Computing Stable Models with Quantified Boolean Formulas: Some Experimental Results*

Uwe Egly, Thomas Eiter, Volker Klotz, Hans Tompits, and Stefan Woltran

Technische Universität Wien,
Institut für Informationssysteme,
Abt. Wissensbasierte Systeme 184/3,
Favoritenstraße 9–11, A–1040 Wien, Austria
e-mail: [uwe,eiter,volker,tompits,stefan]@kr.tuwien.ac.at

Abstract

Quantified boolean formulas (QBFs) are extensions of ordinary propositional formulas which admit efficient representations of many important reasoning tasks. The existence of sophisticated QBF-solvers makes it possible to realize prototype systems for quite different knowledge-representation formalisms in a uniform manner. The system QUIP follows this idea and implements inference tasks from the area of nonmonotonic reasoning by using suitable encodings to QBFs. In this paper, we report experimental results evaluating the performance of QUIP. In particular, we deal here with the disjunctive logic programming module of QUIP, which will be the subject of two kinds of performance tests: First, we compare QUIP with the state-of-the-art logic programming systems `dlv` and `smodels`, and second, we examine the performance of different QBF-solvers on the considered problem classes. As benchmark philosophy we employ classes of disjunctive logic programs which are responsible for the Σ_2^p-hardness of the given decision problems. The results show reasonable performance of the QBF approach and indicate possible improvements of QUIP by exploiting different QBF-solvers as underlying inference engines.

Introduction

In recent years, the area of *answer set programming* has gained a significant amount of popularity. For the most part, this is a direct result of the high level of sophistication which some of the currently available logic programming systems enjoy. Systems like `dlv` (Eiter *et al.* 1997) and `smodels` (Niemelä & Simons 1997) are recognized as the factual state of the art implementing the answer set paradigm. In particular, `dlv` has been conceived as an implementation for the *disjunctive stable model semantics*, i.e., allowing rules with disjunctions in the head, whilst `smodels` originally started as a system for non-disjunctive rules, but has recently been extended to support disjunctions as well (Janhunen *et al.* 2000).

In contrast to these highly specialized inference tools, the system QUIP (Egly *et al.* 2000a; 2000b) is based on a more general philosophy. To wit, the basic idea of QUIP is to provide a *uniform implementation for a wide array of advanced reasoning tasks* by exploiting reductions to so-called *quantified boolean formulas*. By a quantified boolean formula (or "QBF" for short) one understands a term which is constructed like an ordinary propositional formula, except that quantifiers ranging over propositional variables may also occur. The general mechanism of QUIP is to translate a given query into a QBF and then using a sophisticated QBF-evaluator to compute the models of the resultant formula. The rationale behind this approach is given by the fact that the evaluation problem of QBFs—i.e., the problem of determining the truth of a given QBF—is PSPACE-complete, whilst the evaluation problem of QBFs having prenex normal form with i alternating quantifiers is complete for the i-th level of the polynomial hierarchy (Stockmeyer 1976). More precisely, the latter problems are Σ_i^p-complete if the outermost quantifier is existential, and they are Π_i^p-complete if the outermost quantifier is universal (we will refer to these problems as $\mathrm{QSAT}_{i,\exists}$ and $\mathrm{QSAT}_{i,\forall}$, respectively). From these properties it follows immediately that any decision problem which is known to be in PSPACE can be polynomially reduced to some QBF, and any task in Σ_i^p or Π_i^p can likewise be reduced to $\mathrm{QSAT}_{i,\exists}$ or $\mathrm{QSAT}_{i,\forall}$, respectively. (The reader may recall that both Σ_i^p and Π_i^p are subclasses of PSPACE; as well, it holds that $\Sigma_1^p = \mathrm{NP}$ and $\Pi_1^p = \text{co-NP}$.)

Of particular relevance for the development of QUIP were decision problems belonging to the second level of the polynomial hierarchy, like most decision problems from the area of nonmonotonic reasoning. In fact, the current version of QUIP realizes reasoning tasks from logic-based abduction, autoepistemic logic, default logic, stable model semantics for disjunctive logic programs, and circumscription.

In this paper, we deal with the disjunctive logic programming module of QUIP. More specifically, we are interested in an experimental analysis addressing the following two issues: (i) how does the reduction mechanism employed in QUIP perform as compared to dedicated logic programming systems, and (ii) how does the performance of QUIP change if different QBF evaluation algorithms are used. Concerning the first issue, we compare QUIP with the logic programming systems `dlv` and `smodels`, and regarding the second issue, we compare the employed QBF-solver of QUIP with well-known QBF evaluation algorithms proposed by (Cadoli, Giovanardi, & Schaerf 1998), (Rintanen 1999b), and (Feldmann, Monien, & Schamberger 2000). The results

*This work was partially supported by the Austrian Science Fund Project N Z29-INF.

show, on the one hand, that the QBF approach is surprisingly competitive on certain problem instances compared to `dlv` and `smodels`, and, on the other hand, that alternative QBF algorithms enable an increased performance on examples where the currently employed QBF-solver of QUIP is inefficient.

An important point in every evaluation test concerns the question which problem classes should be used as benchmarks in order to obtain indicative results. Since currently there are no generally accepted benchmarks for nonmonotonic theorem provers, different approaches have been chosen in the literature. Often, experimenters choose problems which are viewed as "natural" or "interesting" in terms of real-life applications, but it is not always clear whether these problems are really "hard" for the given formalism. An approach for generating benchmarks for nonmonotonic theorem provers has been proposed by the well-known *TheoryBase* system (Cholewinski *et al.* 1995), originally developed as a test-bed for the default reasoning system DeReS (Cholewinski, Marek, & Truszczyński 1996). TheoryBase provides encodings of various graph problems in terms of default theories or equivalent logic programs. However, the generated problems are at most NP-hard (or co-NP-hard, depending on the reasoning task), and thus do not take full advantage of the expressibility supported by the majority of the nonmonotonic reasoning formalisms.

For the present purpose, we adopt a straightforward method how Σ_2^p-hard benchmark problems for propositional nonmonotonic reasoning formalisms can be generated, following a similar test procedure used to construct benchmarks for theorem provers implementing various propositional modal logics (Massacci 1999). The idea is to use the class of problems establishing the Σ_2^p-hardness of a given reasoning task. Usually, Σ_2^p-hardness of a nonmonotonic formalism is demonstrated by constructing a (polynomial) transformation $\mathcal{S}$ mapping instances I of $\mathrm{QSAT}_{2,\exists}$ into instances $\mathcal{S}(I)$ of the considered nonmonotonic decision problem, say NMP, such that I is a yes-instance of $\mathrm{QSAT}_{2,\exists}$ iff $\mathcal{S}(I)$ is a yes-instance of NMP.[1] Thus, in some sense, the class of problems $\mathcal{S}(I)$ represents "worst-case" examples for the problem NMP, provided hard instances of $\mathrm{QSAT}_{2,\exists}$ are generated. Moreover, these problems are easily scalable by parameterizing different instances of $\mathrm{QSAT}_{2,\exists}$.

The paper is organized as follows. The next section gives background terminology and notation, as well as information on QUIP. Then, details on the performance tests are presented. The paper concludes with a brief discussion.

Background

In what follows, we assume a propositional language $\mathcal{L}$ generated by a finite set of propositional atoms using the standard sentential connectives $\neg$, $\wedge$, $\vee$, and $\rightarrow$. A *quantified boolean formula* (QBF) is a formula which may additionally contain occurrences of quantifiers $\forall, \exists$ ranging over propositional atoms. Informally, a QBF of the form $\forall p\, \exists q\, \Phi$ means that for all truth-assignments of p there is a truth-assignment of q such that Φ is true. For instance, it is easily seen that the QBF

$$\exists p_1\, \exists p_2\, ((p_1 \rightarrow p_2) \wedge \forall p_3 (p_3 \rightarrow p_2))$$

evaluates to true. As noted in the Introduction, the evaluation problem of arbitrary QBFs is PSPACE-complete, and the problems $\mathrm{QSAT}_{i,\exists}$ and $\mathrm{QSAT}_{i,\forall}$ are complete for Σ_i^p and Π_i^p, respectively.

A (*disjunctive*) *logic program*, Π, is a finite set of rules

$$r: \quad H(r) \leftarrow B^+(r), B^-(r)$$

where $H(r)$ is a disjunction of atoms, $B^+(r)$ is a conjunction of atoms, and $B^-(r)$ is a conjunction of negated atoms. Let V denote the set of all atoms occurring in Π. A Herbrand interpretation I of V is a *stable model* of Π (Gelfond & Lifschitz 1988; Przymusinski 1991) if it is a minimal model (with respect to set-inclusion) of the program Π^I resulting from Π as follows: remove each rule r such that $a \in I$ for some $\neg a$ in $B^-(r)$, and remove $B^-(r)$ from all remaining rules (Π^I is known as the *Gelfond-Lifschitz* reduction (Gelfond & Lifschitz 1988)).

The relevant reasoning tasks in the context of stable models are:

- Given a disjunctive logic program Π, is there a stable model of Π?
- *Brave reasoning*: Given a disjunctive logic program Π and some atom p, is there a stable model I of Π such that $p \in I$?
- *Skeptical reasoning*: Given a disjunctive logic program Π and some atom p, does $p \in I$ hold for all stable models I of Π?

Eiter and Gottlob (1995) showed that the first two tasks are complete for Σ_2^p, whilst the third task is complete for Π_2^p. Thus, checking the existence of a stable model as well as brave reasoning can be polynomially reduced to $\mathrm{QSAT}_{2,\exists}$, and skeptical reasoning can be reduced to $\mathrm{QSAT}_{2,\forall}$. In the following, we briefly describe these transformations.

Let Π be a logic program, and let $V = \{v_1, \ldots, v_n\}$ be the atoms occurring in Π. Furthermore, let $V' := \{v_1', \ldots, v_n'\}$ be a set of new variables, and let r' denote the result of replacing every occurrence of an atom p in r by p'. Then, following from results shown in (Eiter & Gottlob 1997), existence of a stable model of Π can be expressed by the QBF $\exists V\, \Psi(\Pi)$, where

$$\Psi(\Pi) = \Big(P \wedge \forall V'((V' < V) \rightarrow \neg P^*)\Big),$$

and

- $P = \bigwedge_{r \in \Pi} [(B^+(r) \wedge B^-(r)) \rightarrow H(r)]$ represents the conjunction of all rules in Π;
- $P^* = \bigwedge_{r \in \Pi} [(B^+(r') \wedge B^-(r)) \rightarrow H(r')]$ takes care of the Gelfond-Lifschitz reduction; and
- $(V' < V)$ is an abbreviation for $\bigwedge_{i=1}^n (v_i' \rightarrow v_i) \wedge \neg \bigwedge_{i=1}^n (v_i \rightarrow v_i')$.

[1] Observe that QUIP implements a given decision problem $\mathrm{NMP}' \in \Sigma_2^p$ precisely in an inverse fashion: given an instance I' of NMP', QUIP translates I' into an instance $\mathcal{T}(I')$ of $\mathrm{QSAT}_{2,\exists}$ such that I' is a yes-instance of NMP' iff $\mathcal{T}(I')$ is a yes-instance of $\mathrm{QSAT}_{2,\exists}$.

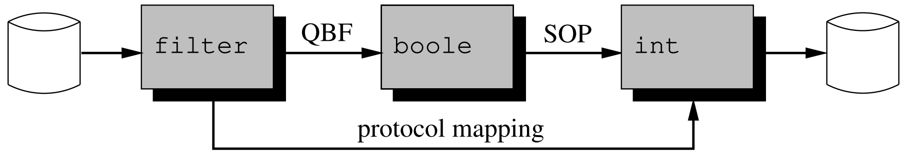

Figure 1: QUIP's system architecture.

Observe that the premise $B^{-}(r)$ in P^{*} contains the atoms from r but not those from r'!

Brave inference of an atom $p \in V$ from Π is expressed by

$$\exists V(\Psi(\Pi) \wedge p),$$

whilst skeptical inference of p from Π is realized by

$$\neg\exists V(\Psi(\Pi) \wedge \neg p).$$

The QBFs discussed so far encode *decision problems*, i.e., they return either *yes* or *no*. However, usually one is more interested in the corresponding *function problems* returning the actual stable models responsible for a given yes-answer. This is easily achieved by simply dropping the outermost quantifier $\exists V$ in the formulas above and computing the respective truth-assignments. More precisely, the propositional models of the formula $\Psi(\Pi)$ correspond to the stable models of the program Π, and the propositional models of $(\Psi(\Pi) \wedge p)$ give all stable models of Π containing the atom p, whilst the propositional models of $(\Psi(\Pi) \wedge \neg p)$ yield stable models *not* including p.

All these reasoning tasks have been implemented in QUIP. The overall architecture of QUIP is depicted in Figure 1. QUIP consists of three parts, namely the `filter` program, the QBF-evaluator `boole`, and the interpreter `int`. The input filter translates the given problem description (in our case, a disjunctive logic program and a specified reasoning task) into the corresponding quantified boolean formula, which is then sent to the QBF-evaluator `boole`. The result of `boole`, usually a formula in disjunctive normal form (often called *sum of products*, SOP), characterizes the truth-assignments of the QBF, and is interpreted by `int`. The latter part associates a meaningful interpretation to the formulas occurring in SOP and provides an explanation in terms of the underlying problem instance (e.g., a stable model). The QBF-evaluator `boole` is a publicly available propositional theorem prover based on *binary decision diagrams* (BDDs), which can be downloaded from the web (http://www.cs.cmu.edu/∼modelcheck/bdd.html).

Comparisons

In this section, we describe our employed test methodology in detail and present the achieved results. First, we introduce the class of problems which are used as benchmarks during the tests. Then, the comparison of QUIP with the dedicated logic programming systems `dlv` and `smodels` is described. Finally, the performance of different QBF evaluation algorithms on the considered benchmark problems is analysed.

Benchmarks

As already argued in the Introduction, we use a class of logic programs encoding the Σ_2^p-complete problem $\mathrm{QSAT}_{2,\exists}$ of determining the truth of QBFs having prenex normal form $\exists P\,\forall Q\,\phi$, where P, Q are lists of propositional variables comprising all variables of the propositional formula ϕ. As is well-known (Wrathall 1976), the problem $\mathrm{QSAT}_{2,\exists}$ remains Σ_2^p-hard if we consider QBFs $\exists P\,\forall Q\,\phi$ where ϕ is in disjunctive normal form such that each clause comprises exactly three literals. Thus, the generated QBFs have the following structure:

$$\exists p_1 \ldots \exists p_n \,\forall q_1 \ldots \forall q_m \bigvee_{i=1}^{t} L_{i,1} \wedge L_{i,2} \wedge L_{i,3} \qquad (1)$$

where

- n is the number of the existentially quantified variables;
- m is the number of the universally quantified variables;
- t is the number of clauses; and
- $L_{i,j}$ are literals r or $\neg r$, where r is some atom among $p_1, \ldots, p_n, q_1, \ldots, q_m$, for $1 \leq i \leq t$ and $j = 1, 2, 3$.

As shown by Eiter and Gottlob (1995), determining the truth of a QBF Φ of form (1) can be expressed by the following disjunctive logic program Π_Φ, where $v_1, \ldots, v_n$, $z_1, \ldots, z_m$, and w are new propositional variables:

$$\begin{array}{ll}
p_i \vee v_i \leftarrow & \text{for } 1 \leq i \leq n \\
q_j \vee z_j \leftarrow \quad q_j \leftarrow w \quad z_j \leftarrow w & \\
w \leftarrow q_j, z_j & \text{for } 1 \leq j \leq m \\
w \leftarrow \sigma(L_{k,1}), \sigma(L_{k,2}), \sigma(L_{k,3}) & \text{for } 1 \leq k \leq t \\
w \leftarrow \ \mathit{not}\ w &
\end{array}$$

The function σ maps literals to atoms as follows:

$$\sigma(L) = \begin{cases} v_i & \text{if } L = \neg p_i, \text{ for some } i = 1, \ldots, n; \\ z_j & \text{if } L = \neg q_j, \text{ for some } j = 1, \ldots, m; \\ L & \text{otherwise.} \end{cases}$$

Programs of this form constitute the benchmark problems used in our tests. It holds that Π_Φ has a stable model iff Φ evaluates to true.

In order to generate QBFs Φ which are hard for $\mathrm{QSAT}_{2,\exists}$ (and thus yielding hard programs Π_Φ), some care is required. As has been pointed out by Gent and Walsh (1999), sets of randomly generated QBFs tend to include an immoderately number of trivial instances and thus are not representative as basis for objective evaluation tests. As a consequence, we generate QBFs according to Model A, advocated by Gent and Walsh (1999), to circumvent this problem. Actually, we adopt a dual variant of this model because the original version has been introduced to deal with formulas in *conjunctive normal form*, whereas we use here formulas in *disjunctive normal form*. In this variant of Model A, every clause is supposed to contain at least two literals from the set of universally quantified variables of the generated QBF, and repetitions are discarded.

QBFs versus Logic Programming Systems

The first test-set comprises comparisons between QUIP, `dlv`, and `smodels`. All results have been achieved using the problem of constructing the stable models of programs Π_Φ generated as described above.

The specific details of the tests are as follows:

- We used `dlv` (release date 2000-04-05) and `smodels v2.25`.
- All evaluations have been performed on an Alpha-server equipped with two 500MHz Alpha-21264 processors, 1 Gigabyte of RAM, and a Linux 2.2.13-SMP kernel (SuSE-Linux 6.3).
- We generated QBFs according to Model A, with the number $k = n+m$ of distinct variables ranging from 20 to 48, the number n of existentially quantified variables ranging from 4 to 18, and the number t of clauses determined by the factor $f = \frac{3 \cdot t}{k}$, where f was chosen 1.1, 1.3, 1.5, and 1.7. For each selected parameter setting, we took 30 samples and calculated the mean value. Time-out was set to 15000s, but was never reached during the experiments.

Results are given in Figures 2–7. Figures 2 and 3 contain diagrams depicting the running time of QUIP vs. `dlv` and `smodels`, respectively, depending on the two parameters k and n, and with fixed factor $f = 1.7$. The graphs reveal a strong dependence of the running time of QUIP on the number k of distinct variables. In contrast, the running times of both `dlv` and `smodels` are more influenced by the number n of existentially quantified variables. Detailed sections of these diagrams are given in Figures 4–7. More specifically, Figure 4 contains the running times of all three systems over the number k of distinct variables, for different values of n and with $f = 1.5$ held constant. Dually, Figure 5 depicts a similar scenario, but for different values of f and with $n = 16$ held constant. These diagrams reiterate the behaviour observed above, namely that the running time of QUIP depends strongly on the number k of distinct variables, but less on the number n of existentially quantified variables, whereas `dlv` and `smodels` show a converse behaviour. The latter two systems are also less influenced by different values of f (Figure 5), whilst QUIP shows a stronger dependence on this factor.

Figures 6 and 7 contain the running times over the number n of existentially quantified variables. In particular, Figure 6 illustrates again the strong dependence of QUIP on k but the weak dependence on n, and the converse behaviour for `dlv` and `smodels`. Also interesting to note is the fact that, for small values of n, QUIP is outperformed by both `dlv` and `smodels`, but wins over these systems as n grows.

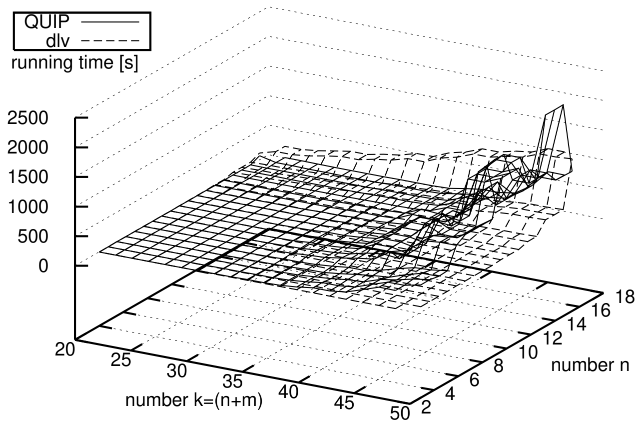

Figure 2: Results for QUIP and `dlv` with $f = 1.7$ constant.

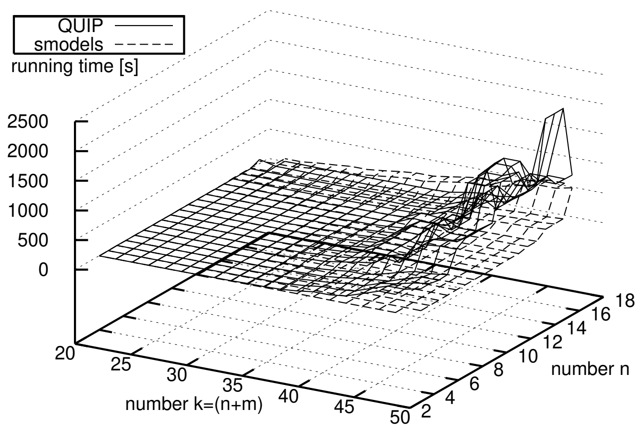

Figure 3: Results for QUIP and `smodels` with $f = 1.7$ constant.

Overall, we can say that QUIP is quite capable to stand against dedicated logic programming systems, at least on certain instances of the considered problem class. For larger-sized problem instances, `boole` works less satisfactorily and both `dlv` and `smodels` are more efficient. This specific problem domain is not detailed in the present diagrams and will be the subject of the following set of evaluation tests.

Comparison of Different QBF-Evaluators

The second category of tests was set out to compare `boole` with the QBF-solvers `Evaluate` (Cadoli, Giovanardi, & Schaerf 1998), `qbf` (Rintanen 1999b), and `ssolve` (Feldmann, Monien, & Schamberger 2000). Thus, in this case,

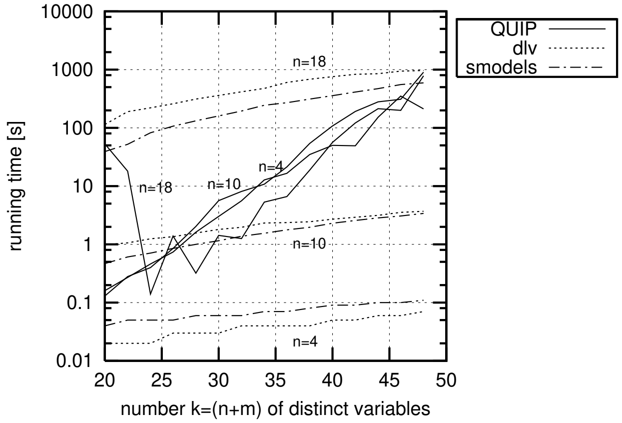

Figure 4: Results for different values of k and $f = 1.5$ constant.

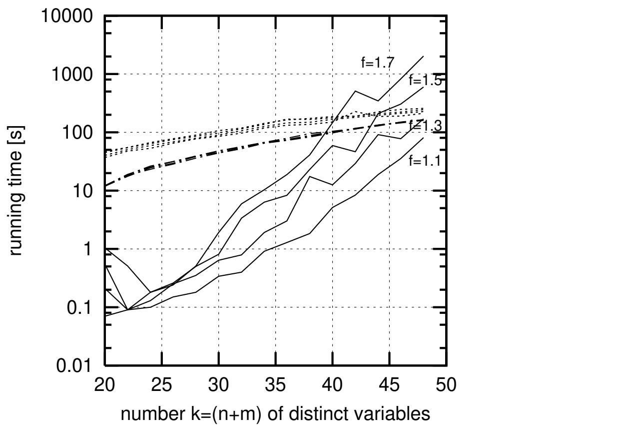

Figure 5: Results for different values of k and $n = 16$ constant.

we are only interested in the evaluation time of the reduced QBFs (resulting from the programs Π_Φ according to the reduction described in the Background section) and not in the overall processing time.

With the exception of `boole`, the considered tools do not accept arbitrary QBFs, but require the input formula to be in *prenex conjunctive normal form*. For these tools, the generated QBFs are translated into *definitional normal form* (Eder 1992; Plaisted & Greenbaum 1986). In contrast to the usual normal form translation based on distributivity laws, the definitional normal form translation is *structure preserving* and polynomial in the length of the input formula.

Another issue concerns the fact that the systems `Evaluate` and `qbf` cannot output the satisfying truth-assignments of the free variables of a given (open) QBF, which, in our approach, is a necessary prerequisite in order to construct the corresponding stable models. Consequently, the present tests perform only *decision problems*, i.e., checking whether there is a stable model of a given program. As well, `Evaluate`, `qbf`, and `ssolve` are currently not available for Alpha-processors, so the two-processor server employed in

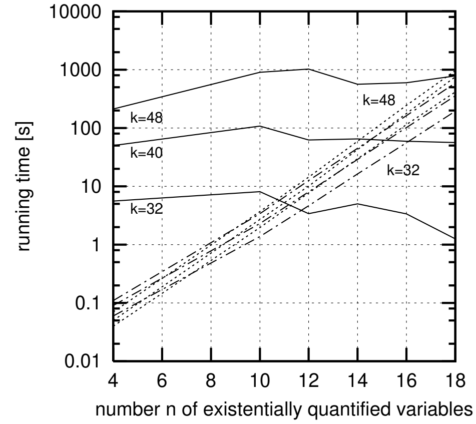

Figure 6: Results for different values of n and $f = 1.5$ constant.

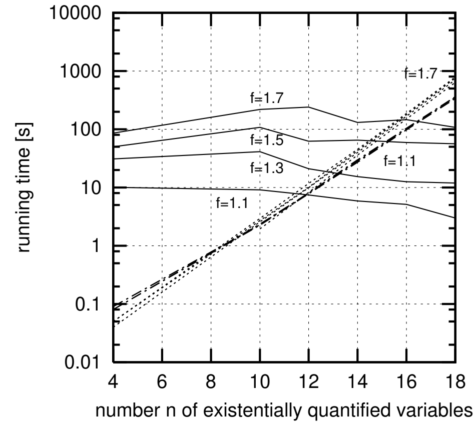

Figure 7: Results for different values of n and $k = 40$ constant.

the previous experiments could not be used here and we ran the tests on a somewhat less powerful SUN workstation under Solaris.

The specific details of the tests are as follows:

- We used `boole`, `ssolve v4.5`, `Evaluate v1.0`, and `qbf v1.0`. The latter package was executed with option `-f`, which leads to a better performance on the considered QBFs.
- The tests have been performed on a SUN ULTRA 60 with 256 MB RAM and SunOS 5.6.
- The input QBFs are of the form $\exists V\, \Psi(\Pi_\Phi)$, expressing the existence of stable models of the program Π_Φ (cf. Background section), where Π_Φ is the class of benchmark programs generated as described previously. Accordingly, the parameters k, n, m, t, and f are to be understood as before. This time, k ranged from 10 to 150, n ranged from 4 to 20, and t ranged from 15 to 515; time-out and sample size are identical to the settings of the first test. In the present case, however, time-out was reached during the experiments and the results are displayed only for those

values which were not affected by instances reaching the time-out.

The results are depicted in Figures 8–10. Figure 8 displays the performance of all four systems in relation to varying values of the factor $f = \frac{3 \cdot t}{k}$, holding both the number k of total variables and the number n of existentially quantified variables constant. Figure 9 depicts the running time for different values of n, with $k = 32$ and $t = 16$ constant (thus representing graphs with $f = 1.5$). Figure 10 gives the running time over the total number k of variables with $t = 51$ and $n = 4$ constant.

The results of Figure 8 show that the running time of `boole` grows strongly with increasing f; in fact, `boole` is unable to compute any instances above $f = 2.5$ due to memory overloading. On the other hand, `ssolve`, `qbf`, and `Evaluate` display quite a different behaviour: they show a typical *phase transition* pattern in the region of $f \approx 5$, representing an easy-hard-easy phenomenon well-documented for SAT-problems. Figure 9 reveals that `ssolve`, `qbf`, and `Evaluate` are like `dlv` and `smodels` strongly dependent on the number n of existentially quantified variables, whereas the graph for `boole` is more or less constant and outperforms the other systems for large numbers of existentially quantified variables. On the other hand, Figure 10 shows (more directly than Figure 8) that `boole` cannot handle problems involving a large totality of variables.[2] This is a well-known weakness of theorem provers based on binary decision diagrams.

In any case, these experiments demonstrate quite clearly that the algorithm `ssolve` (Feldmann, Monien, & Schamberger 2000) is superior to both `qbf` and `Evaluate` on the considered problem class. Also, since `boole` has quite a different behaviour than these QBF-solvers, the inclusion of `ssolve` within the framework of QUIP (by, e.g., invoking it in parallel with `boole`) will improve the overall performance on instances which are difficult to handle for `boole`.

Conclusion

In this paper, we considered the computation of stable models for disjunctive logic programs by means of a reduction to quantified boolean formulas. We compared the realization of this approach with dedicated logic programming systems, and we also evaluated how different QBF-solvers behave on such QBFs. The class of problems used to perform these tests have been chosen purely on a *worst-case basis*, in accord with investigations to generate hard instances of SAT (Selman, Mitchell, & Levesque 1996). The results of our experiments demonstrate that the QBF approach is a viable tool for rapid prototyping applications. Also, different QBF-packages can yield an increased performance for instances where the current BDD-based QBF-solver of QUIP is inefficient. In fact, alternative QBF evaluation algorithms can easily be incorporated within QUIP and substantiate one

[2] One should observe that the parameters indicated in the diagrams reflect the number of variables of the *original* QBFs, whereas the actually tested formulas involve *much more variables* (which are introduced by the reductions and the normal form translation).

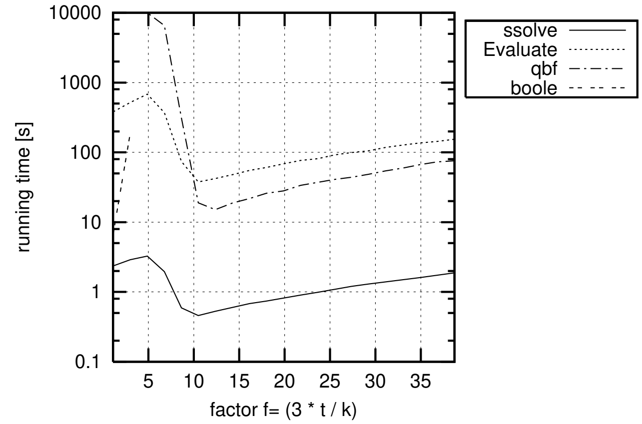

Figure 8: Results for different values of f and $k = 40$, $n = 10$ constant.

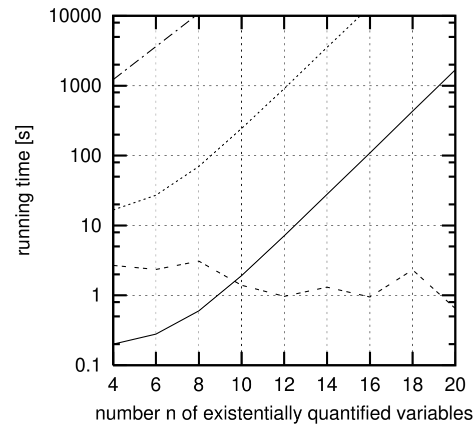

Figure 9: Results for different values of n and $k = 32$, $t = 16$ constant.

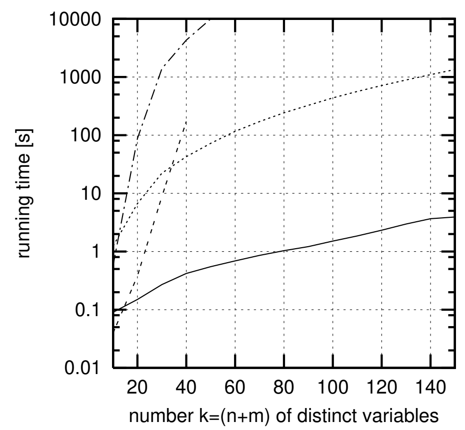

Figure 10: Results for different values of k and $t = 51$, $n = 4$ constant.

of the advantages of our framework. For instance, an ideal candidate of such an extension of QUIP is the distributed version of `ssolve` (Feldmann, Monien, & Schamberger 2000), which runs on PC-clusters yielding an even better performance than the sequential version of `ssolve` considered in the present paper.

In general, the use of QBFs for knowledge representation purposes has been advocated in the literature (Cadoli, Giovanardi, & Schaerf 1998; Rintanen 1999b), and, besides the present system QUIP, applications to planning problems have been realized (Rintanen 1999a).

Our experiments involving the different QBF-solvers were based on a *structure preserving* normal form translation. Such normalization transformations have certain advantages compared to standard translations. For first-order systems, they can yield, e.g., a nonelementary decrease of proof length and search space (Egly 1996). Clearly, different such translations affect the overall performance of a system; a deeper analysis of this issue in the context of the present approach is left for future research. As a matter of fact, we plan to investigate to what extent the mixed normal form translation discussed by Boy de la Tour (1992) can be useful within our framework.

Another point which warrants a more closer investigation concerns the phase transition patterns observed in the present experiments. Such phenomena are well-known for SAT-problems and have also been studied for QBFs (Gent & Walsh 1999; Cadoli *et al.* 2000), but they clearly require a more thorough analysis for the considered problem classes.

References

Boy de la Tour, T. 1992. An Optimality Result for Clause Form Translation. *Journal of Symbolic Computation* 14(4):283–302.

Cadoli, M.; Giovanardi, A.; Giovanardi, M.; and Schaerf, M. 2000. An Algorithm to Evaluate Quantified Boolean Formulae and its Experimental Evaluation. *Journal of Automated Reasoning*. To appear.

Cadoli, M.; Giovanardi, A.; and Schaerf, M. 1998. An Algorithm to Evaluate Quantified Boolean Formulae. In *Proc. AAAI-98*, 262–267.

Cholewinski, P.; Marek, W.; Mikitiuk, A.; and Truszczyǹski, M. 1995. Experimenting with Nonmonotonic Reasoning. In *Proc. ICLP-95*, 267–282.

Cholewinski, P.; Marek, W.; and Truszczyński, M. 1996. Default Reasoning System DeReS. In *Proc. KR-96*, 518–528.

Eder, E. 1992. *Relative Complexities of First-Order Calculi*. Artificial Intelligence. Vieweg Verlag.

Egly, U.; Eiter, T.; Tompits, H.; and Woltran, S. 2000a. QUIP - A Tool for Computing Nonmonotonic Resoning Tasks. In *Proc. 8th International Workshop on Non-Monotonic Reasoning*.

Egly, U.; Eiter, T.; Tompits, H.; and Woltran, S. 2000b. Solving Advanced Reasoning Tasks Using Quantified Boolean Formulas. In *Proc. AAAI-00*, 417–422.

Egly, U. 1996. On Different Structure-preserving Translations to Normal Form. *Journal of Symbolic Computation* 22:121–142.

Eiter, T., and Gottlob, G. 1995. On the Computational Cost of Disjunctive Logic Programming: Propositional Case. *Annals of Mathematics and Artificial Intelligence* 15(3–4):289–323.

Eiter, T., and Gottlob, G. 1997. Expressiveness of Stable Model Semantics for Disjuncitve Logic Programs with Functions. *Journal of Logic Programming* 33(2):167–178.

Eiter, T.; Leone, N.; Mateis, C.; Pfeifer, G.; and Scarcello, F. 1997. A Deductive System for Non-monotonic Reasoning. In *Proc. LPNMR-97*, 363–374.

Feldmann, R.; Monien, B.; and Schamberger, S. 2000. A Distributed Algorithm to Evaluate Quantified Boolean Formulas. In *Proc. AAAI-00*, 285–290.

Gelfond, M., and Lifschitz, V. 1988. The Stable Model Semantics for Logic Programming. In *Proc. ICLP-88*, 1070–1080.

Gent, I. P., and Walsh, T. 1999. Beyond NP: The QSAT Phase Transition. In *Proc. AAAI-99*, 648–653.

Janhunen, T.; Niemelä, I.; Simons, P.; and You, J.-H. 2000. Unfolding Partiality and Disjunctions in Stable Model Semantics. In *Proc. KR-00*, 411–424.

Massacci, F. 1999. Design and Results of the Tableaux-99 Non-classical (Modal) Systems Comparison. In *Proc. TABLEAUX-99*, 14–18.

Niemelä, I., and Simons, P. 1997. Smodels: An Implementation of the Stable Model and Well-Founded Semantics for Normal LP. In *Proc. LPNMR-97*, 420–429.

Plaisted, D. A., and Greenbaum, S. 1986. A Structure Preserving Clause Form Translation. *Journal of Symbolic Computation* 2(3):293–304.

Przymusinski, T. 1991. Stable Semantics for Disjunctive Programs. *New Generation Computing Journal* 9:401–424.

Rintanen, J. 1999a. Constructing Conditional Plans by a Theorem Prover. *Journal of Artificial Intelligence Research* 10:323–352.

Rintanen, J. 1999b. Improvements to the Evaluation of Quantified Boolean Formulae. In *Proc. IJCAI-99*, 1192–1197.

Selman, B.; Mitchell, D. G.; and Levesque, H. J. 1996. Generating Hard Satisfiability Problems. *Artificial Intelligence* 81(1–2):17–29.

Stockmeyer, L. J. 1976. The Polynomial-Time Hierarchy. *Theoretical Computer Science* 3(1):1–22.

Wrathall, C. 1976. Complete Sets and the Polynomial-Time Hierarchy. *Theoretical Computer Science* 3(1):23–33.