementation we utilize the solver `GUROBI` [33] to solve Problems (1) and (2).

Algorithm 2 `wins1`$(\mathcal{Q}\boldsymbol{X} \in \mathcal{D} : \{\Phi_1 \ldots \Phi_\ell\})$ – Solve Multi-Game with a Single Move.

Input: multi-game $\mathcal{Q}\boldsymbol{X} \in \mathcal{D} : \{\Phi_1 \ldots \Phi_\ell\}$ with all Φ_l quantifier free
Output: assignment of $\boldsymbol{X}$ that wins the multi-game or $\perp$ if no winning move exists
1: **if** $\mathcal{Q} = \exists$ **then** $\pi \leftarrow$ Problem (1) **else** $\pi \leftarrow$ Problem (2)
2: **if** π is feasible **then return** solution on π **else return** $\perp$

3.3 Refinement by Expansion

It can be argued that the unique aspect of `RAReQS` and `QFUN` that sets it apart from other CEGAR approaches is that they extend the number of variables being looked at in the abstraction via expansion. This is much easier to spot in the original `RAReQS` than in `QFUN` (our description in Algorithm 1 is based off `QFUN`). In `QFUN` and our description this is achieved by having multiple listed subproblems after the outer quantifier block, as these subgames technically each have separate variables. The function `refine` adds an additional subgame to the abstraction, based on the countermove $\boldsymbol{\mu}$, that was found to beat our move $\boldsymbol{\tau}$ in one of the original subgames.

Given an abstraction $\alpha = \mathcal{Q}\boldsymbol{X} \in \mathcal{D}_{\boldsymbol{X}} : \{\Psi_1, \ldots, \Psi_n\}$, with subgames Ψ_i, $i \in [n]$, for which a winning move $\boldsymbol{\tau}$ was found. Let Φ be another subgame, in which $\boldsymbol{\tau}$ is not a winning move, i.e., there exists a countermove $\boldsymbol{\mu}$ that wins $\Phi[\boldsymbol{\tau}]$. Let the quantification sequence of Φ start with $\bar{\mathcal{Q}}\boldsymbol{Y}$. If $\Phi = \bar{\mathcal{Q}}\boldsymbol{Y} \in \mathcal{D}_{\boldsymbol{Y}} : A^{\exists}\boldsymbol{x} \leq \boldsymbol{b}^{\exists}$, then

$$\texttt{refine}(\alpha, \Phi, \boldsymbol{\mu}) = \mathcal{Q}\boldsymbol{X} \in D_{\boldsymbol{X}} : \left\{\Psi_1, \ldots, \Psi_n, A^{\exists}_{(-\boldsymbol{Y})}\boldsymbol{x}_{(-\boldsymbol{Y})} \leq \boldsymbol{b}^{\exists}_{\boldsymbol{Y}=\boldsymbol{\mu}}\right\},$$

i.e., the refined abstraction contains an additional constraint system accounting for the scenario of the found countermove. If $\Phi = \bar{\mathcal{Q}}\boldsymbol{Y} \in \mathcal{D}_{\boldsymbol{Y}}\mathcal{Q}\boldsymbol{Z} \in \mathcal{D}_{\boldsymbol{Z}} : \Lambda$ for a QIP Λ, then

$$\texttt{refine}(\alpha, \Phi, \boldsymbol{\mu}) = \mathcal{Q}\boldsymbol{X} \in \mathcal{D}_{\boldsymbol{X}}\boldsymbol{Z}_{\Phi}^{(\boldsymbol{Y}=\boldsymbol{\mu})} \in \mathcal{D}_{\boldsymbol{Z}} : \left\{\Psi_1, \ldots, \Psi_n, \Lambda_{\Phi}^{(\boldsymbol{Y}=\boldsymbol{\mu})}[\boldsymbol{\mu}]\right\},$$

where $\boldsymbol{Z}_{\Phi}^{(\boldsymbol{Y}=\boldsymbol{\mu})}$ is a copy of $\boldsymbol{Z}$ that represents the move of $\boldsymbol{Z}$ in case $\boldsymbol{Y}$ is set to $\boldsymbol{\mu}$, and the variables of $\Lambda_{\Phi}^{(\boldsymbol{Y}=\boldsymbol{\mu})}$ are copies of the variables of Λ having the same annotation as $\boldsymbol{Z}_{\Phi}^{(\boldsymbol{Y}=\boldsymbol{\mu})}$. When $\boldsymbol{Z}$ is universal, we also need to make sure it satisfies the uncertainty set and thus $\mathcal{D}_{\boldsymbol{Z}}$ is meant to include the respective constraints on the annotated variables. We sometimes write $\boldsymbol{Z}^{(\boldsymbol{\mu})}$ instead of $\boldsymbol{Z}^{(\boldsymbol{Y}=\boldsymbol{\mu})}$.

▶ **Example 4.** Consider the QIP $\exists x_1 x_2 \forall z_1 z_2 \exists t d : (x_1 + x_2 + t - 2d = 0) \wedge (z_1 + z_2 + t \geq 1) \wedge (-z_1 - z_2 - t \geq -2)$ with all binary domains. Consider the outer level. Initially α, the abstraction, will be empty, which means that any binary assignment of the variables x_1 and x_2 is feasible. If we choose $\boldsymbol{\tau} = (0, 0)$, the universal response is to assign $z_1 = z_2 = 0$, at which point the universal player wins. Hence, a countermove $\boldsymbol{\mu} = (0, 0)$ to $\boldsymbol{\tau}$ is found and consequently the abstraction α is refined to become $\alpha = \exists x_1 x_2 t^{(00)} d^{(00)} : (x_1 + x_2 + t^{(00)} - 2d^{(00)} = 0) \wedge (t^{(00)} \geq 1) \wedge (-t^{(00)} \geq -2)$. The `EQuIPS` call on this refined abstraction will end up in a call of the `wins1` function, as after the initial existential quantifier, no further quantifiers follow. A solution to the respective constraint system is $\boldsymbol{\tau}' = (0, 1, 1, 1)$, which contains assignments of x_1, x_2, $t^{(00)}$, and $d^{(00)}$. Extracting the values of the relevant variables x_1 and x_2 we obtain $\boldsymbol{\tau} = (0, 1)$. We again check whether we find a countermove to $\boldsymbol{\tau}$, in which case $z_1 = z_2 = 1$ is produced. We once again adapt the abstraction by calling

$$\texttt{refine}(\alpha, \forall z_1 z_2 \exists t d : (x_1 + x_2 + t - 2d = 0) \wedge (z_1 + z_2 + t \geq 1) \wedge (-z_1 - z_2 - t \geq -2), (1, 1)),$$

yielding the refined abstraction

$$\exists x_1 x_2 t^{(00)} d^{(00)} t^{(11)} d^{(11)} : \left\{\begin{array}{r}\left((x_1 + x_2 + t^{(00)} - 2d^{(00)} = 0) \wedge (t^{(00)} \geq 1) \wedge (-t^{(00)} \geq -2)\right), \\ \left((x_1 + x_2 + t^{(11)} - 2d^{(11)} = 0) \wedge (t^{(11)} \geq -1) \wedge (-t^{(11)} \geq 0)\right)\end{array}\right\},$$

with two subgames. As each subgames is quantifier free, another call to `wins1` is invoked and the IP solver is called on the constraint system

$$\begin{array}{lll} x_1 + x_2 + t^{(00)} - 2d^{(00)} = 0 & t^{(00)} \geq 1 & -t^{(00)} \geq -2 \\ x_1 + x_2 + t^{(11)} - 2d^{(11)} = 0 & t^{(11)} \geq -1 & -t^{(11)} \geq 0 \\ x_1, x_2, t^{(00)}, d^{(00)}, t^{(11)}, d^{(11)} \in \{0, 1\}. & & \end{array}$$

As this IP is infeasible, we know that there is no move for (x_1, x_2) that wins the abstraction, and hence, there cannot exist a move for (x_1, x_2) that wins the game.

3.4 Optimization

3.4.1 Optimization Method 1: Binary Search

In Section 2, we introduced the QIP optimization problem, for which a search-based solution approach exists [40]. We now want to utilize the presented expansion-based approach, to obtain another solution tool for the optimization problem. To this end, we assume that a linear objective is given with objective coefficients $\boldsymbol{c} \in \mathbb{Z}^n$. For clarity of presentation, we assume an existential starting player in this case. We have already drawn the connection between the QIP optimization problem and its decision version, where the objective function is moved to the constraints and one asks for the existence of a solution with objective value less than or equal to some value z. As all variables are bounded and the objective value only attains integer values, we can compute lower and upper bounds on the objective value. In particular, for the optimal objective value we know $z^\star \in [\min_{\boldsymbol{x}\in\mathcal{D}} \boldsymbol{c}^\top \boldsymbol{x}, \max_{\boldsymbol{x}\in\mathcal{D}} \boldsymbol{c}^\top \boldsymbol{x}]$. Now, we can conduct a binary search to close in on the optimal value as shown in Algorithm 3.

Algorithm 3 Optimization Framework utilizing `EQuIPS`.

Input: QIP optimization problem
Output: $\perp$, if instance is infeasible and otherwise the optimal objective value.

$LB \leftarrow \min_{\boldsymbol{x}\in\mathcal{D}} \boldsymbol{c}^\top \boldsymbol{x}$
$UB \leftarrow \max_{\boldsymbol{x}\in\mathcal{D}} \boldsymbol{c}^\top \boldsymbol{x}$
$z \leftarrow UB$
run `EQuIPS` on QIP decision problem with additional constraint $\boldsymbol{c}^\top \boldsymbol{x} \leq z$
if $\perp$ **then return** *"Instance is infeasible"*
while $UB - LB > 0$ **do**
    $z \leftarrow (LB + UB)/2$
    run `EQuIPS` on QIP decision problem with additional constraint $\boldsymbol{c}^\top \boldsymbol{x} \leq z$
    if feasible **then** $UB \leftarrow z$ **else** $LB \leftarrow z + 1$
return UB

3.4.2 Optimization Method 2: Mixing Methods

Repeatedly running the solver in a binary search can end up being more expensive than necessary. Our working hypothesis is that the existing search-based solver `Yasol` [40] is good at finding solutions, but slow at verifying optimality, while `EQuIPS` suffers from the opposite problem: it is much better at showing inconsistencies from objective value bounds that are too tight, but slow at finding good initial solutions. Therefore we propose to combine the approaches, by letting `Yasol` search for good solutions and use `EQuIPS` to verify optimality. In particular, every time `Yasol` finds a new solution – an improved value for UB – we can call `EQuIPS` and try to verify that no solution with objective value at most $UB - 1$ exists by adding the respective constraint on the objective function. Then, if and only if, UB is optimal, no winning move for the existential player will be found. Doing this sequentially, i.e., stopping the search process of `Yasol` while `EQuIPS` tries to verify optimality, of course can be detrimental, as in the early phase of the optimization process, `Yasol` tends to find better solutions quickly. Thus, calling `EQuIPS` for every newly found solution can become inefficient. We instead think of initiating a parallel process of `EQuIPS` while `Yasol` continues its search. If `EQuIPS` certifies that no better solution exists, `Yasol` can terminate early. Conversely, if `Yasol` finds a new solution, the current `EQuIPS` process can be terminated and restarted with the updated bound.

It is noteworthy, that if EQuIPS returns a winning move with an objective function bounded by z, it cannot be concluded that a solution with objective value z exists, but only that a solution with some value less than or equal to z exists. Hence, Yasol cannot extract a newly found solution from a call to EQuIPS that does not verify optimality.

In this combined approach we also see options to exploit further synergies, where not only Yasol profits from EQuIPS, but also the other way around. When Yasol continues the search process after finding a new incumbent solution, it would incorporate all information it has gathered during the solution process. Running EQuIPS with a new bound on the objective function, on the other hand, is essentially starting again from scratch, which we want to avoid. Since constraints are learned during the search process of Yasol, adding them to EQuIPS is one way to transfer learned information from one solver to the other. Any learned constraint holds information on what variable assignments will not lead to a (improved) solution. It is worth mentioning that Yasol does not explicitly track all its progress through learning constraints, but also implicitly through branching decisions. But each branch of the search tree that is completed without a found (better) solution can be interpreted as a learned constraint. Adding such constraints to the verification instance, has the potential to improve the runtime of EQuIPS, with the risk of increasing the size of the instance too much, making it harder for the underlying IP solver to solve Problems (1) and (2). This issue has similarities to the selection of cutting planes for solving integer programs (see, e.g., [22]) and further research needs to be done for our special case.

▶ **Theorem 5.** *Let $\Pi\phi$ be a QIP with Π a quantifier prefix and ϕ an IP and assume* Yasol *is a correct clause learning algorithm for QIP and also complete, in that it will eventually find the optimal solution. Suppose* Yasol *learns clauses $C_1 \dots C_n$ on the way to learning a solution with objective function value of v. Then $\Pi\phi \wedge F$ is false if and only if $\Pi\phi \wedge C_1 \dots C_n \wedge D \wedge F$ is false, where F is a constraint saying the objective function is strictly less than v.*

Proof. Suppose Yasol has learned clauses $C_1 \dots C_n$ and a solution with value v. Let us consider the QIP feasibility problem $\Pi\phi \wedge C_1 \dots C_n \wedge F$. This QIP is either true, resulting in the existence of a strategy with value less than v, or there is no such strategy, rendering the QIP false. If $\Pi\phi \wedge C_1 \dots C_n \wedge F$ is true, then obviously $\Pi\phi \wedge F$ also must be true, as it contains less existential constraints. Now assume $\Pi\phi \wedge C_1 \dots C_n \wedge F$ is false. Assume $\Pi\phi \wedge F$ is true, which means that there exists a solution with objective value strictly less than v. As Yasol is complete, it eventually has to find this solution. However, $\Pi\phi \wedge C_1 \dots C_n \wedge F$ being false, implies that Yasol can no longer find the solution if it restarted after learning $C_1 \dots C_n$ (which is an option for Yasol to perform after clause learning). Consequently, $\Pi\phi \wedge F$ also must be false. ◀

4 A New Challenging Problem Class: QRandomParity

QRandomParity is a combination of QParity, which are known to be hard for QCDCL based QBF solvers [10], and RandomParity which are hard for CDCL based SAT solvers [17].

Given an integer n and a random permutation σ on $[n]$. Consider the Quantified Boolean problem

$$\begin{aligned}\exists x_1 \dots x_n \forall z \exists t_2 \dots t_n \exists s_2 \dots s_n.\ & t_2 = (x_1 \oplus x_2) \wedge \dots \wedge t_i = (t_{i-1} \oplus x_i), \dots \wedge \\ & s_2 = (x_{\sigma(1)} \oplus x_{\sigma(2)}) \wedge \dots \wedge s_i = (s_{i-1} \oplus x_{\sigma(i)}), \dots \wedge \\ & (z \to \neg t_n) \wedge (\neg z \to s_n)\end{aligned}$$

Both t_n and s_n compute the parity of the x variables but use a different ordering. In particular, $t_n = s_n$ must be fulfilled. If we take the full expansion we get a contradiction, because parity is associative and commutative. These families have been shown hard for CDCL solvers like `CaDiCaL` in both theory and experiments [17]. This is because of the standard encodings of the parity problems into clauses. Typically one encodes $a = (b \oplus c)$ using four clauses $(\neg a \vee b \vee c)$, $(a \vee \neg b \vee c)$, $(a \vee b \vee \neg c)$, and $(\neg a \vee \neg b \vee \neg c)$.

Note that, extension variables can be used to produce formulas easy for resolution. In this case it has been shown [18] that only $O(n \log n)$ many extension variables are needed before we can get linear size resolution refutation. In pseudo-Boolean constraints the extension variables come more naturally into a standard encoding of parity, but require a new variable e for each constraint. Therefore $a = (b \oplus c)$ can be represented by the constraint $a + b + c = 2e$.

This encoding allows a short cutting planes proof. We simply can add all constraints to get $x_1 + \cdots + x_n + t_n + 2t_2 \cdots + 2t_{n-1} - 2e_2 \cdots - 2e_n = 0$ and this can be repeated for $x_1 + \cdots + x_n + s_n + 2s_2 \cdots + 2s_{n-1} - 2f_2 \cdots - 2f_n = 0$.

Subtracting one from the other we get $t_n - s_n + 2\sum_{i=2}^{n}(t_i - e_i - (s_i - f_i)) = 0$. The only integer solution to this equality is to have $t_n = s_n$, as otherwise an odd number would be on the left-hand side of the constraint, and cutting planes finds this via the division rule [29]. The idea is, that IP solvers, which have access to several preprocessing techniques that utilize constraint aggregation (e.g., see [28, 50, 52]), may be able to handle this as well.

The QRandomParity problem in clausal form (as QBF) is given by the quantification sequence $\exists x_1 \dots x_n \forall u \exists t_2 \dots t_n, s_1 \dots s_n$ (with all Boolean variables) and matrix

$$(\bar{x}_1 \vee x_2 \vee t_2)\ (x_1 \vee \bar{x}_2 \vee t_2)\ (x_1 \vee x_2 \vee \bar{t}_2)\ (\bar{x}_1 \vee \bar{x}_2 \vee \bar{t}_2) \tag{3a}$$

$$(\bar{x}_i \vee t_{i-1} \vee t_i)\ (x_i \vee \bar{t}_{i-1} \vee t_i)\ (x_i \vee t_{i-1} \vee \bar{t}_i)\ (\bar{x}_i \vee \bar{t}_{i-1} \vee \bar{t}_i) \text{ for } i = 3 \text{ to } n \tag{3b}$$

$$(\bar{x}_{\sigma(1)} \vee x_{\sigma(2)} \vee s_2)\ (x_{\sigma(1)} \vee \bar{x}_{\sigma(2)} \vee s_2)\ (x_{\sigma(1)} \vee x_{\sigma(2)} \vee \bar{s}_2)\ (\bar{x}_{\sigma(1)} \vee \bar{x}_{\sigma(2)} \vee \bar{s}_2) \tag{3c}$$

$$(\bar{x}_{\sigma(i)} \vee s_{i-1} \vee s_i)\ (x_{\sigma(i)} \vee \bar{s}_{i-1} \vee s_i)\ (x_{\sigma(i)} \vee s_{i-1} \vee \bar{s}_i)\ (\bar{x}_{\sigma(i)} \vee \bar{s}_{i-1} \vee \bar{s}_i) \text{ for } i = 3 \text{ to } n \tag{3d}$$

$$(\bar{u} \vee \bar{t}_n)\ (u \vee s_n), \tag{3e}$$

where each (3a)-(3d) encodes an XOR relation. In linear form (as QIP), all variables are binary and their sequence is given by $\exists x_1 \dots x_n \forall u \exists t_2 \dots t_n, d_2 \dots d_n, s_1 \dots s_n, e_2 \dots e_n$, where a linear encoding of the XOR relation is used, needing auxiliary variables d and e:

$$x_1 + x_2 + t_2 = 2d_2 \qquad x_{\sigma(1)} + x_{\sigma(2)} + s_2 = 2e_2 \tag{4a}$$

$$t_{i-1} + x_i + t_i = 2d_i \qquad s_{i-1} + x_{\sigma(i)} + s_i = 2e_i \qquad \text{for } i \in \{3, \dots, n\} \tag{4b}$$

$$-u - t_n \geq -1 \qquad u + s_n \geq 1. \tag{4c}$$

5 Experimental Evaluation

For our experiments, we compiled `EQuIPS` with `GUROBI` 12.0 and `Yasol`, which utilizes a linear programming solver, is compiled with `CPLEX` 22.1. When installing `QFUN` with the respective SAT solvers, we used the provided script and we used `DepQBF` 6.01 and the latest version of `z3`.

5.1 QRandomParity

As argued in Section 4 we expect our expansion-based solver to perform well on instances of type QRandomParity, compared to search-based algorithms like `Yasol` [40] and `DepQBF` [45]. We also tested the solver `z3` [21], which is capable of handling such instances. Furthermore, we are interested in the comparison of `EQuIPS` against state-of-the-art expansion-based solvers

from the QBF community such as QFUN [44]. QFUN allows the integration of several SAT solvers, and we compiled it once with CaDiCaL [11] and once with CryptoMiniSat [60], which we refer to as $\texttt{QFUN}_{\text{cadi}}$ and $\texttt{QFUN}_{\text{cms}}$, respectively. Here we expect $\texttt{QFUN}_{\text{cms}}$ to perform better on QRandomParity instances, due to the better handling of XOR clauses by CryptoMiniSat compared to other SAT solvers. For Yasol and EQuIPS, we use the QIP Encoding (4), while for the other solvers we use QBF Encoding (3). Notably, we tested both encodings with z3 and observed that it performed better on the QBF formulation than on the QIP formulation. Consequently, we report only its performance on Encoding (3).

For each of the following experiments, we created 100 instances for each n (only varying in the random permutation of the variables). Experiments were conducted on an AMD Ryzen 9 5900X processor (3.70 GHz) with 128 GB RAM, imposing a 1800 seconds time limit per instance and restricting each process to a single thread. In a first experiment, for $n \in \{10, 12, \ldots, 26\}$ we compare all solvers and show the results in a Cactus plot in Figure 1.

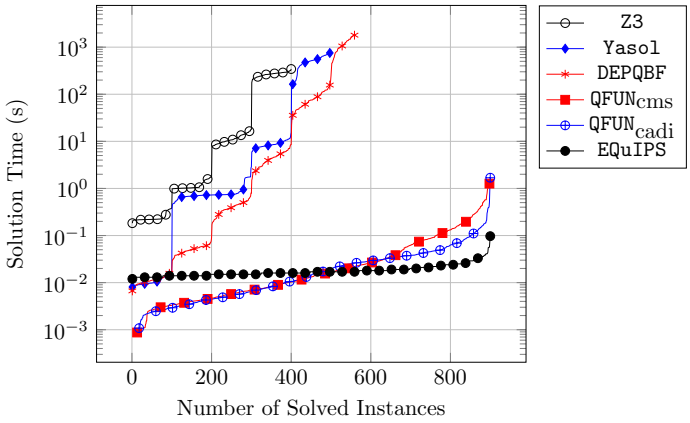

Figure 1 Cactus plot. Algorithms that appear further to the right and closer to the bottom solve more instances faster, indicating better performance.

As expected, we can observe the expansion-based solvers outclass the search-based solvers DepQBF and Yasol as well as z3. $\texttt{QFUN}_{\text{cadi}}$ and $\texttt{QFUN}_{\text{cms}}$ basically have the same performance, while our approach has the most constant behavior, outperforming all solvers for $n \geq 22$.

For even larger values of n, $\texttt{QFUN}_{\text{cadi}}$ quickly reaches its limit, incapable of solving these instances before timeout, while $\texttt{QFUN}_{\text{cms}}$ still can solve such instances easily. In particular, for $n = 100$, no instance was solved by $\texttt{QFUN}_{\text{cadi}}$ before the timeout. This is likely due to the special handling of XOR formulations by CryptoMiniSat, that CaDiCaL lacks. Hence, for larger values of n we restrict our comparison to $\texttt{QFUN}_{\text{cms}}$ and EQuIPS, as shown in Table 1.

Table 1 Median runtimes and (if existent) number of not solved QRandomParity instances.

n	100	200	300	400	500	600	700	800	900	1000
EQuIPS	0.06	0.18	0.37	0.68	0.99	1.31	1.72	2.13	2.64	3.44/10
$\texttt{QFUN}_{\text{cms}}$	1.67	2.21	2.09	2.18	2.37	2.13/1	2.51/2	2.22/3	3.04/4	-/100

For an increasing value of n, our approach consistently outperforms $\texttt{QFUN}_{\text{cms}}$. However, for very large n, both solvers sometimes fail to return a solution within the time limit. This is quite surprising, as runtime trends did not indicate a sharp increase in solution times. We further investigated this by analyzing the behavior of GUROBI on the fully expanded

problem for large values of n. It became evident that for smaller instances, GUROBI could solve them entirely during preprocessing, without initiating a search. However, for larger instances, preprocessing terminated prematurely before infeasibility was detected, forcing the solver into a search phase from which it never returned. For some instances, we were able to fine tune GUROBI parameters, but not to the extend of solving instances with $n \geq 1500$. We suspect a similar phenomenon occurs with CryptoMiniSat. These observations align with our hypothesis that aggregation techniques efficiently prove infeasibility for smaller instances, but as the problem size grows, identifying the right constraints for aggregation becomes increasingly difficult. This, in turn, leads GUROBI to halt preprocessing prematurely and initiate an exhaustive search instead.

For all experiments, QFUN and DepQBF were given instances in the QDIMACS file format. Surprisingly, testing the QCIR format – where we expected $\texttt{QFUN}_{\text{cms}}$ to perform even better – resulted in worse performance. Additionally, we evaluated EQuIPS on the QBF formulation of QRandomParity, where all constraints are clauses (see Encoding (3) in the appendix). While EQuIPS could still solve instances of size 100 within seconds, it failed to solve any instance of size 200. This further supports our claim that leveraging the modeling capabilities of linear constraints can be advantageous.

5.2 Multilevel Critical Node Problem

To evaluate our approach on optimization instances, we consider the multilevel critical node problem (MCN) as introduced in [1]. Details on the problem and the QIP encoding can be found in Appendix A.2. We compare the performance of Yasol, EQuIPS (utilizing binary search), and the baseline column- and row-generation approach ($\texttt{MCN}_{CR}$) from [1]. Note that $\texttt{MCN}_{CR}$ is essentially a scenario generation approach that uses dualization techniques to approximate the optimal solution of the adversary problem. In contrast, the approach provided by Yasol and EQuIPS is much more straightforward and easy to use, requiring only a problem encoding as QIP. Notably, we do not compare our solver to the enhanced techniques from [1], as our goal is to demonstrate the model-and-run potential of QIPs as a baseline for such instances.

The $\texttt{MCN}_{CR}$ algorithm was executed using Python 2.7.18, with mixed-integer linear programs solved via IBM CPLEX 12.9. Experiments were conducted on an AMD EPYC 9474F 48-Core Processor (3.60 GHz) with 256 GB RAM, imposing a two-hour time limit per instance and restricting each process to a single thread. We used the provided instances[2], consisting of randomly generated undirected graphs with $|V| \in \{20, 40, 60, 80, 100\}$, a density of 5%, and various budget limit configurations (Ω, Φ, and Δ). Figure 2 presents a Cactus plot, revealing key performance trends. Our approach exhibits higher run times on instances that are generally solved quickly. However, EQuIPS performs well on larger instances with inherently higher run times. Overall, our method solves the second-highest number of instances (465), compared to 431 solved by Yasol and 475 by $\texttt{MCN}_{CR}$.

5.3 Further Experiments

To further evaluate our expansion-based approach, we conducted additional experiments (see Appendix A.3 for details). The key findings are as follows. First, using EQuIPS to verify the optimality of incumbent solutions found during the search process of Yasol is promising.

[2] Instances, optimal solutions, and algorithms from [1] were provided at https://github.com/mxmmargarida/Critical-Node-Problem.

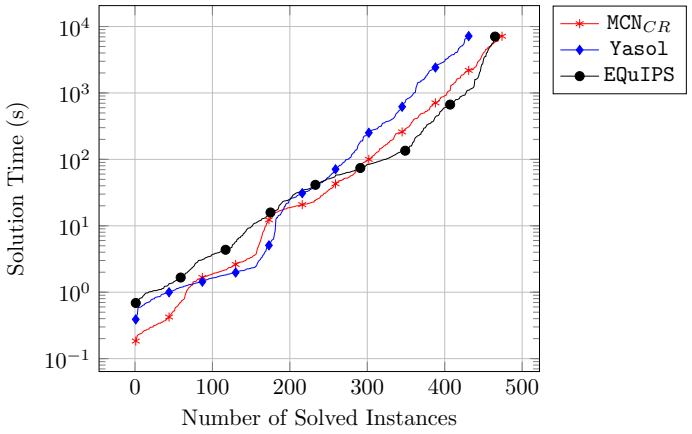

Figure 2 Cactus plot for experiments on MCN test set.

This hybrid approach can be enhanced by incorporating selected learned constraints from `Yasol` into the `EQuIPS` verification instance. Second, `EQuIPS` does not always outperform `Yasol`– especially on instances with multiple quantifier alternations – a result that aligns with similar observations in QBF. Finally, several heuristic improvements remain to be explored. For example, solving the IP relaxation to obtain an initial winning move shows potential but may increase run times when the full expansion is eventually required.

6 Conclusion

In this paper, we presented an expansion-based approach for quantified integer programming, a rarely explored direction in quantified constraint programming. Our method leverages the power of state-of-the-art integer linear programming solvers to handle abstractions – partial expansions of the quantified program.

`EQuIPS` offers advantages over existing QIP solvers like `Yasol` and various QBF approaches. It inherits the benefits of expansion-based solvers over search-based solvers in QBF while also utilizing the modeling flexibility of linear constraints. Additionally, we observe that QBF solvers incorporating XOR reasoning and expansion, such as `QFUN` with `CryptoMiniSat`, perform competitively with `EQuIPS`.

Our experiments show that `EQuIPS` offers advantages in certain cases. We demonstrate performance improvements on different families that other approaches neglect, strengthening the case for combining methods and illustrating how such combinations can be effective in practice.

Future improvements could integrate machine learning techniques to enhance abstraction construction. While `QFUN` employs decision trees, our integer programming setting allows for numerical machine learning methods such as support vector machines. Additionally, the counterexample generation in `wins1` can be heuristically guided by adding an objective to the underlying integer programs, e.g., choosing universal assignments that maximize constraint violation or existential assignments that optimize the objective. To further improve optimization capabilities, strategy extraction as well as directly incorporating optimization aspects into our framework are also promising directions. Furthermore, parallelizing both the exploration of countermoves in each sub-game and the objective-value checks during binary search should improve performance.

References

1 Andrea Baggio, Margarida Carvalho, Andrea Lodi, and Andrea Tramontani. Multilevel approaches for the critical node problem. *Operations Research*, 69(2):486–508, 2021. doi:10.1287/opre.2020.2014.

2 Haniel Barbosa, Andrew Reynolds, Daniel El Ouraoui, Cesare Tinelli, and Clark Barrett. Extending SMT solvers to higher-order logic. In *Automated Deduction–CADE 27: 27th International Conference on Automated Deduction, Natal, Brazil, August 27–30, 2019, Proceedings 27*, pages 35–54. Springer, 2019. doi:10.1007/978-3-030-29436-6_3.

3 Clark Barrett and Cesare Tinelli. Satisfiability modulo theories. *Handbook of model checking*, pages 305–343, 2018. doi:10.1007/978-3-319-10575-8_11.

4 Marco Benedetti, Arnaud Lallouet, and Jérémie Vautard. Quantified constraint optimization. In *International Conference on Principles and Practice of Constraint Programming*, pages 463–477. Springer, 2008. doi:10.1007/978-3-540-85958-1_31.

5 Dimitris Bertsimas and Iain Dunning. Multistage robust mixed-integer optimization with adaptive partitions. *Operations Research*, 64(4):980–998, 2016. doi:10.1287/opre.2016.1515.

6 Dimitris Bertsimas and Angelos Georghiou. Binary decision rules for multistage adaptive mixed-integer optimization. *Mathematical Programming*, 167:395–433, 2018. doi:10.1007/s10107-017-1135-6.

7 Dimitris Bertsimas and Melvyn Sim. Robust discrete optimization and network flows. *Mathematical programming*, 98(1):49–71, 2003. doi:10.1007/s10107-003-0396-4.

8 Olaf Beyersdorff and Benjamin Böhm. Understanding the relative strength of QBF CDCL solvers and QBF resolution. In James R. Lee, editor, *12th Innovations in Theoretical Computer Science Conference, ITCS 2021, January 6-8, 2021, Virtual Conference*, volume 185 of *LIPIcs*, pages 12:1–12:20. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2021. doi:10.4230/LIPIcs.ITCS.2021.12.

9 Olaf Beyersdorff, Leroy Chew, Judith Clymo, and Meena Mahajan. Short proofs in QBF expansion. In *Theory and Applications of Satisfiability Testing–SAT 2019: 22nd International Conference, SAT 2019, Lisbon, Portugal, July 9–12, 2019, Proceedings 22*, pages 19–35. Springer, 2019. doi:10.1007/978-3-030-24258-9_2.

10 Olaf Beyersdorff, Leroy Chew, and Mikolás Janota. New resolution-based QBF calculi and their proof complexity. *ACM Trans. Comput. Theory*, 11(4):26:1–26:42, 2019. doi:10.1145/3352155.

11 Armin Biere, Tobias Faller, Katalin Fazekas, Mathias Fleury, Nils Froleyks, and Florian Pollitt. CaDiCaL 2.0. In Arie Gurfinkel and Vijay Ganesh, editors, *Computer Aided Verification - 36th International Conference, CAV 2024, Montreal, QC, Canada, July 24-27, 2024, Proceedings, Part I*, volume 14681 of *Lecture Notes in Computer Science*, pages 133–152. Springer, 2024. doi:10.1007/978-3-031-65627-9_7.

12 Nikolaj Bjørner, Mikolás Janota, and William Klieber. On conflicts and strategies in QBF. In *LPAR (short papers)*, pages 28–41, 2015. doi:10.29007/4sk1.

13 Nikolaj S Bjørner and Mikolás Janota. Playing with quantified satisfaction. *LPAR (short papers)*, 35:15–27, 2015. doi:10.29007/vv21.

14 Benjamin Böhm and Olaf Beyersdorff. QCDCL vs QBF resolution: Further insights. In Meena Mahajan and Friedrich Slivovsky, editors, *26th International Conference on Theory and Applications of Satisfiability Testing, SAT 2023, July 4-8, 2023, Alghero, Italy*, volume 271 of *LIPIcs*, pages 4:1–4:17. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2023. doi:10.4230/LIPIcs.SAT.2023.4.

15 Hubie Chen. Beyond Q-resolution and prenex form: A proof system for quantified constraint satisfaction. *Logical Methods in Computer Science*, 10, 2014. doi:10.2168/LMCS-10(4:14)2014.

16 Hubie Chen and Martin Pál. Optimization, games, and quantified constraint satisfaction. In *International Symposium on Mathematical Foundations of Computer Science*, pages 239–250. Springer, 2004. doi:10.1007/978-3-540-28629-5_16.

17 Leroy Chew, Alexis de Colnet, Friedrich Slivovsky, and Stefan Szeider. Hardness of random reordered encodings of parity for resolution and CDCL. *Proceedings of the AAAI Conference on Artificial Intelligence*, 38(8):7978–7986, March 2024. doi:10.1609/aaai.v38i8.28635.

18 Leroy Chew and Marijn J. H. Heule. Sorting parity encodings by reusing variables. In Luca Pulina and Martina Seidl, editors, *Theory and Applications of Satisfiability Testing - SAT 2020 - 23rd International Conference, Alghero, Italy, July 3-10, 2020, Proceedings*, volume 12178 of *Lecture Notes in Computer Science*, pages 1–10. Springer, Springer, 2020. doi:10.1007/978-3-030-51825-7_1.

19 Dmitry Chistikov and Christoph Haase. On the complexity of quantified integer programming. In Ioannis Chatzigiannakis, Piotr Indyk, Fabian Kuhn, and Anca Muscholl, editors, *44th International Colloquium on Automata, Languages, and Programming, ICALP 2017, July 10-14, 2017, Warsaw, Poland*, volume 80 of *LIPIcs*, pages 94:1–94:13. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2017. doi:10.4230/LIPIcs.ICALP.2017.94.

20 Abhimanyu Choudhury and Meena Mahajan. Dependency schemes in CDCL-based QBF solving: A proof-theoretic study. *J. Autom. Reason.*, 68(3):16, 2024. doi:10.1007/s10817-024-09707-4.

21 Leonardo Mendonça de Moura and Nikolaj S. Bjørner. Z3: an efficient SMT solver. In C. R. Ramakrishnan and Jakob Rehof, editors, *Tools and Algorithms for the Construction and Analysis of Systems, 14th International Conference, TACAS 2008, Held as Part of the Joint European Conferences on Theory and Practice of Software, ETAPS 2008, Budapest, Hungary, March 29-April 6, 2008. Proceedings*, volume 4963 of *Lecture Notes in Computer Science*, pages 337–340. Springer, Springer, 2008. doi:10.1007/978-3-540-78800-3_24.

22 Santanu S Dey and Marco Molinaro. Theoretical challenges towards cutting-plane selection. *Mathematical Programming*, 170:237–266, 2018. doi:10.1007/s10107-018-1302-4.

23 Thorsten Ederer, Michael Hartisch, Ulf Lorenz, Thomas Opfer, and Jan Wolf. Yasol: an open source solver for quantified mixed integer programs. In *Advances in Computer Games: 15th International Conferences, ACG 2017, Leiden, The Netherlands, July 3–5, 2017, Revised Selected Papers 15*, pages 224–233. Springer, 2017. doi:10.1007/978-3-319-71649-7_19.

24 Pavlos Eirinakis, Salvatore Ruggieri, K Subramani, and Piotr Wojciechowski. On quantified linear implications. *Annals of Mathematics and Artificial Intelligence*, 71(4):301–325, 2014. doi:10.1007/s10472-013-9332-3.

25 Alex Ferguson and Barry O'Sullivan. Relaxations and explanations for quantified constraint satisfaction problems. In *International Conference on Principles and Practice of Constraint Programming*, pages 690–694. Springer, 2006. doi:10.1007/11889205_52.

26 Ian P Gent, Peter Nightingale, Andrew Rowley, and Kostas Stergiou. Solving quantified constraint satisfaction problems. *Artificial Intelligence*, 172(6-7):738–771, 2008. doi:10.1016/j.artint.2007.11.003.

27 Richard Gerber, William Pugh, and Manas Saksena. Parametric dispatching of hard real-time tasks. *IEEE transactions on computers*, 44(3):471–479, 1995. doi:10.1109/12.372041.

28 Fred Glover. Surrogate constraints. *Operations Research*, 16(4):741–749, 1968. doi:10.1287/opre.16.4.741.

29 Stephan Gocht and Jakob Nordström. Certifying parity reasoning efficiently using pseudo-Boolean proofs. *Proceedings of the AAAI Conference on Artificial Intelligence*, 35(5):3768–3777, 2021. doi:10.1609/aaai.v35i5.16494.

30 Marc Goerigk and Michael Hartisch. Multistage robust discrete optimization via quantified integer programming. *Computers & Operations Research*, 135:105434, 2021. doi:10.1016/j.cor.2021.105434.

31 Marc Goerigk and Michael Hartisch. An introduction to robust combinatorial optimization. *International Series in Operations Research and Management Science*, 2024. doi:10.1007/978-3-031-61261-9.

32 Marc Goerigk, Jannis Kurtz, and Michael Poss. Min–max–min robustness for combinatorial problems with discrete budgeted uncertainty. *Discrete Applied Mathematics*, 285:707–725, 2020. doi:10.1016/j.dam.2020.07.011.

33 Gurobi Optimization, LLC. Gurobi Optimizer Reference Manual, 2024. URL: https://www.gurobi.com.

34 Michael Hartisch. *Quantified integer programming with polyhedral and decision-dependent uncertainty*. PhD thesis, University of Siegen, Germany, 2020. doi:10.25819/ubsi/4841.

35 Michael Hartisch. Adaptive relaxations for multistage robust optimization. In *Pacific Rim International Conference on Artificial Intelligence*, pages 485–499. Springer, 2021. doi:10.1007/978-3-030-89188-6_36.

36 Michael Hartisch and Leroy Chew. MichaelHartisch/EQuIPS. Software, DFG-534441421, FWF Project ESP197, swhId: swh:1:dir:29f0d2d80f69841007fd3a315a47c2865669f3a6 (visited on 2025-07-23). URL: https://github.com/MichaelHartisch/EQuIPS, doi:10.4230/artifacts.24095.

37 Michael Hartisch, Thorsten Ederer, Ulf Lorenz, and Jan Wolf. Quantified integer programs with polyhedral uncertainty set. In *Computers and Games: 9th International Conference, CG 2016, Leiden, The Netherlands, June 29–July 1, 2016, Revised Selected Papers 9*, pages 156–166. Springer, 2016. doi:10.1007/978-3-319-50935-8_15.

38 Michael Hartisch and Ulf Lorenz. Mastering uncertainty: towards robust multistage optimization with decision dependent uncertainty. In *PRICAI 2019: Trends in Artificial Intelligence: 16th Pacific Rim International Conference on Artificial Intelligence, Cuvu, Yanuca Island, Fiji, August 26–30, 2019, Proceedings, Part I 16*, pages 446–458. Springer, 2019. doi:10.1007/978-3-030-29908-8_36.

39 Michael Hartisch and Ulf Lorenz. A novel application for game tree search-exploiting pruning mechanisms for quantified integer programs. In *Advances in Computer Games: 16th International Conference, ACG 2019, Macao, China, August 11–13, 2019, Revised Selected Papers 16*, pages 66–78. Springer, 2020. doi:10.1007/978-3-030-65883-0_6.

40 Michael Hartisch and Ulf Lorenz. A general model-and-run solver for multistage robust discrete linear optimization. *arXiv preprint arXiv:2210.11132*, 2022. doi:10.48550/arXiv.2210.11132.

41 Alexey Ignatiev, Mikoláš Janota, and Joao Marques-Silva. Quantified maximum satisfiability. *Constraints*, 21:277–302, 2016. doi:10.1007/s10601-015-9195-9.

42 Mikoláš Janota, William Klieber, João Marques-Silva, and Edmund M. Clarke. Solving QBF with counterexample guided refinement. In Alessandro Cimatti and Roberto Sebastiani, editors, *Proc. 15th International Conference on Theory and Applications of Satisfiability Testing*, volume 7317, pages 114–128. Springer, 2012. doi:10.1007/978-3-642-31612-8_10.

43 Mikoláš Janota and Joao Marques-Silva. Expansion-based QBF solving versus Q-resolution. *Theor. Comput. Sci.*, 577:25–42, 2015. doi:10.1016/j.tcs.2015.01.048.

44 Mikoláš Janota. Towards generalization in QBF solving via machine learning. *Proceedings of the AAAI Conference on Artificial Intelligence*, 32(1):6607–6614, April 2018. doi:10.1609/aaai.v32i1.12208.

45 Florian Lonsing and Uwe Egly. Depqbf 6.0: A search-based QBF solver beyond traditional QCDCL. In *Automated Deduction–CADE 26: 26th International Conference on Automated Deduction, Gothenburg, Sweden, August 6–11, 2017, Proceedings*, pages 371–384. Springer, 2017. doi:10.1007/978-3-319-63046-5_23.

46 Florian Lonsing and Uwe Egly. Evaluating QBF solvers: Quantifier alternations matter. In John Hooker, editor, *Principles and Practice of Constraint Programming*, pages 276–294, Cham, 2018. Springer International Publishing. doi:10.1007/978-3-319-98334-9_19.

47 Ulf Lorenz and Jan Wolf. Solving multistage quantified linear optimization problems with the alpha–beta nested benders decomposition. *EURO Journal on Computational Optimization*, 3:349–370, 2015. doi:10.1007/s13675-015-0038-7.

48 Francesca Maggioni, Fabrizio Dabbene, and Georg Ch. Pflug. Sampling methods for multistage robust optimization problems. *Ann. Oper. Res.*, 347(3):1385–1423, 2025. doi:10.1007/s10479-025-06545-4.

49 Nikos Mamoulis and Kostas Stergiou. Algorithms for quantified constraint satisfaction problems. In *International Conference on Principles and Practice of Constraint Programming*, pages 752–756. Springer, 2004. doi:10.1007/978-3-540-30201-8_60.

50 Hugues Marchand and Laurence A Wolsey. Aggregation and mixed integer rounding to solve MIPs. *Operations research*, 49(3):363–371, 2001. doi:10.1287/opre.49.3.363.11211.

51 Toshihiro Matsui, Hiroshi Matsuo, Marius Calin Silaghi, Katsutoshi Hirayama, Makoto Yokoo, and Satomi Baba. A quantified distributed constraint optimization problem. In *Proc. 9th Int'l. Conf. on Autonomous Agents and Multiagent Systems (AAMAS2010)*, volume 1, pages 1023–1030. Nagoya Institute of Technology, 2010. URL: https://dl.acm.org/citation.cfm?id=1838344.

52 George L. Nemhauser and Laurence A. Wolsey. *Integer and Combinatorial Optimization*. Wiley interscience series in discrete mathematics and optimization. Wiley, 1988. doi:10.1002/9781118627372.

53 Danny Nguyen and Igor Pak. The computational complexity of integer programming with alternations. *Mathematics of Operations Research*, 45(1):191–204, 2020. doi:10.1287/moor.2018.0988.

54 Peter Nightingale. Non-binary quantified CSP: algorithms and modelling. *Constraints*, 14:539–581, 2009. doi:10.1007/s10601-009-9068-1.

55 Jérémy Omer, Michael Poss, and Maxime Rougier. Combinatorial robust optimization with decision-dependent information discovery and polyhedral uncertainty. *Open Journal of Mathematical Optimization*, 5:1–25, 2024. doi:10.5802/ojmo.33.

56 Michael Poss. Robust combinatorial optimization with variable cost uncertainty. *European Journal of Operational Research*, 237(3):836–845, 2014. doi:10.1016/j.ejor.2014.02.060.

57 Krzysztof Postek and Dick den Hertog. Multistage adjustable robust mixed-integer optimization via iterative splitting of the uncertainty set. *INFORMS Journal on Computing*, 28(3):553–574, 2016. doi:10.1287/ijoc.2016.0696.

58 Andrew Reynolds, Tim King, and Viktor Kuncak. Solving quantified linear arithmetic by counterexample-guided instantiation. *Formal Methods in System Design*, 51:500–532, 2017. doi:10.1007/s10703-017-0290-y.

59 Alexander Schrijver. *Theory of linear and integer programming*. Wiley-Interscience series in discrete mathematics and optimization. Wiley, 1999.

60 Mate Soos, Karsten Nohl, and Claude Castelluccia. Extending sat solvers to cryptographic problems. In *International Conference on Theory and Applications of Satisfiability Testing*, pages 244–257. Springer, 2009. doi:10.1007/978-3-642-02777-2_24.

61 Kostas Stergiou. Repair-based methods for quantified CSPs. In *International Conference on Principles and Practice of Constraint Programming*, pages 652–666. Springer, 2005. doi:10.1007/11564751_48.

62 K Subramani. Analyzing selected quantified integer programs. In *International Joint Conference on Automated Reasoning*, pages 342–356. Springer, 2004. doi:10.1007/978-3-540-25984-8_26.

63 Kerian Thuillier, Anne Siegel, and Loïc Paulevé. CEGAR-based approach for solving combinatorial optimization modulo quantified linear arithmetics problems. *Proceedings of the AAAI Conference on Artificial Intelligence*, 38(8):8146–8153, March 2024. doi:10.1609/aaai.v38i8.28654.

64 Phebe Vayanos, Daniel Kuhn, and Berç Rustem. A constraint sampling approach for multistage robust optimization. *Automatica*, 48(3):459–471, 2012. doi:10.1016/j.automatica.2011.12.002.

65 Guillaume Verger and Christian Bessiere. Guiding search in QCSP+ with back-propagation. In *International Conference on Principles and Practice of Constraint Programming*, pages 175–189. Springer, 2008. doi:10.1007/978-3-540-85958-1_12.

66 Lintao Zhang and Sharad Malik. Conflict driven learning in a quantified Boolean satisfiability solver. In *ICCAD*, pages 442–449, 2002. doi:10.1145/774572.774637.

A Appendix

A.1 Underlying Proof System and Correctness

It has been well established that RAReQS works with the QBF proof system ∀Exp+Res [43]. It comes as no surprise that the underlying proof system of EQuIPS acts much the same, instead of describing the SAT oracle as a resolution system, we describe the IP call as a cutting planes proof. Figure 3 describes the ∀Exp+Cutting Planes proof systems which is the underlying proof system for refutations where the first quantifier is existential.

$$\frac{\sum_{k\in[n]:\ Q_k=\exists} a_k x_k + \sum_{k\in[n]:\ Q_k=\forall} a_k x_k \leq b \text{ in matrix}}{\sum_{k\in[n]:\ Q_k=\exists} a_k x_k^{[\boldsymbol{\tau}]} + \sum_{k\in[n]:\ Q_k=\forall} a_k \boldsymbol{\tau}(x_k) \leq b} \text{ (Axiom)}$$

- $\boldsymbol{\tau}$ is a complete assignment to universal variables that satisfies the universal constraint system
- For $x_k^{[\boldsymbol{\tau}]}$, $[\boldsymbol{\tau}]$ takes only the part of $\boldsymbol{\tau}$ that is left of x_k

- **Addition**: From $\sum_{k\in[n]} a_k x_k \leq b$ and $\sum_{k\in[n]} \alpha_k x_k \leq \beta$, derive $\sum_{k\in[n]} (a_k + \alpha_k) x_k \leq b + \beta$.
- **Multiplication**: From $\sum_{k\in[n]} a_k x_k \leq b$, derive $\sum_{k\in[n]} d a_k x_k \leq db$, where $d \in \mathbb{Z}^+$.
- **Division**: From $\sum_{k\in[n]} a_k x_k \leq b$, derive $\sum_{k\in[n]} \frac{a_k}{d} x_k \leq \left\lceil \frac{b}{d} \right\rceil$, where $d \in \mathbb{Z}^+$ divides each a_k.

Figure 3 The proof system ∀Exp+Cutting Planes.

▶ **Theorem 6.** *∀Exp+Cutting Planes is a sound and complete proof system for QIP.*

Proof. Given a QIP, we take a full Shannon expansion on all universal quantifiers. In case a universal constraint system is present, only universal variable assignments satisfying this system are considered. Every potential axiom line is found as a conjunct in this expansion. The full expansion is satisfiability equivalent with the original QIP and contains no universal quantifiers. Therefore given the completeness and soundness of the Cutting Planes proof system, we prove ∀Exp+Cutting Planes is a sound and complete proof system for QIP. ◀

We will now give an overview on the soundness of EQuIPS. We start with an observation about Algorithm 1 and the description of refinement in Section 3.3.

▶ **Observation 7.** *At any point in a run of* EQuIPS *with outer block $\mathcal{Q}\boldsymbol{X} \in D_{\boldsymbol{X}}$, for every subgame of the abstraction there is an assignment $\boldsymbol{\mu}$ to the first inner block variables $\boldsymbol{Y}$ such that the subgame is of one of two forms:*

1. $\Psi_i = A^{\exists}_{(-\boldsymbol{Y})}\boldsymbol{x}_{(-\boldsymbol{Y})} \leq \boldsymbol{b}^{\exists}_{\boldsymbol{Y}=\boldsymbol{\mu}}$ *when* $\Phi_{l_i} = \bar{\mathcal{Q}}\boldsymbol{Y} \in \mathcal{D}_{\boldsymbol{Y}} : A^{\exists}\boldsymbol{x} \leq \boldsymbol{b}^{\exists}$, *or*
2. $\Psi_i = \Lambda^{(\boldsymbol{Y}=\boldsymbol{\mu})}_{\Phi_{l_i}}[\boldsymbol{\mu}]$ *when* $\Phi_{l_i} = \bar{\mathcal{Q}}\boldsymbol{Y} \in \mathcal{D}_{\boldsymbol{Y}}\mathcal{Q}\boldsymbol{Z} \in \mathcal{D}_{\boldsymbol{Z}} : \Lambda$, *where* Λ *itself is a QIP.*

In the first case we have a QIP with only an outer block, in the second case we have a QIP with at least one inner block, Λ *is a QIP representing the rest of blocks and the constraints.*

▶ **Lemma 8.** *If* EQuIPS *returns* $\bot$ *on multi-game* $\forall\boldsymbol{X} \in \mathcal{D}_{\boldsymbol{X}} : \{\Phi_1 \dots \Phi_n\}$ *then this multi-game is won by the existential player.*

Proof. We prove this by induction on the quantifier depth. The base case is if $\Phi_1 \dots \Phi_n$ are quantifier free, and we solve integer linear program (2) as `wins1` is invoked. Then, $\bot$ is returned if and only if the universal player is not able to violate the constraint systems of all subgames as argued in Section 3.2. Therefore $\forall\boldsymbol{X} \in \mathcal{D}_{\boldsymbol{X}} : \{\Phi_1 \dots \Phi_n\}$ is won by the existential player.

Now suppose some Φ_j contains a quantifier. Then, we build an abstraction α of the multi-game and $\bot$ is only returned, if the call to `EQuIPS` in Line 4 on the abstraction returns $\bot$. Note that the first call to Line 4 on the empty abstraction always returns a move $\boldsymbol{\tau}$, as $\mathcal{D}_{\boldsymbol{X}}$ is always non-empty. Thus, a refinement of the abstraction must have led to a returned $\bot$. Hence, we have to show that a call to `EQuIPS` for a refined abstraction $\forall\boldsymbol{X} \in \mathcal{D}_{\boldsymbol{X}}\forall\boldsymbol{Z}^{(\boldsymbol{Y}=\boldsymbol{\mu}_1)}_{\Phi_{l_1}} \in \mathcal{D}_{\boldsymbol{Z}} \dots \boldsymbol{Z}^{(\boldsymbol{Y}=\boldsymbol{\mu}_k)}_{\Phi_{l_k}} \in \mathcal{D}_{\boldsymbol{Z}} : \{\Psi_1 \dots \Psi_k\}$ returns $\bot$, only if the original formula is won by the existential player. We know that the abstraction is existentially feasible by induction hypothesis, in other words there is no assignment to the outer block that makes all subgames infeasible. We now argue that for any $\boldsymbol{\tau} \in \mathcal{D}_{\boldsymbol{X}}$, there must be one subgame that is won by the existential player. Otherwise for each Φ_j, $j \in [n]$ there exists an assignment $\boldsymbol{\tau}_i \in \mathcal{D}_{\boldsymbol{Z}}$ for which $\Psi_i[\boldsymbol{\tau}][\boldsymbol{\tau}_i]$ is lost for the existential player. Then we can construct $\boldsymbol{\tau}' = (\boldsymbol{\tau}, \boldsymbol{\tau}_{l_1}, \dots, \boldsymbol{\tau}_{l_k})$. This is well defined because $\boldsymbol{X}$ (corresponding to $\boldsymbol{\tau}$) are the only shared outer variables between the subgames. $\boldsymbol{\tau}'$ is also a winning move of the abstraction and in particular, each subgame Ψ_i is won by the universal player, against our assumption. Therefore for each $\boldsymbol{\tau} \in \mathcal{D}_{\boldsymbol{X}}$ there is a subgame Ψ_i that is won by the existential player under the assignment to the remaining outer variables. From Observation 7, subgames Ψ_i can have one of the following structures:

1. $\Psi_i = A^{\exists}_{(-\boldsymbol{Y})}\boldsymbol{x}_{(-\boldsymbol{Y})} \leq \boldsymbol{b}^{\exists}_{\boldsymbol{Y}=\boldsymbol{\mu}}$ when $\Phi_{l_i} = \exists\boldsymbol{Y} \in \mathcal{D}_{\boldsymbol{Y}} : A^{\exists}\boldsymbol{x} \leq \boldsymbol{b}^{\exists}$, or
2. $\Psi_i = \Lambda^{(\boldsymbol{Y}=\boldsymbol{\mu})}_{\Phi_{l_i}}[\boldsymbol{\mu}]$ when $\Phi_{l_i} = \exists\boldsymbol{Y} \in \mathcal{D}_{\boldsymbol{Y}}\forall\boldsymbol{Z} \in \mathcal{D}_{\boldsymbol{Z}} : \Lambda$, where Λ itself is a QIP.

We argue that one of these being feasible means that its associated $\boldsymbol{\mu}$ is countermove to $\boldsymbol{\tau}$ in the original game. If $\boldsymbol{\tau}$ allows Φ_{l_i}, then Φ_{l_i} is in the original games. In the first case, $\Psi_i[\boldsymbol{\tau}] = A^{\exists}_{(-\boldsymbol{X}-\boldsymbol{Y})}\boldsymbol{x}_{(-\boldsymbol{X}-\boldsymbol{Y})} \leq \boldsymbol{b}^{\exists}_{\boldsymbol{X}=\boldsymbol{\tau}\boldsymbol{Y}=\boldsymbol{\mu}}$ is feasible so $\boldsymbol{\mu}$ is a countermove in the original game that satisfies $\Phi_{l_i}[\boldsymbol{\tau}]$. In the second case $\Lambda^{(\boldsymbol{X}=\boldsymbol{\tau},\boldsymbol{Y}=\boldsymbol{\mu})}_{\Phi_{l_i}}[\boldsymbol{\tau}, \boldsymbol{\mu}]$ is feasible so $\boldsymbol{\mu}$ must be a countermove in the original game that satisfies $\Phi_{l_i}[\boldsymbol{\tau}]$. Therefore every $\boldsymbol{\tau}$ has an existential countermove. ◀

In the existential case, we can understand soundness through the theoretical existence of proofs. We unfortunately cannot extract proofs at this stage, as we would need to extract cutting planes proofs from `Gurobi` which is not supported.

▶ **Lemma 9.** *If* EQuIPS *returns* $\bot$ *on multi-game* $\exists\boldsymbol{X} \in \mathcal{D}_{\boldsymbol{X}} : \{\Phi_1 \dots \Phi_n\}$ *then there is a* $\forall$*Exp+Cutting Planes refutation of* $\exists\boldsymbol{X} \in \mathcal{D}_{\boldsymbol{X}}.\Phi_1 \wedge \dots \wedge \Phi_n$.

Proof. We can prove this via induction on the quantifier depth of $\exists \boldsymbol{X} \in \mathcal{D}_{\boldsymbol{X}} : \{\Phi_1 \dots \Phi_n\}$.

For the base case, if all $\Phi_1 \dots \Phi_n$ are quantifier free, `wins1` is invoked. Given the outer quantifier is $\exists$, we solve the integer program (1), which is the conjunction of all constraint systems $\Phi_1 \dots \Phi_n$. Since cutting planes is a complete refutational system for Integer Programming [59] there is a cutting planes proof of this refutation.

Now consider the inductive step. Suppose some Φ_i has a quantifier. Then `EQuIPS` goes into the CEGAR loop and only returns $\bot$ if the abstraction α returns $\bot$. We have to show if the abstraction $\exists \boldsymbol{X} \in \mathcal{D}_{\boldsymbol{X}} \exists \boldsymbol{Z}^{(\boldsymbol{Y}=\boldsymbol{\mu}_1)}_{\Phi_{l_1}} \in \mathcal{D}_{\boldsymbol{Z}} \dots \exists \boldsymbol{Z}^{(Y=\boldsymbol{\mu}_k)}_{\Phi_{l_k}} \in \mathcal{D}_{\boldsymbol{Z}} : \{\Psi_1 \dots \Psi_k\}$ returns $\bot$ then we can construct a $\forall$Exp+Cutting Planes refutation of the original formula.

By assuming the induction hypothesis we have an $\forall$Exp+Cutting Planes refutation π of $\exists \boldsymbol{X} \in \mathcal{D}_{\boldsymbol{X}} \exists \boldsymbol{Z}^{(Y=\boldsymbol{\mu}_1)}_{\Phi_{l_1}} \in \mathcal{D}_{\boldsymbol{Z}} \dots \exists \boldsymbol{Z}^{(Y=\boldsymbol{\mu}_k)}_{\Phi_{l_k}} \in \mathcal{D}_{\boldsymbol{Z}} : \{\Psi_1 \dots \Psi_k\}$. Note that the quantified variables in each $\Psi_1 \dots \Psi_k$ are different. The only variables they share are in the outer block variables $\boldsymbol{X}$.

Consider an individual axiom step C of π, which involves taking a constraint from Ψ_i, and a complete assignment $\boldsymbol{\beta}$ to the universal variables of Ψ_i that satisfies the universal constraint system. Note that Ψ_i can only be a subgame in the abstraction for one of two reasons:

1. $\Psi_i = A^{\exists}_{(-\boldsymbol{Y})} \boldsymbol{x}_{(-\boldsymbol{Y})} \leq \boldsymbol{b}^{\exists}_{\boldsymbol{Y}=\boldsymbol{\mu}}$ when $\Phi_{l_i} = \forall \boldsymbol{Y} \in \mathcal{D}_{\boldsymbol{Y}} : A^{\exists} \boldsymbol{x} \leq \boldsymbol{b}^{\exists}$, when $\boldsymbol{\mu} \in \mathcal{D}_{\boldsymbol{Y}}$.
2. $\Psi_i = \Lambda^{(\boldsymbol{Y}=\boldsymbol{\mu})}_{\Phi_{l_i}}[\boldsymbol{\mu}]$ when $\Phi_{l_i} = \forall \boldsymbol{Y} \in \mathcal{D}_{\boldsymbol{Y}} \exists \boldsymbol{Z} \in \mathcal{D}_{\boldsymbol{Z}} : \Lambda$, when $\boldsymbol{\mu} \in \mathcal{D}_{\boldsymbol{Y}}$.

In each case, for every Ψ_i there is a $\boldsymbol{Y} = \boldsymbol{\mu}$ statement that corresponds to it. We take π and rename all the existential variables appearing in the inner blocks to create π', we rename w appearing in Ψ_i to be $w^{(\boldsymbol{Y}=\boldsymbol{\mu})}_{\Psi_i}$.

In the first case, we can take the single row j from $A^{\exists}_{(-\boldsymbol{Y})} \boldsymbol{x}_{(-\boldsymbol{Y})} \leq \boldsymbol{b}^{\exists}_{\boldsymbol{Y}=\boldsymbol{\mu}}$ that was combined with $\boldsymbol{\beta}$ to get C. Note that $\boldsymbol{\beta}$ must be empty, because there are no universals left. We take the same row j of $A^{\exists} \boldsymbol{x} \leq \boldsymbol{b}^{\exists}$ and combine it with $\boldsymbol{\mu}$ (which satisfies the uncertainty set) in an axiom step from our original formula. Then $A^{\exists}_j \boldsymbol{x} \leq \boldsymbol{b}^{\exists}_j$ instantiates to $(A^{\exists}_{(-\boldsymbol{Y})})_j \boldsymbol{x}_{(-\boldsymbol{Y})} \leq (\boldsymbol{b}^{\exists}_{(\boldsymbol{Y}=\boldsymbol{\mu})})_j$, exactly the same as C and the transformation in π' does not change this, because there are no inner existential variables.

In case 2, the inner part of $\Lambda^{(\boldsymbol{Y}=\boldsymbol{\mu})}_{\Phi_{l_i}}$ must have a row j that appears as constraint

$$\sum_{k \in [n]:\ Q_k = \exists} a^{\exists}_{j,k} x_k + \sum_{k \in [n]:\ Q_k = \forall} a^{\exists}_{j,k} x_k \leq (\boldsymbol{b}^{\exists}_{(\boldsymbol{\mu})})_j$$

that combines with $\boldsymbol{\beta}$ to get axiom

$$\sum_{k \in [n]:\ Q_k = \exists} a^{\exists}_{j,k} x^{[\boldsymbol{\beta}]}_k + \sum_{k \in [n]:\ Q_k = \forall} a^{\exists}_{j,k} \boldsymbol{\beta}(x_k) \leq (\boldsymbol{b}^{\exists}_{(\boldsymbol{\mu})})_j.$$

In π' we can write it as

$$\sum_{\substack{k \in [n] \\ x_k \in \boldsymbol{X}}} a^{\exists}_{j,k} x_k + \sum_{\substack{k \in [n] \\ x_k \in \boldsymbol{Z}^{(\boldsymbol{Y}=\boldsymbol{\mu})}_{\Phi_{l_i}}}} a^{\exists}_{j,k} x_k + \sum_{\substack{k \in [n]:\ Q_k = \exists \\ x_k \text{ inner}}} a^{\exists}_{j,k} x_{k\,\Phi_{l_i}}^{(\boldsymbol{Y}=\boldsymbol{\mu})[\boldsymbol{\beta}]} + \sum_{k \in [n]:\ Q_k = \forall} a^{\exists}_{j,k} \boldsymbol{\beta}(x_k) \leq (\boldsymbol{b}^{\exists}_{(\boldsymbol{\mu})})_j.$$

But notice that all $x_k \in \boldsymbol{Z}^{(\boldsymbol{Y}=\boldsymbol{\mu})}_{\Phi_{l_i}}$ match the renaming annotation in π'. Simplifying again gets us

$$\sum_{\substack{k \in [n] \\ x_k \in \boldsymbol{X}}} a^{\exists}_{j,k} x_k + \sum_{\substack{k \in [n]:\ Q_k = \exists \\ x \notin \boldsymbol{X}}} a^{\exists}_{j,k} x_{k\,\Phi_{l_i}}^{(\boldsymbol{Y}=\boldsymbol{\mu})[\boldsymbol{\beta}]} \leq (\boldsymbol{b}^{\exists}_{(\boldsymbol{\mu} \sqcup \boldsymbol{\beta})})_j.$$

We now take the same row in the inner part of Λ and combine it with $\boldsymbol{\mu} \sqcup \boldsymbol{\beta}$ to get the axiom

$$\sum_{k \in [n]:\ Q_k = \exists} a_k x_k^{[\boldsymbol{\mu} \sqcup \boldsymbol{\beta}]} + \sum_{k \in [n]:\ Q_k = \forall} a_k \boldsymbol{\beta}(x_k) \leq (\boldsymbol{b}^{\exists})_j.$$

Note that $\boldsymbol{\mu} \sqcup \boldsymbol{\beta}$ is a disjoint union because $\boldsymbol{Y}$ is already assigned in Ψ_i, and each universal block satisfies its domain given by bounds and universal constraint system. Subtracting both sides ends us with the axiom exactly as it was in π', because $\boldsymbol{X}$ variables are not changed under $[\boldsymbol{\mu} \sqcup \boldsymbol{\beta}]$ and the $\boldsymbol{\mu}$ annotation is already present in the π' proof. π' is therefore a refutation in the original formula. ◀

A.2 Multilevel Critical Node Problem

We consider the multilevel critical node problem [1]. Given a directed graph $G = (V, E)$. Two agents act on G: The *attacker* selects a set of nodes she wants to infect and the *defender* tries to maximize the number of saved nodes. The defender can *vaccinate* nodes before any infection occurs and *protect* a set of nodes after the attack. An infection triggers a cascade of further infections that propagates via the graph neighborhood, only stopped by vaccinated or protected nodes. For each action (vaccination, infection, protection) a budget (Ω, Φ, Λ, respectively) exists limiting the number of chosen nodes. For any node $v \in V$ binary variables z_v, y_v, and x_v are used to indicated its vaccination, infection, and protection, respectively. Variables $\alpha_v \in \{0, 1\}$ indicate whether node $v \in V$ is saved eventually. Only the variables $\boldsymbol{y}$ are universally quantified. Their domain is restricted by a budget constraint, i.e., $\mathcal{U}_{\boldsymbol{y}} = \{\boldsymbol{y} \in \{0,1\}^V \mid \sum_{v \in V} y_v \leq \Phi\}$. Hence, the universal constraint system only contains the single budget constraint. A QIP with polyhedral uncertainty set can be stated as follows:

$$\max_{\boldsymbol{z} \in \{0,1\}^V} \min_{\boldsymbol{y} \in \mathcal{U}_{\boldsymbol{y}}} \max_{\substack{\boldsymbol{x} \in \{0,1\}^V \\ \boldsymbol{\alpha} \in \{0,1\}^V}} \sum_{v \in V} \alpha_v \tag{5a}$$

$$\text{s.t. } \exists \boldsymbol{z} \in \{0,1\}^V\ \forall \boldsymbol{y} \in \mathcal{U}_{\boldsymbol{y}}\ \exists \boldsymbol{x} \in \{0,1\}^V\ \boldsymbol{\alpha} \in \{0,1\}^V : \tag{5b}$$

$$\sum_{v \in V} z_v \leq \Omega \tag{5c}$$

$$\sum_{v \in V} x_v \leq \Lambda \tag{5d}$$

$$\alpha_v \leq 1 + z_v - y_v \qquad \forall v \in V \tag{5e}$$

$$\alpha_v \leq \alpha_u + x_v + z_v \qquad \forall (u, v) \in E \tag{5f}$$

Constraint (5e) ensures that infected nodes cannot be saved, unless they were vaccinated and Constraint (5f) describes the propagation of the infection to neighboring nodes that are neither vaccinated nor protected. This model corresponds exactly to the trilevel program presented in [1] with the key difference, that we are able to directly plug this model into our solver to obtain the optimal solution, without having to dualize, reformulate or develop domain specific algorithms. The same is true for the QIP solver `Yasol`.

A.3 Further Experiments

A.3.1 Mixing Methods

As outlined in Section 3.4.2, there is potential to link search-based and expansion-based approaches. To demonstrate this, we conducted an experiment using multilevel critical node instances. 1. Run `Yasol` and record the time t_{opt} at which the optimal solution $z^\star$ is first

found (its existence is verified, not its optimality). 2. Run `EQuIPS`, separately, with the objective function constraint bound $z^\star + 1$ (note the maximization objective function) and record the verification time t_{ver}. This hypothetical solver – running `Yasol` in parallel with `EQuIPS` for verification – with runtime $t_{\text{opt}} + t_{\text{ver}}$ solves 27 more instances than `Yasol` alone and has strictly lower runtime on 196 instances.

We also tested whether adding learned constraints from `Yasol`'s search benefits the expansion-based solver. We modified `Yasol` to extract every detected conflict as a constraint until the optimal solution was found (extracting beyond this point might prune the optimal solution). For a single instance, we obtained 73 learned constraints and created three variations of the verification instance: one without added constraints, one with all 73 constraints, and one with three hand-picked constraints. Table 2 reports the runtimes and the number of IP calls in the `wins1` function. For this instance, incorporating learned

Table 2 Comparison of three verification instances.

instance type	original verification instance	org. + 73 constraints	org. + 3 constraints
runtime	36.3s	44.4s	22.2s
calls to IP solver	1034	648	550

constraints reduced the iterations (and thus IP calls) for the expansion-based solver. However, too many constraints may increase IP solver runtime; therefore, selectively transferring the "most beneficial" constraints to `EQuIPS` can significantly enhance the verification process. Although we expected the 73-constraint instance to have fewer IP calls than the one with three constraints, this discrepancy likely results from testing only a single instance. Averaged over a larger set, more constraints should decrease the number of IP calls.

A.3.2 Performance of `EQuIPS` vs. `Yasol` on other optimization test sets

While the computational experiments in Sections 5.1 and 5.2 show promising results – suggesting that our expansion-based approach can effectively compete with the search-based solver `Yasol`– this cannot be stated as a general conclusion, particularly for optimization instances. We conducted tests on 1800 multistage robust assignment instances from [30] and 270 multistage robust scheduling instances from [34]. The former involve combinatorial matching problems under cost uncertainty, while the latter model aircraft scheduling with uncertain arrival times. Both datasets include instances with up to seven decision stages. In both cases, instances can be encoded either with or without polyhedral uncertainty, labeled with QIP_{PU} and QIP, respectively.

Table 3 Number of solved instances and median runtimes on different test sets.

	solved instances					median run times (seconds)				
	Assignment		Scheduling		MCN	Assignment		Scheduling		MCN
	QIP	QIP_{PU}	QIP	QIP_{PU}		QIP	QIP_{PU}	QIP	QIP_{PU}	
`Yasol`	1800	1800	262	264	431	0.4	0.4	56.7	29.0	247.8
`EQuIPS`	1599	1760	194	247	465	10.6	2.7	287.9	51.3	83.8

Table 3 presents the number of instances solved within a 1800-second time limit as well as the median runtime, highlighting that `EQuIPS` struggles with these problem types. Several factors may explain this behavior. First, for assignment instances without a universal

constraint system, the structure requires existentially quantified variables to adapt to any changes in universally quantified variables. As a result, nearly the entire expansion must be constructed before a solution can be determined, drastically increasing computational complexity. Additionally, many of the tested instances feature multiple levels of universally quantified variables, unlike the MCN instances, which contain only a single universal level.

This aligns with expansion-based solvers in the QBF domain, which perform well with few quantifier alternations but struggle as the number of universal levels increase.

A.3.3 Adapted `wins1` Function in Case of Empty Abstraction

Several aspects of Algorithm 1 allow for a choice between standard techniques and more sophisticated implementations. One such aspect is selecting a winning move for the empty abstraction, occurring at Line 4. The call to `EQuIPS` leads to `wins1`, which, for $\mathcal{Q} = \exists$, returns any assignment satisfying the domain constraints. In our implementation, we initially returned the lower bounds of existentially quantified variables, avoiding the IP solver. While computationally inexpensive, this can result in an assignment that violates the existential constraints, making the first countermove less meaningful.

An obvious alternative is to use the IP relaxation, where we solve the existential constraint system without considering quantification. The goal is to obtain stronger moves that lead to more relevant countermoves, ultimately reducing the number of subgames to be considered. Note, that as we observed superior runtimes when using this existential IP relaxation, all reported results so far, are based on this implementation.

However, solving an IP is more computationally expensive than assigning lower bounds. We investigated this trade-off using the same test sets as in Appendix A.3.2. Results in Table 4 show that fewer instances of the assignment test set are solved within the 1800-second time limit with the existential IP relaxation. However, for other test sets, `EQuIPS` performance improves, with median runtimes decreasing overall.

Table 4 Number of solved instances and median runtimes, with `EQuIPS` using lower bounds (LB) vs. solving the existential IP relaxation (exist. IP) to find a winning move for the empty abstraction.

	solved instances					median run times (seconds)				
	Assignment		Scheduling		MCN	Assignment		Scheduling		MCN
	QIP	QIP_{PU}	QIP	QIP_{PU}		QIP	QIP_{PU}	QIP	QIP_{PU}	
LB	1600	1770	175	223	461	12.3	3.8	590.9	127.8	121.9
exist. IP	1599	1760	194	247	465	10.6	2.7	287.9	51.3	83.8