Improvements to the Evaluation of Quantified Boolean Formulae

Jussi Rintanen
Universität Ulm, Fakultät für Informatik
Albert-Einstein-Allee, 89069 Ulm
Germany

Abstract

We present a theorem-prover for quantified Boolean formulae and evaluate it on random quantified formulae and formulae that represent problems from automated planning. Even though the notion of quantified Boolean formula is theoretically important, automated reasoning with QBF has not been thoroughly investigated. Universal quantifiers are needed in representing many computational problems that cannot be easily translated to the propositional logic and solved by satisfiability algorithms. Therefore efficient reasoning with QBF is important. The Davis-Putnam procedure can be extended to evaluate quantified Boolean formulae. A straightforward algorithm of this kind is not very efficient. We identify universal quantifiers as the main area where improvements to the basic algorithm can be made. We present a number of techniques for reducing the amount of search that is needed, and evaluate their effectiveness by running the algorithm on a collection of formulae obtained from planning and generated randomly. For the structured problems we consider, the techniques lead to a dramatic speed-up.

1 Introduction

Many computational problems can be conveniently formulated in classical propositional logic. Examples of such are many constraint satisfaction problems, forms of planning, and many problems in graph-theory. A common property of these problems is that they belong to the complexity class NP and that there are therefore polynomial-time translations from them to the satisfiability of formulae in the classical propositional logic.

However, there are many important problems that are (under plausible complexity-theoretic assumptions) outside the complexity class NP, and therefore cannot be in general efficiently translated to the problem of satisfiability of propositional formulae. Many such problems are PSPACE-hard or harder and can be forced to NP only by making restrictions that make the problems lose most of their practically interesting aspects. There are, however, a number of problems that belong to the classes in the polynomial hierarchy, many of them to its lower levels. These include determining the truth-value of quantified Boolean formulae [Meyer and Stockmeyer, 1972], computing the radius of a covering code [McLoughlin, 1984], many decision problems in nonmonotonic logics [Gottlob, 1992], some forms of abduction [Eiter and Gottlob, 1995], and determining the Vapnik-Červonenkis dimension in probability theory [Schäfer, 1996].

Like satisfiability algorithms can be used for solving problems in NP, a similar approach is applicable to problems higher in the polynomial hierarchy: implement an efficient decision procedure for one problem, and give good polynomial-time translations from other problems to that problem. The use of QBF this way has not been investigated partly because there have been no efficient decision procedures. Translation to QBF has recently been proposed as a solution method for conditional planning [Rintanen, 1999].

In this work we present an algorithm for determining the truth of quantified Boolean formulae, and evaluate it with randomly generated quantified Boolean formulae and formulae from conditional planning.

2 Preliminaries

Quantified Boolean formulae are of the form $q_1 x_1 \cdots q_n x_n \phi$ where ϕ is a propositional formula and the prefix consists of universal $\forall$ and existential $\exists$ quantifiers q_i and the propositional variables x_i occurring in ϕ. Define $\phi[\psi/x]$ as the formula obtained from ϕ by replacing occurrences[1] of the propositional variable x by the formula ψ. The truth of formulae is defined recursively as follows. The truth of a formula that does not contain variables, that is, that consists of connectives and the constants true $\top$ and false $\bot$, is defined by the truth-tables for the connectives. A formula $\exists x \phi$ is true if and only if $\phi[\top/x]$ or $\phi[\bot/x]$ is true. A formula $\forall x \phi$ is true if and only if $\phi[\top/x]$ and $\phi[\bot/x]$ are true. Examples of true formulae are $\forall x \exists y (x \leftrightarrow y)$ and $\exists x \exists y (x \wedge y)$. The formulae $\exists x \forall y (x \leftrightarrow y)$ and $\forall x \forall y (x \vee y)$ are false.

[1]We assume that nested quantifiers do not quantify the same variable.

```
PROCEDURE decide($e, \langle M_1, M_2, \ldots, M_n\rangle, C$)
BEGIN
    $C$ := unit($C$);
    IF $\emptyset \in C$ THEN RETURN false;
    IF $n = 0$ THEN RETURN true;
    remove from $M_1$ all variables not in $C$;
    IF $M_1 = \emptyset$ THEN
        RETURN decide(not $e, \langle M_2, \ldots, M_n\rangle, C$);
    $x$ := a member of $M_1$;
    $M_1 := M_1 \backslash \{x\}$;
    IF $e$ THEN
        IF decide($e, \langle M_1, \ldots, M_n\rangle, C \cup \{\{x\}\}$)
        THEN RETURN true;
    ELSE
        IF not decide($e, \langle M_1, \ldots, M_n\rangle, C \cup \{\{x\}\}$)
        THEN RETURN false;
    RETURN decide($e, \langle M_1, \ldots, M_n\rangle, C \cup \{\{\neg x\}\}$)
END
```

Figure 1: The algorithm

3 The Basis Algorithm

We have designed and implemented an algorithm that determines the truth-value of quantified Boolean formulae. The Davis-Putnam procedure [Davis *et al.*, 1962] is a special case of the algorithm. The main differences are that instead of only or-nodes, the search tree for quantified Boolean formulae contains also and-nodes that correspond to universally quantified variables, and that the order of the variables in the prefix constrains the order in which the variables generate a search tree.

The main procedure of the algorithm sketched in Figure 1 takes three parameters.[2] The variable e is true if the first quantifier in the prefix of the formula is $\exists$. The sequence $\langle M_1, \ldots, M_n\rangle$ represents the prefix. For example, if the prefix is $\exists x_1 \exists x_2 \forall x_3 \exists x_4$, then $M_1 = \{x_1, x_2\}, M_2 = \{x_3\}$ and $M_3 = \{x_4\}$. The set C consists of clauses $\{l_1, \ldots, l_n\}$ where $n \geq 0$ and l_i are literals. The empty clause $\emptyset$ is false.

The subprocedure *unit* performs simplification by unit resolution and unit subsumption; unit(S) is defined as the fixpoint of F under a set S of clauses.

$$\begin{array}{rcl} F(C) & = & \{c \backslash \{l\} | c \in C, \{\bar{l}\} \in C, l \in c\} \\ & \cup & \{c \in C | l \notin c \text{ and } \bar{l} \notin c \text{ for all } \{l\} \in C\} \\ & \cup & \{\{l\} \in C\} \end{array}$$

So for the purposes of this paper, our definition of unit subsumption retains unit clauses in the clause set.

4 Pruning Techniques

An important difference between the Davis-Putnam procedure and the algorithm for QBF is that the order in which branching variables are chosen is dependent on their location in the prefix. The branching variable is always quantified by the outermost quantifier. In structured problems it is often the case that many of the first variables occur only in clauses that also contain variables quantified by inner quantifiers, and an almost exhaustive search through all truth-values for the variables may be needed because unit propagation does not yield the truth-values. Already for a small number of variables this can be prohibitively expensive. For example in a formula $\exists X \forall Y \exists Z \Phi$, if $|X| = 20$, the variables in X induce a search tree with 2^{20} leaf nodes, with a potentially difficult QBF to evaluate in each.

[2]The algorithm is simplified because we just want to indicate what the main differences from the Davis-Putnam procedure are. For example, we do not require that the variable x does not have a truth-value when it is branched on.

```
let $A_1, \ldots, A_n$ be consistent subsets of $Y \cup \{\neg y | y \in Y\}$;
FOR $i := 1$ TO $n$ DO
    $C'$ := unit($C \cup U(A_i)$);
    $R := C' \cap U(X \cup \{\neg x | x \in X\})$;
    IF $\emptyset \notin C'$ THEN $C$ := unit($C \cup R$);
    FOR EACH $p \in X$ DO
        $C''$ := unit($C' \cup \{\{\neg p\}\}$);
        IF $\emptyset \in C''$ THEN $C$ := unit($C \cup \{\{p\}\}$);
        IF $\emptyset \in C$ THEN RETURN false;
        (* Similarly for $p$ *)
        $C''$ := unit($C' \cup \{\{p\}\}$);
        IF $\emptyset \in C''$ THEN $C$ := unit($C \cup \{\{\neg p\}\}$);
        IF $\emptyset \in C$ THEN RETURN false;
    END
END
```

Figure 2: Inversion of quantifiers

In this section we first propose two pruning techniques that are based on reasoning with variables that are *not* quantified by the current outermost quantifier. The third technique is an extension of the use of unit propagation for detecting failed literals, and the fourth reduces the computation needed in going through all valuations of universally quantified variables.

4.1 Inverting Quantifiers

Given $\exists X \forall Y \Phi$, it is useful to look also at the formula $\forall Y \exists X \Phi$.[3] If for some valuation of variables in Y only certain valuations of X are possible, these valuations are the only possible ones also for the formula $\exists X \forall Y \Phi$.

The technique we propose is based on the above idea, and we therefore call it the inversion of the quantifiers. We randomly assign some truth-values to variables in Y (it may be possible and useful to try out all possible valuations of Y), and then try to detect failed literals with unit propagation [Li and Anbulagan, 1997]. Any truth-values obtained for variables in X are those that must be chosen when evaluating $\exists X \forall Y \Phi$. Inversion is only applicable to formulae with $\exists$ as the first quantifier. The program code in Figure 2 outlines how quantifier inversion is performed. This code is run before the first call to the main procedure is made. The function $U(L) = \{\{l\} | l \in L\}$ produces a set of unit clauses from a set of literals.

[3]The formula Φ may contain quantifiers.

```
FOR EACH $p \in X$ such that $\{\{p\},\{\neg p\}\} \cap C = \emptyset$
  FOR i := 1 TO numberOfSamples DO
    $R$ := random consistent subset of $Y \cup \{\neg y | y \in Y\}$;
    $C'$ := unit($C \cup U(\{p\} \cup R)$);
    IF $\emptyset \in C'$ THEN $C$ := unit($C \cup \{\{\neg p\}\}$);
    IF $\emptyset \in C$ THEN RETURN false;
    (* Similarly for $\neg p$ *)
    $R$ := random consistent subset of $Y \cup \{\neg y | y \in Y\}$;
    $C'$ := unit($C \cup U(\{\neg p\} \cup R)$);
    IF $\emptyset \in C'$ THEN $C$ := unit($C \cup \{\{p\}\}$);
    IF $\emptyset \in C$ THEN RETURN false;
  END
END
```

Figure 3: Sampling

4.2 Sampling

When choosing a truth-value for a variable $x \in X$ in a formula $\exists X \forall Y \Phi$, it would be useful to check that the choice is possible under all valuations of variables in Y. This is, however, expensive. Nevertheless, it has turned out that trying out at least some valuations of Y and performing unit propagation helps in the detection of wrong choices. The valuations of a small number of variables in Y are chosen randomly. We call this sampling of Y. This technique is similar to the first one, but because it is performed for many variables in every node of the search tree, valuations of Y cannot be covered very exhaustively.

Sampling is best combined with the use of unit propagation in choosing branching variables and detecting failed literals, but here we describe it separately. The program code in Figure 3 outlines how sampling is performed. The code is inserted right after the first call to the function *unit* in the main procedure.

4.3 Failed Literal Detection

The third technique is the use of unit propagation for detecting failed literals [Li and Anbulagan, 1997] also for variables quantified by inner quantifiers. This proceeds by adding a literal to the clause set, and then performing unit propagation. If an empty clause is obtained and the literal is existential, the complement of the literal must be true and it is added to the clause set. If the literal is universal, the formula is false.

In the quantified case also the occurrence of a clause with universal variables only may yield a failed literal. Here the ordering of variables in the prefix has to be observed. Consider the formula $\forall x \exists y (x \leftrightarrow y)$ which is in clausal form $\forall x \exists y ((x \vee \neg y) \wedge (\neg x \vee y))$. If we try to see whether y is a failed literal by adding the literal y to the clause set and performing unit propagation, we get the unit clause x with a universal variable. This would seem to indicate that y should be assigned false. However, the value of y is a function of the value of x, and y may assume different values for different values of x.

Lemma 1 *Let $F = \exists X_1 \forall Y_1 \exists X_2 \forall Y_2 \cdots \Phi$ be a QBF. Assume $\Phi \cup \{l\} \models l_1 \vee \ldots \vee l_n$ such that the propositional variables $p, p_1, \ldots, p_n$ occur respectively in literals $l, l_1, \ldots, l_n$. Then F is false if $\{p_1, \ldots, p_n\} \subseteq Y_j \cup Y_{j+1} \cup \cdots$ for some j and $p \in X_k$ for some $k \leq j$.*

4.4 Splitting Clause Sets

Apart from the above techniques specific to QBF, we maintain a graph describing the dependencies between variables: there is an edge between two variables if they occur in the same clause. In each node of the search tree, for separate connected components separate recursive calls to the main procedure of the algorithm are made. This has a noticeable effect on the efficiency of the algorithm on certain random problems as well as on some structured problems. For example, for a true $\forall\exists$ QBF not all combinations of truth-values for universally quantified variables have to be considered if the variables occupy different connected components. This is often the case for random QBF with a low clauses/variables ratio. Consider a formula with n universal variables. Normally a tree with 2^n leaves has to be traversed, but if the clause set is split so that universal variables are evenly in m components, m search trees with only $2^{n/m}$ leaves have to be traversed. For example, for $n = 20$ and $m = 2$ this means a decrease from about 10^6 to 2000.

5 Experiments

We have tested the implementation of our algorithm and the new techniques on a number of problems from conditional planning and on a number of random quantified Boolean formulae. Our Davis-Putnam implementation that we have extended to handle QBF and on top of which all the new techniques have been implemented, is closest to Li and Anbulagan's [1997] Sat0, but slower.

The basis of the generation of random quantified Boolean formulae is the fixed clause length model [Mitchell *et al.*, 1992] for generation of random propositional formulae, in which 3-literal clauses are generated by randomly choosing three variables from a set of N variables and negating each with probability 0.5. The presence of clauses with less than two existential variables asymptotically causes all formulae to be false when the number of variables increases [Gent and Walsh, 1998]. This is because the probability of a clause with universal literals only or two clauses with complementary existential literals and both with two universal literals becomes very high. Hence for the generation of random quantified formulae we use model A of Gent and Walsh [1998] in which no clauses with less than two existential literals are generated. In our test runs $N = 50$ and the QBF have the prefix $\exists\forall\exists$. Effect of the techniques on formulae with prefixes $\forall\exists$ and $\forall\exists\forall\exists$ is similar. We ran our program by varying two parameters, the proportion of universal variables in the prefix and the number of clauses. The proportion of universal variables was increased by 3 per cent steps and the clauses/variables ratio by steps of 0.2. If we have 100 variables and 50 per cent of them are universal, the first quantifier quantifies the variables 1–25, the second quantifier the variables 26–75, and the third quantifier the variables 76-100. We

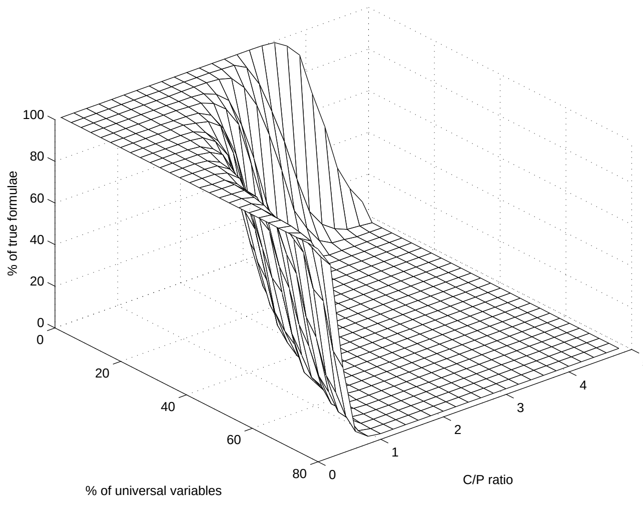

Figure 4: Truth of random ∃∀∃ formulae

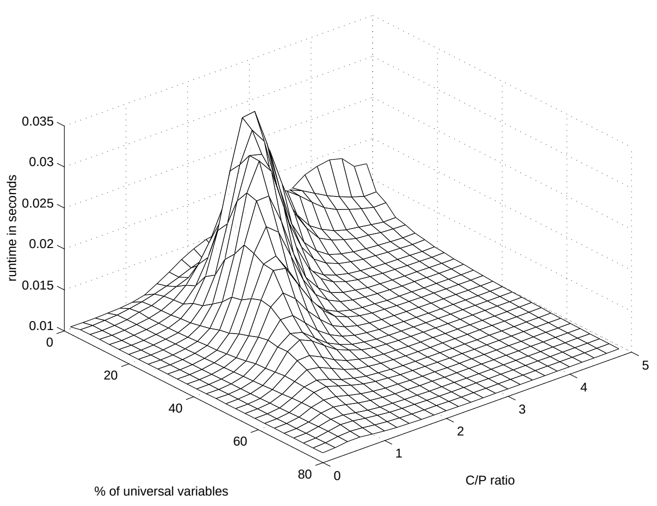

Figure 6: Runtimes on random formulae without S4.3

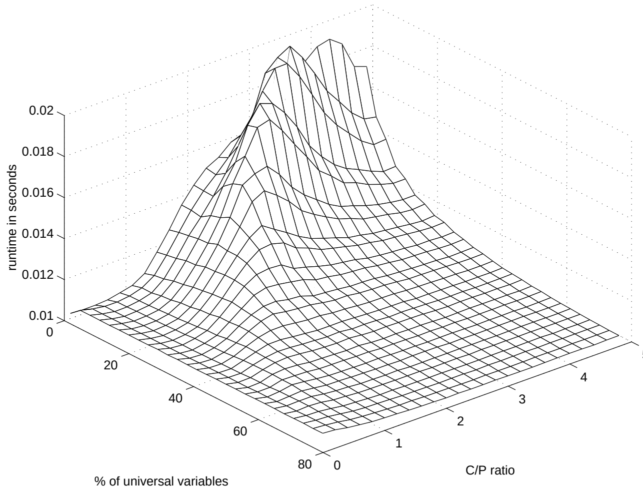

Figure 5: Runtimes on random formulae

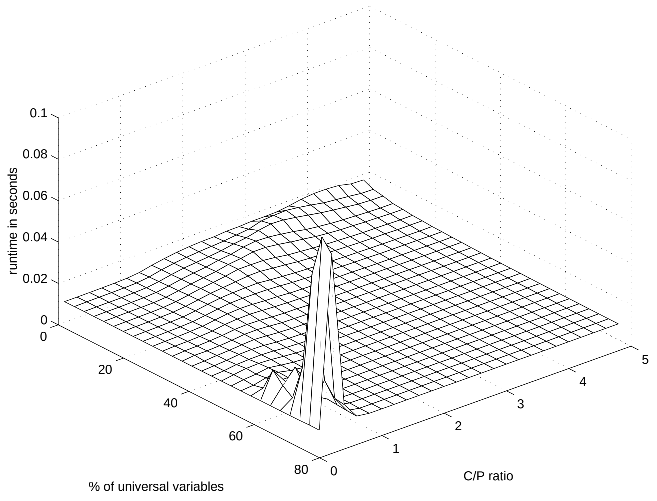

Figure 7: Runtimes on random formulae without S4.4

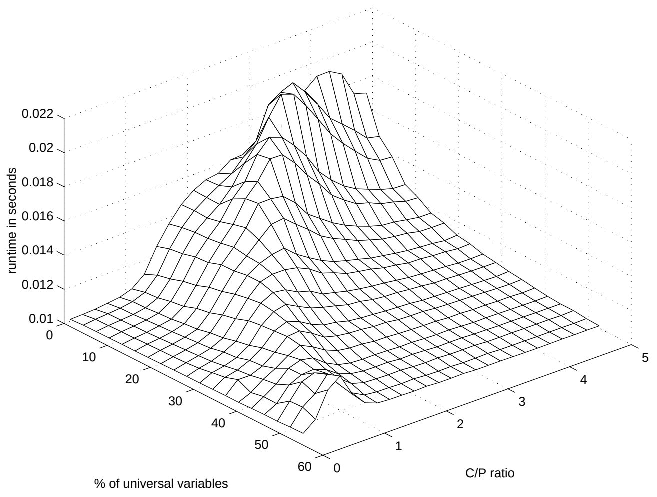

Figure 8: Runtimes on random formulae without S4.4

have observed that (in the presence of the other techniques) both inversion and sampling – the techniques that turn out to be most effective for the structured problems we consider – affect very little the number of nodes in search trees. Therefore the test runs were with these techniques disabled.

The results from the test runs are depicted in Figures 4, 5, 6, 7 and 8. The runtime for every combination of parameter values is the average of runtimes for 60000 formulae. All the runs were on a Sun Ultra II workstation with a 296 MHz processor. The execution times are not very accurate because the smallest measurable unit of CPU time was 10 milliseconds. Figure 4 depicts the proportion of true formulae for all parameter values considered. When increasing the proportion of universal variables, the region of formulae that are more difficult to evaluate shifts to the direction of lower clauses/variables ratios. The phase transition region from true to false shifts similarly. Figure 5 shows the runtimes with full detection of failed literals (Section 4.3) and partitioning of clause sets (Section 4.4.) Figure 6 shows runtimes without full detection of failed literals and Figures 7 and 8 runtimes without partitioning. The peak in Figure 7 is due to a very small number of difficult formulae. As is obvious from a comparison between Figures 5 and 8, the effect of partitioning on formulae with few universal variables is small.

The diagrams show that detection of failed literals for all variables is useful, although the effect is not dramatic. Partitioning of clause sets reduces the amount of computation dramatically for a small number of formulae with many universal variables and a small clauses/variables ratio. This is because partitioning produces several small clause sets with disjoint variables, and hence considering all 2^n valuations of n universal variables takes much less than $O(2^n)$ time.

The structured problems we consider are translations from conditional planning to QBF [Rintanen, 1999]. The most basic form of planning is the identification of sequences of operations that reach a goal state from a given initial state. That problem has been successfully solved by algorithms that test propositional satisfiability [Kautz and Selman, 1996]. In conditional planning, plans that work under several different circumstances are constructed. A conditional plan is essentially a program in a simple programming language with conditional statements that chooses at execution time which operations to apply. When there is only one initial state, planning can be represented by formulae $\exists p_1 \cdots p_n \exists e_1 \cdots e_m \Phi$ that say that there is a plan (represented by the variables p_i) such that its execution (represented by the variables e_i) produces a goal state. Conditional planning can be represented by formulae $\exists p_1 \cdots p_n \forall c_1 \cdots c_p \exists e_1 \cdots e_m \Phi$ where variables c_i represent the different circumstances the plan has to work in.

Runtimes for formulae from conditional planning are given in Table 1. These formulae encode the problem of existence of conditional plans for taking three or four blocks from all possible initial configurations to a unique goal configuration. We have evaluated the effect of each technique separately. The first column identifies the formulae in question. The second column gives the number of propositional variables in each QBF, and the third the number of clauses. The fourth column gives the truth-value of the formula. The fifth gives the runtime of our theorem-prover with all techniques enabled. The sixth gives the runtime with inversion disabled. The seventh with sampling disabled. The eighth with detection of failed literals for inner quantifiers disabled. The ninth without partitioning of clause sets. For sampling the number of samples was 10 with at most 16 variables in each sample (the formulae all have less than 16 universal variables), and for inversion the number of valuations considered was at most 100.

In some cases disabling one of the pruning techniques leads to a better runtime, but these are the easier formulae and the differences are relatively small. The only technique that in most cases does not have any effect is the partitioning of clause sets. Only for three formulae there is a significant decrease in the runtimes, from 14.6 to 1.7, from 165.1 to 76.1 and from 13802.7 to 192.5. We would expect that on many structured problems the decrease is small. When none of the techniques is used, none of the runs end in 4 hours. Sampling and inversion are the most effective techniques: without them four of the formulae could not be evaluated in 4 hours.

6 Related work

Automated reasoning with quantified Boolean formulae has been investigated by Kleine Büning et al. [1995] who define a resolution rule for quantified Boolean formulae and a polynomial time decision procedure for quantified Horn clauses. Aspvall et al. [1979] give a polynomial time decision procedure for quantified 2-literal clauses.

Cadoli et al. [1998] extend the Davis-Putnam procedure to handle quantified Boolean formulae. Their algorithm is similar to the one in Section 3. Cadoli et al. generalize techniques familiar from the Davis-Putnam procedure to QBF. For example, they introduce the pure literal rule for universal variables and a rule that concludes that formulae with clauses without existential variables are false. A technique not employed by us is testing whether the set of clauses with all universal literals deleted is satisfiable. If it is, the formula is true. Cadoli et al. perform this test in every node of the search tree. It seems to us that on random problems with many universal variables and a low clauses/variables ratio it may be beneficial like the technique we present in Section 4.4. The algorithm by Cadoli et al. does not evaluate any of the formulae in Table 1 in 16 hours. We also ran an experiment on a small set of random $\exists\forall\exists$ formulae with 150 variables and parameters varying like in the experiments reported in this paper. A small number of the formulae were much more difficult for the program by Cadoli et al. and some of the very easy formulae were evaluated faster. The sum of the runtimes for our program was 85.7 seconds versus 4965.28 seconds for the Cadoli et al. program. For a bigger set of random formulae with 100 variables the runtimes were 25.36 seconds versus 410.79 seconds.

7 Conclusions and Future Work

Future work includes a systematic study of computational properties of QBF arising from applications, including AI planning, automated theorem-proving and computer-aided verification, and then improving the QBF algorithm accordingly. There are several degrees of freedom in the use of the new techniques, and only by analyzing a wider range of problems it is possible to determine what is the best way of using them.

As an important research topic we see the reduction of exhaustive search needed in handling universal quantifiers. The technique presented in this paper is applicable only in relatively simple cases.

formula	vars	clauses	value	all	no S4.1	no S4.2	no S4.3	no S4.4
BLOCKS3i.4.4	288	2928	F	0.6	0.7	0.2	1.3	0.6
BLOCKS3i.5.3	286	2892	F	334.2	> 14400	> 14400	1045.6	331.9
BLOCKS3i.5.4	328	3852	T	76.1	> 14400	> 14400	302.3	165.1
BLOCKS3ii.4.3	247	2533	F	0.3	0.3	0.2	0.5	0.3
BLOCKS3ii.5.2	282	2707	F	1.0	1.2	0.6	1.1	1.0
BLOCKS3ii.5.3	304	3402	T	3.1	35.1	11.7	8.5	5.0
BLOCKS3iii.4	202	1433	F	0.2	0.2	0.1	0.2	0.2
BLOCKS3iii.5	256	1835	T	1.7	4.7	7.1	2.4	14.6
BLOCKS4i.6.3	779	15872	F	68.3	56.1	37.4	135.1	67.7
BLOCKS4i.7.2	783	15219	?	> 14400	> 14400	> 14400	> 14400	> 14400
BLOCKS4i.7.3	863	18768	?	> 14400	> 14400	> 14400	> 14400	> 14400
BLOCKS4ii.6.3	838	15061	F	13.6	14.0	42.3	106.5	14.1
BLOCKS4ii.7.2	915	15047	F	85.5	111.6	80.8	174.7	85.5
BLOCKS4ii.7.3	969	17944	T	241.3	> 14400	> 14400	596.9	353.2
BLOCKS4iii.6	727	9661	F	8.8	8.9	21.3	80.7	8.9
BLOCKS4iii.7	855	11303	T	192.5	> 14400	> 14400	346.6	13802.7

Table 1: Runtimes of our program on formulae from planning (in seconds)

Acknowledgements

This research was funded by the Deutsche Forschungsgemeinschaft through the SFB 527. We thank the referees for comments on the generation of random QBF and Marco Cadoli et al. for access to their program.

References

[Aspvall *et al.*, 1979] Bengt Aspvall, Michael F. Plass, and Robert Endre Tarjan. A linear time algorithm for testing the truth of certain quantified Boolean formulas. *Information Processing Letters*, 8(3):121–123, 1979. Erratum in 14(4):195, June 1982.

[Cadoli *et al.*, 1998] M. Cadoli, A. Giovanardi, and M. Schaerf. An algorithm to evaluate quantified Boolean formulae. In *Proceedings of the 15th National Conference on Artificial Intelligence (AAAI-98) and the 10th Conference on Innovative Applications of Artificial Intelligence (IAAI-98)*, pages 262–267. AAAI Press, July 1998.

[Davis *et al.*, 1962] M. Davis, G. Logemann, and D. Loveland. A machine program for theorem proving. *Communications of the ACM*, 5:394–397, 1962.

[Eiter and Gottlob, 1995] Thomas Eiter and Georg Gottlob. The complexity of logic-based abduction. *Journal of the ACM*, 42(1):3–42, 1995.

[Gent and Walsh, 1998] Ian Gent and Toby Walsh. Beyond NP: the QSAT phase transition. Technical Report APES-05-1998, University of Strathclyde, Department of Computer Science, July 1998.

[Gottlob, 1992] Georg Gottlob. Complexity results for nonmonotonic logics. *Journal of Logic and Computation*, 2(3):397–425, June 1992.

[Kautz and Selman, 1996] Henry Kautz and Bart Selman. Pushing the envelope: planning, propositional logic, and stochastic search. In *Proceedings of the 13th National Conference on Artificial Intelligence and the 8th Innovative Applications of Artificial Intelligence Conference*, pages 1194–1201. AAAI Press, August 1996.

[Kleine Büning *et al.*, 1995] Hans Kleine Büning, Marek Karpinski, and Andreas Flögel. Resolution for quantified Boolean formulas. *Information and Computation*, 117:12–18, 1995.

[Li and Anbulagan, 1997] Chu Min Li and Anbulagan. Heuristics based on unit propagation for satisfiability problems. In Martha Pollack, editor, *Proceedings of the 15th International Joint Conference on Artificial Intelligence*, pages 366–371. Morgan Kaufmann Publishers, August 1997.

[McLoughlin, 1984] A. McLoughlin. Complexity of computing the covering radius of a code. *IEEE Transactions on Information Theory*, 30(6):800–804, 1984.

[Meyer and Stockmeyer, 1972] A. R. Meyer and L. J. Stockmeyer. The equivalence problem for regular expressions with squaring requires exponential time. In *Proceedings of the 13th Annual Symposium on Switching and Automata Theory*, pages 125–129. IEEE Computer Society, 1972.

[Mitchell *et al.*, 1992] David Mitchell, Bart Selman, and Hector Levesque. Hard and easy distributions of SAT problems. In William Swartout, editor, *Proceedings of the 10th National Conference on Artificial Intelligence*, pages 459–465, San Jose, California, July 1992. The MIT Press.

[Rintanen, 1999] Jussi Rintanen. Constructing conditional plans by a theorem-prover. *Journal of Artificial Intelligence Research*, 10:323–352, 1999.

[Schäfer, 1996] Marcus Schäfer. Deciding the Vapnik-Červonenkis dimension is Σ_3^P-complete. In *Proceedings of the 11th Conference on Computational Complexity*, pages 77–80, 1996.