The KR System dlv: Progress Report, Comparisons and Benchmarks

Thomas Eiter
Institut für Informatik
Universität Gießen
Arndtstrasse 2, D-35392
Gießen, Germany

Nicola Leone
Institut für
Informationssysteme
TU Wien
A-1040 Wien, Austria

Cristinel Mateis
Institut für
Informationssysteme
TU Wien
A-1040 Wien, Austria

Gerald Pfeifer
Institut für
Informationssysteme
TU Wien
A-1040 Wien, Austria

Francesco Scarcello
ISI-CNR, c/o DEIS
Università della Calabria
I-87030 Rende, Italy

Abstract

`dlv` is a knowledge representation system, based on disjunctive logic programming, which offers front-ends to several advanced KR formalisms. The system has been developed since one year at the Technical University of Vienna in an ongoing project funded by the Austrian Science Funds. After a report on the current state of the art in the implementation of `dlv` and of its application front-ends, the paper compares `dlv` with other knowledge representation systems. Both a qualitative and a quantitative comparison are carried out. The first compares the representational power of the systems (intended as the ability to represent problems in a natural and simple fashion). The latter compares the performances of the systems. The `dlv` system turns out to be very powerful from the knowledge representation side and, on the other hand, quite good also in computational power.

Keywords: Implemented KR&R Systems: Reports, Comparisons, Evaluations.

1 INTRODUCTION

The extension of logic programming by disjunction and true negation has been pointed out as a necessary requirement for knowledge representation [16, 2]. This view has been confirmed by results which prove that without disjunction, the expressive capability of logic programming is limited such that relevant problems can not be expressed [12, 7]. Moreover, disjunction has been recognized as an important feature for a declarative KR language, which allows to express knowledge in a simple and natural way. For example, the rule

```
genotype(P,T1) v genotype(P,T2) :-
parent(C,P), heterozygot(C,T1,T2).
```

from a blood group knowledge base[1] may express that the genotype of a parent P of a person C is either $T1$ or $T2$, if C is heterozygot with types $T1$ and $T2$.

After a body of work on the theoretical foundations, more recently implementations of nonmonotonic reasoning systems have been tackled [19, 9, 1, 23]. The `dlv` system is an effort in this vein, which aims at providing a KR tool based on disjunctive logic programming (DLP) under the stable model semantics [15, 16, 21] and, for true negation, under the answer set semantics [16]. The system supports front-ends to KR applications, and can serve as a declarative knowledge programming language.

The general approach to the system has been described in [13]. In this paper, we describe the state of the art of the implementation and present a quantitative and qualitative comparison to similar KR systems. Our results look promising and show that the current implementation is competitive among the fastest systems. Moreover, by its supreme expressive capability, `dlv` embodies the most convenient and powerful KR system among those considered here.

2 SYSTEM OVERVIEW

The system architecture is shown in Figure 1. The internal system language is function-free DLP with true negation [16] and integrity constraints. The kernel is an efficient engine for computing all or some stable models of a program. Various frontends, i.e. translators for specific applications into the internal language, have been developed on top of it (cf. Section 3). A Graphical User Interface (GUI) provides convenient access to both the system and its front-ends.

The heart of the system is the *Query Processor* (QP). It controls the system execution and – together with the integrated frontends – it performs pre-processing of the input and post-processing of the generated models, respectively. Upon startup, QP reads the (nonground) program and passes it to the *Rules and Graphs Handler* (RGH), which splits it into subprograms. These subprograms are then, together with facts from a database (Oracle or Access files), submitted to the *Intelligent Grounding Module*, which efficiently generates a subset of the grounded input program which has the same stable models, but is much smaller in general. (For stratified programs, for example, the Grounding already computes the single stable model.)

QP then again invokes RGH, which generates two partitionings into components of the grounded program. They are used by the *Model Generator* (MG) and the *Model Checker*, and enable a modular program evaluation. This often yields a tremendous speedup.

Then, the heart of the computation is performed by the Model Generator and Model Checker. Roughly, the Model Generator produces some "candidates" stable models, whose stability is verified by the Model Checker.

The generation of the stable models of a program LP relies on a monotonic operator $\mathcal{W}_{LP}$ [17] that extends to the disjunctive case the well-founded operator of [26]. It is defined in terms of a proper notion of *unfounded set*. Intuitively, an unfounded set for a disjunctive program $\mathcal{P}$ w.r.t. an interpretation I is a set of positive literals that definitely cannot be derived from $\mathcal{P}$ assuming the facts in I [17].

Briefly, the algorithm works as follows. $\mathcal{W}^{\omega}_{LP}(\emptyset)$ is first computed, which is contained in every stable model. If $\mathcal{W}^{\omega}_{LP}(\emptyset)$ is a total model, it is returned as the unique stable model. Otherwise, moving from $\mathcal{W}^{\omega}_{LP}(\emptyset)$ towards the stable models, a conjunction of literals (called *possibly-true conjunction* in [17]), the truth of which allows to infer new atoms, is assumed true. The computation proceeds by iteratively applying the inflationary version of

[1] Courtesy of Dietmar Seipel.

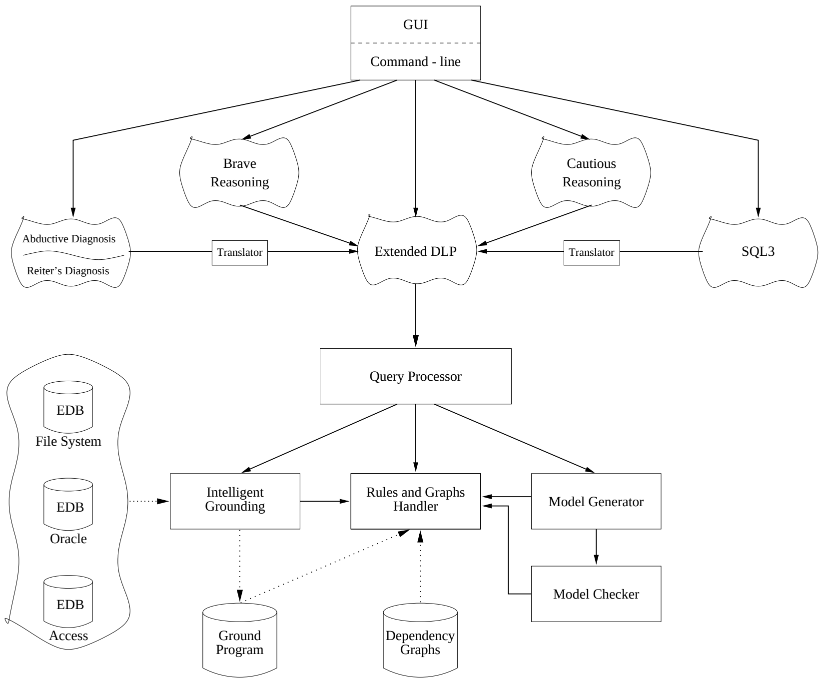


Figure 1: The System Architecture of `dlv`

the immediate consequence operator, until either a total model of LP is reached or two contradictory literals are generated. In the latter case, backtracking is performed; in the former case, the Model Checker is invoked.

The Model Checker (MC) verifies whether the model at hand is a stable model. This task is very hard in general, because checking the stability of a model is well known to be co-NP-complete (cf. [12]). However, recent studies [3, 4, 17] unveiled that minimal model checking — the hardest part of stable model checking— can be efficiently performed for the relevant class of *head-cycle-free* (*HCF*) programs [3].

The MC satisfies the above complexity bounds. Indeed (a) it terminates in polynomial time on every HCF program; and (b) it always runs in polynomial space and single exponential time. Moreover, even on general (non-HCF) programs, the MC limits the inefficient part of the computation to the modules that are not HCF (Note that it may well happen that only a very small part of the program is not HCF).

Finally, once a stable model has been found, control is returned to QP, which performs post-processing and possibly invokes the MG to look for further models.

More details on the basic algorithms implemented in the system can be found in [17, 13, 10]. For space reasons, we can not describe here ongoing refinements.

3 THE APPLICATION FRONT-ENDS

3.1 THE NATIVE `dlv` INTERFACE

The native interface of the system offers the full power of the internal language of `dlv`: function-free disjunctive logic programs extended by allowing both integrity constraints and true negation [16]. As will be evident in Section 4, the presence of these constructs makes our language well suited to easily represent a wide range of problems in a natural and highly declarative way.

A `dlv` rule has the following form:

$$L_1 \vee \ldots \vee L_k \quad :- \quad L_{k+1}, \ldots, L_m, \; not\ L_{m+1}, \ldots, not\ L_n$$

where $1 \leq k \leq m \leq n$ and each L_i is a literal, i.e., an atom $a(t_1, \ldots, t_o)$ or its "classical" negation $\neg a(t_1, \ldots, t_o)$, a being the predicate name of that atom, and each t_i $(1 \leq i \leq o)$ being either a constant or a variable. The convention on names is the one of Prolog.

A *constraint* is like a rule with $k = 0$ and $n \geq 1$. A `dlv` program P is a pair $< R, C >$, where R is a set of `dlv` rules and C a set of constraints. A *stable model* of P is any consistent answer set of R [16] satisfying all constraints in C.

3.2 BRAVE AND CAUTIOUS REASONING

The frontends for brave and cautious reasoning are an extension of the native interface described above. In addition to the program, the user also specifies a query, which is essentially the body of a rule followed by a question mark:

$$b_1, \ldots, \; b_m, \; \texttt{not} \; b_{m+1}, \ldots, \; \texttt{not} \; b_n \; \texttt{?}$$

where each b_i $(1 \leq i \leq n)$ is a ground literal.

The brave reasoning frontend is invoked by the `-FB` command line option. If the query Q evaluates to true in at least one stable model M, then the system replies: "Q is bravely true, evidenced by M", i.e., returns a stable model that witnesses this fact; otherwise the answer is "Q is bravely false".

Similarly, the cautious frontend is invoked by `-FC`, and the system answers "Q is cautiously true" if and only if Q is true in all stable models. In case of negative answer, we get: "Q is cautiously false, evidenced by M", where M is a stable model witnessing the falsity of Q.

3.3 DIAGNOSIS FRONTENDS

`dlv` provides two kind of diagnosis frontends: the *abductive diagnosis* frontend [20, 11], and the *Reiter's diagnosis* frontend [22].

Our frontend for *abductive diagnosis* currently supports three different modes: general diagnosis, where all diagnoses are computed, subset minimal diagnosis[2], and single failure diagnosis. These modes are invoked by the command line options `-FD`, `-FDmin` and `-FDsingle`, respectively.

The diagnostic theory obeys the syntax described in Section 3.1. Hypothesis (resp. observations) are lists of atoms (resp. literals) separated by a dot (`.`) and are stored in files whose names carry the extension `.hyp` (resp. `.obs`).

Basically, the diagnostic frontend works as follows: After reading the input, disjunctive rules for guessing possible diagnosis candidates are generated from the hypotheses, while the observations become constraints that forbid the generation of inconsistent diagnoses. In the case of subset minimal diagnosis, further rules are added for pruning non-minimal solutions. Finally, the grounding (and subsequently the MG etc.) are invoked, and for each reported stable model, the corresponding diagnosis is returned.

Example 1 (Network Diagnosis) Suppose that machine a is online in the computer network below, but we observe it cannot reach machine e. Which machines are offline? This can be easily modelled as a diagnosis problem, where the theory is

$$\begin{array}{lll} reaches(X, X) & :- & node(X), not\ offline(X). \\ reaches(X, Z) & :- & reaches(X, Y), \\ & & connected(Y, Z), \\ & & not\ offline(Z). \end{array}$$

and the observations and hypotheses are respectively

```
not offline(a).  not reaches(a,e).
```

[2] For positive non-disjunctive theories only.

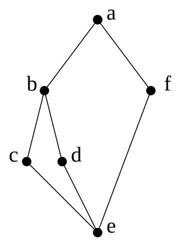


Figure 2: Computer network

and

```
offline(a).offline(b).offline(c).
offline(d).offline(e).offline(f).
```

The frontend for *consistency-based diagnosis* as described by Reiter [22] works similarly. The command line options are `-FR`, `-FRmin` and `-FRsingle`, respectively.

3.4 FURTHER FRONTENDS

The system has a SQL3-frontend based on the current working draft for the standard. It supports recursive query processing such as the bill-of-materials problem.

Other frontends are currently under way. We have started work on a frontend for inheritance reasoning based on the approach in [14]; a frontend for extended logic programs with preferences as in [5] will be tackled in the near future.

4 COMPARISONS AND BENCHMARKS

In this section, we compare `dlv` with two prominent and effective recent systems for nonmonotonic reasoning, DeReS [9] and *Smodels* [19] on a diversified collection of benchmark problems. Basically, we follow the methodology for testing nonmonotonic systems suggested in TheoryBase [8].

4.1 OVERVIEW OF THE COMPARED KR SYSTEMS

Smodels [19] has been recently recognized as the best system to compute stable models for normal (i.e., non-disjunctive) logic programs [25]. Upon request, the system returns the stable models of the input program. *Smodels* supports a kind of brave reasoning by allowing the user to specify a set of atoms that must belong to every returned stable model. It is also possible to specify a maximum number of desired stable models. Thus, *Smodels* also supports nondeterministic semantics: by selecting only one stable model, we look for any nondeterministically chosen stable model of the given program.

DeReS [9] is a KR system implementing classical default logic. It supports several basic reasoning tasks such as testing whether extensions exist, and finding one or all extensions. If the input theory amounts to a logic program, DeReS computes its stable models and supports queries on membership of an atom in some or all stable models. *DeReS* handles only propositional theories. Thus, no uniform problem representation for varying background facts is supported. A preliminary instantiation is needed to emulate programs with variables.

4.2 ENCODING OF BENCHMARK PROBLEMS

Next we present several problems, drawn from different domains, for comparing the KR systems. Some examples have been repeatedly used for system evaluation. However, we also present some novel benchmark problems which are challenging for KR Systems.

Ancestor. Given the parent relationship $parent(\_,\_)$, find the genealogy tree of each person in the database.

This is a classical deductive database example, which has been used to compare the performances of several DDB systems. We consider it to test the suitability of the systems for handling DDB applications and double recursion. For each system, the encoding looks like:

```
ancestor(X,Y) :- parent(X,Y).
ancestor(X,Y) :- ancestor(X,U),
                 ancestor(U,Y).
```

Graph 3-colorability. Given a graph, represented by facts of the form `node(_)` and `edge(_,_)`, assign each node one of three colors such that no two adjacent nodes have the same color.

3-colorability is a classical NP-complete problem, which has been used as a benchmark problem for DeReS [19] and *Smodels* [9]. In *dlv*, the problem can be encoded in a very easy and natural way by means of disjunction and constraints:

$$\texttt{col(X,r)} \vee \texttt{col(X,g)} \vee \texttt{col(X,b)} :- \texttt{node(X)}.$$

```
:- edge(X,Y), col(X,C), col(Y,C).
```

The disjunctive rule nondeterministically chooses a color for each node X in the graph; the constraint enforces that the choices are legal.

In *Smodels*, the problem can be easily encoded as well, but we cannot use either constraints or disjunction, which must be simulated by unstratified negation.

$$\begin{array}{lll}
col(X,r) & :- & node(X),\ not\ col(X,g), \\
 & & not\ col(X,b). \\
col(X,g) & :- & node(X),\ not\ col(X,r), \\
 & & not\ col(X,b). \\
col(X,b) & :- & node(X),\ not\ col(X,g), \\
 & & not\ col(X,r). \\
bad & :- & edge(X,Y),\ col(X,C), \\
 & & col(Y,C),\ not\ bad.
\end{array}$$

Evidently, `dlv` allows for a more elegant problem representation.

The encoding used for DeReS is Δ^3_{col} as described in [8]. For space reasons, we can not list this encoding here.

Prime Implicants. Find the prime implicants of a propositional CNF formula $\Phi = \bigwedge_{i=1}^{n}(d_{i1} \vee \ldots \vee d_{ic_i})$, where the $d_{i,j}$ are literals over the variables $x_1, \ldots, x_m$.

Recall that a *prime implicant* of Φ is a consistent conjunction I of literals such that $I \models \Phi$ and for no subformula I' of I we have $I' \models \Phi$. As easily seen, a prime implicant exists if and only if Φ is satisfiable. Thus, finding a prime implicant of Φ subsumes the satisfiability problem for Φ.

The representation in `dlv` is natural and simple. Here, true negation in the language plays a key role. we have a literal translation of Φ into a `dlv` program P:

$$\mathtt{d_{11}} \vee \ldots \vee \mathtt{d_{1c_1}}. \qquad \ldots \qquad \mathtt{d_{n1}} \vee \ldots \vee \mathtt{d_{nc_n}}.$$

It is easy to see that the stable models of P (cf. Section 3) correspond 1-1 to the prime implicants of Φ.

Smodels has no true negation nor disjunction. We thus rewrite P to a normal logic program as follows.First, each literal $\neg x_i$ is replaced by a new atom x'_i, and a constraint

$$\mathtt{bad} \quad :- \quad \mathtt{x_i},\ \mathtt{x'_i},\ \mathtt{not\ bad}.$$

is added, which enforces that no stable model can contain both x_i and x'_i, for any $1 \leq i \leq m$. Then any disjunctive rule corresponding to the i-th clause $(1 \leq i \leq n)$

$$\mathtt{d_{i1}} \vee \ldots \vee \mathtt{d_{ic_i}}.$$

is replaced by c_i non-disjunctive rules

$$\begin{array}{lll}
d_{il} & :- & not\ d_{i1}, \ldots,\ not\ d_{i(l-1)}, \\
 & & not\ d_{i(l+1)}, \ldots,\ not\ d_{ic_i}.
\end{array}$$

where $1 \leq l \leq c_i$.

For DeReS, the input theory for Φ has been generated by taking the ground instance for the *Smodels* program, which is then translated into default rules using the translation which maps a rule $\mathtt{A} :- \mathtt{B_1}, \ldots, \mathtt{B_n}, \mathtt{not\ C_1}, \ldots, \mathtt{not\ C_k}$ into the default $\frac{B_1 \wedge \cdots \wedge B_n\ :\ notC_1, \ldots, notC_k}{A}$.

Strategic Companies. The strategic companies problem is from [7]; it is, to the best of our knowledge, the only Σ_2^P-complete KR problem from the business domain. No experimental results for any Σ_2^P-complete KR problems are known.

Briefly, a holding owns companies, each of which produces some goods. Moreover, several companies may have jointly control over another company. Now, some companies should be sold, under the constraint that all goods can be still produced, and that no company is sold which would still be controlled by the holding after the transaction. A company is *strategic*, if it belongs to a *strategic set*, which is a minimal set of companies satisfying these constraints. Those sets are expressed by the following natural program:

```
strategic(C1) v strategic(C2) :-
           produced_by(P,C1,C2).
strategic(C) :-
           controlled_by(C,C1,C2,C3),
           strategic(C1),
           strategic(C2),
           strategic(C3).
```

Here strategic(C) means that C is strategic, produced_by($P, C1, C2$) that product P is produced by companies $C1$ and $C2$, and controlled_by($C, C1$, $C2, C3$) that C is jointly controlled by $C1, C2$ and $C3$; we have adopted here from [7] that each product is produced by at most two companies and each company is jointly controlled by at most three other companies.

The problem is to find out the set of all strategic companies (i.e., under brave reasoning, for which C the fact strategic(C) is true).

Note that this problem can not be expressed by a fixed normal logic program uniformly on all collections of facts produced_by($p, c1, c2$) and controlled_by($c, c1$, $c2, c3$) (unless NP = Σ_2^P, an unlikely event). The only system capable of this problem besides ours is DeReS, provided that a default theory is constructed ad hoc by instantiation for an instance.

For DeReS, we have translated the above dlv program into a default theory based on the translation tr_1 [18] (which is appropriate for our purposes). We have also tried a variant of the translation which uses CWA defaults.

Smodels and similar systems can not express **Strategic Companies**. They must use, by current methods, ad hoc programs for particular instances, which need in general exponential time for construction.

4.3 EXPERIMENTAL RESULTS

The results of our experiments are displayed in Figure 6. They were run on a SUN Ultra 1/143 under Solaris 2.5.1, using gcc 2.7.2. The inputs for **Prime Implicants** and **Strategic Companies** were randomly generated; all systems received the same random input.

Since DeReS cannot handle non-ground theories, we generated ground theories using the TheoryBase tool [8]. The time for this grounding is not included in the figures, as it is difficult to assess. It is based on disk saving, which takes on average about one minute total time for larger theories, of which about 15% is CPU time. Thus, the actual total time for DeReS is higher.

For **Ancestor**, we used instances in which the ancestor relation is as depicted in Figure 3. The x-axis of the diagram in Figure 6 shows the number n of people, while the y-axis shows the total time. It appears that both dlv and *Smodels* show a similar time behavior, but *Smodels* has a steeper increase; for 250 people, dlv is doing about 25% faster than *Smodels*. Note that **Ancestor** is mainly a grounding and forward chaining benchmark; as DeReS has no variables, it is not considered here.

For **Graph 3-Colorability**, we have considered two classes of graphs, namely *ladder graphs* and *simplex graphs* (see Figure 4 and Figure 5). The results (see Figure 6) show that *Smodels* and dlv have a similar behavior, but the increase for dlv is slower; for ladder graphs, it runs about 40% faster and for simplex graphs, about 50% faster; moreover, it scales up better to large graphs. DeReS is com-

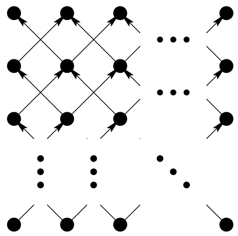


Figure 3: Parent relation

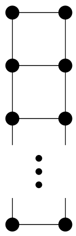


Figure 4: Ladder Graph

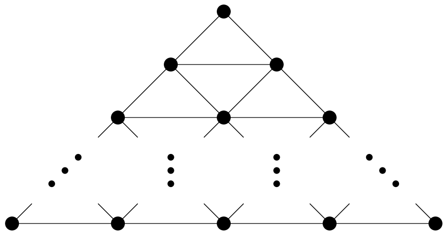


Figure 5: Simplex Graph

parable to dlv, but could only handle instances up to about 1100 nodes resp. 650 nodes in total, while the system capacity was exceeded on larger ones. If auxiliary files generated by TheoryBase (whose purpose is not evident) are removed, then DeReS is running considerably slower on both problems.

For **Prime Implicants**, we have randomly generated CNF formulas on n variables using the system of Selman and Kautz [24], where the ratio between the number of clauses and the number of variables is 4.3 (these tend to be hard SAT instances).

Here, *Smodels* has unbeatable performance which nearly parallels the x-axis. dlv has reasonable performance up to $n = 25$, but then the time increases rapidly. (Note that we do not have a smooth curve, since the instances are random.) This may be explained by current weaknesses of dlv in backtracking in the process of model generation. The implementation of intelligent backtracking strategies and modular evaluation is under way (and will be completed soon), which will improve the performance significantly. Surprisingly, DeReS has very long running time with steep increase, which is not plotted here. We have no explanation for this.

For **Strategic Companies**, we have generated two kinds of tests, each with instances for n companies and $3n$ products.

For the first test, whose results are reported in the left graph, we uniform randomly chose the produced_by and the controlled_by relations, where each company is controlled by a single joint consortium obeying the obvious constraint that no company can control itself. In the second test, reported in the right graph, each company is controlled by one to five companies (Two consortia have to have at least one member in common), where again the actual number of companies is uniform randomly chosen. On average there are 1.5 controlled_by relations per company.

As mentioned above, **Strategic Companies** is a problem which cannot be represented in *Smodels*, and in DeReS only via preliminary instantiation. The results for dlv are interesting. Up to roughly 50 companies, the evaluation in the first case is very fast, and then the running time increases similar as in case of **Prime Implicants**. In the second case performances are somewhat worse and it seems that very hard instances can be met even with fewer companies. We believe that further similar tests may prove useful to better understand the complex nature of Σ_2^P-hard problems.

For DeReS, we observed very long running times with steep increase, and only small instances could be handled. The results are not included in the figure.

5 DISCUSSION AND CONCLUSION

The encoding of benchmark problems in the various KR systems, reported in the previous section, highlights quite clearly that `dlv` is the most powerful system from the perspective of knowledge representation power.

All problems we considered are represented in `dlv` in a simpler and more natural fashion than in the other systems. This result confirms the usefulness of extending logic programming by disjunction, true negation and integrity constraints. Roughly, disjunction and integrity constraints allowed an elegant and highly declarative encoding of **3-Colorability**, and true negation played a key role for the representation of **Prime Implicants**. Moreover, the encoding of **Strategic Companies** was possible only thanks to the presence of disjunction (indeed, this problem can not be solved by *Smodels*).

Even if `dlv` is young (the project started only one year ago), and still misses some optimization techniques (that are currently under development), on average it compared positively to the other systems also from the viewpoint of efficiency. Indeed, `dlv` showed the best performances on **Ancestor** and on **3-Coloring** for simplex graphs; it was evidently slower only in the case of **Prime Implicants** (and, even in this case, only compared to *Smodels*).

Importantly, `dlv` was the only system able to handle **Strategic Companies**. This example is important since it definitely requires the use of disjunction in knowledge representation, and can not be expressed without. Thus, the experimental results that we obtained are also interesting from the respect what running time behaviors the need for disjunction as a modelling construct may invoke. As it appeared, up to some realistic instance size, the problem could be handled in reasonable time. This indicates that the expressive power of our `dlv` system is quite convenient: We may express the problem on all possible inputs, without bothering about restrictions on the problem instance under which we *can* write a program (like in *Smodels*), while the actual running time on instances is still acceptable. Further experiments in this direction will be taken.

The experiments show that DeReS can be profitably used when it can exploit the encoding provided by TheoryBase. Indeed, on both "3-coloring" examples (where we used the TheoryBase encoding for DeReS) the system was comparable to `dlv`. However, without the TheoryBase encoding (that was not available on **Prime Implicants** and **Strategic Companies**) the system was very slow. Moreover, even on **3-Coloring**, it was not able handle large instances.

In our opinion, this is because DeReS is a propositional system and has been thought to be used with TheoryBase. *Smodels* confirmed to be a very solid implementation of the stable model semantics for normal (non-disjunctive) programs. On **Ancestor** and both **3-Coloring** its performances were comparable to `dlv` (only slightly worse), and *Smodels* was the most efficient system on **Prime Implicants**. The latter problem evidences that `dlv` is not yet efficient on all domains and that the ongoing work on *modular evaluation* and *intelligent backtracking* is definitely needed to make the system very solid.

Currently, we are improving the system along the lines of [10], in particular, the Model Generator, which will boost the performance on problems like **Prime Implicant** and problems with unstratified negation. Another important ongoing activity concerns the development of efficient data structures that will support linear time implementation of our disjunctive operators. Moreover, we are working also on linguistic extensions, adding weak constraints to the system [6]. This will increase the expressive power of the system beyond Σ_2^P, which enables us to represent problems of higher complexity (in particular, diagnostic examples). Furthermore, these constraints allow for a more natural problem representation in many cases. In addition, we are pursuing the development of further KR frontends, which exploit dlv as a declarative knowledge representation system.

For up-to-date information and a download of the system as well as some further examples, please visit the project homepage at http://www.dbai.tuwien.ac.at/proj/dlv/.

Acknowledgements

The work was partially supported by *FWF (Austrian Science Funds)* under the project P11580-MAT "A Query System for Disjunctive Deductive Databases" and by the Italian MURST under the project for Italy-Austria cooperation *Advanced Formalisms and Systems for Nonmonotonic Reasoning*.

The authors are deeply indebted to Wolfgang Faber, who designed and implemented the automated benchmark tool for dlv, and helped in conducting the experiments.

References

[1] C. Aravindan, J. Dix, and I. Niemelä. Dislop: A research project on disjunctive logic programming. *AI Communications*, 10(3/4):151–165, 1997.

[2] C. Baral and M. Gelfond. Logic Programming and Knowledge Representation. *Journal of Logic Programming*, 19/20:73–148, 1994.

[3] R. Ben-Eliyahu and R. Dechter. Propositional Semantics for Disjunctive Logic Programs. *Annals of Mathematics and Artificial Intelligence*, 12:53–87, 1994.

[4] R. Ben-Eliyahu and L. Palopoli. Reasoning with Minimal Models: Efficient Algorithms and Applications. In *Proceedings Fourth International Conference on Principles of Knowledge Representation and Reasoning (KR-94)*, pages 39–50, 1994.

[5] G. Brewka. Preferred Answer Sets. In *Proceedings Workshop on Logic Programming and Knowledge Representation, at ILPS '97*, Port Jefferson, October 1997.

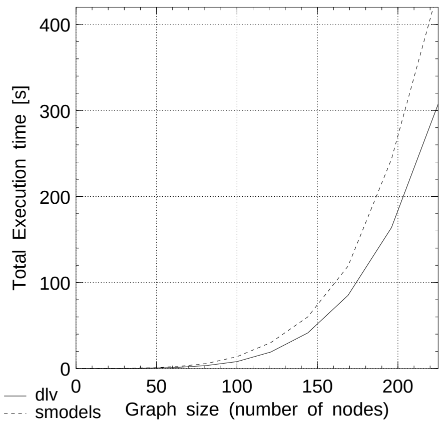
[6] F. Buccafurri, N. Leone, and P. Rullo. Strong and Weak Constraints in Disjunctive Datalog. In *Proceedings of the 4th International Conference on Logic Programming and Non-Monotonic Reasoning (LPNMR '97)*, Dagstuhl, Germany, July 1997.

[7] M. Cadoli, T. Eiter, and G. Gottlob. Default Logic as a Query Language. *IEEE Transactions on Knowledge and Data Engineering*, 9(3):448–463, May/June 1997.

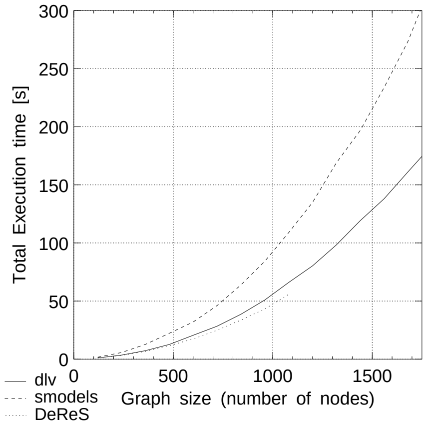
[8] P. Cholewiński, V. W. Marek, A. Mikitiuk, and M. Truszczyński. Experimenting with Nonmonotonic Reasoning. In *Proceedings of the 12th International Conference on Logic Programming (ICLP-95)*, pages 267–281, Shonan Village Center, Japan, June 1995. MIT Press.

[9] P. Cholewiński, V. W. Marek, and M. Truszczyński. Default Reasoning System DeReS. In *Proceedings of International Conference on Principles of Knowledge Representation and Reasoning (KR '96)*, Cambridge, Massachusetts, USA, 1996.

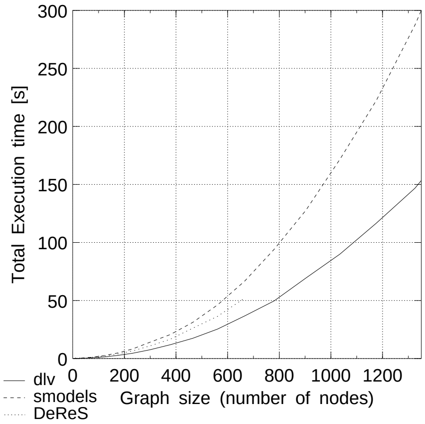
[10] S. Citrigno, T. Eiter, W. Faber, G. Gottlob, C. Koch, N. Leone, C. Mateis, G. Pfeifer, and F. Scarcello. The dlv System: Model Generator and Application Frontends. In *Proceedings of the 12th Workshop on Logic Programming (WLP '97), Research Report PMS-FB10*, pages 128–137, München, Germany, September 1997. LMU München.



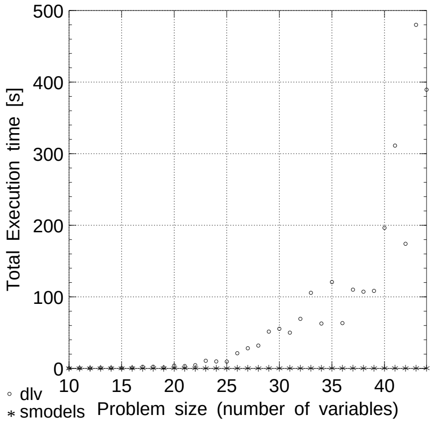
Results for **Ancestor**



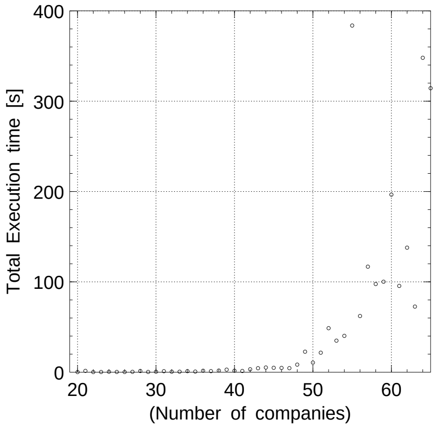
Results for **3-COL Ladder Graphs**



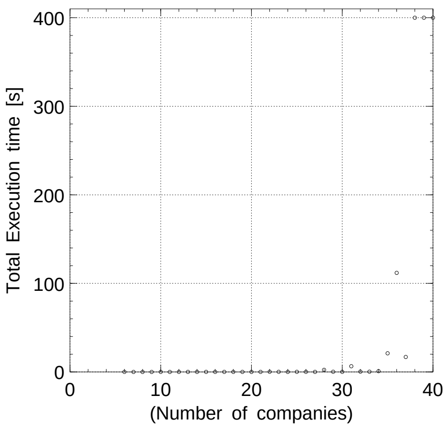
Results for **3-COL Simplex Graphs**



Results for **Prime Implicants**



Results for **Strategic Companies**



Results for **Strategic Companies**

Figure 6: Experimental Results

[11] T. Eiter, G. Gottlob, and N. Leone. Abduction From Logic Programs: Semantics and Complexity. *Theoretical Computer Science*, 189(1–2):129–177, December 1997.

[12] T. Eiter, G. Gottlob, and H. Mannila. Disjunctive Datalog. *ACM Transactions on Database Systems*, 22(3):315–363, September 1997.

[13] T. Eiter, N. Leone, C. Mateis, G. Pfeifer, and F. Scarcello. A Deductive System for Nonmonotonic Reasoning. In J. Dix, U. Furbach, and A. Nerode, editors, *Proceedings of the 4th International Conference on Logic Programming and Nonmonotonic Reasoning (LPNMR '97)*, number 1265 in Lecture Notes in AI (LNAI), Berlin, 1997. Springer.

[14] M. Gelfond and A. Gabaldon. From Functional Specifications to Logic Programms. In *Proceedings of the International Logic Programming Symposium (ILPS '97)*, pages 355–369, 1997.

[15] M. Gelfond and V. Lifschitz. The Stable Model Semantics for Logic Programming. In *Logic Programming: Proceedings Fifth Intl Conference and Symposium*, pages 1070–1080, Cambridge, Mass., 1988. MIT Press.

[16] M. Gelfond and V. Lifschitz. Classical Negation in Logic Programs and Disjunctive Databases. *New Generation Computing*, 9:365–385, 1991.

[17] N. Leone, P. Rullo, and F. Scarcello. Disjunctive stable models: Unfounded sets, fixpoint semantics and computation. *Information and Computation*, 135(2):69–112, June 1997.

[18] W. Marek and M. Truszczyński. Stable semantics for logic programs and default theories. In *Proceedings NACLP-89*. MIT Press, 1989.

[19] I. Niemelä and P. Simons. Efficient Implementation of the Well-founded and Stable Model Semantics. In *Proceedings of the 1996 Joint International Conference and Symposium on Logic Programming*, pages 289–303, Bonn, Germany, Sept. 1996.

[20] D. Poole. Explanation and Prediction: An Architecture for Default and Abductive Reasoning. *Computational Intelligence*, 5(1):97–110, 1989.

[21] T. C. Przymusinski. Stable Semantics for Disjunctive Programs. *New Generation Computing*, 9:401–424, 1991.

[22] R. Reiter. A Theory of Diagnosis From First Principles. *Artificial Intelligence*, 32:57–95, 1987.

[23] D. Seipel and H. Thöne. DisLog – A System for Reasoning in Disjunctive Deductive Databases. In *Proceedings International Workshop on the Deductive Approach to Information Systems and Databases (DAISD'94)*, 1994.

[24] B. Selman and H. Kautz, 1997. URL `ftp.research.att.com/dist/ai`.

[25] M. Truszczyński. Automated reasoning with non-monotonic logics (invited talk). In *Proceedings of the 4th International Conference on Logic Programming and Non-Monotonic Reasoning (LPNMR '97)*, pages 112–114, 1997.

[26] A. van Gelder, K. Ross, and J. Schlipf. The Well-Founded Semantics for General Logic Programs. *Journal of the ACM*, 38(3):620–650, 1991.