INFSYS
RESEARCH
REPORT

INSTITUT FÜR INFORMATIONSSYSTEME

ABTEILUNG WISSENSBASIERTE SYSTEME

SOLVING HARD PROBLEMS FOR THE SECOND LEVEL OF THE POLYNOMIAL HIERARCHY: HEURISTICS AND BENCHMARKS

Wolfgang Faber Nicola Leone Francesco Ricca

INFSYS RESEARCH REPORT 1843-05-09

OCTOBER 2005

Institut für Informationssysteme
Abtg. Wissensbasierte Systeme
Technische Universität Wien
Favoritenstraße 9-11
A-1040 Wien, Austria
Tel: +43-1-58801-18405
Fax: +43-1-58801-18493
sek@kr.tuwien.ac.at
www.kr.tuwien.ac.at

TECHNISCHE UNIVERSITÄT WIEN

INFSYS RESEARCH REPORT

INFSYS RESEARCH REPORT 1843-05-09, OCTOBER 2005

SOLVING HARD PROBLEMS FOR THE SECOND LEVEL OF THE POLYNOMIAL HIERARCHY: HEURISTICS AND BENCHMARKS

Wolfgang Faber[1,2]   Nicola Leone[1]   Francesco Ricca[1]

Abstract. *Recent research on answer set programming (ASP) systems, has mainly focused on solving* NP *problems more efficiently. Yet, disjunctive logic programs allow for expressing every problem in the complexity classes* Σ_2^P *and* Π_2^P*. These classes are widely believed to be strictly larger than* NP*, and several important AI problems, like conformant and conditional planning, diagnosis and more are located in these classes.*
In this paper we focus on improving the evaluation of Σ_2^P/Π_2^P*-hard ASP programs. To this end, we define a new heuristic* h_{DS} *and describe its implemention in the (disjunctive) ASP system* DLV*. The definition of* h_{DS} *is geared towards the peculiarites of hard programs, while it maintains the benign behaviour of the well-assessed heuristic of* DLV *for* NP *problems.*
We have conducted extensive experiments with the new heuristic. h_{DS} *significantly outperforms the previous heuristic of* DLV *on hard 2QBF problems. We also compare the* DLV *system (with* h_{DS}*) to the QBF solvers which performed best in the QBF evaluation of 2004. The results of the comparison indicate that ASP systems currently seem to be the best choice for solving* Σ_2^P/Π_2^P*-complete problems.*

[1]Department of Mathematics, University of Calabria. 87030 Rende (CS), Italy E-mail: ricca, faber, leone@mat.unical.it

[2]Funded by an APART grant of the Austrian Academy of Sciences.

Acknowledgements: This work was supported by the European Commission under projects IST-2002-33570 INFOMIX, and IST-2001-37004 WASP.

1 Introduction

Answer Set Programming (ASP) is a novel programming paradigm, which has been recently proposed in the area of nonmonotonic reasoning and logic programming. The idea of answer set programming is to represent a given computational problem by a logic program whose answer sets correspond to solutions, and then use an answer set solver to find such a solution [20]. The knowledge representation language of ASP is very expressive in a precise mathematical sense; in its general form, allowing for disjunction in rule heads and nonmonotonic negation in rule bodies, ASP can represent *every* problem in the complexity class Σ_2^P and Π_2^P (under brave and cautious reasoning, respectively) [8]. Thus, ASP is strictly more powerful than SAT-based programming, as it allows us to solve problems which cannot be translated to SAT in polynomial time. The high expressive power of ASP can be profitably exploited in AI, which often has to deal with problems of high complexity. For instance, problems in diagnosis and planning under incomplete knowledge are complete for the complexity class Σ_2^P or Π_2^P [23, 7], and can be encoded in ASP [1, 16].

Most of the optimization work on ASP systems has focused on the efficient evaluation of non-disjunctive programs (whose power is limited to NP/co-NP), whereas the optimization of full (disjunctive) ASP programs has been treated in fewer works (e.g., in [13, 14]). In particular, we are not aware of any work concerning heuristics for Σ_2^P/Π_2^P-hard ASP programs, which is fundamentally important for the efficiency of an ASP system.

In this paper, we address the following two questions:

▶ Can the heuristics of ASP systems be refined to deal more efficiently with Σ_2^P/Π_2^P-hard ASP programs?

▶ On hard Σ_2^P/Π_2^P problems, can ASP systems compete with other AI systems, like QBF solvers?

We define a new heuristic h_{DS} for the (disjunctive) ASP system DLV. The new heuristic aims at improving the evaluation of Σ_2^P/Π_2^P-hard ASP programs, but it is designed to maintain the benign behaviour of the well-assessed heuristic of DLV on NP problems like 3SAT and Blocks-World, on which it proved to be very effective [10]. We experimentally compare h_{DS} against the DLV heuristic on hard 2QBF instances, generated following recent works presented in the literature that describe transition phase results for QBFs [5, 12]. h_{DS} significantly outperforms the heuristic of DLV on 2QBF.

To check the competitiveness of ASP w.r.t. QBF solvers on hard problems, we carry out an experimental comparison of the DLV system (with the new heuristic h_{DS}) with four prominent QBF solvers, which performed best at the 2004 QBF evaluation[22]: SSolve, Semprop, Quantor, yQuaffle. The results of the comparison, performed on instances used in the QBF competition and on a set of randomly generated instances for the *Strategic Companies* problem, indicate that ASP systems currently perform better than QBF systems on Σ_2^P/Π_2^P-hard problems.

It is worthwhile noting that, besides DLV [15], there are many other efficient ASP systems, including GnT [13], Smodels [24], ASSAT [21], and Cmodels [19]. We did not experiment our heuristics with these systems, because they are either limited to NP/co-NP (e.g., Smodels, ASSAT) or employ another system as a black-box (e.g., GnT, Cmodels).

2 Answer Set Programming Language

2.1 ASP Programs

A *(disjunctive) rule* r is a formula

$$a_1 \ \vee \ \cdots \ \vee \ a_n :\!- \ b_1, \cdots, b_k, \ \texttt{not} \ b_{k+1}, \cdots, \ \texttt{not} \ b_m.$$

where $a_1, \cdots, a_n, b_1, \cdots, b_m$ are atoms and $n \geq 0$, $m \geq k \geq 0$. The disjunction $a_1 \vee \cdots \vee a_n$ is the *head* of r, while the conjunction $b_1, \cdots, b_k, \texttt{not}\ b_{k+1}, \cdots, \texttt{not}\ b_m$ is the *body*, $b_1, \cdots, b_k$ the *positive body*, and $\texttt{not}\ \ b_{k+1}, \cdots, \texttt{not}\ \ b_m$ the *negative body* of r.

An *(ASP) program* $\mathcal{P}$ is a finite set of rules. An object (atom, rule, etc.) is called *ground* or *propositional*, if it contains no variables.

2.2 Answer Sets

Given a program $\mathcal{P}$, let the *Herbrand Universe* $U_{\mathcal{P}}$ be the set of all constants appearing in $\mathcal{P}$ and the *Herbrand Base* $B_{\mathcal{P}}$ be the set of all possible ground atoms which can be constructed from the predicate symbols appearing in $\mathcal{P}$ with the constants of $U_{\mathcal{P}}$.

Given a rule r, $Ground(r)$ denotes the set of rules obtained by applying all possible substitutions σ from the variables in r to elements of $U_{\mathcal{P}}$. Similarly, given a program $\mathcal{P}$, the *ground instantiation* $Ground(\mathcal{P})$ of $\mathcal{P}$ is the set $\bigcup_{r \in \mathcal{P}} Ground(r)$.

A set L of ground literals is said to be *consistent* if, for every atom $\ell \in L$, its complementary literal $\texttt{not}\ \ell$ is not contained in L. An interpretation I for $\mathcal{P}$ is a consistent set of ground literals over atoms in $B_{\mathcal{P}}$.A ground literal ℓ is *true* w.r.t. I if $\ell \in I$; ℓ is *false* w.r.t. I if its complementary literal is in I; ℓ is *undefi ned* w.r.t. I if it is neither true nor false w.r.t. I. Interpretation I is *total* if, for each atom A in $B_{\mathcal{P}}$, either A or $\texttt{not}\ A$ is in I (i.e., no atom in $B_{\mathcal{P}}$ is undefined w.r.t. I). A total interpretation M is a *model* for $\mathcal{P}$ if, for every $r \in Ground(\mathcal{P})$, at least one literal in the head is true w.r.t. M whenever all literals in the body are true w.r.t. M. X is an *answer set* for a positive program $\mathcal{P}$ if it is minimal w.r.t. set inclusion among the models of $\mathcal{P}$.

Example 1 For the program $\mathcal{P}_1 = \{a \vee b \vee c.\,,\ :\!-a.\}$, $\{b, \texttt{not}\ a, \texttt{not}\ c\}$ and $\{c, \texttt{not}\ a, \texttt{not}\ b\}$ are the answer sets. For the program $\mathcal{P}_2 = \{a \vee b \vee c.\,,\ :\!-a.\,,\ b{:}\!-c.\,,\ c{:}\!-b.\}$, $\{b, c, \texttt{not}\ a\}$ is the only answer set. ∎

The *reduct* or *Gelfond-Lifschitz transform* of a general ground program $\mathcal{P}$ w.r.t. an interpretation X is the positive ground program $\mathcal{P}^X$, obtained from $\mathcal{P}$ by (i) deleting all rules $r \in \mathcal{P}$ whose negative body is false w.r.t. X and (ii) deleting the negative body from the remaining rules.

An answer set of a general program $\mathcal{P}$ is a model X of $\mathcal{P}$ such that X is an answer set of $Ground(\mathcal{P})^X$.

Example 2 Given the (general) program $\mathcal{P}_3 = \{a \vee b{:}\!-c.\,,\ b{:}\!-\texttt{not}\ a, \texttt{not}\ c.\,,\ a \vee c{:}\!-\texttt{not}\ b.\}$ and $I = \{b, \texttt{not}\ a, \texttt{not}\ c\}$, the reduct $\mathcal{P}_3^I$ is $\{a \vee b{:}\!-c., b.\}$. I is an answer set of $\mathcal{P}_3^I$, and for this reason it is also an answer set of $\mathcal{P}_3$. ∎

Given an interpretation I for a ground program $\mathcal{P}$,we say that a ground atom A is *supported* in I if there exists a *supporting* rule $r \in ground(\mathcal{P})$ such that the body of r is true w.r.t. I and A is the only true atom in the head of r.

Proposition 3 *[17]* If M is an answer set of a program $\mathcal{P}$, then all atoms in M are supported.

3 Answer Set Computation

In this section, we describe the main steps of the computational process performed by ASP systems. We will refer particularly to the computational engine of the DLV system, which will be used for the experiments, but also other ASP systems, like Smodels, employ a similar procedure.

An answer set program $\mathcal{P}$ in general contains variables. The first step of a computation of an ASP system eliminates these variables, generating a ground instantiation $ground(\mathcal{P})$ of $\mathcal{P}$.[1] The hard part of the computation is then performed on this ground ASP program generated by the instantiator.

```
Function ModelGenerator(I: Interpretation): Boolean;
begin
    I := DetCons(I);
    if I = $\mathcal{L}$ then return False; (* inconsistency *)
    if no atom is undefi ned in I then return IsAnswerSet(I);
    Select an undefi ned ground atom $A$ according to a heuristic;
    if ModelGenerator($I \cup \{A\}$) then return True;
    else return ModelGenerator($I \cup \{\texttt{not } A\}$);
end;
```

Figure 1: Computation of Answer Sets

The heart of the computation is performed by the Model Generator, which is sketched in Figure 1. Roughly, the Model Generator produces some "candidate" answer sets. The stability of each of them is subsequently verified by the function *IsAnswerSet(I)*, which verifies whether the given "candidate" I is a minimal model of the program $Ground(\mathcal{P})^I$ obtained by applying the GL-transformation w.r.t. I. *IsAnswerSet(I)* returns True if the computation should be stopped and False otherwise.

The ModelGenerator function is first called with parameter I set to the empty interpretation.If the program $\mathcal{P}$ has an answer set, then the function returns True setting I to the computed answer set; otherwise it returns False. The Model Generator is similar to the Davis-Putnam procedure employed by SAT solvers. It first calls a function DetCons(), which returns the extension of I with the literals that can be deterministically inferred (or the set of all literals $\mathcal{L}$ upon inconsistency). This function is similar to a unit propagation procedure employed by SAT solvers, but exploits the peculiarities of ASP for making further inferences (e.g., it exploits the knowledge that every answer set is a minimal model). If DetCons does not detect any inconsistency, an atom A is selected according to a heuristic criterion and ModelGenerator is called on $I \cup \{A\}$ and on $I \cup \{\texttt{not } A\}$. The atom A plays the role of a branching variable of a SAT solver. And indeed, like for SAT

[1]Note that $ground(\mathcal{P})$ is usually a subset of $Ground(\mathcal{P})$.

solvers, the selection of a "good" atom A is crucial for the performance of an ASP system. In the next section, we describe a number of heuristic criteria for the selection of such branching atoms.

Remark 1 On hard ASP programs (programs which are not Head Cycle Free (HCF) [2]), a very large part of the computation-time may be consumed by function *isAnswerSet(I)*, since it performs a co-NP-complete task if the program is non-HCF. ∎

4 Heuristics

Throughout this section, we assume that a ground ASP program $\mathcal{P}$ and an interpretation I have been fixed. Here, we describe the two heuristic criteria that will be compared in Section 5. We consider "dynamic heuristics" (the ASP equivalent of UP heuristics for SAT[2]), that is, branching rules where the heuristic value of a literal Q depends on the result of taking Q true and computing its consequences. Given a literal Q, $ext(Q)$ will denote the interpretation resulting from the application of DetCons (see previous section) on $I \cup \{Q\}$; without loss of generality, we assume that $ext(Q)$ is consistent, otherwise Q is automatically set to false and the heuristic is not evaluated on Q at all.

The Heuristic of DLV **(h_{UT}).** The heuristic employed by the DLV system was proposed in [10], where it was shown to be very effective on relevant problems like 3Satisfiability, Hamilthonian Path, Blocks World, and Strategic Companies.

A peculiar property of answer sets is *supportedness*: For each true atom A of an answer set I, there exists a rule r of the program such that the body of r is true w.r.t. I and A is the only true atom in the head of r. Since an ASP system must eventually converge to a supported interpretation, ASP systems try to keep the interpretations "as much supported as possible" during the intermediate steps of the computation. To this end, the DLV system counts the number of *UnsupportedTrue (UT)* atoms, i.e., atoms which are true in the current interpretation but still miss a supporting rule (further details on UTs can be found in [9] where they are called MBTs). For instance, the rule $:- not\ x$ implies that x must be true in every answer set of the program; but it does not give a "support" for x. Thus, in the DLV system x is taken true to satisfy the rule, and it is added to the set of UnsupportedTrue; it will be removed from this set once a supporting rule for x will be found (e.g., $x \vee b :- c$ is a supporting rule for x in the interpretation $I = \{x, \texttt{not}\ b, c\}$). Given a literal Q, let $UT(Q)$ be the number of UT atoms in $ext(Q)$. Moreover, let $UT_2(Q)$ and $UT_3(Q)$ be the number of UT atoms occurring, respectively, in the heads of exactly 2 and 3 unsatisfied rules w.r.t. $ext(Q)$. The heuristic h_{UT} of DLV considers $UT(Q)$, $UT_2(Q)$ and $UT_3(Q)$ in a prioritized way, to favor atoms yielding interpretations with fewer $UT/UT_2/UT_3$ atoms (which should more likely lead to a supported model). If all UT counters are equal, then the heuristic considers the total number $Sat(Q)$ of rules which are satisfied w.r.t. $ext(Q)$.

The heuristic h_{UT} is "balanced", that is, the heuristic values of an atom Q depends on both the effect of taking Q and $\texttt{not}\ Q$.

[2]The UP heuristic for SAT adds for each variable x a unit clause x and -x, respectively, and makes two independent unit propagations. The choice is then based on information thus obtained.

For an atom Q, let $UT'(Q) = UT(Q) + UT(\texttt{not}\ Q)$, $UT_2'(Q) = UT_2(Q) + UT_2(\texttt{not}\ Q)$, $UT_3'(Q) = UT_3(Q) + UT_3(\texttt{not}\ Q)$, and, finally, $Sat'(Q) = Sat(Q) + Sat(\texttt{not}\ Q)$. Given two atoms A and B, $A <_{h_{UT}} B$ holds if:

1. $UT'(A) > UT'(B)$, or
2. $UT'(A) = UT'(B)$ and $UT_2'(A) > UT_2'(B)$, or
3. $UT'(A) = UT'(B)$, $UT_2'(A) = UT_2'(B)$ and $UT_3'(A) > UT_3'(B)$, or
4. $UT'(A) = UT'(B)$, $UT_2'(A) = UT_2'(B)$, $UT_3'(A) = UT_3'(B)$ and $Sat'(A) < Sat'(B)$.

A $<_{h_{UT}}$-maximum atom A is selected by the heuristic h_{UT} of DLV; A is taken positive or negative, by comparing the values of $UT(A)$, $UT_2(A)$, $UT_3(A)$, and $Sat(A)$, with $UT(\texttt{not}\ A)$, $UT_2(\texttt{not}\ A)$, $UT_3(\texttt{not}\ A)$, and $Sat(\texttt{not}\ A)$, respectively, as above.

Example 4 Consider program $\mathcal{P}_4 = \{a \vee b \vee c.\,,\ d \vee e \vee f.\,,\ {:-}\texttt{not}\ w.\,,\ w{:-}a.\,,\ w{:-}d.\,,\ a \vee z{:-}w.\,,\ b \vee z{:-}w.\,,\ {:-}d, z.\,,\ {:-}a, z.\}$, and let the current interpretation $I = \{w\}$; atom w is UT. a and d are the $<_{h_{UT}}$-maxima, as only assuming their truth can eliminate the UT w. Indeed, any other choice would be poor. ∎

The New Heuristic (h_{DS}). The unsupported true atoms are, in a sense, the hardest constraints occurring in an ASP program. Indeed, as pointed out above, an unsupported true atom x is intuitively like a unary constraint ${:-}\texttt{not}\ x$, which must be satisfied. By minimizing the UT atoms and maximizing the satisfied rules, the heuristic h_{UT} tries to drive the DLV computation toward a *supported model* (i.e., all rules are satisfied and no UT exists). Intuitively, supported models have good chances to be answer sets (while unsupported models are guaranteed to be not answer sets), and, for simple classes of programs (e.g., tight stratified disjunctive programs) the supported models are precisely the answer sets. If the program is not tight and stratified, then supported models are not guaranteed to be answer sets; but answer-set checking can be done efficiently if the program is HCF [2].

For hard ASP programs (i.e., non-HCF programs – they express Σ_2^P-complete problems under brave reasoning), supported models are often not answer sets. Answer-set checking is computationally expensive (co-NP-complete), and may consume a large portion of the resources needed for computing an answer set.

The heuristic h_{DS}, described next, tries to drive the computation toward supported models having higher chances to be answer sets, reducing the overall number of the expensive answer-set checks. Models having a “higher degree of supportedness” are preferred, where the degree of supportedness is the average number of supporting rules for the true atoms (note that this number is higher than one, on supported models). Intuitively, if all true atoms have many supporting rules in a model M, then the elimination of an atom from the model would violate many rules, and it becomes less likely to find a subset of M which is a model of $\mathcal{P}^M$, in order to disprove that M is an answer set.

We next formalize this intuition to define the new heuristic h_{DS}. Given a literal Q, let $True(Q)$ be the number of true non-HCF atoms in $ext(Q)$, and let $SuppRules(Q)$ be the number of all supporting rules for non-HCF atoms w.r.t. $ext(Q)$. Intuitively, the heuristic maximizes the “degree of supportedness" of the interpretation, intended as the ratio between the number of supporting rules and the number of true atoms. Also in this case, the heuristic is "balanced", it takes into account both the atom and its complement.

Moreover, it is defined as a refinement of the heuristic h_{UT} (i.e., $A <_{h_{UT}} B \Rightarrow A <_{h_{DS}} B$). In this way, h_{DS} keeps the same nice behaviour as the well-assessed h_{UT} on NP problems like 3SAT and Blocks-World, where h_{UT} proved to be very effective [10]; while, as we will see in Section 5 it sensibly improves on h_{UT} on hard 2QBF problems (Σ_2^P-complete). Given two atoms A and B, $A <_{h_{DS}} B$ holds if:

1. $A <_{h_{UT}} B$, or
2. $B \not<_{h_{UT}} A$ and $DS(A) < DS(B)$

where $DS(Q) = SuppRules(Q)/(True(Q) + 1) + SuppRules(\texttt{not}\ Q)/(True(\texttt{not}\ Q) + 1)$.

The heuristic selects a $<_{h_{DS}}$-maximum atom A; A is taken positive or negative, by comparing the degree of supportedness of A and $\texttt{not}\ A$.

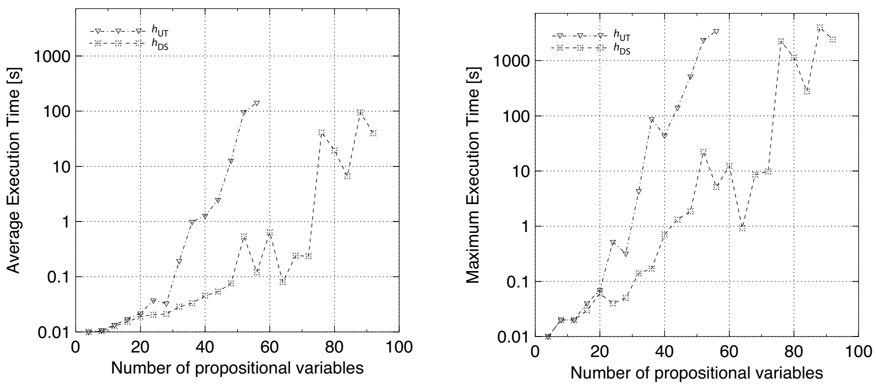

Figure 2: Running Times on Random QBF problems

Example 5 Reconsider Example 4 with the interpretation being $I = \{w\}$. We get $ext(a) = \{w, a, b, \texttt{not}\ z, \texttt{not}\ c\}$ and $ext(d) = \{w, d, a, b, \texttt{not}\ z, \texttt{not}\ c, \texttt{not}\ e, \texttt{not}\ f\}$ We get $DS(a) = 3/3$, since $w \leftarrow a$; $a \vee z \leftarrow w$ and $b \vee z \leftarrow w$ are supporting rules for the three true non-HCF atoms w, a, b. On the other hand, $DS(d) = 4/3$, since $w \leftarrow d$ is an additional supporting rule for the same three true non-HCF atoms w, a, b. Therefore $a <_{h_{DS}} d$ holds. Indeed, d is a better choice than a, as it leads immediately to an answer set. a would require at least another choice, and choosing e or f would cause a failing model check. ∎

5 Comparing h_{UT} vs h_{DS}: Experiments

The proposed heuristic aims at improving the performance of DLV on hard (Σ_2^P-complete) ASP programs. While there are many experimental works benchmarking ASP systems on NP-complete problems, less is available for Σ_2^P-complete problems. We resort to 2QBF, the canonical problem, and one of the few Σ_2^P-hard problems for which some transition phase results are known [5, 12].

The problem here is to decide whether a quantified Boolean formula (QBF) $\Phi = \exists X \forall Y \phi$, where X and Y are disjoint sets of propositional variables and $\phi = C_1 \vee \ldots \vee C_k$ is a 3DNF formula over $X \cup Y$, is valid. The transformation from 2QBF to disjunctive logic programming is based on a reduction used in [6]. The propositional disjunctive logic program $\mathcal{P}_\phi$ produced by the transformation requires $2 * (|X| + |Y|) + 1$ predicates (with one dedicated predicate w).

Our benchmark instances were generated following recent works presented in the literature that describe transition phase results for QBFs [5, 12], see [15], for a thorough discussion. In all generated instances, the number of $\forall$-variables in any formula is the same as the number of $\exists$-variables (that is, $|X| = |Y|$) and each disjunct contains at least two universal variables. Moreover, the number of clauses is $((|X| + |Y|)/2)^{0.5}$.

	DLV	Quantor	Semprop	yQuaffle	SSolve
$Robot$	32 (100%)	10 (31%)	17 (53%)	21 (67%)	22 (69%)
$Random$	108 (100%)	14 (13%)	96 (89%)	55 (51%)	103 (95%)
$Tree$	2 (100%)	2 (100%)	2 (100%)	2 (100%)	2 (100%)
$Pan - Kph$	1 (100%)	1 (100%)	1 (100%)	1 (100%)	1 (100%)
$Total$	143 (100%)	27 (19%)	116 (81%)	79 (55%)	128 (90%)

Table 1: Number (and percentage) of instances solved within the allowed time.

Experiments were performed on a PentiumIV 1500 MHz machine with 256MB RAM running SuSe Linux 9.0. Time measurements have been done using the `time` command shipped with SuSe Linux 9.0.

We generated 100 random QBF instances for each problem size. The results of our experiments are displayed in Fig. 2 (the horizontal axis depicts $|X| + |Y|$). For each instance, we allowed a maximum time of 7200 seconds (two hours). The line of a system stops whenever some problem instance was not solved within this time limit. On the vertical axis, we report, respectively, the average and the maximum running time in seconds over the 100 instances of the respective size, in logarithmic scale.

It is evident that the new heuristic h_{DS} outperforms the heuristic h_{UT} in these experiments. Heuristic h_{UT} stopped at size 56; while heuristic h_{DS} solved all instances up to size 92. To solve an instance of size 56, h_{UT} took 3455.85s; while h_{DS} required at most 5.13s and 0.12s on average for instances of this size. Heuristic h_{UT} could not solve a 60-variables instance within 2 hours of cpu time; while h_{DS} took at most 12.41s and 0.64s on average for solving them.

6 ASP vs QBF Solvers

One may wonder whether ASP systems are competitive with other systems on Σ_2^P/Π_2^P-hard problems. Currently it seems that QBF solvers are the most prominent (and efficient) non-ASP-systems capable of this task. These systems can also solve harder problems.

In order to answer this question, we carry out an experimental comparison of DLV (with h_{DS}) with QBF solvers which performed best at the 2004 QBF evaluation [22]: SSolve [11] (in the version used at the 2004 QBF evaluation), Semprop [18] (version v01.06.04), Quantor [3] (version

1.3pre1), and yQuaffle [25] (version 093004). We use two different sets of benchmarks, which we describe in the following sections.

6.1 QBF Evaluation

The first group of benchmarks constitute the Σ_2^P- and Π_2^P-complete QBF instances of the 2004 QBF evaluation, which we obtained from the qbflib web site [22]. These instances are of four different kinds: Narizzano-robot, hard random-instances, Letz-tree, and Pan-Kph, see [22] for details. In total, our suite contains 143 2QBF instances: 2 Letz-tree, 32 Narizzano-robot, 1 Pan-Kph, and 108 random instances. For DLV we used a standard propositional encoding as described in Sec. 5, while for the QBF systems we used the qDimacs format.

The experiments were performed on the same machine as those of Sec. 5. For each instance, we have allowed a maximum running time of 1800 seconds (30 minutes). Again, we have limited the process size to 256MB to avoid swapping.

Table 1 displays, for each system, the number and percentage of instances which have been solved under the resource limitations. Summarizing, DLV could solve all instances (100%) and is therefore clearly the best among the compared systems. Among the QBF solvers, SSolve and Semprop could solve 90% and 81% of the instances, respectively, and thus performed significantly better than both yQuaffle (55%) and Quantor (19%). It should be noted that practically all of the unsolved instances for Quantor are due to excessive memory consumption, while for the other systems they are due to time-outs. Indeed, we have analyzed the runs of Quantor on some of its unsolved instances: Within the first minute of CPU time (several minutes real-time due to swapping), it had typically allocated around 500MB, and after two minutes (around half an hour in real time) more than 700MB, still growing. We then aborted the test to avoid a machine lock-up.

	SSolve	Semprop	Quantor	yQuaffle
# solved	128	116	27	79
solver avg	43,86s	68,18s	4,74s	55,24s
DLV avg	38,95s	43,50s	10,94s	49,05s

Table 2: Average time on instances solved by QBF solvers.

While SSolve and Semprop did significantly better on the random instances than on the ”Narizzano-robot”instances, the situation is inverse for Quantor and yQuaffle.

Also when comparing the average runtime between DLV and each QBF solver (on the instances solved by the respective system), DLV usually has an edge, as Table 2 shows. The average runtime of DLV is larger only when compared to Quantor; but given that this comparison is based only on 19% of all instances, this is not significant.

6.2 Strategic Companies

The second group of benchmarks is made up of randomly generated instances for the *Strategic Companies* problem, as defined in [4]. We use the same DLV program and generation method as in [15].

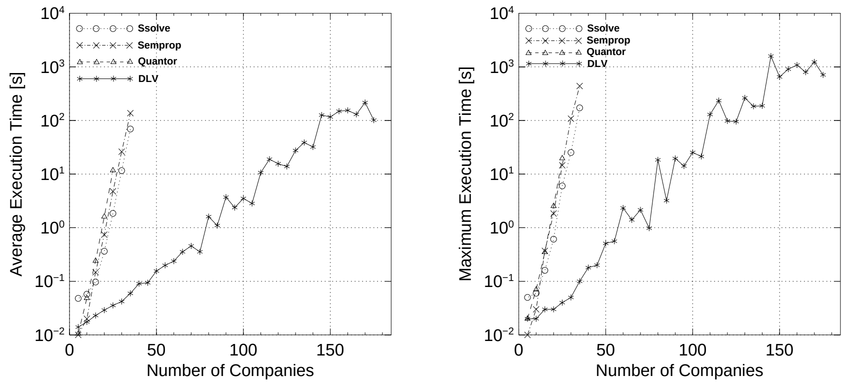

Figure 3: Average (left) and maximum (right) timings for Strategic Companies

Here, we generated tests as in [15] with 20 instances each size for m companies ($5 \leq m \leq 200$), $3m$ products, 10 uniform randomly chosen $contr\_by$ relations per company (up to four controlling companies), and uniform randomly chosen $prod\_by$ relations (up to four producers per product). The problem is deciding whether two fixed companies (1 and 2, w.l.o.g.) are strategic.

For the QBF solvers we have produced the following formula: $\exists c_1, \ldots, c_n : \forall c'_1, \ldots, c'_n : ((I \wedge NE) \rightarrow (R \wedge R') \wedge c_1 \wedge c_2)$ where I stands for $(c'_1 \rightarrow c_1) \wedge \ldots \wedge (c'_n \rightarrow c_n)$, NE for $\neg((c'_1 \leftrightarrow c_1) \wedge \ldots \wedge (c'_n \leftrightarrow c_n))$, R for $\bigwedge_{i=1}^{m}((\bigwedge_{c_j \in O_i} c_j) \rightarrow c_i) \wedge \bigwedge_{i=1}^{n}(\bigvee_{g_i \in C_j} c_j)$ (O_i contains the controlling companies of c_i, while C_j contains the companies producing good j. R' is defined analogous to R on the primed variables.

Unfortunately this formula is not in CNF, as required by the qDimacs format. In order to avoid a substantial blowup of the formula by a trivial normalization, we have used the tool *qst* of the *traquasto* suite [26], which transforms a formula into qDimacs by introducing additional "label variables" to avoid exponential formula growth. However, these additional variables are existentially quantified at the inner level and thus would turn the formula above into a 3QBF. To avoid this, we consider the negated formula $\forall c_1, \ldots, c_n : \exists c'_1, \ldots, c'_n : \neg((I \wedge NE) \rightarrow (R \wedge R') \wedge c_1 \wedge c_2)$, that is a 2QBF.

In the same experimental setting as before, we obtained the results of Fig. 3.[3] It is evident that DLV scales significantly better than the QBF solvers (note that the vertical axis is logarithmic), and can solve all instances of up to 175 companies, while the QBF solvers fail to solve instances of 40 companies.

7 Conclusion

In this paper, we have presented a new heuristic method for ASP systems, which is geared towards hard problems on the second level of the polynomial hierarchy. We have implemented this method in the state-of-the-art system DLV, and showed that it is beneficial for the performance of the system.

[3]yQuaffle is excluded, as some input files failed abnormally.

To our knowledge, this is the first work dealing with heuristics dedicated for Σ_2^P/Π_2^P-hard ASP programs. Previous optimization techniques for this segment have been concerned with the model checking portion, which is important for this class of problems. In our work, we attack the problem earlier, in the model generation phase, and can therefore cut on the model checks. Importantly, this heuristics has been incorporated in a way such that the benign behavior on NP/co-NP programs w.r.t. the previous heuristic of DLV is maintained.

We experimentally verified that the new heuristic significantly improves the DLV system performance on randomly generated hard 2QBF instances, reducing the average execution time, enlarging the maximum solvable size of these problems for a fixed time limit.

We also carried out an experimental comparison of DLV (with the heuristic described in this paper) with the best QBF solvers of the 2004 QBF evaluation [22]: SSolve [11], Semprop [18], Quantor [3], and yQuaffle [25]. This comparison was done on benchmark instances of the 2004 QBF evaluation, and Strategic Companies. In both cases, DLV could outperform the QBF solvers, often significantly. We therefore conclude that ASP systems are currently the best choice for solving Σ_2^P/Π_2^P-complete problems. All benchmark data is available at `http://www.dlvsystem.com/examples/tests-sigma2-2005.tar.gz`.

References

[1] C. Baral. *Knowledge Representation, Reasoning and Declarative Problem Solving*. CUP, 2002.

[2] R. Ben-Eliyahu and R. Dechter. Propositional Semantics for Disjunctive Logic Programs. *AMAI*, 12:53–87, 1994.

[3] A. Biere. Resolve and Expand. 2004. SAT'04.

[4] M. Cadoli, T. Eiter, and G. Gottlob. Default Logic as a Query Language. *IEEE TKDE*, 9(3):448–463, 1997.

[5] M. Cadoli, A. Giovanardi, and M. Schaerf. Experimental Analysis of the Computational Cost of Evaluating Quantifi ed Boolean Formulae. In *AI*IA 97*, pp. 207–218, Italy, 1997.

[6] T. Eiter and G. Gottlob. On the Computational Cost of Disjunctive Logic Programming: Propositional Case. *AMAI*, 15(3/4):289–323, 1995.

[7] T. Eiter and G. Gottlob. The Complexity of Logic-Based Abduction. *JACM*, 42(1):3–42, 1995.

[8] T. Eiter, G. Gottlob, and H. Mannila. Disjunctive Datalog. *ACM TODS*, 22(3):364–418, 1997.

[9] W. Faber, N. Leone, and G. Pfeifer. Pushing Goal Derivation in DLP Computations. In *LPNMR'99*, pp. 177–191.

[10] W. Faber, N. Leone, and G. Pfeifer. Experimenting with Heuristics for Answer Set Programming. In *IJCAI 2001*, pp. 635–640.

[11] Rainer Feldmann, Burkhard Monien, and Stefan Schamberger. A Distributed Algorithm to Evaluate Quantifi ed Boolean Formulae. In *Proceedings National Conference on AI (AAAI'00)*, pp. 285–290, Austin, Texas, 30-August 3 2000. AAAI Press.

[12] I. Gent and T. Walsh. The QSAT Phase Transition. In *AAAI*, 1999.

[13] T. Janhunen, I. Niemelä, P. Simons, and J.-H. You. Partiality and Disjunctions in Stable Model Semantics. In *KR 2000, 12-15*, pp. 411–419.

[14] C. Koch, N. Leone, and G. Pfeifer. Enhancing Disjunctive Logic Programming Systems by SAT Checkers. *Artifi cial Intelligence*, 15(1–2):177–212, 2003.

[15] N. Leone, G. Pfeifer, W. Faber, T. Eiter, G. Gottlob, S. Perri, and F. Scarcello. The DLV System for Knowledge Representation and Reasoning. *ACM TOCL*, 2005. To appear.

[16] N. Leone, R. Rosati, and F. Scarcello. Enhancing Answer Set Planning. In *IJCAI-01 Workshop on Planning under Uncertainty and Incomplete Information*, pp. 33–42, 2001.

[17] N. Leone, P. Rullo, and F. Scarcello. Disjunctive Stable Models: Unfounded Sets, Fixpoint Semantics and Computation. *Information and Computation*, 135(2):69–112, 1997.

[18] R. Letz. Lemma and Model Caching in Decision Procedures for Quantified Boolean Formulas. In *TABLEAUX 2002*, pp. 160–175, Denmark, 2002.

[19] Yuliya Lierler and Marco Maratea. Cmodels-2: SAT-based Answer Set Solver Enhanced to Non-tight Programs. In *LPNMR-7*, LNCS, pp. 346–350. 2004.

[20] V. Lifschitz. Answer Set Planning. In *ICLP'99*, pp. 23–37.

[21] F. Lin and Y. Zhao. ASSAT: Computing Answer Sets of a Logic Program by SAT Solvers. In *AAAI-2002*, AAAI Press / MIT Press.

[22] M. Narizzano and A. Tacchella. QBF Solvers Evaluation page, 2002. `http://www.qbflib.org/qbfeval/index.html/`.

[23] Jussi Rintanen. Constructing Conditional Plans by a Theorem-Prover. *JAIR*, 10:323–352, 1999.

[24] P. Simons, I. Niemelä, and T. Soininen. Extending and Implementing the Stable Model Semantics. *Artificial Intelligence*, 138:181–234, 2002.

[25] L. Zhang and S. Malik. Towards a Symmetric Treatment of Satisfaction and Conflicts in Quantified Boolean Formula Evaluation. In *CP 2002*, pp. 200–215, NY, USA, 2002.

[26] Michael Zolda. Comparing Different Prenexing Strategies for Quantified Boolean Formulas. Master's thesis, TU Wien, 2005.