Default Reasoning System DeReS

Paweł Cholewiński
Victor W. Marek
Mirosław Truszczyński

Department of Computer Science
University of Kentucky
Lexington, KY 40506-0046

{pawel|marek|mirek}@cs.engr.uky.edu

Topic areas: Implemented KR&R systems, nonmonotonic logics

Keywords: automated reasoning, default logic, logic programming, stable models, algorithms, benchmark problems

Correspondence:

Mirosław Truszczyński
Department of Computer Science
University of Kentucky
Lexington, KY 40506-0046
mirek@cs.engr.uky.edu

Abstract

In this paper, we describe an automated reasoning system, called DeReS. DeReS implements default logic of Reiter by supporting several basic reasoning tasks such as testing whether extensions exist, finding one or all extensions (if at least one exists) and querying if a formula belongs to one or all extensions. If an input theory is a logic program, DeReS computes stable models of this program and supports queries on membership of an atom in some or all stable models. The paper contains an account of our preliminary experiments with DeReS and a discussion of the results. We show that a choice of a propositional prover is critical for the efficiency of DeReS. We also present a general technique that eliminates the need for some global consistency checks and results in substantial speedups. We experimentally demonstrate the potential of the concept of stratification for making automated reasoning systems practical.

1 Introduction

The area of nonmonotonic logics was originated about twenty years ago [Rei78, Rei80, MD80, McC80] in an effort to build efficient knowledge representation formalisms. Since then solid theoretical foundations of nonmonotonic logics have been established. The efforts of the past two decades culminated in several research monographs [Eth88, Bes89, Bre91, MT93] describing major nonmonotonic systems: default logic, logic programming with negation as failure, autoepistemic logics and circumscription.

It was expected that nonmonotonic logics would be able to model formal aspects of commonsense reasoning and that their computational properties would be better than those of classical logic. Computational complexity results obtained in recent years were, however, discouraging. Decision problems associated with nonmonotonic reasoning, even when restricted to the propositional case, are very complex. For example, in the case of logic programming with stable model semantics they are NP-complete or co-NP-complete. In the case of default logic, they are Σ_2^P-complete or Π_2^P-complete.

However, the complexity results do not necessarily disqualify nonmonotonic logics as a computational knowledge representation mechanism. Decision problems in classical logic are also highly computationally complex — NP- and co-NP-complete in the propositional case. These tasks become even more complex in the case of quantified Boolean formulas [MS72]. In the same time, recent experimental results on satisfiability indicate that propositional logic can serve as a computational tool, and is capable of handling large collections of variables and clauses [SLM92, SKC93]. Complexity results alone are also insufficient to determine whether classical logic or nonmonotonic logics are better suited as the basis for an automated reasoning system. In particular, the results of [CDS94, GKPS95] show that higher computational complexity of nonmonotonic logics may be offset by more concise encodings than those possible with propositional logic.

Hence, despite the tremendous progress in our understanding of the basic principles of nonmonotonic logics, a major question of their viability as a computational tool remains unresolved. Systematic implementation and experimentation effort is necessary to provide us with better insights into the computational properties of nonmonotonic logics. Despite evident importance of experimental studies of nonmonotonic logics, there has been little work reported in the literature. While several algorithms were published and some implementations described [MW88, BNN$^+$93, BNN$^+$94, BEP94, NS95] the results are far from conclusive. This state of affairs can be attributed to the lack of systematic experimentation with implemented systems. A possible reason for this was the absence of commonly accepted benchmarking systems. In particular, a convincing and useful for experimentation model of a random logic program has yet to be proposed.

To address the issue of benchmarks we developed a system, called TheoryBase, generating propositional logic programs and default theories [CMMT95]. This system allows the users to systematically generate interesting families of logic programs and default theories for experimentation. A key feature of TheoryBase is that it provides an identifier (label) for every theory it generates. This can greatly facilitate experimental comparisons between implemented systems. TheoryBase has already been used by several research

groups around the world. Most notably, Niemelä and Simmons [Nie95, NS95] used the ideas behind TheoryBase to study their implementation of a system for computing stable models of logic programs.

In this paper we report on the project to design and implement a program, Default Reasoning System DeReS, that supports basic automated reasoning tasks for default logic [Rei80] and for logic programming with stable model semantics [GL88]. We describe basic features of DeReS. We also present experimental results obtained by running DeReS on default theories encoding graph problems generated by TheoryBase[1].

Our current version of DeReS is built around the notion of *stratification* [ABW88, Cho95c, Cho95a]. Stratification allows us to use a *divide-and-conquer* approach when computing extensions. An original default theory is partitioned into several smaller subtheories, called *strata*. The extensions of the original theory are then reconstructed from the extensions of its strata. The notion of stratification considered here is a relaxed version of the original concept as introduced in [ABW88]. In particular, a theory stratified in our sense may possess no extension (stable model) or, if it does, not necessarily a unique one. In the paper, we show that stratification leads to dramatic speedups. Thus, whenever possible, problems should be encoded by default theories so that to maximize the effects of stratification.

In the paper we also study the effects of different propositional theorem provers on the efficiency of DeReS. We observe that full theorem provers, which check for global consistency when deciding whether a theory proves a formula, result in performing prohibitive amount of redundant computation. A weaker notion of a *limited prover*, sound but not complete, can also be used to correctly implement DeReS and results in substantial improvements in time performance.

Our results show that there are classes of theories that DeReS can handle very efficiently. However, if stratification does not yield a partition of an input theory into small strata, the efficiency of DeReS may be poor. In this context, it is interesting to relate our work to the work of Niemelä and Simmons [NS95]. The two implementations are difficult to compare as they attack different aspects of the same problem. While our research focused on techniques to exploit stratification and limit the number of global consistency checks, Niemelä and Simmons studied techniques to deal with individual strata. It seems that the next generation implementations of nonmonotonic systems, in order to be effective in a large range of different applications, must combine the techniques developed in both projects.

The paper is organized as follows. In the next section, we describe DeReS. We briefly describe basic properties of stratification, introduce limited provers and present the benchmark problem generator, TheoryBase. Then, in Section 3 we show how to use these tools in our implementation of DeReS. We also present and discuss the results of our experimentation. The last section contains conclusions.

[1]To date, both systems: TheoryBase and DeReS, have been installed and tested in Bern, Ithaca, Karlsruhe, Lexington, Riverside and San Diego. They are currently available by anonymous ftp from al.cs.engr.uky.edu. The file names are: `TheoryBase.tar.gz` and `DeReS.tar.gz` in the directory `/cs/software/logic`.

2 Overview of DeReS

DeReS is a comprehensive software package for nonmonotonic reasoning. The focus of DeReS is on automated reasoning with default logic and logic programming (although some other nonmonotonic formalisms are also supported). The programs are written in C and run under UNIX operating system. The package was developed on a Sun SPARC-station 20 with the SunOS 5.4 operating system. Nevertheless, no special features of this environment are used and DeReS can be installed on any machine running SYSV or BSD version of UNIX. The main components of DeReS are:
user interface — the module facilitating interactive work with stored default theories,
scanner — the module accepting default theory, checking for errors, and building internal data structures,
default engine — a library of routines for reasoning with a given default theory,
provers — the module providing several propositional theorem provers that can be called by the default engine module,
graphical interface — an optional graphical interface for displaying the progress of computation and the results.

DeReS computes extensions for finite propositional default theories. There are no syntactic restrictions on formulas which can occur in the input. In particular, DeReS can compute the list of all extensions for a given default theory or check whether a given default theory has an extension. It also outputs basic time characteristics for each solved query – the total time spent processing the query and the number of calls to propositional provability routine. The user communicates with DeReS via its shell, which accepts user commands and starts desired tasks.

We will now focus on the main aspects of DeReS. We will also provide a short overview of TheoryBase. Two main questions that we study are:
(1) What is the effect of propositional provers on the efficiency of DeReS?
(2) What is the role of stratification in efficient implementations of nonmonotonic logics?
These questions are further expanded in Sections 2.1 and 2.2. We discuss TheoryBase in Section 2.3. Section 3 contains the description of the experiments and our findings.

2.1 Propositional provers

The prover module of DeReS system is used as a propositional provability oracle by all reasoning routines. Since the computational complexity of basic decision problems in nonmonotonic reasoning is very high, special care is needed to design efficient provers, or to design techniques to use provers more efficiently. DeReS is equipped with the following propositional provability procedures:
(1) **Full prover** — sound and complete propositional tableaux theorem prover,
(2) **Limited prover** — local provability propositional tableaux theorem prover (sound but not complete),
(3) **df-lookup** — table lookup method for disjunction-free theories.

Full prover, implemented using the propositional tableaux method, was provided to allow DeReS to work with arbitrary propositional theories. Limited prover is also based on the tableaux method. It differs from the full prover in that it does not, in general, perform global consistency checks. Finally, a *df-lookup prover* is a table lookup method applicable for *disjunction-free* theories. A default theory (D, W) is disjunction-free if all formulas in W, all prerequisites, justifications and conclusions of defaults in D are conjunctions of literals. For disjunction-free theories provability problems can be decided by determining membership of a literal a in a set of literals T (assuming that T does not contain contradictory literals — in such a case, the df-lookup returns **true**). Clearly, the df-lookup can be implemented in time $O(1)$, which results in dramatic performance improvement. The class of disjunction-free default theories is simple yet quite powerful. In particular, it subsumes the class of extended logic programs.

We will now describe the idea behind a limited prover. Although this approach does not improve the worst case complexity of the reasoning algorithms it usually yields significant speedups. Let $\mathcal{L}$ be any propositional language. Consider a propositional theory T contained in $\mathcal{L}$. Let φ be a formula in $\mathcal{L}$ and let $Var(\varphi)$ denotes the set of propositional variables in the formula φ.

A theory T proves a formula φ ($T \models \varphi$) in standard propositional logic if either the information contained in T and concerning the propositional variables occurring in φ allows us to derive φ, or if T is inconsistent.

Definition 2.1 *Let T be a set of propositional formulas. The* incidence graph G_T *is a simple undirected graph $G_T = (T, E)$ such that for any $\varphi, \psi \in T$*

$$\{\varphi, \psi\} \in E \quad \textit{if and only if} \quad Var(\varphi) \cap Var(\psi) \neq \emptyset$$

By T_φ we denote the set of vertices of the connected component of G_T which contains φ.

Intuitively, local provability of φ by a theory T means that φ is entailed by the part of T consisting of formulas containing variables relevant to φ, and not because T contains inconsistent data. A careful formalization of this idea yields the following definition.

Definition 2.2 *A theory T* locally proves *a formula φ (denoted $T \models_{loc} \varphi$) if $(T \cup \{\varphi\})_\varphi \setminus \{\varphi\} \models \varphi$*

Let us consider now standard propositional provability routines, such as tableaux or resolution based algorithms. Suppose we want to use them to answer the query $T \models_{loc} \varphi$. The modifications required in these algorithms to implement local provability are very simple. All one has to do is to block expanding the tableaux tree or the resolution list by formulas which have no common variables with the variables mentioned in the structure (the tree or the list) so far. This way the prover remains restricted to the component of $G_{T \cup \{\varphi\}}$ containing the formula φ.

Clearly, local provers are faster than full provers as, in general, they do not consider all formulas in the input theory. The exact amount of savings depends on the structure of the underlying theory T and the input formula φ or, more precisely, on the structure of the incidence graph $G_{T\cup\{\varphi\}}$. We gather some immediate consequences of Definition 2.2 in the following theorem.

Theorem 2.1 *For arbitrary theory T and formula φ the following holds:*
(1) If $T \models_{loc} \varphi$ then $T \models \varphi$.
(2) If T is consistent then $T \models \varphi$ if and only if $T \models_{loc} \varphi$.
(3) If $G_{T\cup\{\varphi\}}$ is connected then $T \models \varphi$ if and only if $T \models_{loc} \varphi$.
(4) $T \models \varphi$ if and only if either T is inconsistent or $T \models_{loc} \varphi$. □

Theorem 2.1 yields a technique to compute extensions using a local prover directly. Let (D, W) be an input default theory. Assume that the task is to find an extension (or generate all of them). To simplify the discussion, we will assume that each default in D has at least one justification. Under this assumption, (D, W) has either a unique inconsistent extension (if W is inconsistent), or all extensions of (D, W) are consistent (if W is consistent). Hence, a single call to a full propositional prover is sufficient to decide which of the two possibilities holds. If it is the first one, we stop. So, from now on we will assume that W is consistent.

It is well known that extensions of a default theory (D, W) are determined by the sets of formulas of the form $W \cup c(U)$, where $U \subseteq D$ and $c(U)$ consists of the consequents of defaults in U ([MT93]). In order to verify whether $W \cup c(U)$ generates an extension, DeReS has to decide several provability problems of the form $W \cup c(U) \vdash \varphi$. If $W \cup c(U)$ is consistent, the local prover can be used instead of the full prover.

To search through all such sets, DeReS considers systematically all subsets U of D, starting with $U = \emptyset$. Then $W \cup c(U) = W$ which, by our assumption, is consistent. Hence, the local prover can be used to decide if $W \cup c(U)$ generates an extension. Each next set of defaults, say U', is obtained from the current set U by removing or adding one default, say d. In the first case, the resulting set of formulas $W \cup c(U')$ is consistent and the local prover can be used to decide whether it generates an extension. In the second case, $W \cup c(U')$ is consistent if and only if $W \cup c(U) \not\vdash \neg c(d)$. Since $W \cup c(U)$ is consistent, this condition can be tested by a call to the local prover. If $W \cup c(U')$ is inconsistent, DeReS removes U', as well as all its supersets, from the search space and backtracks to $W \cup c(U)$. Otherwise, $W \cup c(U')$ is consistent and, again, the local prover can be used to test whether $W \cup c(U')$ is an extension [2].

2.2 Stratification

To improve efficiency of reasoning with default logic one can check if a given theory belongs to a subclass for which reasoning can be performed faster. A common technique

[2]Eliminating global consistency checks by local provers is analogous to ommitting occur check in logic programming.

is to stratify a theory. Stratification consists of partitioning a given theory into a sequence of smaller theories for which extensions can be computed faster. This approach was studied in the cases of logic programming [CH82, ABW88, VG88, Prz88] and autoepistemic logic [Gel87, MT91]. Some results of applying this approach to default logic were obtained by Etherington [Eth88] and Kautz and Selman [KS89]. DeReS uses a relaxed variant of stratification adopted for general default theories. That is, no syntactic restrictions on formulas appearing in defaults are imposed. Instead more restrictive conditions on dependencies between defaults are required. See [Cho95c] and [Cho95a] for details on stratification for default theories.

Given an input default theory (D, W) and its stratification $D_1, \ldots, D_k$, DeReS builds extensions gradually, dealing with consecutive strata "one at a time". DeReS assumes a fixed linear order (topological sort) of strata. The nodes of the search tree are pairs (D_i, E) where D_i is the current stratum and E is an extension for the default theory $(D_1 \cup \ldots \cup D_{i-1}, W)$. The children of this node are of the form (D_{i+1}, E') where D_{i+1} is the next stratum and E' is an extension for the default theory $(D_1 \cup \ldots \cup D_i, W)$. It is known [Cho95c] that all such extensions are exactly the extensions for (D_i, E). DeReS uses an additional (local) search tree to find extensions for each default theory (D_i, E).

2.3 TheoryBase

To test and experiment with an automated reasoning system, one requires a diversified collection of benchmark theories. In our case, to test DeReS, we needed default theories and logic programs. They should be easily generated, realistic and meaningful. They should be easy to reproduce and disseminate to facilitate experimental comparisons of different implementations of nonmonotonic reasoning systems. These requirements prompted another project – TheoryBase [CMMT95]. TheoryBase automatically generates families of default theories and logic programs. It is based on the observation that graph problems such as existence of colorings, kernels and hamilton cycles can be encoded as default theories and logic programs, with extensions (stable models) corresponding to combinatorial objects in question. Hence, the TheoryBase first generates a graph and then produces an encoding — a default theory or a logic program — of a combinatorial problem for this particular graph. Several such encodings were described in [CMMT95]. Most important of them are:
(1) existence of a kernel,
(2) existence of a coloring,
(3) existence of a hamiltonian cycle.
In order to generate graphs, TheoryBase uses the system, Stanford GraphBase, developed by Knuth [Knu93]. This software system can generate a large variety of graphs, each with a unique identifier. The system of Stanford GraphBase identifiers is extended in TheoryBase to default theories and logic programs. Each default theory and logic program generated by TheoryBase receives an identifier which allows for an easy reconstruction of the corresponding theory or program.

TheoryBase can easily be extended by new encodings of combinatorial problems as

default theories, logic programs, disjunctive logic programs, as well as propositional theories. Our intention is for TheoryBase to evolve into a commonly accepted benchmarking system to support research in automated reasoning.

3 Results

In this section we discuss some of the experiments that were run on DeReS. The test theories were generated by TheoryBase and their identifiers are provided here. Two main questions that we studied were (1) the impact of a propositional prover on the efficiency of DeReS and (2) the impact of stratification of the efficiency of DeReS. In our tests we focused on two tasks: to decide the existence of extensions, and to generate all extensions of an input default theory.

In the presentation of the results we use the following notation: $time_f$ denotes the CPU time for queries processed with full propositional tableaux, $time_l$ denotes the CPU time for queries processed with limited propositional tableaux, $time_a$ denotes CPU time for queries processed with df-lookup prover. Further, NCPP stands for the number of calls to propositional provability routine, CAND stands for the number of candidate theories tested for an extension. We denote the total number of extensions for the input theory by EXT. By D we denote the set of defaults of default theories used in experimentation. We set $N = |D|$. Finally, V and E denote the vertex set and the edge set of the graphs underlying default theories used in experimentation.

All times are given in seconds. A bar (–) symbol in tables indicates that the program was running for more than 2 hours without reaching the final answer and then was interrupted. To capture the reasoning time we measure the CPU time from the point when an input default theory is already represented and stored, together with its stratification, as a DeReS data structure to the point when the answer is returned.

3.1 Computing graph kernels with DeReS

We start our presentation of the experiments with DeReS with the problem of computing kernels in graphs. We considered the family of graphs $G_{n,m} = board(n, m, 0, 0, 5, 3, 1)$ from Stanford GraphBase. The graph $G_{n,m}$ is the graph of chess-knight moves on a wrapped $n \times m$ chessboard. Moreover, the edges are directed according to the lexicographical order on the pairs of coordinates for the chess-board squares.

To compute kernels in the graphs $G_{n,m} = board(n, m, 0, 0, 5, 3, 1)$ the encodings with identifiers $kernel.board\_n, m, 0, 0, 5, 3, 1\_$ were generated by TheoryBase.[3] In particular, we studied the graphs $G_{8,m}$. The size of these graphs grows proportionally to m. The same holds for the size of the TheoryBase encodings of the kernel existence problem for these graphs. We propose this sequence of default theories as a benchmark problem for non-monotonic reasoning.

[3]Since the theories are saved as UNIX files, the system of identifiers uses underscores instead of parentheses.

kernel.board_8,m,0,0,5,3,1_, single solution								
m	$\|V\|$	$\|E\|$	N	NCPP	CAND	$time_f$	$time_l$	$time_a$
2	16	64	112	1939	851	2.34	0.14	0.01
4	32	128	224	14804	6845	69.95	1.45	0.07
6	48	192	336	121249	56298	1532.87	16.49	0.57
8	64	256	448	308910	143677	–	53.40	1.45
10	80	320	560	1982796	921464	–	421.74	9.14

Table 1: Searching for a kernel in $board(8, m, 0, 0, 5, 3, 1)$.

kernel.board_8,m,0,0,5,3,1_, all solutions									
m	$\|V\|$	$\|E\|$	N	NCPP	CAND	$time_f$	$time_l$	$time_a$	EXT
2	16	64	112	3337	1473	4.03	0.24	0.02	2
4	32	128	224	65704	30016	365.59	6.70	0.32	6
6	48	192	336	421082	192175	5239.54	57.34	1.90	5
8	64	256	448	4130579	1888829	–	720.22	18.97	134
10	80	320	560	31630658	14466688	–	6819.35	141.74	267

Table 2: Computing all kernels in $board(8, m, 0, 0, 5, 3, 1)$.

In Table 1 we show times which DeReS needed to find one kernel in graphs $G_{8,m} = board(8, m, 0, 0, 5, 3, 1)$, $2 \leq m \leq 10$. In Table 2 we show the times required to compute all the kernels. In both cases time grows exponentially with the size of the underlying default theory. In the same time, this example illustrates that DeReS is capable to deal with default theories containing hundreds of defaults. It is due to the effective use of stratification in the encoding Δ^3_{ker}, which results in the partition of the theory into relatively small clusters. Consequently, the provability routine performs very efficiently and the answers are found within seconds.

Let us look more closely at times corresponding to different provability routines. In all cases, the total time required to decide all relevant provability problems using the df-lookup prover is proportional to the total number of calls to the prover. Hence, the time per call is constant. The time grows exponentially and has order $\Theta(3^{N/56})$ (recall that N is the number of defaults in D). That is, it grows at a much smaller rate than the theoretical bound $O(N^2\ 2^N)$ [MT93]. Similarly, the times $time_a(m)$ for finding one extension are of the order $\Theta(2^{N/56})$. These speedups are due to the fact that we were able to stratify input default theories into chains of clusters of defaults and construct extensions by considering several subproblems of small sizes instead of one large set set of defaults.

When tableau provers are used, the times are larger because more time is needed for each call to propositional provability. In the case of limited prover, the time required

kernel.b.board_4,m,0,0,5,3,1_, single solution								
m	$\|V\|$	$\|E\|$	N	NCPP	CAND	$time_f$	$time_l$	$time_a$
2	8	32	32	52931	2351	8.34	1.28	0.13
3	12	48	48	241868873	9814692	–	–	438.46
4	16	64	64	–	–	–	–	–

Table 3: Searching for a kernel in $board(4, m, 0, 0, 5, 3, 1)$.

for a single call grows proportionally with m which implies the total time of the order $\Theta(N\ 3^{N/56})$.

Full propositional prover scans the input theory $O(m)$ times per each query. Consequently, the time needed to solve each such query is $O(m^2)$ (quadratic in the size of theory), and the total time of finding all extensions is of the order $\Theta(N^2\ 3^{N/56})$.

These results show that efficiency of provers have substantial effect on the performance of DeReS. One general lesson is that local provers should be used instead of full provers, especially in view of the fact that local provers do not impose any restrictions on input theories. Although our results were obtained for a full and local tableaux provers, we believe the same speedups would be obtained if any other full prover is replaced by its local version. That is, consistency checks performed each time when the prover is called can and should be avoided.

Secondly, even though the theories that arise in this example are very simple, both local and full tableaux provers are outperformed by the df-lookup prover. This indicates that when using DeReS as a knowledge representation tool, it is useful to encode domain knowledge as a disjunction-free theory, as this allows DeReS to refer to the df-lookup prover when reasoning.

To discuss in more detail the effects of stratification, we will consider the same class of graphs but the kernel existence problem will be encoded differently. This time, we will use the encoding Δ^1_{ker}, also described in [CMMT95]. The corresponding theories generated by TheoryBase have identifiers $(kernel.b.board\_n, m, 0, 0, 5, 3, 1\_)$. Table 3 contains the experimental results. It is clear that DeReS performs much worse on these theories. The lack of good stratification for the theories $(kernel.b.board\_n, m, 0, 0, 5, 3, 1\_)$ is to blame. Hence, encoding domain knowledge in DeReS, as in any other knowledge representation language, requires some skill. In the case of DeReS, the same problem can be encoded efficiently (Δ^3_{ker}) and inefficiently (Δ^1_{ker}). A general lesson is that while encoding domain knowledge in DeReS, encodings that admit fine stratification should be used.

3.2 Coloring ladder graphs

In [NS95], the problem to find a 3-coloring of a ladder graph was proposed as a benchmark for testing non-monotonic reasoning systems. The ladder graph, L_n, is defined as

color3.board_n,2,0,0,1,0,0_, one solution								
n	$\|V\|$	$\|E\|$	N	NCPP	CAND	$time_f$	$time_l$	$time_a$
200	400	598	2994	7988	2594	1427.50	6.45	0.18
400	800	1198	5994	15988	5194	–	28.72	0.34
600	1200	1798	8994	23988	7794	–	68.74	0.50
800	1600	2398	11994	31988	10394	–	128.17	0.68
1000	2000	2998	14994	39988	12994	–	209.14	0.84

Table 4: Finding a coloring for $L_n = board(n, 2, 0, 0, 1, 0, 0)$.

follows. Its vertices are points (x, y) in the real plane such that, $x = 0, 1, \ldots, n$ and $y = 0, 1$. Two vertices are joined by an edge if the euclidean distance between them equals 1. Graph L_n can be generated using The Stanford GraphBase and has label $board(n, 2, 0, 0, 1, 0, 0)$. We use default theory $chrom_3(G)$ from [CMMT95] to encode 3-colorings for graph G. These encodings were generated using TheoryBase system and have TheoryBase labels $color3.board\_n, 2, 0, 0, 1, 0, 0\_$.

DeReS times obtained for the query *find one extension* are shown in Table 4. In this case, the number of calls to propositional provability routine and the number of tested candidates grow linearly with respect to the size (the number of vertices) of the graph. All three provers which we use behave in the same way as in the previous example. That is, df-lookup prover runs in constant time per call, limited tableau prover in linear time and full tableau prover in quadratic time. Consequently, the time for finding an extension (coloring of the ladder graph L_n) is $\Theta(n)$ for df-lookup prover, $\Theta(n^2)$ for limited tableau and $\Theta(n^3)$ for full tableau method.

Hence, when efficient provability method is used, DeReS can find a 3-coloring for a ladder graph within a second even when the corresponding encoding contains almost 15 thousand defaults. The success of DeReS is again due to the structure of the encoding used — it allows for stratification into small strata. The method for computing stable models of logic programs, developed in [NS95], also did very well for this particular class of problems. Hence, both our results and the results in [NS95] indicate that the 3-coloring problem for ladder graphs is too easy to serve as a useful benchmark problem for nonmonotonic reasoning systems. This is due to the large number of 3-colorings that the ladder graphs admit. Hence, any search method is likely to find a 3-coloring quickly. In fact, there are 3^{n-1} non-isomorphic 3-colorings for L_n. This yields 2×3^n extensions for default theory *color3.board_n,2,0,0,1,0,0_*. This is also the reason why we did not provide time for the task of finding all extensions of theories *color3.board_n,2,0,0,1,0,0_*. Times to find a single extension are short but the list of all solutions is exponentially large.

kernel.board_n,0,0,0,1,1,1_, compute extensions								
n	$\|V\|$	$\|E\|$	N	NCPP	CAND	$time_f$	$time_l$	$time_a$
175	175	175	700	5235	2618	414.07	1.46	0.08
375	375	375	1500	11235	5618	4294.84	6.50	0.15
575	575	575	2300	17235	8618	–	15.06	0.24
775	775	775	3100	23235	11618	–	27.25	0.32
975	975	975	3900	29235	14618	–	42.98	0.39

Table 5: Searching for kernels in $C_n = board(n, 0, 0, 0, 1, 1, 1)$.

3.3 Computing with DeReS when no extensions exist

To get a complete picture of the performance of an automated reasoning method (or any search-based algorithm) it is necessary to test the method on instances for which solution do not exist and, hence, the method has to run a complete search. As a benchmark for such tests we use the problem of finding kernels in directed cycles with odd number of vertices. Default theories which we use to encode this problem have no extensions. The objective of this experiment was to find out how long will it take for DeReS to answer that no solution exists. In this case the reasoning time is the same for the query *find one extension* and for the query *find all extensions*.

We use directed cycles C_n of odd length. In Stanford GraphBase C_n is labeled *board(n,0,0,0,1,1,1)*. To encode kernels we use the TheoryBase encoding Δ^3_{ker} and the resulting theories have labels *kernel.board_n,0,0,0,1,1,1_*. In this case, intermediate extensions exist for all strata but the last one. Hence, the entire D must be analyzed before the conclusion that no extensions exist can be drawn. Times needed by DeReS to conclude that no solution exists are shown in Table 5. Since in this case whole set of defaults must be considered to find out that there are no extensions number of calls to propositional provability procedure is not constant. It grows linearly – 100 additional nodes in the circuit require exactly 3000 extra calls to propositional provability routine. This yields linear reasoning time, $O(N)$, when df-lookup prover is used and quadratic, $O(N^2)$ reasoning time, in the case of limited tableau method. The results presented in this section show that stratification plays a critical role also when determining that no solutions exist.

Also in this case it is important to use an appropriate encoding. Representing the existence of kernel problem for graphs *board(n,0,0,0,1,1,1)* by theories with the labels *kernel.b.board*_n,0,0,0,1,1,1_ (Δ^1_{ker} in [CMMT95]) leads to a dramatic loss of performance.

4 Conclusions

We have presented several experimental results of using default logic to solve problems from a domain of independent interest – graph theory. In our study we focused on the

kernel.b.board_n,0,0,0,1,1,1_, compute extensions								
n	$\|V\|$	$\|E\|$	N	NCPP	CAND	$time_f$	$time_l$	$time_a$
15	15	15	15	36178	2583	2.53	0.79	0.10
19	19	19	19	318800	17710	30.79	7.27	0.80
23	23	23	23	2670648	121392	334.62	62.49	6.46
27	27	27	27	21633042	832039	3509.69	521.95	51.67
31	31	31	31	171086612	5702886	–	4288.25	403.07

Table 6: Searching for kernels in $C_n = board(n, 0, 0, 0, 1, 1, 1)$.

following two aspects of default reasoning:
(1) how the ability to stratify a default theory affects the computational time,
(2) which methods of propositional provability are suitable as an oracle for default reasoning.
Consequently, we did not consider any specific method of dealing with a single cluster of defaults. We used a generic reduct-based algorithm to compute extensions for a single stratum of defaults. We expect that stratification would affect the performance of DeReS in the same way for any other method to process clusters. This claim, however, still needs to be confirmed experimentally.

The results presented in this paper confirmed the theoretical results on stratification given in [Cho95c]. That is, using stratification we were able to find extensions for default theories consisting of hundreds and thousands of defaults. Of course, this level of performance is not guaranteed unless the input theory admits a fine stratification, that is, when there is a constant bound on the size of the largest stratum. We showed that stratification, in general, does not eliminate the exponential time complexity but rather reduces the complexity so that substantial instances can be solved in reasonable time.

We investigated three different methods of implementing provability procedure which is used as an oracle in default reasoning. The three algorithms which we used could have solved a single provability query in amortized times $O(1)$, $O(N)$ and $O(N^2)$ per query. Our results did not depend on any particular algorithm for propositional provability. The same (up to constants) results can be obtained under the assumption that we have three provers which run in constant, linear and quadratic time per query. Two important conclusions are:
(1) Performance of DeReS and other nonmonotonic systems can be dramatically improved if limited provers are used allowing us to reduce the number of global consistency checks.
(2) To keep reasoning time as small as possible, the simplest prover which can handle the class of formulas appearing in an input default theory should be used.

Finally, this work demonstrates usability of TheoryBase as a benchmarking system for nonmonotonic reasoning.

References

[ABW88] K. Apt, H.A. Blair, and A. Walker. Towards a theory of declarative knowledge. In J. Minker, editor, *Foundations of deductive databases and logic programming*, pages 89–142, Los Altos, CA, 1988. Morgan Kaufmann.

[BEP94] R. Ben-Eliyahu and L. Palopoli. Reasoning with minimal models: Efficient algorithms and applications. In *Proceedings of KR'94*, San Francisco, CA, 1994. Morgan Kaufmann.

[Bes89] P. Besnard. *An introduction to default logic*. Springer-Verlag, Berlin, 1989.

[BF91] N. Bidoit and C. Froidevaux. General logical databases and programs: default logic semantics and stratification. *Information and Computation*, 91:15–54, 1991.

[BNN+94] C.Bell, A. Nerode, R.Ng, V.S. Subrahmanian. Mixed Integer Programming Methods for Computing Non-Monotonic Deductive Databases, *Journal of the ACM*, 41:1178–1215, 1994.

[BNN+93] C. Bell, A. Nerode, R. Ng, W. Pugh, and V.S. Subrahmanian. Implementing stable semantics by linear programming. In A. Nerode and L. Pereira, editors, *Logic programming and non-monotonic reasoning*. MIT Press, 1993.

[Bre91] G. Brewka. *Nonmonotonic reasoning: logical foundations of commonsense*. Cambridge University Press, Cambridge, UK, 1991.

[CDS94] M. Cadoli, F. M. Donini, and M. Schaerf. Is intractability of non-monotonic reasoning a real drawback? In *Proceedings of the Twelfth National Conference on Artificial Intelligence (AAAI-94)*, pages 946–951, Seattle, USA, 1994.

[CH82] A.K. Chandra and D. Harel. Structure and complexity of relational queries. *Journal of Computer and System Sciences* 25:99-128, 1982.

[Cho95a] P. Cholewiński. Reasoning with stratified default theories. In *Proceedings of LPNMR'95*. Berlin: Springer-Verlag, 1995. Lecture Notes in Computer Science 928.

[Cho95c] P. Cholewiński. Stratified default theories. In *Proceedings of CSL'94*. Berlin: Springer-Verlag, 1995. Lecture Notes in Computer Science 933.

[CMMT95] P. Cholewiński, W. Marek, A. Mikitiuk, and M. Truszczyński. Experimenting with nonmonotonic reasoning. In *Proceedings of the 12th International Conference on Logic Programming*, Cambridge, MA, 1995. MIT Press.

[Eth88] D. W. Etherington. *Reasoning with incomplete information*. Pitman, London, 1988.

[Gel87] M. Gelfond. On stratified autoepistemic theories. In *Proceedings of AAAI-87*, pages 207–211, Los Altos, CA, 1987. American Association for Artificial Intelligence, Morgan Kaufmann.

[GKPS95] G. Gogic, H. Kautz, Ch. Papadimitriou, and B. Selman. Compactness of knowledge representation: A comparative analysis. In *Proceedings of IJCAI-95*, pages 862–869. Morgan Kaufmann, 1995.

[GL88] M. Gelfond and V. Lifschitz. The stable semantics for logic programs. In R. Kowalski and K. Bowen, editors, *Proceedings of the 5th international symposium on logic programming*, pages 1070–1080, Cambridge, MA, 1988. MIT Press.

[Knu93] D. E. Knuth. *The Stanford GraphBase: a platform for combinatorial computing*. Addison-Wesley, 1993.

[KS89] H.A. Kautz and B. Selman. Hard problems for simple default logics. In *Proceedings of the 1st international conference on principles of knowledge representation and reasoning, KR '89*, pages 189–197, San Mateo, CA, 1989. Morgan Kaufmann.

[McC80] J. McCarthy. Circumscription — a form of non-monotonic reasoning. *Artificial Intelligence*, 13:27–39, 1980.

[MD80] D. McDermott and J. Doyle. Nonmonotonic logic I. *Artificial Intelligence*, 13:41–72, 1980.

[MS72] A.R. Meyer and L.J. Stockmeyer. The equivalence problem for regular expressions with squaring requires exponential time. In *Proceedings of the 13th annual symposium on switching and automata theory*, pages 125–129. IEEE Computer Society, 1972.

[MT91] W. Marek and M. Truszczyński. Autoepistemic logic. *Journal of the ACM*, 38:588–619, 1991

[MT93] W. Marek and M. Truszczyński. *Nonmonotonic logics; context-dependent reasoning*. Berlin: Springer-Verlag, 1993.

[MW88] D. Maier and D. S. Warren. *Computing with logic. Logic programming with Prolog*. The Benjamin/Cummings Publishing Company, Inc., 1988.

[Nie95] I. Niemelä. Towards efficient default reasoning. In *Proceedings of IJCAI-95*, pages 312–318. Morgan Kaufmann, 1995.

[NS95] I. Niemelä and P. Simmons. Evaluating an algorithm for default reasoning. In *Proceedings of the IJCAI-95 Workshop on Applications and Implementations of Nonmonotonic Reasonings Systems*, 1995.

[Prz88] T.C. Przymusinski. On the declarative semantics of deductive databases and logic programs. In J .Minker, editor, *Foundations of deductive databases and logic programming*, pages 193–216. Los Altos, CA: Morgan Kaufmann, 1988.

[Rei78] R. Reiter. On closed world data bases. In H. Gallaire and J. Minker, editors, *Logic and data bases*, pages 55–76. Plenum Press, 1978.

[Rei80] R. Reiter. A logic for default reasoning. *Artificial Intelligence*, 13:81–132, 1980.

[SKC93] B. Selman, H.A. Kautz, and B. Cohen. Local search strategies for satisfiability testing. In *Proceedings 1993 DIMACS Workshop on Maximal Clique, Graph Coloring and Satisfiability*, 1993.

[SLM92] B. Selman, H. Levesque, and D. Mitchell. A new method for solving hard satisfiability problems. In *Proceedings of AAAI-92*, pages 440 – 446, Los Altos, CA, 1992. American Association for Artificial Intelligence, Morgan Kaufmann.

[VG88] A. Van Gelder, Negation as failure using tight derivations for general logic programs, in: J. Minker, ed, Foundations of deductive databases and logic programming, pages 1149–1176, Morgan Kaufmann, Los Altos, 1988.