Antichain-based QBF Solving

T. Brihaye[1], V. Bruyère[2], L. Doyen[3], M. Ducobu[1], and J.-F. Raskin[4]

[1] Institut de Mathématique – Université de Mons, Belgique
[2] Institut d'Informatique – Université de Mons, Belgique
[3] LSV, ENS Cachan & CNRS, France
[4] Département d'Informatique – Université Libre de Bruxelles, Belgique

Abstract. We consider the problem of QBF solving viewed as a reachability problem in an exponential And-Or graph. Antichain-based algorithms for reachability analysis in large graphs exploit certain subsumption relations to leverage the inherent structure of the explored graph in order to reduce the effect of state explosion, with high performance in practice.
In this paper, we propose simple notions of subsumption induced by the structural properties of the And-Or graphs for QBF solving. Subsumption is used to reduce the size of the search tree, and to define compact representations of certificates (in the form of antichains) both for positive and negative instances of QBF. We show that efficient exploration of the reduced search tree essentially relies on solving variants of Max-SAT and Min-SAT. Preliminary stand-alone experiments of this algorithm show that the antichain-based approach is promising.

1 Introduction

The problem of evaluating the truth value of a quantified Boolean formula (QBF) is one of the most popular PSPACE-complete problems, like SAT (the satisfiability problem for Boolean formulas) is the typical NP-complete problem. QBF is a simple and elegant formalism in which many problems of practical interest can be encoded, in a large number of areas such as automated planning, artificial intelligence, logic reasoning, and verification [14, 6]. For instance, QBF can encode reachability problems more succinctly than SAT with a formula that is logarithmic in the diameter of the system when it can only be done linearly in the diameter with a SAT formula [18]. As another example, SAT and QBF can be integrated for bounded model-checking where the existence of a path is encoded by SAT, and termination is checked with QBF [10].

The simple form of QBF makes the problem appealing and accessible to a large community. However, despite its apparent simplicity, the design of efficient algorithmic solutions remains challenging. Recent progress has been observed in the practical approaches to this problem. In particular, generalizations of heuristics and optimizations used in SAT solving have been applied to QBF with some success [24, 27].

Many algorithms have been proposed in the literature to solve QBF and competitive events like QBFEVAL aim at assessing the advances in reasoning about QBF [27, 15]. Several leading QBF solvers are search-based. They typically use pruning techniques that extend the DPLL search strategy from SAT to QBF [8]. Common heuristics are unit propagation, conflict learning and back-jumping, which are implemented in tools like QuBE [16, 17] and DepQBF [22, 23]. Recent works have focused on certifying (rather

$$\psi = \underbrace{(x_1 \vee y_4 \vee y_7)}_{c_1} \wedge \underbrace{(x_2 \vee \bar{y}_4 \vee x_6)}_{c_2} \wedge \underbrace{(x_1 \vee x_3 \vee y_5 \vee \bar{y}_7)}_{c_3} \wedge$$

$$\underbrace{(\bar{x}_3 \vee \bar{y}_5 \vee \bar{x}_6 \vee y_7)}_{c_4} \wedge \underbrace{(\bar{x}_1 \vee x_2 \vee y_4)}_{c_5} \wedge \underbrace{(\bar{x}_2 \vee \bar{y}_7)}_{c_6} \wedge \underbrace{(x_1 \vee x_2 \vee x_3 \vee y_7)}_{c_7}$$

Fig. 1. The CNF formula ψ for the QBF formula $f = \forall x_1 x_2 x_3 \cdot \exists y_4 y_5 \cdot \forall x_6 \cdot \exists y_7 \cdot \psi$.

than just evaluating) QBF formulas, as certificates can help in extracting error traces in QBF-encoded problems. The tool suites ChEQ and sKizzo/ozziKs evaluate and certify QBF formulas [3, 17, 24].

In verification and automata theory, a typical PSPACE-complete problem is the universality problem for nondeterministic finite automata. Despite its worst-case exponential complexity, dramatic performance improvements have been obtained recently for this problem by *antichain algorithms* [11, 12]. One key idea of antichain algorithms is to exploit the underlying structure of automata constructions (classically, powerset-based constructions) to define *subsumption relations*, yielding compact symbolic representations, as well as sound pruning of the search space. Although QBF is also a PSPACE-complete problem, this natural idea has never been used in QBF solving.

In this paper, we identify structural properties of QBF and we define pruning strategies to obtain antichain algorithms for QBF. The purpose is to define and evaluate antichain-based techniques for QBF solving, and to suggest that their integration in other search-based solvers could be valuable. We take the classical view of QBF as a reachability problem in an exponential And-Or graph, where the nodes represent subformulas (the And-nodes correspond to universal quantifications, and the Or-nodes to existential quantifications). We illustrate the main ideas of the algorithm on the following running example. Let $f = \forall x_1 x_2 x_3 \cdot \exists y_4 y_5 \cdot \forall x_6 \cdot \exists y_7 \cdot \psi$ where ψ (shown in Fig. 1) is a CNF formula viewed as the set of clauses $\{c_1, c_2, \dots, c_7\}$. The And-Or graph for f is a DAG where each level corresponds to a block of quantifiers (see a partial expansion in Fig. 2 where each clause c_i is identified with its index i). Nodes of the DAG are subsets $\varphi \subseteq \psi$ of clauses which remain to be satisfied in the evaluation game. The root of the DAG is the set ψ of all clauses. The successors of a node at level i are the sets of clauses obtained by assigning the variables quantified in the ith block of the formula. In the game interpretation, two players choose the successor of the nodes (player $\mathsf{P}_\forall$ in And-nodes, and $\mathsf{P}_\exists$ in Or-nodes) by assigning the variables quantified in the block of the level of the node. The goal of player $\mathsf{P}_\exists$ is to reach a node $\varnothing$ where all clauses are satisfied, and to avoid nodes where all literals of a clause are false (denoted by $\perp$). The QBF formula is true if and only if $\mathsf{P}_\exists$ has a winning strategy to reach $\varnothing$ from the initial node ψ in the game. A key observation is that *set inclusion* is a subsumption relation that can be used to substantially reduce the size of the search tree: if player $\mathsf{P}_\exists$ has a winning strategy from a node ψ_1 at level i, then player $\mathsf{P}_\exists$ also has a winning strategy from all nodes $\psi_2 \subseteq \psi_1$ at level i, because in ψ_2 less clauses remain to be satisfied. This has two implications in the search through the DAG. First, player $\mathsf{P}_\exists$ should only consider valuations that make true a *maximal subset* of the remaining clauses, while player $\mathsf{P}_\forall$ should

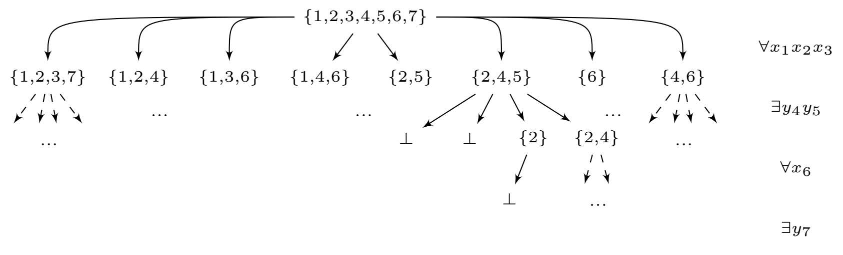

Fig. 2. Search tree for the formula of Fig. 1.

make true a *minimal subset*. Computing such variable assignments reduces to solving variants of Max-SAT and Min-SAT problems [19, 21]. Second, the set of winning nodes at level i is downward-closed, and the set of losing nodes at level i is upward-closed. Therefore, antichains of incomparable sets of clauses are the appropriate representation of winning and losing nodes. We exploit this structure when backward propagating the information collected during the exploration of the DAG, and we never explore a node which is smaller than a winning node (or greater than a losing node) at the same level. Finally, the information stored in the antichains at the end of the search is so rich that it immediately provides compact certificates for both positive and negative instances of QBF. Note that compact certificates represented by antichains is a new notion.

We propose an antichain algorithm which is search-based and reduces the search space using antichains of winning and losing nodes of the And-Or graph. Antichains can be viewed as compact symbolic representations which can be exponentially succinct, thus it also has the flavor of symbolic procedures. In contrast, traditional symbolic QBF solvers rely on binary decision diagrams (BDD) and they are based on quantifier elimination, such as Skolemization-based approaches [2] (with the aim at eliminating existentially quantified variables), or symbolic quantifier elimination by clause resolution or BDD algorithms [25]. The tool sKizzo falls in this category of solvers [5].

We have implemented the ideas presented in this paper in a stand-alone prototype in order to push and evaluate the approach, independently of the established heuristics commonly used in search-based QBF solvers. While some benchmarks are solved more efficiently with our prototype (e.g., see Fig. 5), the results are encouraging beyond the absolute performance. In particular the experiments and comparison with state-of-the-art solvers show that:

- the search trees constructed by our algorithm are generally much smaller (by orders of magnitude) as compared to the entire search space, thanks to subsumption (e.g., see Table 1);
- our algorithm automatically provides certificates with no additional cost, whereas in other approaches, additional computation is required to extract certificates after evaluation of the formula;
- difficult instances (several hundreds of variables, thousands of clauses) are solved by our prototype (e.g., see Table 1), and on several families of formulas, the overall behaviour of our algorithm scales better or similarly as the size of the formulas increases (e.g., see Figs. 5-8).

2 Preliminaries

2.1 Notations and QBF problem

Let $V = \{x_1, x_2, \ldots, x_m\}$ be a set of m Boolean variables, we use $X, X_1, X_2, \ldots$ to denote subsets of V. A *literal* ℓ is either a variable $x \in V$ or the negation $\bar{x}$ of a variable $x \in V$, and a *clause* c is a disjunction of literals, or equivalently a set of literals. We use notations such as $\ell \in c$, $\bar{x} \in c$, etc. A CNF formula is a conjunction of clauses, or equivalently a set of clauses. The empty CNF formula is denoted by 1, and the empty clause by 0. In the figures we use the notations $\varnothing$ and $\perp$ instead of 1 and 0 respectively. Given a set $X \subseteq V$ and a CNF formula ψ over V, we denote by $\pi_X(\psi)$ the projection of ψ over X, with $\pi_X(\psi) = \bigcup_{c \in \psi} \pi_X(c)$ and $\pi_X(c) = \{l \in c \mid l = x \text{ or } l = \overline{x} \Rightarrow x \notin X\}$.

A *quantified Boolean formula* (QBF) is an expression $\mathsf{Q}_1 X_1 \cdot \mathsf{Q}_2 X_2 \cdots \mathsf{Q}_n X_n \cdot \psi$ where each $\mathsf{Q}_i \in \{\exists, \forall\}$ for $1 \leq i \leq n$, the sets $X_1, \ldots, X_n$ (called *blocks*) form a partition of V, and ψ is a CNF formula over V. We also write $\mathsf{Q}_1 x_1 \cdot \mathsf{Q}_2 x_2 \cdots \mathsf{Q}_n x_n \cdot \psi$ when each block X_i contains one variable ($X_i = \{x_i\}$). Since ψ is in CNF we assume w.l.o.g. that the last block is *existential* (i.e., $\mathsf{Q}_n = \exists$). The truth value of a QBF formula is defined as usual. The QBF *evaluation problem* is to decide whether a given QBF formula is true or false. This problem is PSPACE-complete [26].

A *valuation* for $X \subseteq V$ is a function $v : X \rightarrow \{0, 1\}$. The domain of v is $\mathsf{dom}(v) = X$. If $X = \{x_1, \ldots, x_k\}$, a valuation $v : X \rightarrow \{0, 1\}$ can be identified with a word $a_1 a_2 \cdots a_k \in \{0, 1\}^{|X|}$ such that $a_l = v(x_l)$ for all $1 \leq l \leq k$. The empty word ϵ corresponds to $\mathsf{dom}(v) = \varnothing$. Given a partition $P = X_1 \cup X_2 \cup \cdots \cup X_n$ of V, let $X_{\leq i}$ be the set of variables $X_1 \cup X_2 \cdots \cup X_i$ (with $X_{\leq 0} = \varnothing$), and let $X_{\geq i} = V \setminus X_{\leq i-1}$. Given the valuations $v : X_{\leq i-1} \rightarrow \{0, 1\}$ and $w : X_i \rightarrow \{0, 1\}$, let vw be the valuation identified with the concatenation of the words representing v and w.

A clause c is *satisfied* by a valuation v (written $v \models c$) if there exists $x \in \mathsf{dom}(v)$ such that either $x \in c$ and $v(x) = 1$, or $\bar{x} \in c$ and $v(x) = 0$. Given a CNF formula ψ, we denote by $\mathsf{sat}_v(\psi)$ the set of clauses $c \in \psi$ such that $v \models c$. We denote by $\psi[v]$ the CNF formula obtained by replacing in ψ each variable $x \in \mathsf{dom}(v)$ by its value $v(x)$. Formula $\psi[v]$ is supposed to be simplified using the laws $c \vee 1 = 1$, $c \vee 0 = c$ with c being a clause, and $\varphi \wedge 1 = \varphi$, $\varphi \wedge 0 = 0$ with φ being a CNF formula.

Let ψ be an unsatisfiable CNF formula. An *unsatisfiable core* ψ' of ψ is any subset of clauses of ψ, minimal for the inclusion, such that ψ' is still unsatisfiable.

2.2 QBF problem as a game

It is classical to view the QBF evaluation problem as reachability in an And-Or graph, or equivalently as a two-player reachability game [26]. For the formula $f = \mathsf{Q}_1 x_1 \cdot \mathsf{Q}_2 x_2 \cdots \mathsf{Q}_m x_m \cdot \psi$ over $V = \{x_1, x_2, \ldots, x_m\}$, the game is played in m rounds (numbered $1, \ldots, m$) by the existential player $\mathsf{P}_\exists$ and the universal player $\mathsf{P}_\forall$. In round i, the truth value of the variable x_i is chosen by player $\mathsf{P}_{\mathsf{Q}_i}$. After m rounds, the players have constructed a valuation $v : V \rightarrow \{0, 1\}$, and player $\mathsf{P}_\exists$ wins if $\psi[v] = 1$ (all clauses are satisfied by v), otherwise player $\mathsf{P}_\forall$ wins. It is easy to see that $\mathsf{P}_\exists$ has a winning strategy in this game iff the formula f is true. Note that instead of having one round for each variable, we can also consider a game with one round for each block of variables, such that the blocks correspond to quantifier alternations in f. The players then choose a valuation for all the variables in the block at once, and the number of rounds is equal to the number of quantifier alternations. As the algorithms proposed in this paper are based on this game metaphor, we present the And-Or graph on which the game is played.

Let $P = X_1 \cup X_2 \cup \cdots \cup X_n$ be a partition of $V = \{x_1, x_2, \ldots, x_m\}$, and let $f = \mathsf{Q}_1 X_1 \cdot \mathsf{Q}_2 X_2 \cdots \mathsf{Q}_n X_n \cdot \psi$ be a QBF formula over V. We define the *And-Or graph* $G_f = (S, S_\exists, S_\forall, s_0, E, F)$ where:

- $S = \{\psi[v] \mid \mathsf{dom}(v) = X_{\leq i-1},\ \text{for } i, 1 \leq i \leq n+1\}$;
- $S_\exists = \{\psi[v] \mid \mathsf{dom}(v) = X_{\leq i-1} \wedge \mathsf{Q}_i = \exists,\ \text{for } i, 1 \leq i \leq n\}$ is the set of $\mathsf{P}_\exists$ nodes;
- $S_\forall = \{\psi[v] \mid \mathsf{dom}(v) = X_{\leq i-1} \wedge \mathsf{Q}_i = \forall,\ \text{for } i, 1 \leq i \leq n\}$ is the set of $\mathsf{P}_\forall$ nodes;
- $s_0 = \psi$ is the initial node;
- $E = \{(\psi[v], \psi[vw]) \mid \mathsf{dom}(v) = X_{\leq i-1} \wedge \mathsf{dom}(w) = X_i,\ \text{for } i, 1 \leq i \leq n\}$ is the set of edges;
- $F = \{\psi[v] \in S \mid \psi[v] = 1\}$ is the set of final nodes.

The set S is naturally partitioned into *levels* as follows: $S = \mathsf{Level}_1 \cup \mathsf{Level}_2 \cup \cdots \cup \mathsf{Level}_{n+1}$ where $\mathsf{Level}_i = \{\psi[v] \mid \mathsf{dom}(v) = X_{\leq i-1}\}$ for each $1 \leq i \leq n+1$. The objective of player $\mathsf{P}_\exists$ is to reach the set F of nodes $\psi[v]$ such that all clauses of ψ are satisfied by v. The game starts in node s_0 and player P_Q ($\mathsf{Q} \in \{\exists, \forall\}$) chooses the successor of node s if $s \in S_\mathsf{Q}$. Thus if $s = \psi[v] \in S_\exists$ and $\mathsf{dom}(v) = X_{\leq i-1}$, then player $\mathsf{P}_\exists$ chooses one of the $2^{|X_i|}$ possible successors of s in E, corresponding to a valuation $w : X_i \to \{0, 1\}$. A node s is *winning* for player $\mathsf{P}_\exists$ if he has a strategy to force reaching a node in F from s, no matter the choices of $\mathsf{P}_\forall$; otherwise it is *losing*. We denote by W the set of winning nodes for player $\mathsf{P}_\exists$, and by $L = S \setminus W$ the set of losing nodes for $\mathsf{P}_\exists$. We say that $\mathsf{P}_\exists$ is *winning the game* if $s_0 \in W$. In the sequel, we use the notations W_i (resp. L_i) to denote $W \cap \mathsf{Level}_i$ (resp. $L \cap \mathsf{Level}_i$).

Proposition 1. *A QBF formula f is true iff player $\mathsf{P}_\exists$ is winning the game G_f.*

Note that in the graph G_f, each node $\psi[v]$ with $\mathsf{dom}(v) = X_{\leq i-1}$ can be associated with the formula $\mathsf{Formula}(\psi[v]) \equiv \mathsf{Q}_i X_i \cdots \mathsf{Q}_n X_n \cdot \psi[v]$, and we can strengthen the previous proposition as follows.

Proposition 2. *Given a QBF formula f, the set of winning nodes in the graph G_f is $W = \{\psi[v] \in S \mid \mathsf{Formula}(\psi[v])$ is true$\}$, and the set of losing nodes is $L = \{\psi[v] \in S \mid \mathsf{Formula}(\psi[v])$ is false$\}$.*

2.3 Structure in the And-Or graph and antichains

We present in the next section an algorithm to solve the game played on G_f which exploits the following *subsumption* relation on QBF formulas. We write $f_1 \sqsubseteq f_2$ if $f_1 = \mathsf{Q}_i X_i \cdots \mathsf{Q}_n X_n \cdot \psi_1$ and $f_2 = \mathsf{Q}_i X_i \cdots \mathsf{Q}_n X_n \cdot \psi_2$ are two QBF formulas with the same quantifier prefix, and $\psi_1 \subseteq \psi_2$. Intuitively, f_1 is more promising than f_2 for player $\mathsf{P}_\exists$ because all strategies that are winning from ψ_2 are also winning from ψ_1.

Proposition 3. *Suppose that $f_1 \sqsubseteq f_2$. We have: if f_2 is true, then f_1 is true; and if f_1 is false, then f_2 is false.*

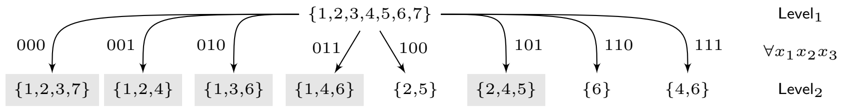

Fig. 3. Level$_1$ and Level$_2$ of G_f and the 5 minimal valuations of $\mathsf{P}_\forall$.

As a direct consequence of Propositions 2 and 3, we obtain the next corollary.

Corollary 1. *In the graph G_f, for all nodes $s_1, s_2 \in \mathsf{Level}_i$ such that $s_1 \subseteq s_2$, i.e.* $\mathsf{Formula}(s_1) \sqsubseteq \mathsf{Formula}(s_2)$*, if $s_2 \in W_i$, then $s_1 \in W_i$; and if $s_1 \in L_i$, then $s_2 \in L_i$.*

Hence, W_i is $\subseteq$-downward closed and L_i is $\subseteq$-upward closed. The set of $\subseteq$-maximal elements of W_i, noted $\lceil W_i \rceil$, is an *antichain* for the partial order $\subseteq$ (i.e. a set of pairwise incomparable elements) that canonically and compactly represents W_i. Similarly, the set of $\subseteq$-minimal elements of L_i, noted $\lfloor L_i \rfloor$, is an antichain that canonically and compactly represents L_i. Elements of these antichains are denoted α, β.

3 Algorithms

In Section 3.1, we discuss the computation of optimal valuations to explore only the most promising nodes, and in Section 3.2, we propose an antichain-based algorithm for solving the QBF evaluation game.

3.1 Maximal and minimal valuations

According to Corollary 1, when it is the turn for $\mathsf{P}_\exists$ to play in node $s = \varphi$ in Level_i, he can restrict his choices among valuations $w : X_i \to \{0,1\}$ that *maximize* the set of clauses of φ that are satisfied. Symmetrically, player $\mathsf{P}_\forall$ can restrict his choices among valuations $w : X_i \to \{0,1\}$ that *minimize* the set of clauses of φ that are satisfied.

We define the notion of maximal and minimal valuations as follows. Let φ be a CNF formula over $X_{\geq i}$. A valuation $w : X_i \to \{0,1\}$ is φ-*maximal* if for all $w' : X_i \to \{0,1\}$, $\mathsf{sat}_w(\varphi) \subseteq \mathsf{sat}_{w'}(\varphi)$ implies $\mathsf{sat}_w(\varphi) = \mathsf{sat}_{w'}(\varphi)$. Symmetrically, w is φ-*minimal* if for all $w' : X_i \to \{0,1\}$, $\mathsf{sat}_{w'}(\varphi) \subseteq \mathsf{sat}_w(\varphi)$ implies $\mathsf{sat}_w(\varphi) = \mathsf{sat}_{w'}(\varphi)$.

Example 1. Consider the CNF formula ψ of Fig. 1, viewed as the set of clauses $\{c_1, c_2, \ldots, c_7\}$. In Level$_1$, we have $X_1 = \{x_1, x_2, x_3\}$ which are universal variables. Among the $2^3 = 8$ valuations, 5 are ψ-minimal (shaded in Fig. 3, where each clause c_i is identified with i). Remember that the nodes in the And-Or graph are the clauses that *remain* to be satisfied, thus maximal such sets correspond to minimal valuations. Note also that we may need to compute *all* maximal (or minimal) valuations in a node.

Maximal and minimal valuations can be computed by multiple calls to a SAT solver. Let us give the intuition for maximal valuations (details are given in the appendix). Let

φ be a set of clauses over $X_{\geq i}$. First notice that a valuation $w : X_i \to \{0,1\}$ is φ-maximal if and only if it is $\pi_{X_i}(\varphi)$-maximal. Thus we can assume w.l.o.g. that φ is a set of clauses over X_i (instead of $X_{\geq i}$). Using a set of new variables $Y = \{y_c \mid c \in \varphi\}$, called selectors, we transform the set of clauses φ into a set of clauses φ' over $X_i \cup Y$ such that any valuation $w : X_i \cup Y \to \{0,1\}$ with $w(y_c) = 1$ implies that w satisfies c. By a first call to a SAT solver on φ', we get a valuation w and a subset C of clauses of φ that are satisfied by w. Then we modify φ' into φ'' by imposing additional constraints on the variables of Y in a way that a second call to a SAT solver provides a subset of satisfied clauses of φ that strictly contains C. Iterating this procedure, we finally obtain a valuation that satisfies a maximal set of clauses in φ.

Computing maximal and minimal valuations can also be computed thanks to solvers for variants of the *Maximum Satisfiability* (Max-SAT) and *Minimum Satisfiability* (Min-SAT) problems [19, 21]. Given a CNF formula φ, the Max-Sat problem asks to compute a valuation that maximizes the *number* of satisfied clauses in φ (Min-Sat is defined symmetrically). Note that such a valuation is φ-maximal but the converse is not necessarily true. Given a CNF fomula $\varphi = \varphi_h \wedge \varphi_s$ where φ_h represents the *hard* clauses and φ_s represents the *soft* clauses, the *partial Max-SAT* problem consists in finding a valuation such that all hard clauses are satisfied and the number of satisfied soft clauses is maximized. This variant of Max-SAT can be used to generate *all* φ-maximal valuations as follows. The first φ-maximal valuation is computed by a call to a Max-SAT solver. The next ones are computed thanks to a partial Max-SAT solver, such that hard clauses with selectors impose that for each already computed φ-maximal valuation w, at least one new clause $c \notin \mathsf{sat}_w(\varphi)$ is satisfied (see details in the appendix).

3.2 Antichain-based algorithm

In this section we present an antichain-based algorithm to evaluate a QBF formula $f = \mathsf{Q}_1 X_1 \cdots \mathsf{Q}_n X_n \cdot \psi$. It is a search-based algorithm of the And-Or graph G_f with backward propagation of the information collected during the exploration. Such a forward-backward exploration was also used with success in timed games [9].

Our algorithm consists of two recursive procedures named $\mathsf{ATCSearch}\exists(\varphi, i)$ and $\mathsf{ATCSearch}\forall(\varphi, i)$ where φ is a node of G_f and i is the recursion level (see Algorithms 1 and 2). Initially, we make a call to $\mathsf{ATCSearch}\exists(\psi, 1)$ if $\mathsf{Q}_1 = \exists$, and to $\mathsf{ATCSearch}\forall(\psi, 1)$ if $\mathsf{Q}_1 = \forall$. These procedures determine whether a node φ is winning or losing for $\mathsf{P}_\exists$, i.e. whether $\varphi \in W_i$ or $\varphi \in L_i$. The sets W_i and L_i are updated as global variables and compactly stored by antichains $\lceil W_i \rceil$ and $\lfloor L_i \rfloor$ respectively. They are used to prune the search by the *subsumption* checks (see lines 8, 11 in Algorithm 1).

The details of $\mathsf{ATCSearch}\exists(\varphi, i)$ are as follows. If φ is not even satisfiable, then it is a losing node; if φ is satisfiable and $i = n$, then it is winning since φ belongs to $S_\exists$; otherwise, the procedure enumerates the φ*-maximal* valuations w (line 7) and checks if $\varphi[w]$ is winning at level $i+1$. For player $\mathsf{P}_\exists$, maximal valuations are sufficient because the set of winning nodes is downward-closed (see Corollary 1). The recursive call to $\mathsf{ATCSearch}\forall(\varphi[w], i+1)$ can be avoided if $\varphi[w]$ is in the downward-closure of $\lceil W_{i+1} \rceil$ (line 8), or if $\varphi[w]$ is in the upward-closure of $\lfloor L_{i+1} \rfloor$ (line 11). Finally, if all φ-maximal valuations have been explored, then φ is losing (line 17).

Algorithm 1 ATCSearch$\exists(\varphi, i)$

```
Require: node $\varphi \in S_\exists \cap \mathsf{Level}_i$, $i \leq n$.
Ensure: Win if $\varphi \in W_i$, Lose if $\varphi \in L_i$.
if $\neg\mathsf{IsSat}(\varphi)$ then
    Add($\varphi$, $\lfloor L_i \rfloor$)
    return Lose
if $i = n$ then
    Add($\varphi$, $\lceil W_i \rceil$)
    return Win
for each $\varphi$-maximal valuation $w : X_i \to \{0,1\}$ do
    if $\exists\alpha \in \lceil W_{i+1} \rceil$ s.t. $\varphi[w] \subseteq \alpha$ then
        Add($\varphi$, $\lceil W_i \rceil$)
        return Win
    if $\neg(\exists\alpha \in \lfloor L_{i+1} \rfloor$ s.t. $\alpha \subseteq \varphi[w])$ then
        R ← ATCSearch$\forall(\varphi[w], i+1)$
        if R = Win then
            Add($\varphi$, $\lceil W_i \rceil$)
            return Win
Add($\varphi$, $\lfloor L_i \rfloor$)
return Lose
```

Algorithm 2 ATCSearch$\forall(\varphi, i)$

```
Require: node $\varphi \in S_\forall \cap \mathsf{Level}_i$, $i < n$.
Ensure: Win if $\varphi \in W_i$, Lose if $\varphi \in L_i$.
if $\neg\mathsf{IsSat}(\varphi)$ then
    Add($\varphi$, $\lfloor L_i \rfloor$)
    return Lose
for each $\varphi$-minimal valuation $w : X_i \to \{0,1\}$ do
    if $\exists\alpha \in \lfloor L_{i+1} \rfloor$ s.t. $\alpha \subseteq \varphi[w]$ then
        Add($\varphi$, $\lfloor L_i \rfloor$)
        return Lose
    if $\neg(\exists\alpha \in \lceil W_{i+1} \rceil$ s.t. $\varphi[w] \subseteq \alpha)$ then
        R ← ATCSearch$\exists(\varphi[w], i+1)$
        if R = Lose then
            Add($\varphi$, $\lfloor L_i \rfloor$)
            return Lose
Add($\varphi$, $\lceil W_i \rceil$)
return Win
```

The procedure ATCSearch$\forall(\varphi, i)$ for nodes $\varphi \in S_\forall$ is dual. Note that the case $i = n$ is not relevant since $\mathsf{Q}_n = \exists$. By a symmetrical argument, $\mathsf{P}_\forall$ needs to consider only the φ-minimal valuations.

In these two procedures, the φ-maximal and φ-minimal valuations are computed as explained in Section 3.1, by either using a SAT solver or a partial Max-SAT solver. Procedure $\mathsf{IsSat}(\varphi)$ tests whether formula φ is satisfiable by a call to a SAT solver. The antichains $\lceil W_i \rceil$ and $\lfloor L_i \rfloor$ (for $1 \leq i \leq n$) are initially empty. The antichain structure is maintained by the procedure Add which computes $\lceil \{\varphi\} \cup W_i \rceil$ and $\lfloor \{\varphi\} \cup L_i \rfloor$.

Example 2. Consider the CNF formula ψ of Fig. 1. Since $\mathsf{Q}_1 = \forall$, the algorithm starts with ATCSearch$\forall(\psi, 1)$ which needs to explore the 5 minimal valuations of Fig. 3. Assume that the first valuation is $(x_1 \mapsto 0, x_2 \mapsto 0, x_3 \mapsto 0)$, denoted 000, which satisfies clauses $4, 5, 6$. Then, the game proceeds to the node $\psi[w] = \psi[000] = \{1, 2, 3, 7\}$ of remaining clauses where the turn is to player $\mathsf{P}_\exists$. The subgraph of G_f rooted at $\psi[000]$ is shown in the first tree of Fig. 4. Among the 4 possible valuations for player $\mathsf{P}_\exists$, only 2 are maximal, namely 01 and 11. For valuation 01, only clauses 1 and 7 remain. At this point, all variables are instantiated except x_6 and y_7, and player $\mathsf{P}_\exists$ wins by choosing $y_7 \mapsto 1$ which satisfies ψ no matter the value of x_6 chosen by player $\mathsf{P}_\forall$.

Thus nodes $\{1, 7\}$ at Level_3, and $\{1, 2, 3, 7\}$ at Level_2 are winning, and the related antichains are updated as follows: $\lceil W_3 \rceil = \{\{1, 7\}\}$ and $\lceil W_2 \rceil = \{\{1, 2, 3, 7\}\}$.

At the root node of G_f, the valuation 000 is not a good choice for player $\mathsf{P}_\forall$, and no conclusion can be drawn yet for this node (Fig. 3). We need to explore another choice

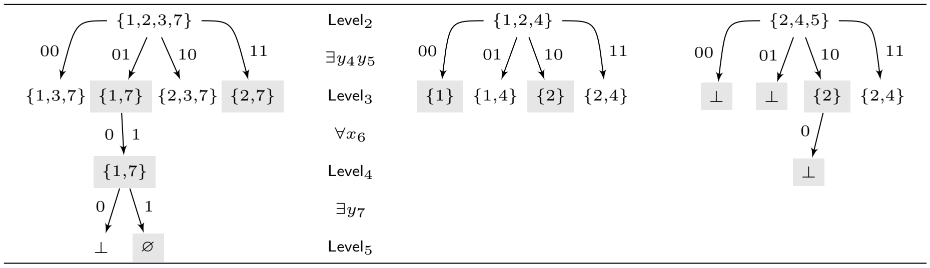

Fig. 4. Subgames rooted at $\psi[000]$, $\psi[001]$, and $\psi[101]$

for player $\mathsf{P}_\forall$. The second tree of Fig. 4 shows the subgame rooted at node $\psi[001]$. In this case, with minimal valuation 00, player $\mathsf{P}_\exists$ reaches node $\{1\}$ in Level$_3$. Since $\{1\} \in W_3$ (indeed $\{1\} \subseteq \{1,7\} \in \lceil W_3 \rceil$, and W_3 is the downward-closure of $\lceil W_3 \rceil$), he knows immediately that he is winning without further exploring the graph. This situation illustrates the power of the subsumption which allows to prune the search for nodes smaller than previously visited winning ones. The antichain $\lceil W_2 \rceil$ is then updated to $\{\{1,2,3,7\},\{1,2,4\}\}$. Valuation 001 is again a bad choice for $\mathsf{P}_\forall$.

One can check that valuations 010 and 011 are bad choices for $\mathsf{P}_\forall$ and their exploration leads to the following update of the antichains: $\lceil W_2 \rceil = \{\{1,2,3,7\},\{1,2,4\},$ $\{1,3,6\},\{1,4,6\}\}$ and $\lceil W_3 \rceil = \{\{1,7\},\{6\}\}$. The last minimal valuation is 101 and for all choices of player $\mathsf{P}_\exists$, there is a choice of player $\mathsf{P}_\forall$ to falsify the formula (see the last tree in Fig. 4). Therefore $\mathsf{P}_\exists$ is losing the game and the formula f is false.

The correctness of this algorithm is established using the notion of certificate presented in the next section.

Theorem 1. *Let f be a QBF formula. Applying Algorithms 1 and 2 on f returns* `Win` *if and only if f is true.*

4 Certificates

In the previous section we have described a search-based algorithm to evaluate a QBF formula f. This algorithm computes the sets of winning nodes and losing nodes for each level of the graph G_f, and these sets are compactly represented by antichains.

Our algorithm gathers enough information in these antichains to easily build *compact certificates* for both true and false QBF formulas. The certificates for true formulas differ from the certificates for false formula. Intuitively, if f is true, that is, player $\mathsf{P}_\exists$ is winning the game G_f, then a certificate is given by the antichains $\lceil W_i \rceil$, $1 \leq i \leq n$, and for each $\alpha \in W_i$ by the maximal valuation computed by Algorithm 1 when α has been declared winning. To the best of our knowledge they are different from the certificates considered in the literature [3, 24].

We first define positive certificates as a witness for true QBF formulas. A *positive certificate* for a formula $f \equiv \mathsf{Q}_1 X_1 \cdots \mathsf{Q}_n X_n \cdot \psi$ is a pair $\langle (C_i^+)_{1 \leq i \leq n}, \mathsf{w} \rangle$ such that:

- each C_i^+ is a set of nodes at the ith level of the graph G_f, that is, $C_i^+ \subseteq \mathsf{Level}_i$;
- for each i such that $\mathsf{Level}_i \subseteq S_\exists$, w is a function that assigns a valuation $\mathsf{w}(\alpha) : X_i \rightarrow \{0,1\}$ to each $\alpha \in C_i^+$;
- and the following properties are verified:
  1. $C_1^+ = \{\psi\}$.
  2. for each $i < n$ such that $\mathsf{Level}_i \subseteq S_\exists$, for all $\alpha \in C_i^+$, there exists $\beta \in C_{i+1}^+$ such that $\alpha[\mathsf{w}(\alpha)] \subseteq \beta$.
  3. for each $i < n$ such that $\mathsf{Level}_i \subseteq S_\forall$, for all $\alpha \in C_i^+$, for all $w : X_i \rightarrow \{0,1\}$, there exists $\beta \in C_{i+1}^+$ such that $\alpha[w] \subseteq \beta$.
  4. for $i = n$, for all $\alpha \in C_i^+$, $\alpha[\mathsf{w}(\alpha)] = 1$.

Clearly, there exists a nondeterministic polynomial time algorithm to recognize pairs $\langle (C_i^+)_{1 \leq i \leq n}, \mathsf{w} \rangle$ that are not positive certificate. All the verification related to Conditions 1, 2 and 4 can be done in deterministic polynomial time while Condition 3 requires nondeterminism. Therefore, verifying the validity of a positive certificate is a problem in coNP.

The next two lemmas are proved in the appendix.

Lemma 1. *If $\langle (C_i^+)_{1 \leq i \leq n}, \mathsf{w} \rangle$ is a positive certificate for a QBF formula f, then f is true.*

Lemma 2. *Let $\lceil W_i \rceil$, $1 \leq i \leq n$, be the antichains built by the execution of Algorithms 1 and 2 on formula f. For each i such that $\mathsf{Level}_i \subseteq S_\exists$, for all $\alpha \in \lceil W_i \rceil$, let $\mathsf{w}(\alpha)$ be the valuation used by Algorithms 1 when α has been declared winning. If f is true, then $\langle (\lceil W_i \rceil)_{1 \leq i \leq n}, \mathsf{w} \rangle$ is a positive certificate for f.*

The next theorem directly follows from the two previous lemmas.

Theorem 2. *Let f be a QBF formula. Then f is true if and only if there exists a positive certificate for f.*

Negative certificates are defined in a way similar to positive certificates; they are a witness for false formulas f. The reader is referred to the appendix for more details.

5 Optimizations

5.1 Guiding the search to promising valuations

We recall that in Algorithm 1, when φ is a satisfiable formula, and $i \leq n-1$, player $\mathsf{P}_\exists$ traverses all the maximal valuations $w : X_i \rightarrow \{0,1\}$ in the hope to find w such that $\varphi[w]$ is winning. The first optimization that we consider tries to guide the search to *promising* maximal valuations. The new algorithm for player $\mathsf{P}_\exists$ is described in the appendix. Symmetrical improvements also exist for player $\mathsf{P}_\forall$.

Improving $\mathsf{ATCSearch}\exists$. When considering all the maximal valuations $w : X_i \rightarrow \{0,1\}$, we observe that player $\mathsf{P}_\exists$ is winning as soon as he can find a valuation w such that $\varphi[w] \subseteq \alpha$ for some $\alpha \in \lceil W_{i+1} \rceil$ (see Corollary 1). So, it is wise for $\mathsf{P}_\exists$ to first try to find such a valuation w. Suppose now that player $\mathsf{P}_\exists$ cannot win by exploiting the elements of $\lceil W_{i+1} \rceil$. Then he should avoid considering maximal valuations w such that $\alpha \subseteq \varphi[w]$ for some $\alpha \in \lfloor L_{i+1} \rfloor$ as $\varphi[w]$ is losing (see Corollary 1). These two observations are exploited by the new algorithm.

Improving ATCSearch$\forall$. We can improve the choice of minimal valuations for player $\mathsf{P}_\forall$ in a symmetric manner. Indeed, $\mathsf{P}_\forall$ should first look for a valuation w such that there exists $\alpha \in \lfloor L_{i+1} \rfloor$ with $\alpha \subseteq \varphi[w]$. If such a valuation does not exist, then he should only consider minimal valuations that avoid the set $\lceil W_{i+1} \rceil$ of winning nodes.

5.2 Improving the information about losing and winning nodes

Our algorithms can be seen as mixing a forward exploration of the And-Or graph G_f with a backward propagation of the information about the winning and losing nodes. We present below several ways of improving the propagation phase.

At initialization. Recall that the antichain $\lfloor L_i \rfloor$ is initially empty in Algorithms 1 and 2. We can add some useful information to $\lfloor L_i \rfloor$ in the following case. Consider the initial node ψ of G_f, and let i, $1 \leq i \leq n-1$. Suppose that the projection $d = \pi_{X \geq i}(c)$ of a clause c in ψ on $X_{\geq i}$ has all its literals *universally quantified*. Then the clause d must be satisfied by a valuation generated before reaching Level_i since otherwise player $\mathsf{P}_\forall$ can falsify it. So, at initialization we can add $\{d\}$ to $\lfloor L_i \rfloor$ for any such d (all sets of clauses that contain d are losing at Level_i). An example is given in the appendix.

Some other information can be initially stored in the antichains $\lceil W_i \rceil$ and $\lfloor L_i \rfloor$ to better guide the search. Consider the initial node ψ of G_f, and its projection $\varphi = \pi_{X \geq i}(\psi)$. If φ is unsatisfiable, then any unsatisfiable core of φ can be extracted and added to $\lfloor L_i \rfloor$. On the other hand, if φ is satisfiable and we further restrict φ to the existentially quantified variables, then we can compute a φ-maximal valuation w to get a maximal subset of satisfied clauses $\varphi' = \mathsf{sat}_w(\varphi)$. The set φ' can be added to $\lceil W_i \rceil$ because player $\mathsf{P}_\exists$ has a winning strategy at Level_i if the set of clauses not in φ' are satisfied when the game enters this level.

When updating $\lceil W_i \rceil$ *and* $\lfloor L_i \rfloor$. We now give an optimization that can be applied when updating the sets of winning and losing nodes during the search. We consider two scenarios. First, assume that node φ is declared winning by the algorithm ATCSearch$\exists$ in Level_i. This means that there exists a valuation $w : X_i \to \{0, 1\}$ such that either $\varphi[w] \subseteq \alpha$ for some $\alpha \in \lceil W_{i+1} \rceil$, or $\varphi[w]$ is declared winning by the recursive call to ATCSearch$\forall$. Notice that φ is a subset of the set of clauses in the root ψ (projected on variables $X_{\geq i}$). It may happen that other clauses $c \in \psi$ are also satisfied by the valuation w. Thus, in a way to have bigger elements in $\lceil W_i \rceil$, it is preferable to add the set $\varphi' = \varphi \cup \{c \mid c \in \mathsf{sat}_w(\psi)\}$ instead of φ to $\lceil W_i \rceil$. An example is given in the appendix.

Second, assume that φ is declared losing by Algorithm ATCSearch$\exists$ in Level_i with $i = n$. This means that φ is unsatisfiable. Instead of adding φ to $\lfloor L_i \rfloor$, we can add any unsatisfiable core $\varphi' \subseteq \varphi$ instead.

6 Experimental Results

Setting. We have implemented the algorithms of Section 3 with the optimizations of Section 5 in a stand-alone prototype in order to evaluate the impact of the antichain approach. Thus none of the classical heuristics used in search-based QBF solvers, like backjumping, unit propagation, and monotone literals elimination [8] has been integrated. The code is written partly in C for the low level operations on the data structures, and partly in Python to implement high level operations like the exploration of the And-Or graph, and for the construction of CNF formulas submitted to the SAT or partial Max-SAT solvers. We use Python to facilitate the fast evaluation of different ideas even if some price has to be paid at the performance level. We use the SAT solver MiniSat [13] and the partial Max-SAT solver Akmaxsat [20] to compute the maximal and minimal valuations as explained in Section 3.1, and PicoSAT [7] to compute unsatisfiable cores as described in Section 5.

In a preprocessing step, the formula is simplified with PreQuel [28] and then presented in a tree-like structure [4]. This is standard practice in QBF solving. The experiments were run on a PC equipped with a Intel i7 2.8GHz processor, 6 GB of RAM and running Linux Ubuntu 2.6.

Instance	Var	Cl	Blocs	Value	Nodes	ATC	QuBE	DepQBF	sKizzo
k-path-n-01	108	275	7	1	14	0.22	0.01	0.005	0.01
k-path-n-02	180	481	9	1	43	0.93	0.01	0.02	0.05
k-path-n-05	384	1051	15	1	120	4.17	2.53	39.97	0.08
k-path-n-06	456	1257	17	1	142	7.54	13.74	/	0.26
k-path-n-10	732	2033	25	1	247	16.33	/	/	45.87
k-path-n-11	804	2234	27	1	269	27.36	/	/	445.71
k-path-n-20	1428	3992	45	1	503	95.19	/	/	/
k-path-n-21	1488	4155	47	1	452	88.81	/	/	/

Table 1. Family k-path-n.

Instances. We tested our algorithm on several instances proposed during the seventh QBF solvers evaluation (QBFEVAL'10) [27] and compared the results with the ones obtained with three state-of-the-art QBF solvers: QuBE-7.0, sKizzo-v0.8.2-beta and DepQBF-0.1 (winner of QBFEVAL'10). QuBE and DepQBF are search-based solvers, whereas sKizzo uses symbolic Skolemization.

The families k-$*$ correspond to the encoding of the satisfiability problem for modal K formulas into QBF, and they are known to give difficult instances for search-based solvers. The families k-$*$ have a deep level of quantifier alternations while the families Toilet$*$ and aim-$*$ have three quantifier alternations.

Results. In Table 1, **Var** (resp. **Cl**, **Blocs**) gives the number of variables (resp. clauses, blocs) of the instance and **Value** is its truth value. **Nodes** is the number of nodes of the And-Or graph that have been explored. The columns **ATC**, **QuBE**, **DepQBF**, and **sKizzo** present the execution times in seconds (with a timeout of 600 seconds) of our antichain-based solver and the three other solvers. The experimental results obtained with our solver are encouraging.

- According to pure performance, our prototype already performs better than the three state-of-the-art solvers for the families k-path-n and k-path-p.
  For other families in k-$*$, QuBE and DepQBF solvers (which are search-based) have often an execution time beyond 600 seconds (see the appendix).
  See Table 1, Fig. 5 and 6 (the complete table for k-path-n is given in the appendix).
- The antichain approach leads to search trees that are *amazingly small* as compared to the size of the entire search space (see the **Nodes** entry in Table 1 as compared

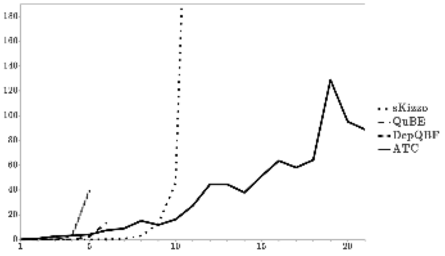

Fig. 5. Family k-path-n

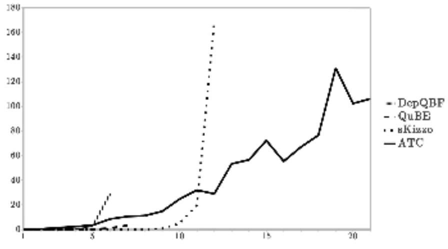

Fig. 6. Family k-path-p

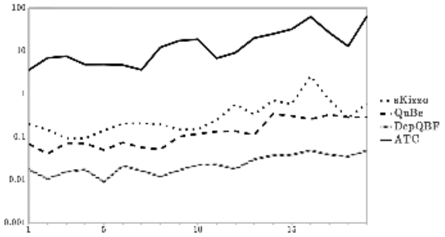

Fig. 7. Family aim-100 (true - log scale)

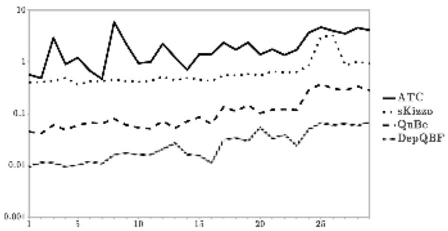

Fig. 8. Family aim-100 (false - log scale)

to the size of the entire search space in $O(2^{\mathbf{Var}})$); the antichains are thus also very small as they are composed of nodes of the search tree.

- For several families from the QBFLIB library [15] (including Toilet$*$ and aim-$*$), the execution times of our solver follow the same shape of curve (at logarithmic scale, with a timeout of 600 seconds) as for the other three state-of-the-art solvers, see for instance the family aim-100 in Fig. 7 (true instances) and Fig. 8 (false instances). Other tables and figures for families Toilet$*$ and aim-$*$ are given in the appendix.

As explained in Section 3.1, the maximal and minimal valuations can be computed either with a SAT solver of with a partial Max-SAT solver. In Section 5.2, we described several ways to improve the information about losing and winning nodes in the antichains. These different approaches have been tested, and the experiments show the best approach can vary with the family of formulas that is tested (the table and figures always present the results for the best approach). For instance, depending on the family, the size of the search space can be either increased or decreased when using a partial MaxSAT solver instead of a SAT solver.

Along with deciding if a given QBF formula is true or false, our solver provides compact certificates in the form of antichains both for positive and negative instances of QBF, and *without any additional computation* (see Section 4). DepQBF solver does not construct certificates. For sKizzo and QuBE solvers, additional computation is required to extract certificates [3, 17, 24]. The log produced by sKizzo is evaluated by ozziKs to construct certificates (only for true instances); QuBE-cert (in the suite ChEQ) is an extension of QuBE that adds to QuBE the instrumentation required to generate certificates (for both true and false instances). In [3], experiments have been done with the family adder∗ with "*the surprising phenomenon that the time taken to reconstruct a model may overcome the time needed to solve the instance*".

7 Conclusion and Perspectives

The approach presented in this paper for QBF solving is inspired by previous works on effective antichain algorithms for PSPACE-complete problems in automata theory which are based on simple subsumption relations [11, 12]. While the And-Or graph of QBF formulas enjoys such a subsumption relation, this idea has not been exploited in search-based QBF solvers. In a prototypical implementation, we have evaluated the feasibility of antichain-based algorithms for QBF. Experimental results show that on several benchmarks the size of the search tree is drastically reduced, and that some instances are solved more efficiently than by the leading QBF solvers. This shows that the antichain approach is promising and it provides a new research direction in the area. Its integration in standard search-based QBF solvers is worth investigating. On the other hand, our algorithm provides automatically compact certificates represented by antichains with no additional cost, and for both true and false instances.

We now plan to improve our algorithm with respect to the computation of the valuations. Indeed, we observed on the experiments that the execution time is partly spent when computing maximal and minimal valuations. Our current solver computes the *best* valuations since they are restricted to maximal/minimal ones, and exploit as much as possible the information stored in the antichains. We could instead compute *approximate* valuations in a way to decrease the execution time while keeping the advantages of the antichains [1].

Our work also provides a new application of the Max-SAT problem. We intend to submit instances coming from our experiments to the *Evaluation of Max-SAT Solvers* that is yearly organized as an affiliated event of the International Conference on Theory and Applications of Satisfiability Testing (SAT)[1]. We believe that the difficult instances we produce could be of interest to the Max-SAT community and that antichain-based QBF solving would benefit from their improvements.

Acknowledgements We thank Marco Benedetti, the author of sKizzo, for his great help, Florian Lonsing for explanations about DepQBF, the QuBE's team for answers about their solver, and Nicolas Maquet for his guidance in the implementation.

References

1. T. Asano and D. P. Williamson. Improved approximation algorithms for max sat. *J. Algorithms*, 42(1):173–202, 2002.
2. M. Benedetti. Evaluating QBFs via Symbolic Skolemization. In F. Baader and A. Voronkov, editors, *LPAR*, volume 3452 of *LNCS*, pages 285–300. Springer, 2004.
3. M. Benedetti. Extracting Certificates from Quantified Boolean Formulas. In L. P. Kaelbling and A. Saffiotti, editors, *IJCAI*, pages 47–53. Professional Book Center, 2005.

[1] See `http://www.maxsat.udl.cat/`

4. M. Benedetti. Quantifier Trees for QBFs. In F. Bacchus and T. Walsh, editors, *SAT*, volume 3569 of *LNCS*, pages 378–385. Springer, 2005.
5. M. Benedetti. sKizzo: A Suite to Evaluate and Certify QBFs. In R. Nieuwenhuis, editor, *CADE*, volume 3632 of *LNCS*, pages 369–376. Springer, 2005.
6. M. Benedetti and H. Mangassarian. Qbf-based formal verification: Experience and perspectives. *JSAT*, 5(1-4):133–191, 2008.
7. A. Biere. PicoSAT Essentials. *JSAT*, 4(2-4):75–97, 2008.
8. M. Cadoli, A. Giovanardi, and M. Schaerf. An algorithm to evaluate quantified Boolean formulae. In *Proc. of AAAI-98/IAAI-98*, pages 262–267. MIT Press, 1998.
9. F. Cassez, A. David, E. Fleury, K. G. Larsen, and D. Lime. Efficient on-the-fly algorithms for the analysis of timed games. In M. Abadi and L. de Alfaro, editors, *CONCUR*, volume 3653 of *LNCS*, pages 66–80. Springer, 2005.
10. B. Cook, D. Kroening, and N. Sharygina. Verification of boolean programs with unbounded thread creation. *Theor. Comput. Sci.*, 388(1-3):227–242, 2007.
11. M. De Wulf, L. Doyen, T. A. Henzinger, and J.-F. Raskin. Antichains: A New Algorithm for Checking Universality of Finite Automata. In *Proc. of CAV*, LNCS 4144, pages 17–30. Springer, 2006.
12. L. Doyen and J.-F. Raskin. Antichain Algorithms for Finite Automata. In J. Esparza and R. Majumdar, editors, *TACAS*, volume 6015 of *LNCS*, pages 2–22. Springer, 2010.
13. N. Eén and N. Sörensson. An extensible SAT-solver. In E. Giunchiglia and A. Tacchella, editors, *SAT*, volume 2919 of *LNCS*, pages 502–518. Springer, 2003.
14. U. Egly, T. Eiter, H. Tompits, and S. Woltran. Solving Advanced Reasoning Tasks Using Quantified Boolean Formulas. In *Proc. of IAAI*, pages 417–422. AAAI Press, 2000.
15. E. Giunchiglia, M. Narizzano, and A. Tacchella. Quantified Boolean Formulas satisfiability library (QBFLIB), 2001. www.qbflib.org.
16. E. Giunchiglia, M. Narizzano, and A. Tacchella. QuBE++: An Efficient QBF Solver. In *Proc. of FMCAD*, LNCS 3312, pages 201–213. Springer, 2004.
17. E. Giunchiglia, M. Narizzano, and A. Tacchella. Clause/Term Resolution and Learning in the Evaluation of Quantified Boolean Formulas. *J. Artif. Intell. Res.*, 26:371–416, 2006.
18. T. Jussila and A. Biere. Compressing BMC Encodings with QBF. *Electron. Notes Theor. Comput. Sci.*, 174:45–56, May 2007.
19. R. Kohli, R. Krishnamurti, and P. Mirchandani. The Minimum Satisfiability Problem. *SIAM J. Discrete Math.*, 7(2):275–283, 1994.
20. A. Kügel. Improved Exact Solver for the Weighted Max-SAT problem, 2011. Accepted at the workshop Pragmatics of SAT. To appear in easychair electronic proceedings.
21. C. M. Li and F. Manyà. MaxSAT, Hard and Soft Constraints. In *Handbook of Satisfiability*, Frontiers in Artificial Intelligence and Applications 185, pages 613–631. IOS Press, 2009.
22. F. Lonsing and A. Biere. DepQBF: A dependency-aware QBF solver (System Description). *Journal on Satisfiability, Boolean Modeling and Computation*, 7:71–76, 2010.
23. F. Lonsing and A. Biere. Integrating Dependency Schemes in Search-Based QBF Solvers. In *Proc. of SAT*, LNCS 6175, pages 158–171. Springer, 2010.
24. M. Narizzano, C. Peschiera, L. Pulina, and A. Tacchella. Evaluating and certifying QBFs: A comparison of state-of-the-art tools. *AI Commun.*, 22:191–210, December 2009.
25. G. Pan and M. Y. Vardi. Symbolic Decision Procedures for QBF. In M. Wallace, editor, *CP*, volume 3258 of *LNCS*, pages 453–467. Springer, 2004.
26. C. H. Papadimitriou. *Computational complexity*. Addison-Wesley Publishing Company, Reading, MA, 1994.
27. C. Peschiera, L. Pulina, A. Tacchella, U. Bubeck, O. Kullmann, and I. Lynce. The Seventh QBF Solvers Evaluation (QBFEVAL'10). In *Proc. of SAT*, LNCS 6175, pages 237–250. Springer, 2010.
28. H. Samulowitz, J. Davies, and F. Bacchus. Preprocessing QBF. In F. Benhamou, editor, *CP*, volume 4204 of *LNCS*, pages 514–529. Springer, 2006.

Appendix

Details concerning Subsection 3.1

Maximal valuations. To compute φ-maximal valuations $w : X_i \to \{0,1\}$ for a CNF formula φ over $X_{\geq i}$, we adopt a classical approach that performs multiple calls to a SAT solver, and can be summarized as follows.

First recall that w.l.o.g. we can suppose that $\varphi = \{c_1, c_2, \ldots, c_k\}$ is a formula over X_i (instead of $X_{\geq i}$). For each clause $c_i \in \varphi$, we introduce a new variable y_i called the *selector* of c_i. We denote by Y this set of new variables. We modify each clause $c_i \in \varphi$ as $c'_i = \overline{y_i} \vee c_i$. Notice that if c'_i is satisfied by some valuation $v : X_i \cup Y \to \{0,1\}$ and $v(y_i) = 1$, then c_i must be satisfied by v. Then we construct the CNF formula

$$\varphi' = \bigwedge_{c_i \in \varphi} c'_i \wedge \bigvee_{y_i \in Y} y_i.$$

If φ' is unsatisfiable, then there is no clause c_i in φ that can be made true, and all valuations $w : X_i \to \{0,1\}$ are φ-maximal (because $\mathsf{sat}_w(\varphi) = \varnothing$). Otherwise, let $v_1 : X_i \cup Y \to \{0,1\}$ be a valuation such that $\varphi'[v_1] = 1$. From v_1, we know that all clauses c_i such that $v_1(y_i) = 1$ are satisfied by v_1. Let $C_{v_1}(\varphi) = \{c_i \in \varphi \mid v_1(y_i) = 1\}$. This set gives a first subset of clauses in φ that can be satisfied *all together*. Now, we want to know if there exists a *superset* of this set that can be still satisfied. In this aim we construct a new formula

$$\varphi'(v_1) = \bigwedge_{c_i \in \varphi} c'_i \wedge \bigwedge_{y_i \in Y, v_1(y_i)=1} y_i \wedge \left(\bigvee_{y_i \in Y, v_1(y_i)=0} y_i \right).$$

Suppose that $\varphi'(v_1)$ is satisfiable and let v_2 be a valuation that evaluates it to true. The second part of $\varphi'(v_1)$ forces that all the clauses c_i satisfied by v_1 are still satisfied by v_2. The last part of the formula imposes that at least one additional clause c_i is satisfied by v_2. It follows that $C_{v_1}(\varphi) \subset C_{v_2}(\varphi)$, and we can iterate the process until formula $\varphi'(v_l)$ becomes unsatisfiable for some $l \geq 2$. Valuation $w = v_{l-1}$ identifies a maximal subset of satisfiable clauses c_i, and w is thus a φ-maximal valuation.

Minimal valuations. The approach to find minimal valuations uses an iteration schema similar to the one for maximal valuations. Let $\varphi = \{c_1, c_2, \ldots, c_k\}$ be a formula and let Y be the set of selectors as above. To compute a φ-minimal valuation $w : X_i \to \{0,1\}$, we first transform each clause of $c_i \in \varphi$ into the formula $\overline{c_i} \vee y_i$ which is expressed in CNF as the clause $c'_i = \bigwedge_{\ell \in c_i} \neg \ell \vee y_i$. Notice that if c'_i is satisfied by some valuation $v : X_i \cup Y \to \{0,1\}$ and $v(y_i) = 0$, then c_i cannot be satisfied by v. Then, we consider the formula

$$\varphi' = \bigwedge_{c_i \in \varphi} c'_i \wedge \bigvee_{y_i \in Y} \neg y_i.$$

If φ' is unsatisfiable, then no clause in φ can be made false and any valuation $w : X_i \to \{0,1\}$ is φ-minimal. Otherwise, let $v_1 : X_i \cup Y \to \{0,1\}$ be a valuation such that $\varphi'[v_1] = 1$. Valuation v_1 identifies the set of clauses $C_{v_1}(\varphi) = \{c_i \in \varphi \mid v_1(y_i) = 0\}$ such that these clauses are unsatisfied by v_1 all together. With an iteration schema similar to the one defined above, we can find a set of valuations $\{v_1, v_2, \ldots, v_l\}$ such that $\varphi'(v_l)$ is unsatisfiable and $C_{v_1}(\varphi) \subset C_{v_2}(\varphi) \subset \cdots \subset C_{v_{l-1}}(\varphi)$. Valuation $w = v_{l-1}$ identifies a minimal subset of satisfiable clauses c_i in φ (given by the complement of $C_{v_{l-1}}(\varphi)$) and it is thus a φ-minimal valuation.

Variants of MaxSAT problem. We have explained above how to use a SAT solver to compute a maximal or a minimal valuation. Another approach is to use a Max-SAT solver and a Partial Max-SAT solver. Let $\varphi = \{c_1, c_2, \ldots c_k\}$ be a CNF formula over X_i. A first φ-maximal valuation is computed by one call to a Max-SAT solver. Suppose that the set $\{v_1, v_2, \ldots, v_l\}$ of φ-maximal valuations has already been computed. A new φ-maximal valuation is computed as follows using a Partial Max-SAT solver. For each clause $c_i \in \varphi$, we introduce a selector y_i. The CNF formula $\varphi_h \wedge \varphi_s$ is submitted to the solver such that $\varphi_s = \varphi$ (soft clauses) and

$$\varphi_h = \bigwedge_{i=1}^{k} (\overline{y_i} \vee c_i) \ \wedge \ \bigwedge_{j=1}^{l} (\bigvee_{v_j(y_i)=0} y_i)$$

(hard clauses). The hard clauses express that for each already computed valuation v_j, at least one new clause $c \notin \mathsf{sat}_{v_j}(\varphi)$ is satisfied.

Details concerning Section 4

Proof (of Lemma 1). By induction we show that for all i, $1 \leq i \leq n$, for all $\alpha \in C_i^+$, $\mathsf{Formula}(\alpha)$ is true. This is sufficient as $f = \mathsf{Formula}(\psi)$, and $\psi \in C_1^+$.

We start with the base case $i = n$. Let $\alpha \in C_n^+$, so $\mathsf{Formula}(\alpha) = \exists X_n \cdot \alpha$. By definition of positive certificates, the valuation $\mathsf{w}(\alpha) : X_n \to \{0,1\}$ is such that $\alpha[\mathsf{w}(\alpha)] = 1$. So the valuation $\mathsf{w}(\alpha)$ satisfies all the clauses of α. It is thus a witness that the formula $\exists X_n \cdot \alpha$ is true.

We now proceed with the induction case, under the hypothesis that for all for all $\alpha \in C_{i+1}^+$, $\mathsf{Formula}(\alpha)$ is true. We consider two cases.

Suppose that $\mathsf{Level}_i \subseteq S_\exists$, and let $\alpha \in C_i^+$. By definition of positive certificates, there exists $\beta \in C_{i+1}^+$ such that $\alpha' = \alpha[\mathsf{w}(\alpha)]$ and $\alpha' \subseteq \beta$. So $\mathsf{Formula}(\alpha') \sqsubseteq \mathsf{Formula}(\beta)$. By induction hypothesis $\mathsf{Formula}(\beta)$ is true; by Proposition 3, $\mathsf{Formula}(\alpha')$ is also true. Therefore, as $\mathsf{Q}_i = \exists$, it follows that $\mathsf{Formula}(\alpha)$ is true.

Suppose that $\mathsf{Level}_i \subseteq S_\forall$, and let $\alpha \in C_i^+$. By definition of positive certificates, for all valuations $w : X_i \to \{0,1\}$, there exists $\beta \in C_{i+1}^+$ such that $\alpha[w] \subseteq \beta$, and so $\mathsf{Formula}(\alpha[w]) \sqsubseteq \mathsf{Formula}(\beta)$. By induction hypothesis, we know that $\mathsf{Formula}(\beta)$ is true, and by Proposition 3, $\mathsf{Formula}(\alpha[w])$ is also true. Since this holds for all valuations $w : X_i \to \{0,1\}$ and $\mathsf{Q}_i = \forall$, we have that $\mathsf{Formula}(\alpha)$ is true.

Proof (of Lemma 2). We reason by induction on the number of blocks in formula f. Let i, $1 \leq i \leq n$, we show that $\langle(\{\alpha\}, \lceil W_{i+1} \rceil, \ldots, \lceil W_n \rceil), \mathsf{w}\rangle$ is a positive certificate for $\mathsf{Formula}(\alpha)$, for all $\alpha \in \mathsf{Level}_i$ treated by the algorithm and such that $\mathsf{Formula}(\alpha)$ is true. In this certificate, $\lceil W_{i+1} \rceil, \ldots, \lceil W_n \rceil$ are the current antichains when α has just been treated. Notice that at the end of the execution, the last treated node α is the root $\psi \in \mathsf{Level}_1$ and $\lceil W_1 \rceil = \{\alpha\}$.

For the base case $i = n$, we have $\alpha \in \mathsf{Level}_n$ and $\mathsf{Formula}(\alpha) = \exists X_n \cdot \alpha$. If $\mathsf{Formula}(\alpha)$ is true then there exists $w : X_n \to \{0,1\}$ such that $w \models \alpha$. Let $\mathsf{w}(\alpha) = w$ with w being the valuation constructed by Algorithm 1. Clearly $\langle (\{\alpha\}), \mathsf{w} \rangle$ is a positive certificate for $\mathsf{Formula}(\alpha)$.

For the induction case, we assume that the property holds for level $i+1$. Let $\alpha \in \mathsf{Level}_i$ such that $\mathsf{Formula}(\alpha)$ is true. Assume that the first quantifier of $\mathsf{Formula}(\alpha)$ is existential, that is, $\mathsf{Formula}(\alpha) = \exists X_i \cdot \forall X_{i+1} \cdots \exists X_n \cdot \alpha$ (the proof is similar for the universal case). As $\mathsf{Formula}(\alpha)$ is true, there exists a valuation $w : X_i \to \{0,1\}$ such that $\mathsf{Formula}(\alpha[w])$ is true, and in particular there exists a maximal valuation w with this property. Algorithm 1 considers such a maximal valuation w and makes a recursive call to Algorithm 2 on node $\alpha[w]$. By induction hypothesis, the algorithm will construct a positive certificate $\langle (\{\alpha[w]\}, \lceil W_{i+2} \rceil, \ldots, \lceil W_n \rceil), \mathsf{w} \rangle$ for $\mathsf{Formula}(\alpha[w])$. As $\alpha[w]$ is not necessarily a maximal winning node in Level_{i+1}, this node may have been replaced by another winning node β in $\lceil W_{i+1} \rceil$ such that $\alpha[w] \subset \beta$, this is compatible with Property 3 of the definition of positive certificates. So, $\langle (\{\alpha\}, \lceil W_{i+1} \rceil, \ldots, \lceil W_n \rceil), \mathsf{w}' \rangle$ where w' is extending w for α by $\mathsf{w}'(\alpha) = w$, is a positive certificate for $\mathsf{Formula}(\alpha)$.

Negative certificates are defined as follows and they enjoy properties similar to those for positive certificates. A *negative certificate* for a QBF formula $f = \mathsf{Q}_1 X_1 \cdots \mathsf{Q}_n X_n \cdot \psi$ is a pair $\langle (C_i^-)_{1 \leq i \leq n}, \mathsf{w} \rangle$ such that:

- each C_i^- is a set of nodes such that $C_i^- \subseteq \mathsf{Level}_i$
- for each i such that $\mathsf{Level}_i \subseteq S_\forall$, w is a function that assigns a valuation $\mathsf{w}(\alpha) : X_i \to \{0,1\}$ to each $\alpha \in C_i^-$
- and the following properties are verified:
  1. $C_1^- = \{\psi'\}$ with $\psi' \subseteq \psi$
  2. for each $i < n$ such that $\mathsf{Level}_i \subseteq S_\forall$, for all $\alpha \in C_i^-$, there exists $\beta \in C_{i+1}^-$ such that $\alpha[\mathsf{w}(\alpha)] \supseteq \beta$
  3. for each $i < n$ such that $\mathsf{Level}_i \subseteq S_\exists$, for all $\alpha \in C_i^-$, for all $w : X_i \to \{0,1\}$, there exists $\beta \in C_{i+1}^-$ such that $\alpha[w] \supseteq \beta$
  4. for $i = n$, for all $\alpha \in C_i^-$, for all $w : X_i \to \{0,1\}$, $\alpha[w] = 0$.

Example 3. Consider the running example of Fig. 1. As f is false, the following negative certificate $\langle (C_i^-)_{1 \leq i \leq 4}, \mathsf{w} \rangle$ is built by the execution of Algorithms 1 and 2 (see also Example 2):

$$\begin{array}{ll}
C_1^- = \lfloor L_1 \rfloor = \{\psi\}, & \mathsf{w}(\psi) = 101, \\
C_2^- = \lfloor L_2 \rfloor = \{\{2,4,5\}\}, & \\
C_3^- = \lfloor L_3 \rfloor = \{\{2\},\{5\}\}, & \mathsf{w}(\{2\}) = 0, \mathsf{w}(\{5\}) = 0 \text{ or } 1, \\
C_4^- = \lfloor L_4 \rfloor = \{\{2\}\}. &
\end{array}$$

Details concerning Subsection 5.1

Improving $\mathsf{ATCSearch}\exists$. When considering all the maximal valuations $w : X_i \to \{0,1\}$, we observe that player $\mathsf{P}_\exists$ is winning as soon as he can find a valuation w such that $\varphi[w] \subseteq \alpha$ for some $\alpha \in \lceil W_{i+1} \rceil$ (see Corollary 1). So, it is wise for $\mathsf{P}_\exists$ to first try

Algorithm 3 ATCSearchImproved$\exists(\varphi, i)$

```
Require: node $\varphi \in S_\exists \cap \mathsf{Level}_i$, $i \leq n$.
Ensure: Win if $\varphi \in W_i$, Lose if $\varphi \in L_i$.
 1: if $\neg$IsSat($\varphi$) then
 2:     Add($\varphi$, $\lfloor L_i \rfloor$)
 3:     return Lose
 4: if $i = n$ then
 5:     Add($\varphi$, $\lceil W_i \rceil$)
 6:     return Win
 7: ExistWinVal $\leftarrow$ TestWinVal($\varphi$, $\lceil W_{i+1} \rceil$)
 8: if ExistWinVal then
 9:     Add($\varphi$, $\lceil W_i \rceil$)
10:     return Win
11: (ExistVal, $w$) $\leftarrow$ MaxVal($\varphi$, $\lfloor L_{i+1} \rfloor$)
12: while ExistVal do
13:     R $\leftarrow$ ATCSearchImproved$\forall$($\varphi[w]$, $i+1$)
14:     if R = Win then
15:         Add($\varphi$, $\lceil W_i \rceil$)
16:         return Win
17:     (ExistVal, $w$) $\leftarrow$ MaxVal($\varphi$, $\lfloor L_{i+1} \rfloor$)
18: Add($\varphi$, $\lfloor L_i \rfloor$)
19: return Lose
```

to find such a valuation w. This approach is followed by the new algorithm (line 7 in Algorithm 3). Procedure TestWinVal$(\varphi, \lceil W_i \rceil)$ tests, using a SAT solver, if player $\mathsf{P}_\exists$ can choose a valuation w such that $\varphi[w] \subseteq \alpha$ for some $\alpha \in \lceil W_i \rceil$. In that case it returns true, otherwise false.

Now, suppose that player $\mathsf{P}_\exists$ cannot win by exploiting the elements of $\lceil W_{i+1} \rceil$. Then he should avoid considering (maximal) valuations w such that $\alpha \subseteq \varphi[w]$ for some $\alpha \in \lfloor L_{i+1} \rfloor$ as $\varphi[w]$ is losing (see Corollary 1). In this aim, he must ensure to only consider valuations w that satisfy at least one clause of each $\alpha \in \lfloor L_{i+1} \rfloor$. Procedure MaxVal$(\varphi, \lfloor L_{i+1} \rfloor)$ checks in line 11, using a SAT solver, if such a valuation w exists. If yes it returns (true, w), otherwise $(\mathsf{false}, \cdot)$. Assume that a valuation w generated by Procedure MaxVal leads to line 17 of the algorithm, this means that $\varphi[w]$ is losing and has been added to $\lfloor L_{i+1} \rfloor$. Thus a call to MaxVal in line 17 generates a new maximal valuation (if it exists) which is incomparable with w as it avoids $\lfloor L_{i+1} \rfloor$.

The rest of the new algorithm is the same as in Algorithm 1.

Details concerning Subsection 5.2

We present below two examples of improving the propagation of the information about the winning and losing nodes.

At initialization. For example, consider the clause $c_2 = x_2 \vee \bar{y}_4 \vee x_6$ of our running example (see Fig. 1). When projected on the variables $X_{\geq 3} = \{x_6, y_7\}$, this clause reduces to x_6 which is a universally quantified variable. If this clause has not been satisfied by a valuation generated before reaching Level_3, then clearly player $\mathsf{P}_\forall$ has a strategy to falsify it. Therefore, this clause can be added to the antichain $\lfloor L_3 \rfloor$

When updating $\lceil W_i \rceil$ *and* $\lfloor L_i \rfloor$. As an example, consider the first tree in Fig. 4 of our running example. We see that at node $\{1, 7\}$ in Level_4, player $\mathsf{P}_\exists$ has a winning strategy by choosing value 1 for variable y_7. With this choice, clause 1 and clause 7 are satisfied but also clause 4. It means that in Level_4, both $\{1, 7\}$ and $\{1, 4, 7\}$ are winning. So instead of adding $\{1, 7\}$ to $\lceil W_4 \rceil$, we can add $\{1, 4, 7\}$.

Details concerning Section 6

Table 2 presents the experimental results (with a timeout of 600 seconds) obtained for some instances of the families Toilet$*$, aim-$*$ and k-$*$, taken from the seventh QBF solvers evaluation (QBFEVAL'10) [27].

Instance	Var	Cl	Blocs	Value	Nodes	ATC	QuBE	DepQBF	sKizzo
toilet-a-06-01.9	186	1044	3	0	35	2.39	0.03	0.01	0.01
toilet-a-06-04.4	214	1637	3	1	19	2.87	0.02	0.01	0.01
toilet-a-10-05.4	330	12685	3	1	106	191.79	11.53	0.69	0.51
toilet-a-10-05.4	170	11315	3	1	38	24.18	9.48	0.02	0.61
toilet-c-10-01.08	260	1219	3	0	33	3.29	0.01	0.007	0.01
toilet-c-10-01.18	580	2709	3	0	99	411.86	42.13	/	10.87
toilet-c-10-05.4	324	2555	3	1	106	141.08	0.95	0.03	0.02
aim-50-2-0-yes1-1-50	248	473	3	0	2	0.26	0.02	0.005	0.10
aim-50-2-0-yes1-2-90	210	415	3	1	85	2.81	0.02	0.01	0.06
aim-100-3-4-yes1-1-00	738	1657	3	0	2	2.35	0.13	0.03	0.55
aim-100-3-4-yes1-1-90	560	1390	3	1	184	17.37	0.10	0.01	0.14
aim-200-3-4-yes1-2-90	1124	2787	3	0	2	5.50	0.44	0.12	4.18
aim-200-3-4-yes1-4-50	1276	3013	3	1	167	122.61	0.44	0.07	2.41
k-d4-p-03	215	587	15	0	378	12.45	0.05	0.19	0.01
k-d4-p-13	815	2417	35	0	3303	299.64	/	/	0.11
k-dum-n-03	214	522	17	1	312	6.36	0.01	0.23	0.02
k-dum-n-12	620	1594	35	1	598	15.72	/	231.06	0.06
k-dum-p-04	215	556	15	0	506	12.27	0.01	0.12	0.01
k-dum-p-15	641	1666	33	0	1645	160.31	4.96	/	0.07
k-poly-n-10	816	1841	65	1	312	24.64	0.03	/	0.42
k-poly-p-10	843	1902	67	0	437	40.11	0.02	/	0.31

Table 2. Families Toilet$*$, aim-$*$ and k-$*$.

Table 3 presents the results for the family k-path-n. The curves for both families k-path-n (true instances) and k-path-p (false instances) are given in Fig. 9. They show that our solver performs better than the three state-of-the-art solvers.

The experimental results for other families in Toilet$*$, aim-$*$ and k-$*$ are given in Figs. 10-19. The execution times of our solver follow the same shape of curve (at logarithmic scale, with a timeout of 600 seconds) as for the other three state-of-the-art solvers. For families in k-$*$, QuBE and DepQBF solvers (which are search-based) have often an execution time beyond 600 seconds.

Instance	Var	Cl	Blocs	Value	Nodes	ATC	QuBE	DepQBF	sKizzo
k-path-n-01	108	275	7	1	14	0.22	0.01	0.005	0.01
k-path-n-02	180	481	9	1	43	0.93	0.01	0.02	0.05
k-path-n-03	252	682	11	1	79	2.65	0.04	0.27	0.027
k-path-n-04	324	888	13	1	103	3.30	0.13	3.59	0.04
k-path-n-05	384	1051	15	1	120	4.17	2.53	39.97	0.08
k-path-n-06	456	1257	17	1	142	7.54	13.74	/	0.26
k-path-n-07	528	1458	19	1	180	8.95	/	/	0.72
k-path-n-08	600	1664	21	1	226	15.17	/	/	3.17
k-path-n-09	660	1827	23	1	195	11.85	/	/	12.66
k-path-n-10	732	2033	25	1	247	16.33	/	/	45.87
k-path-n-11	804	2234	27	1	269	27.36	/	/	445.71
k-path-n-12	876	2440	29	1	322	44.62	/	/	/
k-path-n-13	936	2603	31	1	314	44.59	/	/	/
k-path-n-14	1008	2809	33	1	361	37.98	/	/	/
k-path-n-15	1080	3010	35	1	363	51.30	/	/	/
k-path-n-16	1152	3216	37	1	427	63.58	/	/	/
k-path-n-17	1212	3379	39	1	409	58.11	/	/	/
k-path-n-18	1284	3585	41	1	460	64.24	/	/	/
k-path-n-19	1356	1786	43	1	461	128.84	/	/	/
k-path-n-20	1428	3992	45	1	503	95.19	/	/	/
k-path-n-21	1488	4155	47	1	452	88.81	/	/	/

Table 3. Family k-path-n.

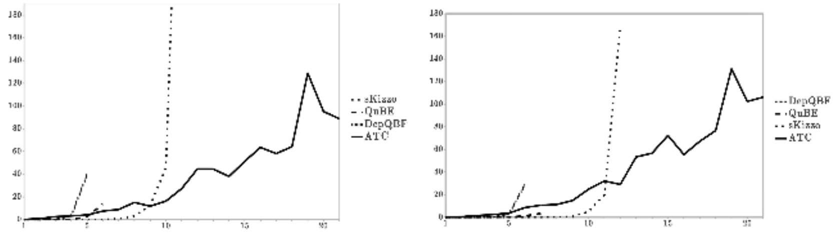

Fig. 9. Family k-path-∗, true and false instances

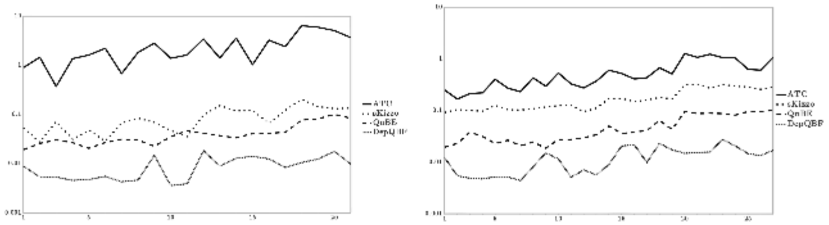

Fig. 10. Family aim-50, true and false instances, log scale

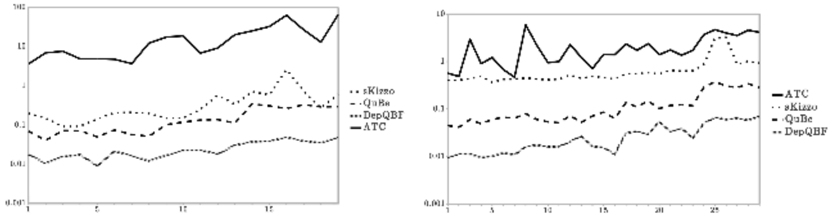

Fig. 11. Family aim-100, true and false instances, log scale

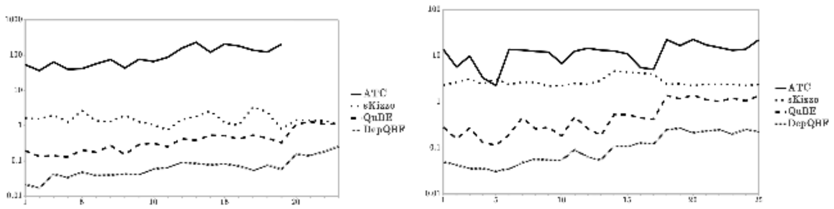

Fig. 12. Family aim-200, true and false instances, log scale

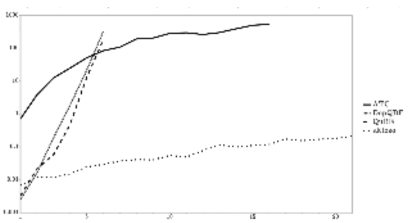

Fig. 13. Family k-d4-p, false instances, log scale

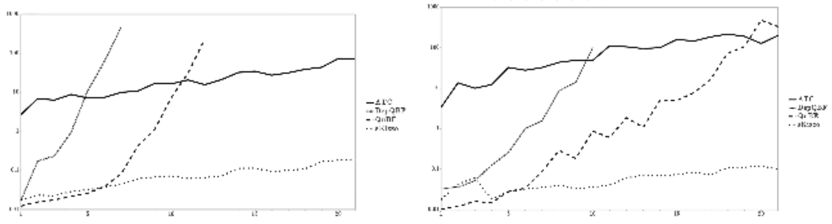

Fig. 14. Family k-dum-∗, true and false instances, log scale

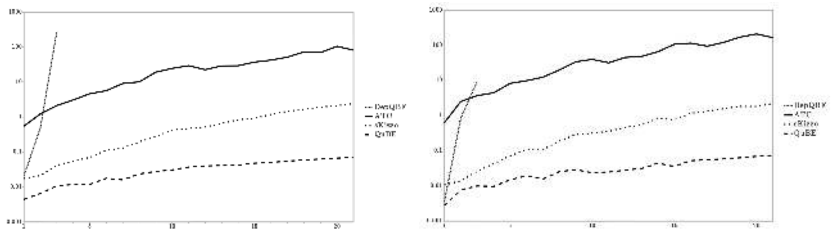

Fig. 15. Family k-poly-∗, true and false instances, log scale

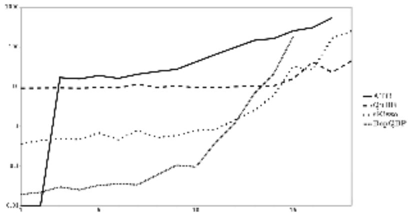

Fig. 16. Family toilet-a-10-01, false instances, log scale

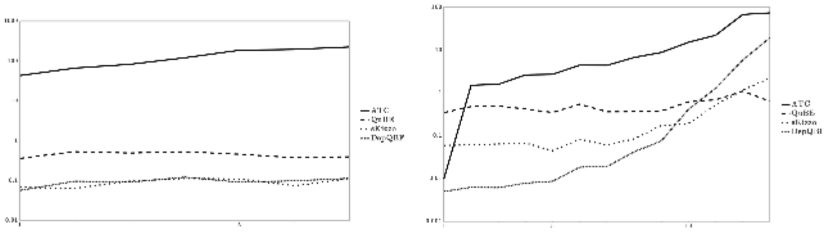

Fig. 17. Family toilet-a-8-∗, true and false instances, log scale

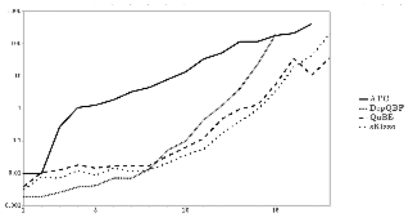

Fig. 18. Family toilet-c-10-01, false instances, log scale

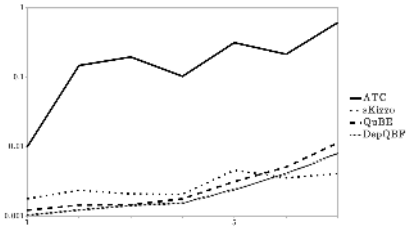

Fig. 19. Family toilet-g, true instances, log scale