Heuristics for Hard ASP Programs *

Wolfgang Faber† and **Nicola Leone** and **Francesco Ricca**
Department of Mathematics, University of Calabria, I-87030 Rende (CS), Italy
Email: {faber,leone,ricca}@mat.unical.it

Abstract

We define a new heuristic h_{DS} for ASP, and implement it in the (disjunctive) ASP system DLV. The new heuristic improves the evaluation of Σ_2^P/Π_2^P-hard ASP programs while maintaining the benign behaviour of the well-assessed heuristic of DLV on NP problems. We experiment with the new heuristic on QBFs. h_{DS} significantly outperforms the heuristic of DLV on hard 2QBF problems. We compare also the DLV system (with the new heuristic h_{DS}) to three prominent QBF solvers. The results of the comparison, performed on instances used in the last QBF competition, indicate that ASP systems can be faster than QBF systems on Σ_2^P/Π_2^P-hard problems.

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 [Lifschitz, 1999]. 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) [Eiter *et al.*, 1997]. Thus, ASP is strictly more powerful than SAT-based programming, as it allows us to solve even 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 the complexity class Σ_2^P or Π_2^P, and can be naturally encoded in ASP [Baral, 2002; Leone *et al.*, 2001].

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 [Janhunen *et al.*, 2000; Koch *et al.*, 2003]). In particular, we are not aware of any work concerning heuristics for Σ_2^P/Π_2^P-hard ASP programs.

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, aiming at improving the evaluation of Σ_2^P/Π_2^P-hard ASP programs, but maintaining the benign behaviour of the heuristic of DLV on NP problems. We experimentally compare h_{DS} against the DLV heuristic on hard 2QBF instances, showing a clear benefit. We also experiment the competitiveness of ASP w.r.t. QBF solvers on hard problems, indicating that ASP systems are very competitive with QBF systems on Σ_2^P/Π_2^P-hard problems.

2 Answer Set Computation and Heuristics

We first recall the main steps of the computational process performed by ASP systems, in particular the DLV system, which will be used for the experiments.

An answer set program $\mathcal{P}$ in general contains variables. The first step of a computation of an ASP system eliminates these variables, then the following algorithm is invoked:

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

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 GL-transformed program and outputs the model, if so. *IsAnswer-*

*This work was partially supported by the European Commission under projects IST-2002-33570 (INFOMIX) and IST-2001-37004 (WASP), and by FWF (Austrian Science Funds) under projects P16536-N04 and P17212-N04.

†Funded by APART of the Austrian Academy of Sciences.

Set(I) returns True if the computation should be stopped and False otherwise.

The function DetCons() computes an extension of I with the literals that can be deterministically inferred (or the set of all literals $\mathcal{L}$ upon inconsistency). 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.

The heuristic h_{UT}, proposed in [Faber *et al.*, 2001] is currently employed in DLV. It is mostly based on the number of *UnsupportedTrue (UT)* atoms (called MBTs in [Faber *et al.*, 2001]), i.e., atoms which are true in the current interpretation but miss a supporting rule, trying to minimize UT atoms and hence more likely arrive at supported models.

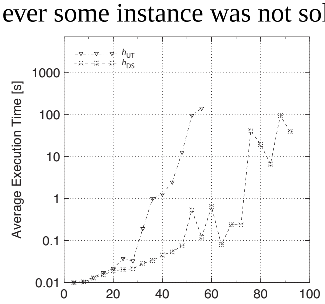
For hard ASP programs (i.e., non-HCF programs [Ben-Eliyahu and Dechter, 1994] – they express Σ_2^P-complete problems under brave reasoning), supported models are often not answer sets. Moreover, answer-set checking is computationally expensive (co-NP), and may consume a large portion of the resources needed for computing an answer set.

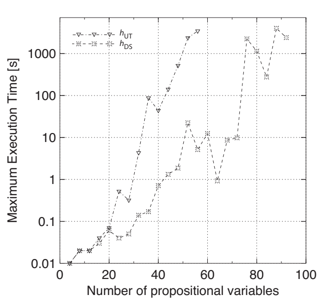
We therefore propose the new heuristic h_{DS}, which tries in addition to maximize the *degree of supportedness*, the average number of supporting rules for non-HCF true atoms. Intuitively, if all true atoms have many supporting rules in a model M, then the elimination of an atom from the model would violate many rules, and it becomes less likely to find a subset of M which is a model of the reduct $\mathcal{P}^M$, disproving that M is an answer set. We define h_{DS} 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 behaviour of the well-assessed h_{UT} on NP problems while, as we will see in Section 3, it sensibly improves over h_{UT} on hard 2QBF problems (Σ_2^P-complete).

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

We generated randomly a data set of 100 2QBF formulas, following [Gent and Walsh, 1999], and used the ASP encoding described in [Leone *et al.*, 2005].

Experiments were performed on a PentiumIV 1500 MHz machine with 256MB RAM running SuSe Linux 9.0. For every instance, we allowed a maximum running time of 7200 seconds (two hours). The results of our experiments are displayed in the following graphs, in which a line stops whenever some instance was not solved within the time limit.





It is clear that the new heuristic h_{DS} outperforms the heuristic h_{UT} in these experiments, advancing the "maximum solvable-size" from 56 up to size 92, and reducing the average execution times of the smaller instances.

4 ASP vs QBF Solvers

The main goal of this paper is to improve the performance of ASP systems for problems located at the second level of the polynomial hierarchy. One may wonder whether, on such Σ_2^P/Π_2^P-hard problems, ASP systems are competitive with other AI systems, like the QBF solvers. In order to give a first answer to this question, we have also performed a comparison with QBF solvers **Quantor** [Biere, 2004], **Semprop** [Letz, 2002], and **yQuaffle** [Zhang and Malik, 2002] on the set of all Σ_2^P- and Π_2^P-complete QBF formulas of the last QBF competition. The results below report the number of instances solved within 660s and show that DLV (with heuristic h_{DS}) generally outperformed the QBF solvers.

	$DLV(h_{DS})$	$Quantor$	$Semprop$	$yQuaffle$
$Robot$	27 (84%)	10 (31%)	13 (41%)	17 (53%)
$Random$	108 (100%)	14 (12%)	90 (83%)	53 (49%)
$Tree$	2 (100%)	2 (100%)	2 (100%)	2 (100%)
KPH	1 (100%)	1 (100%)	1 (100%)	1 (100%)
$Total$	137 (96%)	26 (18%)	105 (73%)	72 (50%)

References

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

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

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

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

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

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

[Janhunen *et al.*, 2000] 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.

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

[Leone *et al.*, 2001] 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.

[Leone *et al.*, 2005] 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.

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

[Lifschitz, 1999] V. Lifschitz. Answer Set Planning. In *ICLP'99*, pp. 23–37.

[Zhang and Malik, 2002] 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.