# Comparing Subsumption Optimizations

**Ian Horrocks**
University of Manchester, Manchester, UK

**Peter F. Patel-Schneider**
Bell Labs Research, Murray Hill, NJ, U.S.A.

**Abstract**

Effective systems for expressive description logics require a heavily-optimised subsumption checker incorporating a range of optimisation techniques. Because of the correspondence between description logics and propositional modal logic most of these techniques carry over into propositional modal logic satisfiability checking. Some of the techniques are extremely effective on various test suites for propositional modal satisfiability and others are less effective. Further, the effectiveness of a technique depends on the test performed.

Description logic systems spend much of their time computing subsumption relationships between descriptions. If the system is based on an expressive description logic then the amount of time spent computing subsumption can be intolerable, even for small knowledge bases, unless steps are taken to heavily optimise this task. The total time spent in subsumption checking comes from the number of subsumption checks required to process a knowledge base as well as from the time spent in performing the hardest of these subsumption checks—with an expressive description logic, the time taken by a small number of hard subsumption checks can dominate the total time.

Two systems based on expressive description logics, KRIS [1] and CRACK [3], have incorporated a number of optimisations to achieve better performance of their subsumption checkers. These systems use various techniques to avoid performing subsumption checks, and they also optimise the subsumption check itself. Two other systems that explore the optimisations required to build an expressive description logic system are FaCT [9], a full description logic system, and DLP [12], an experimental system providing only a limited description logic interface. The subsumption checkers for both FaCT and DLP incorporate a range of known, adapted and novel optimisation techniques including lexical normalisation, semantic branching search, boolean constraint propagation, dependency directed backtracking, heuristic guided search and caching.

These optimisation techniques make a dramatic difference to the performance of the overall system. As evidence, KRIS is not able to load (a modified version of) a large medical terminology knowledge base from the GALEN project [13] because it gets stuck trying to perform one of the thousands of required subsumption tests. FaCT and DLP, which have higher levels of optimisation, are able to easily load this knowledge base, classifying over two thousand definitions in about two hundred seconds.

| Syntax | Semantics |
|---|---|
| $\mathsf{A}$ | $\mathsf{A}^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}}$ |
| $\top$ | $\Delta^{\mathcal{I}}$ |
| $\bot$ | $\emptyset$ |
| $\neg C$ | $\Delta^{\mathcal{I}} - C^{\mathcal{I}}$ |
| $C \sqcap D$ | $C^{\mathcal{I}} \cap D^{\mathcal{I}}$ |
| $C \sqcup D$ | $C^{\mathcal{I}} \cup D^{\mathcal{I}}$ |
| $\exists R.C$ | $\{d \in \Delta^{\mathcal{I}} \mid R^{\mathcal{I}}(d) \cap C^{\mathcal{I}} \neq \emptyset\}$ |
| $\forall R.C$ | $\{d \in \Delta^{\mathcal{I}} \mid R^{\mathcal{I}}(d) \subseteq C^{\mathcal{I}}\}$ |
| $\exists T.C$ | $\{d \in \Delta^{\mathcal{I}} \mid T^{\mathcal{I}}(d) \cap C^{\mathcal{I}} \neq \emptyset\}$ |
| $\forall T.C$ | $\{d \in \Delta^{\mathcal{I}} \mid T^{\mathcal{I}}(d) \subseteq C^{\mathcal{I}}\}$ |

Table 1: Semantics of $\mathcal{ALC}_{R^+}$ concept expressions

Because FaCT and DLP incorporate several optimisations we have investigated which of these optimisations are most effective. We would have liked to perform this investigation using a sample of description logic knowledge bases that incorporate hard problems for description subsumption. Unfortunately such knowledge bases are currently uncommon, largely because existing description logic systems have been unable to effectively process them.

However, there are other sources of hard description logic subsumption problems! FaCT and DLP implement a superset of the propositional modal logic $\mathbf{K4_{(m)}}$. A number of testing methodologies have been established for propositional modal logics [8; 7; 10] and we have used these to perform experiments comparing the effectiveness of the various optimisations built into FaCT and DLP.

## The Description Logic $\mathcal{ALC}_{R^+}$

Both FaCT and DLP implement a superset of $\mathcal{ALC}_{R^+}$, but this DL is adequate to demonstrate their optimisations. $\mathcal{ALC}_{R^+}$ is built up from atomic concepts and two kinds of atomic roles, non-transitive roles and transitive roles. Concepts in $\mathcal{ALC}_{R^+}$ are formed using the grammar $\mathsf{A} \mid \top \mid \bot \mid \neg C \mid C \sqcap D \mid C \sqcup D \mid \exists R.C \mid \forall R.C \mid \exists T.C \mid \forall T.C$, where $\mathsf{A}$ is an atomic concept, $C$ and $D$ are concept expressions, $R$ is a non-transitive role, and $T$ is a transitive role.

The semantics of $\mathcal{ALC}_{R^+}$ is a standard extensional semantics, as given in Table 1.

Systems that are based on description logics like $\mathcal{ALC}_{R^+}$ generally determine whether a subsumption holds by transforming the subsumption question into a satisfiability question in the obvious manner. They then attempt to construct a model for this concept, using a tableaux method. During this process, various nodes are created, where each node represents an individual, and tells whether the individual belongs to various concepts. This set of concepts is said to form the label of the node—denoted $\mathcal{L}(x)$.

The basic algorithm starts out with a single node representing an individual that must be in the extension of the concept being tested for satisfiability. This concept is expanded to produce simpler concepts that must have the individual in their extension. Disjunctive concepts give rise to choice points in the algorithm.

Each existential role concept, $\exists R.C$, causes the creation of a new, related node representing another individual which must be in the extension of $C$. If a node is related to another node via role $R$, the second node is called an $R$-successor of the first.

Universal role concepts augment the concepts that these individuals must belong to. In order to guarantee termination, transitive roles require *blocking*: a check to ensure that no other node has the same set of concepts—if so, the two nodes can be collapsed into a cycle.[1]

If the algorithm constructs a collection of nodes where there are no concept expressions that have not been expanded and where there are no obvious contradictions, called *clashes*, at any of the nodes, then the collection of nodes corresponds to a model for the initial concept. If the algorithm fails to construct such a collection then the initial concept is unsatisfiable.

The details of the basic algorithm are fairly standard, and can be found in [14].

## Optimisation Techniques

A naive implementation of the tableaux method would be much too slow to be used for subsumption testing in a description logic. DLP (and FaCT)[2] therefore employ a range of known, adapted and novel optimisations that improve the performance of the satisfiability testing algorithm.

DLP simplifies all concept expressions and converts them into a lexically normalised form. In this form, concept expressions consist only of (possibly negated) atomic concepts, conjunction concepts and universal role concepts: expressions of the form $\exists R.C$ are transformed into $\neg(\forall R.\neg C)$ and expressions of the form $(D_1 \sqcup \ldots \sqcup D_n)$ are transformed into $\neg(\neg D_1 \sqcap \ldots \sqcap \neg D_n)$. In addition, the sub-expressions forming conjunctive concepts are sorted, and any duplicates eliminated. The normalisation process also identifies and simplifies sub-expressions which are obviously satisfiable or obviously unsatisfiable, replacing them with $\top$ or $\bot$ respectively. In extreme cases the need for a tableau expansion can be completely eliminated.

[1]In this description logic all cycles are good—they can be interpreted as valid cyclical models.

[2]From now on we will often refer to DLP only, as it has a larger set of optimizations, and incorporates the ideas from FaCT.

Lexically identical concepts are uniquely stored so that a clash can be detected as soon as an expression and its lexical negation occur in the same node label. This can lead to clashes being detected much earlier, eliminating the (possibly costly) expansion which would have been required in order to generate node label(s) containing clashing atomic concepts.

Description logic satisfiability tests typically deal with an unexpanded disjunction $(D_1 \sqcup \ldots \sqcup D_n) \in \mathcal{L}(x)$ by searching the possible models obtained by separately adding each of $D_1, \ldots, D_n$, a technique known as syntactic branching [7]. In contrast, DLP uses a semantic branching search technique adapted from the Davis-Putnam-Logemann-Loveland procedure (DPLL). In semantic branching the branches are obtained by separately adding $D_i$ and $\neg D_i$.

Because the two possible models generated at a semantic branching point are strictly disjoint, there is no possibility of wasted search. An additional advantage of using a DPLL-based search technique is that a great deal is known about the implementation and optimisation of this algorithm. In particular, both boolean constraint propagation and heuristic guided search can be used to try to minimise the size of the search tree.

Boolean constraint propagation (BCP) is a technique used to maximise deterministic expansion, and thus pruning of the search tree via clash detection [6]. Before semantic branching is applied to the label of a node $x$, BCP deterministically expands disjunctions in $\mathcal{L}(x)$ which present only one expansion possibility and detects a clash when a disjunction in $\mathcal{L}(x)$ has no expansion possibilities: in effect, BCP uses the inference rule $\frac{\neg C, C \sqcup D}{D}$ to simplify the expression represented by $\mathcal{L}(x)$. This can dramatically reduce the size of the search space, particularly when used in conjunction with semantic branching.

Inherent unsatisfiability concealed in sub-problems can lead to large amounts of unproductive backtracking search known as thrashing. DLP tackles this problem by adapting a form of dependency directed backtracking called *backjumping*, which has been used in solving constraint satisfiability problems [2].

Backjumping labels concept expressions with a dependency set indicating the branch points on which they depend. When a clash is discovered, the dependency sets can be used to identify the most recent branch point where exploring the other branch might alleviate the cause of the clash. The algorithm can then jump back over intervening branch points *without* exploring alternative branches. Backjumping can lead to dramatic reductions in the search space, but there is some overhead caused by the evaluation and storage of the dependency sets.

During a tableau expansion many identically labelled nodes may be created, particularly as the $R$-successors for a node $x$ each have the same concept expressions for the universal role concepts in $\mathcal{L}(x)$. DLP takes advantage of the repetitive structure of a typical tableau by caching the satisfiability result for each node label as it is evaluated. If a label recurs, then there is no need to reevaluate the satisfiability of that node: the previous satisfiability result can simply be reused.

DPLL SAT algorithms often use heuristics to guide the search by selecting the next disjunct on which to branch. These heuristics typically try to maximise the effectiveness of BCP by selecting disjuncts which occur frequently in small disjunctions [5]. However, these techniques do not work well with modal problems because they rely for their effectiveness on finding the same disjuncts recurring in multiple disjunctions. This is likely in non-modal problems, otherwise most problems would be trivially satisfiable, but it is less likely in modal problems where unsatisfiability can be caused by modal sub-problems.

DLP tackles this problem by using a heuristic which tries to maximise the effectiveness of backjumping. This is done by branching first on a disjunction with the oldest dependencies [9]. The disjunct within these disjunctions is chosen using Jeroslow and Wang's weighted occurrences heuristic [11] (the JW heuristic hereafter), a BCP-maximising heuristic. In addition, DLP reduces the size of the search space by using similar heuristic techniques to select the order in which $R$-successors of a node are expanded.

## Comparing Optimisations

In order to determine which optimisations are effective, DLP has configuration options to turn on and off or vary all of the above optimisations. We have run DLP in various configurations on several test suites. All statistics reported for DLP are for runs on machines with approximately the speed of a SPARC Ultra 1 and with 128MB of main memory.

The configurations that we tested are:

**Oldest-JW:** Select an oldest disjunction and use the JW heuristic to select a disjunct in it and whether to branch positive or negative first. This is the basis for the other configurations below.

**JW:** Use the JW heuristic to select a disjunct from all disjunctions and whether to branch positive or negative first.

**Random, negative:** Select a disjunct at random, and branch negative on it first.

**Random, positive:** Select a disjunct at random, but do the positive (instead of the negative) branch first.

**No caching:** Turn off caching.

**No backjumping:** Turn off backjumping

**No semantic branching:** Turn off semantic branching

**No BCP:** Turn off boolean constraint propagation

**No normalisation:** Turn off normalization

Unfortunately, we are not yet satisfied with the kinds of tests that we have been able to perform. We would prefer to test on actual description logic knowledge bases, as that is what DLP is designed for. However, there are very few description logic knowledge bases that use the more-powerful constructs provided by DLP. Most of our testing has thus been against test suites for propositional modal logics, using the propositional modal logic interface for DLP. We have tested against the test suite for the Tableaux'98 propositional modal logic comparison [8] and against a collection of random formulae initially generated by Hustadt and Schmidt [10].

The Tableaux'98 test suite consists of several classes of formulae (e.g. branch), in both provable and non-provable forms, for each of $\mathbf{K}$, $\mathbf{KT}$, and $\mathbf{S4}$. For each class of formula, 21 examples of supposedly exponentially increasing difficulty are automatically generated from a basic pattern which incorporates features intended to make the formulae hard to solve. The test methodology is to ascertain the number of the largest formula of each type which the system is able to solve within 100 seconds of CPU time.

The complete test suite contains 1,134 problems; Table 2 shows the total number of problems solved (within 100s of CPU time) by various configurations of DLP and how many of the provable $\mathbf{S4}$ formulae were solved within the time limit.

There is a wide variability between the different types of formula, with some optimisations dramatically changing the behaviour both quantitatively, in solving much more difficult problems, and qualitatively, in changing from an exponential growth in solution time to an almost-constant solution time. This is illustrated by Fig. 1 which shows the actual solution times for two classes of formulae with various optimisations disabled. In one of these examples the qualitative improvement is due to caching; in the other it is due to semantic branching and backjumping.

The results indicate that the most effective optimisation for this test suite is semantic branching. The next-most-effective technique is caching, followed by backjumping. The benefits from semantic branching, however, are concentrated in a few classes of formulae, such as branch and t4p in Table 2, which were designed to have large amounts of redundant syntactic search. Semantic branching avoids this redundant search and thus does much better on these classes of formulae. Caching is very effective on this test suite because of its large amount of structure, which results in the frequent repetition of sub-problems.

The heuristics were only effective for some classes of problem in this test suite. In some classes the JW heuristic was good, such as s5 in Table 2, and in others it resulted in worse performance, but overall it actually resulted in fewer problems being solved than using the simpler oldest-first heuristic. This is probably due to the fact that the JW heuristic is designed for non-modal problems whereas the oldest-first heuristic enhances backjumping.

Our second propositional modal logic test suite uses a common method for testing SAT decision procedures [4] that has been adapted for use with propositional modal $\mathbf{K}$ by Giunchiglia and Sebastiani [7], and further refined by Hustadt and Schmidt [10]. The method uses a random generator to produce formulae, with the characteristics of the formulae being controlled by a number of parameters. Each formula is a conjunction of $L$ $K$-clauses, where a $K$-clause is a disjunction of $K$ elements, each element being negated with a probability of 0.5. An element is either a modal atom of the form $\forall R.C$, where $C$ is itself a $K$-clause, or at the maximum modal depth $D$, a propositional variable chosen from the $N$

| Configuration | Total | S4 class | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| | | 45 | branch | grz | ipc | md | path | ph | s5 | t4p |
| Oldest-JW | 928 | 21 | 21 | 21 | 10 | 3 | 13 | 4 | 19 | 21 |
| JW | 915 | 21 | 18 | 21 | 10 | 3 | 8 | 5 | 21 | 21 |
| Random, negative | 907 | 21 | 18 | 21 | 10 | 3 | 12 | 7 | 3 | 21 |
| Random, positive | 868 | 21 | 18 | 21 | 10 | 3 | 9 | 7 | 4 | 21 |
| No caching | 839 | 21 | 21 | 21 | 8 | 7 | 9 | 4 | 8 | 21 |
| No backjumping | 863 | 21 | 21 | 21 | 7 | 3 | 3 | 4 | 5 | 19 |
| No semantic branching | 738 | 12 | 4 | 21 | 7 | 3 | 2 | 7 | 4 | 6 |
| No BCP | 910 | 21 | 21 | 21 | 9 | 3 | 13 | 4 | 17 | 21 |
| No normalisation | 901 | 21 | 21 | 21 | 10 | 3 | 8 | 6 | 13 | 21 |

Table 2: Total Tableaux'98 problems solved and Provable **S4** problems solved

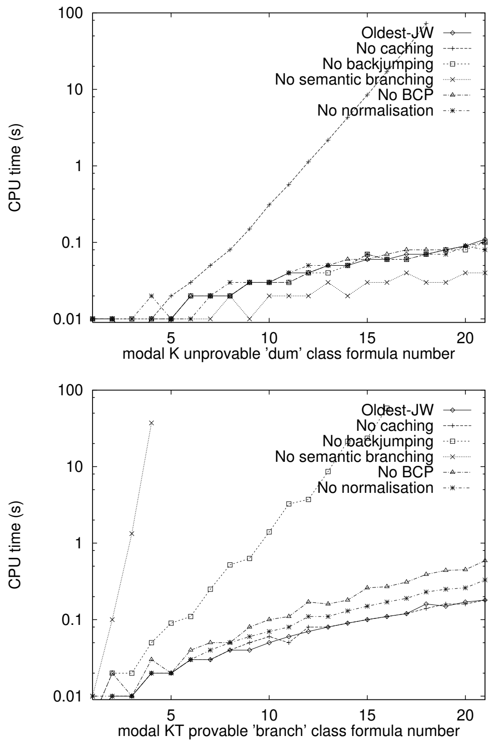


Figure 1: Solution times for two Tableaux'98 tests

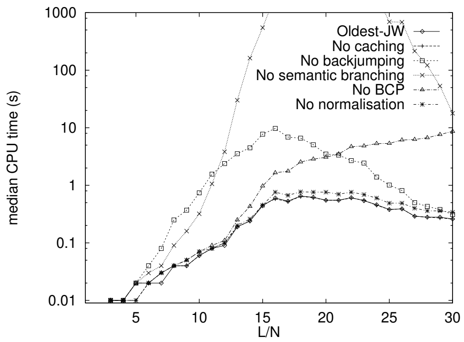


Figure 2: Median solution times for **PS12** formulae

propositional variables which appear in the formula. Hustadt and Schmidt used two sets of formulae, denoted **PS12** and **PS13**, choosing $N = 4$ and $N = 6$ respectively, with $K = 3$ and $D = 1$ in both cases. The test sets are created by varying $L$ from $N$ to $30N$, giving formulae with a probability of satisfiability varying from $\approx 1$ to $\approx 0$, and generating 100 formulae for each integer value of $L/N$.

The median times required to test the satisfiability of the **PS12** formulae using various configurations of DLP are given in Figure 2. The results for **PS13** are generally similar.

The most effective optimisation for this test suite is again semantic branching. Random problems can have large amounts of overlapping search between the different disjuncts in a disjunction, which semantic branching avoids. The two next-most-effective optimisations are backjumping and boolean constraint propagation, with backjumping being more effective for intermediate values of $L/N$, where the "harder" problems arise, and boolean constraint propagation being more effective for the larger values of $L/N$, where the formulae are severely overconstrained, so there is considerable scope for simplification whenever a branching choice is made. The effectiveness of boolean constraint propagation also helps to explain the effectiveness of semantic branching for the overconstrained formulae, as syntactic branching does not allow as much boolean constraint propagation.

The other optimisations are much less effective in this suite. In particular, caching is not effective at all. This is because, with such a small number of literals, the purely propositional problems at depth 1 can always be solved deterministically, and performance is therefore dependent on the efficiency of propositional reasoning at depth 0. Caching is thus ineffective because there are no hard modal sub-problems to cache. Normalisation is largely ineffective here because the formulae are already in conjunctive normal form.

None of the heuristics are particularly effective with this test suite, so changing heuristics made little difference. The JW heuristic is ineffective because the disjuncts are randomly generated modal sub-formulae, and the large number of different possibilities means that any given sub-formula is unlikely to occur in many disjunctions. The oldest-first heuristic is ineffective because, for formulae in conjunctive normal form, every disjunction at depth 0 has the same "age".

Although the Tableaux'98 and random test suites show how our optimisations perform on propositional modal log-

| Configuration | Time (s) |
|---|---|
| Oldest-JW | 264 |
| JW | 487 |
| Random, negative | 472 |
| Random, positive | 203 |
| No caching | 4808 |
| No backjumping | >10000 |
| No semantic branching | 199 |
| No BCP | 251 |
| No normalisation | 223 |

Table 3: Times for the GALEN KB

ics, neither is very good for our purposes. One test with a knowledge base that we have been able to do is to take the GALEN knowledge base and construct versions of it that are acceptable to FaCT, DLP and KRIS. KRIS was unable to process this knowledge base within four hours, but both FaCT and DLP can process it in about 200 seconds. The times for the various configurations of DLP loading this knowledge base are given in Table 3.

In this test the most important optimisation is backjumping, followed by caching. Semantic branching is not effective in this knowledge base—in fact, turning semantic branching off results in the fastest configuration. We do not understand why this is—perhaps in the GALEN knowledge base there are seldom any cases where adding the negation of a disjunct affects the other elements of the disjunction and the results simply reflect the additional complexity of semantic branching.

## Summary

The collection of optimizations we have described are effective in improving the speed of modal propositional logic reasoners, as shown by the results we have given above. They can also dramatically improve the speed of subsumption reasoning on description logic knowledge bases. To our knowledge some of these improvements have not been investigated in the modal propositional reasoning literature. The combination appears to be unique and, moreover, results in a powerful reasoner for the propositional modal logics $\mathbf{K}$, $\mathbf{KT}$, and $\mathbf{S4}$.

The optimisations are not uniformly effective. In particular, semantic branching is extremely effective on constructed hard problems and on random satisfiability problems, but not on the GALEN knowledge base. We plan to perform more experiments on knowledge bases to see if semantic branching is indeed ineffective on them. The two other optimisations that are the most effective are backjumping and caching. These two optimisations make the difference between acceptable and ridiculous performance in many of the tests. Their absence in previous description logic systems has made them unacceptablely slow.

We, along with Enrico Franconi of IRST, are embarking on a project to create a description logic system for a description logic that corresponds to a propositional dynamic logic. This project will require more optimisation, as propositional dynamic logic is harder than the logics we are currently handling, and will give us further opportunities to investigate the optimisation of satisfiability reasoners. We are also performing more testing of the optimisations we are putting into our provers and we plan to create a test suite that emphasizes the modal nature of description logics.

FaCT is available at http://www.cs.man.ac.uk/ horrocks; the DLP prover is currently under development, but the version used in these tests is available at http://www-db.research.bell-labs.com/user/pfps.

## References


[1] F. Baader and B. Hollunder. KRIS: Knowledge representation and inference system. *SIGART Bulletin*, 2(3):8–14, 1991.

[2] A. B. Baker. *Intelligent Backtracking on Constraint Satisfaction Problems: Experimental and Theoretical Results*. PhD thesis, University of Oregon, 1995.

[3] P. Bresciani, E. Franconi, and S. Tessaris. Implementing and testing expressive description logics: a preliminary report. In G. Ellis, R. A. Levinson, A. Fall, and V. Dahl, editors, *Knowledge Retrieval, Use and Storage for Efficiency*, pages 28–39, 1995.

[4] J. Franco and M. Paull. Probabilistic analysis of the Davis-Putnam procedure for solving the satisfiability problem. *Discrete Applied Mathematics*, 5:77–87, 1983.

[5] J. W. Freeman. *Improvements to propositional satisfiability search algorithms*. PhD thesis, Department of Computer and Information Science, University of Pennsylvania, Philadelphia, PA, USA, 1995.

[6] J. W. Freeman. Hard random 3-SAT problems and the Davis-Putnam procedure. *Artificial Intelligence*, 81:183–198, 1996.

[7] F. Giunchiglia and R. Sebastiani. A SAT-based decision procedure for $\mathcal{ALC}$. In *KR'96*, pages 304–314. Morgan Kaufmann, November 1996.

[8] A. Heuerding and S. Schwendimann. A benchmark method for the propositional modal logics K, KT, S4. Technical report IAM-96-015, University of Bern, Switzerland, October 1996.

[9] I. Horrocks. *Optimising Tableaux Decision Procedures for Description Logics*. PhD thesis, University of Manchester, 1997.

[10] U. Hustadt and R. A. Schmidt. On evaluating decision procedures for modal logic. Technical Report MPI-I-97-2-003, Max-Planck-Institut Für Informatik, Saarbrücken, Germany, February 1997.

[11] R. Jeroslow and J. Wang. Solving propositional satisfiability problems. *Annals of Mathematics and Artificial Intelligence*, 1:167–187, 1990.

[12] P. F. Patel-Schneider. System description: DLP. Bell Labs Research, Murray Hill, NJ, December 1997.

[13] A. Rector and I. Horrocks. Experience building a large, reusable medical ontology using a description logic with transitivity and concept inclusions. In *Proceedings of the Workshop on Ontological Engineering*. AAAI Press, 1997.

[14] U. Sattler. A concept language extended with different kinds of transitive roles. In G. Görz and S. Hölldobler, editors, *20. Deutsche Jahrestagung für Künstliche Intelligenz*, number 1137 in LNAI, pages 333–345. Springer Verlag, 1996.