Evaluating Optimised Decision Procedures for Propositional Modal $\mathbf{K}_{(\mathbf{m})}$ Satisfiability

Ian Horrocks
Department of Computer Science, University of Manchester, UK
*(*horrocks@cs.man.ac.uk*)*

Peter F. Patel-Schneider
Bell Labs Research, Murray Hill, NJ, U.S.A.
*(*pfps@research.bell-labs.com*)*

Abstract. Heavily-optimised decision procedures for propositional modal satisfiability are now becoming available. Two systems incorporating such procedures for modal **K**, DLP and KSATC, are tested on randomly-generated CNF formulae with several sets of parameters, varying the maximum modal depth and ratio of propositional variable to modal subformulae. The results show some easy-hard-easy behaviour, but there is as yet no sharp peak as in propositional satisfiability.

Keywords: satisfiability, optimization, tableaux

1. Introduction

Propositional modal satisfiability has lagged behind standard propositional satisfiability in that until recently there have been no heavily-optimised propositional modal systems. This has changed in the last few years with the appearance of several heavily-optimised systems that can be used to determine the satisfiability of formulae in propositional modal logics.

One factor driving the development of such propositional modal systems is the relationship between propositional modal logics and description logics. A system that correctly reasons with information in an expressive description logic includes a decision procedure for an expressive propositional modal logic. This decision procedure must be heavily optimised if it is to be able to usefully reason with knowledge bases of any complexity. Description logic systems that incorporate a heavily-optimised propositional modal logic decision procedure include FACT [24], DLP [36], and HAM-ALC [21].

Another factor that has lead to the development of heavily-optimised propositional modal systems is the availability of heavily-optimised decision procedures for non-modal propositional logic. It is possible to use such a procedure as the core of a decision procedure for a propositional modal logic. Modal decision procedures built in this fashion include KSAT and its successor, KSATC [19, 20, 18].

These systems perform well on hand-generated problems. In the comparison at Tableaux'98 [6, 27], DLP solved many of the supposedly-difficult problems without even performing any search. Even when search was required, the optimisations in DLP resulted in very short execution times for many of the problems. A KSAT system also did well on the Tableaux'98 problems, although it did not perform as well as DLP.

There has also been some testing of decision procedures for propositional modal logics on random formulae. Giunchiglia *et al* [19, 20, 18] designed a random generator for formulae in the modal logic K and tested their systems, KSAT and KSATC, on formulae produced by the generator. Unfortunately, some versions of their generator suffered from problems which made the results of their testing suspect. The generator was modified by Hustadt and Schmidt[31, 32] to remove some of its problems and used to test the performance of their system, TA, a decision procedure for various propositional modal logics.

We have created a new generator that extends the previous generators, and tested both DLP and KSATC on the formulae produced by this new generator. The results show that it is easy to produce modal formulae that are difficult for both DLP and KSATC, even when there are only six propositional variables and the modal depth is limited to one or two. Nevertheless, both systems performed much better than previous decision procedures for propositional modal logic. The two systems have quite different behaviour on the generated formulae, with neither dominating the other, showing that their optimisations are effective for different kinds of formulae.

We also used the new generator to test the two systems on modal formulae with a maximum modal depth of 100. To our knowledge, this is the first time that propositional modal satisfiability procedures have been tested with such highly modal formulae. Although such problems are mostly very easy, the test did highlight an important difference between the two systems: KSATC finds a small number of these formulae very hard to solve, whereas DLP found them uniformly easy.

2. DLP and KSATC

The two systems we compare are similar in that they both systematically search for a model of a formula. Thus they are complete, non-stochastic procedures. They are quite different from stochastic systems like GSAT [38], or from the TA system, which translates propositional modal formulae into a fragment of first-order logic (basic path logic) and tests the satisfiability of the resulting formulae using a first-

order prover (FLOTTER/SPASS [40]) that always terminates on this fragment.

Although DLP and KSATC both use search techniques, they are otherwise quite different. DLP was developed as the core of a description logic reasoner. It includes a complete decision procedure for a very expressive description logic; one that includes propositional dynamic logic as a subset. Because DLP can reason in a superset of propositional dynamic logic, it can be used as a decision procedure for many (multi) modal logics, including $\mathbf{K}_{(\mathbf{m})}$ and $\mathbf{K4}_{(\mathbf{m})}$, and, through simple embeddings, $\mathbf{KT}_{(\mathbf{m})}$ and $\mathbf{S4}_{(\mathbf{m})}$. DLP is similar in capabilities and design to other description logic systems, including FACT and HAM-ALC.

The system KSATC was designed as a reasoner for the propositional modal logic $\mathbf{K}_{(\mathbf{m})}$. Through the standard embedding it can be used to reason in the propositional modal logic $\mathbf{KT}_{(\mathbf{m})}$. However, KSATC cannot handle transitive modalities, so it is unable to reason in $\mathbf{K4}_{(\mathbf{m})}$, $\mathbf{S4}_{(\mathbf{m})}$, or propositional dynamic logic, except by using a complex encoding that produces very difficult to solve formulae.

2.1. DLP

DLP [36] is based on a modal tableaux procedure. Given an input formula, it attempts to create a modal structure that is a witness to the satisfiability of the input formula. This modal structure consists of a collection of nodes and binary relations between the nodes. Each relation corresponds to a particular modality, and if a node x is related to a note y by a relation r, then y is said to be an r-successor of x. Each node x is annotated with a set of subformulae called its label (denoted $\mathcal{L}(x)$), and a structure is said to be contradictory (or contain a *clash*) if some label contains both a formula and its negation. DLP incorporates and extends the implementation philosophy of FACT [24], a description logic system written in LISP. Disjunctions and other choice points are handled via a backtracking mechanism. If the procedure terminates with a suitable (non-contradictory) structure, the formula is satisfiable; if the procedure cannot find such a structure, the formula is unsatisfiable.

Because DLP allows transitive roles and the transitive closure of roles, it has to be able to handle loops in the modal structure. It does this by looking for nodes with the same label (annotated with the same collection of subformulae) and checking for satisfiability or unsatisfiability of the loop. As the testing analysed in this paper does not have transitive features we will not detail this mechanism of DLP, nor will we detail the other mechanisms that support transitive features.

Each diamond formula ($\Diamond_r \phi$) that is true at a node gives rise to an r-successor node labelled with $\{\phi\}$. Each box formula ($\Box_r \phi$) that is true at a node adds the formula ϕ to the label of each r-successor node. (Transitivity modifies this somewhat.) DLP delays the generation of successor nodes until there is no more propositional processing to be performed at a node. This delay is important for some of the optimisations and enhancements that DLP incorporates.

The details of the algorithm, including details of the processing of the various formulae constructors and the precise termination conditions, are fairly standard, and can be found in [37]. DLP is implemented in Standard ML of New Jersey, using a mostly-functional implementation style.

The basic algorithm sketched above is much too slow to form the basis of a useful decision procedure. DLP thus includes a variety of optimisations to improve its performance. These optimisations were initially chosen in an attempt to improve the processing of description logic knowledge bases but have since been revised based on several experiments with DLP. The optimisations include lexical normalisation, semantic branching search, unit resolution, dependency directed backtracking, caching, and heuristic guided search.

Theoretical descriptions of tableaux algorithms often assume that the formula to be tested is in negation normal form, with negations applying only to propositional variables [23, 3, 7]. Assuming that formulae are in negation normal form simplifies the (description of the) algorithm, but then clashes will only be detected when a propositional variable and its negation appear in the same node label, as in negation normal form negation only applies to propositional variables. If there are complex subformulae that occur both positively and negatively in the input formula, there are opportunities to detect clashes earlier.

Clashes can be detected early by transforming formulae into a lexically normalised form that does not move negation inward. Then a clash can be detected whenever any formula, not just propositional variables, and its negation occur in the same node label.

In the lexically normalised form used in DLP, formulae consist only of propositional variables, conjunctions, and diamond formulae, plus their negations. All other formulae are transformed into one of these forms using the obvious equivalences. The normalisation process used in DLP also removes repeated conjuncts and replaces conjunctions that directly contain a formulae and its negation with the negation of the empty conjunction. Further, conjunction is treated as working on a set of conjuncts, and nested conjunctions are flattened. Tableau expansion

of formulae in this form is no more complex than if they are in negation normal form [25].

Unrestricted branching on disjunctions can be another source of inefficiency in tableaux procedures. If disjunctions are expanded by searching the different models obtained by adding each of the disjuncts, the alternative branches of the search tree are not disjoint, so there is nothing to prevent the recurrence of an unsatisfiable disjunct in different branches, as noted in [19]. The resulting wasted expansion could be costly if discovering the unsatisfiability requires the solution of a complex sub-problem.

This source of inefficiency is dealt with in DLP by using a semantic branching technique adapted from the Davis-Putnam-Logemann-Loveland procedure (DPLL) commonly used to solve propositional satisfiability (SAT) problems [9, 14]. Instead of choosing an unexpanded disjunction in $\mathcal{L}(x)$, a single disjunct ϕ is chosen from one of the unexpanded disjunctions in $\mathcal{L}(x)$. The two possible sub-trees obtained by adding either ϕ or $\neg\phi$ to $\mathcal{L}(x)$ are then searched. Because the two subtrees are strictly disjoint, there is no possibility of wasted search as in syntactic branching. There are other methods for reducing repeated work in tableaux procedures, including improved analytic tableaux [39].

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 *unit resolution* and *heuristic guided search* can be used to try to minimise the size of the search tree.

Unit resolution (UR) is a technique used to maximise deterministic expansion, and thus pruning of the search tree via clash detection, before performing non-deterministic expansion (branching) [14]. Before semantic branching is applied to the label of a node x, UR 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, UR is using the inference rule

$$\frac{\neg\phi, \phi \vee \psi}{\psi}$$

to simplify the expression represented by $\mathcal{L}(x)$.

Inherent unsatisfiability concealed in sub-problems can easily lead to large amounts of unproductive backtracking search known as thrashing. The problem is exacerbated when all propositional reasoning is performed at a node before any successors are examined.

This problem is addressed in DLP by adapting a form of dependency directed backtracking called *backjumping*, which has been used in solving constraint satisfiability problems [5] (a similar technique was

also used in the HARP theorem prover [35]). Backjumping works by labeling formulae with a dependency set indicating the branching points on which they depend. A formula $\phi \in \mathcal{L}(x)$ depends on a branching point when ϕ was added to $\mathcal{L}(x)$ at the branching point or when $\phi \in \mathcal{L}(x)$ depends an another formula $\psi \in \mathcal{L}(y)$, and $\psi \in \mathcal{L}(y)$ depends on the branching point. A formula $\phi \in \mathcal{L}(x)$ depends on a formula $\psi \in \mathcal{L}(y)$ when ϕ was added to $\mathcal{L}(x)$ by a deterministic expansion which used $\psi \in \mathcal{L}(y)$, e.g., if $p \in \mathcal{L}(x)$ was derived from the expansion of $(p \wedge q) \in \mathcal{L}(x)$, then $p \in \mathcal{L}(x)$ depends on $(p \wedge q) \in \mathcal{L}(x)$.

When a clash is discovered, the dependency sets of the clashing formulae can be used to identify the most recent branching point where exploring the other branch might alleviate the cause of the clash. The algorithm can then jump back over intervening branching points *without* exploring alternative branches, which can result in considerable savings. It is possible for this backjumping to jump out of nodes into their ancestors.

During a satisfiability check there may be many successor nodes created. These nodes tend to look considerably alike, particularly as the labels of the r-successors for a node x each contain the same formulae derived from the box formulae $(\Box_r \phi)$ in $\mathcal{L}(x)$. Considerable time can thus be spent performing computations on nodes that have the same label, and as the satisfiability algorithm only cares whether a node is satisfiable or not, this time is wasted.

If the modal successors of a node are only created when all purely propositional processing at that node is exhausted, then all the formulae that would be added to a successor node because of modal formulae in the parent node can be determined when the successor node is first created. The satisfiability status of the successor node is then completely determined by this set of formulae. Under this condition, if there exists another node with the same set of "initial" formulae, then the two nodes will have the same satisfiability status [10].

If the satisfiability status of these sets of formulae are suitably stored, or *cached*, they can then be later retrieved and used to determine the satisfiability status of any new node whose set of initial formulae has already been encountered. Each time this cache is used to successfully determine the satisfiability status of a new node, a considerable amount of processing can be saved, as not only is the work at the new node saved, but also the work at any successors it may have had.

One downside of caching is that the dependency information required for backjumping cannot be effectively calculated for nodes whose processing is eliminated. This happens because the dependency set of any clash detected depends on the dependency sets of the incom-

ing formulae, which will differ between the two nodes. Backjumping can still be performed, however, by combining the dependency sets of all incoming formulae and using that as the dependency set for the unsatisfiable node. This combination may underestimate the amount of backjumping possible, but is generally better than not performing backjumping. Another problem with caching is that it requires that nodes, or at least sets of formulae, be retained until the end of a satisfiability test, changing the storage requirements of the algorithm from polynomial to exponential in the worst case.

Heuristic techniques can be used to guide the search in a way which tries to minimise the size of the search tree. A method which is widely used in DPLL SAT algorithms is to branch on the disjunct which has the Maximum number of Occurrences in disjunctions of Minimum Size—the well known MOMS heuristic [13] and its variants, including the heuristic from Jeroslow and Wang [33]. By choosing a disjunct which occurs frequently in small disjunctions, this heuristic tries to maximise the effect of UR. For example, if the label of a node x contains the unexpanded disjunctions $\phi \vee \psi_1, \ldots, \phi \vee \psi_n$, then branching on ϕ leads to their deterministic expansion in a single step: when ϕ is added to $\mathcal{L}(x)$, all of the disjunctions are fully expanded and when $\neg\phi$ is added to $\mathcal{L}(x)$, UR will expand all of the disjunctions. Branching first on any of $\psi_1, \ldots, \psi_n$, on the other hand, would only cause a single disjunction to be expanded.

Unfortunately this heuristic interacts adversely with the backjumping optimisation by overriding any "oldest first" order for choosing disjuncts: older disjuncts are those that resulted from earlier branching points and will thus lead to more effective pruning if a clash is discovered [24]. Moreover, the heuristic itself is of little value in many interesting modal formulae because it relies for its effectiveness on finding the same disjuncts recurring in multiple unexpanded disjunctions: this is likely in SAT problems, where the disjuncts are propositional variables, and where the number of different variables is usually small compared to the number of disjunctive clauses (otherwise problems would, in general, be easily satisfiable); it is unlikely in modal satisfiability problems, where the disjuncts can be modal formulae, and where the number of different modal formulae is usually large compared to the number of disjunctive clauses. As a result, the heuristic will often discover that all disjuncts have similar or equal priorities, and the guidance it then provides is not particularly useful.

An alternative strategy is to employ a heuristic that tries to maximise the effectiveness of backjumping by using dependency sets to guide the expansion. Whenever a choice is presented, the heuristic

chooses the formula whose dependency set includes the earliest branching points. This technique can be used both when selecting disjuncts on which to branch and when selecting the order in which r-successors are expanded.

The current version of DLP allows for various heuristics. The heuristics used in the tests are to use the oldest-first heuristic to select some disjunctions, and then use MOMS along with some weighting based on the size and kind of a formula to select the best-looking disjunct from amongst these disjunctions.

2.2. KSATC

KSATC is based on Böhm's C implementation of the DPLL algorithm. It implements a decision procedure for $\mathbf{K}_{(\mathbf{m})}$ (multi-modal $\mathbf{K}$). KSATC requires its input formulae to be in extended conjunctive normal form, where each formula is a conjunction of disjunctions of (possibly negated) propositional variables and box formulae. The subformula of these box formulae are disjunctions as above.

Böhm's DPLL algorithm was the winner of a 1992 SAT competition [8]. The algorithm is a standard DPLL algorithm that takes a propositional formula in conjunctive normal form and determines whether the formula is satisfiable by giving assignments to the propositional variables in the formula. It uses a very efficient set of data structures to minimise overhead in determining which variable to assign next and to modify the assignments of other variables. It also implements a collection of smart heuristics to determine the order of variable assignments, including the MOMS heuristic.

In KSATC this DPLL algorithm is modified by allowing modal formulae to masquerade as propositional variables as far as the DPLL algorithm is concerned. Thus the DPLL algorithm assigns values to both propositional variables and modal formulae. Further, the pure literal rule is removed, as it is not valid if the literal is a modal formula. Finally, the DPLL algorithm is modified to allow for the possibility of many instances of the algorithm being active at once.

KSATC uses this DPLL algorithm to determine the propositional satisfiability of a node. When a propositional assignment is found, successor nodes are generated as necessary, and their satisfiability is determined by a recursive call. If these successors are all satisfiable, then the node itself is satisfiable; if not, then the DPLL algorithm resumes control and continues to look for other propositional assignments.

KSATC includes one other important optimisation. Successor nodes are not only investigated when a complete propositional assignment is found, they are also investigated just before a new propositional vari-

able is chosen as a branching variable. Only those successors that have new information are actually investigated, however. If some successor is unsatisfiable at this point, then both branches will be unsatisfiable and do not need to be explored, potentially saving a considerable amount of processing.

KsatC incorporates lexical normalisation, much as Dlp does, although, of course, only for formulae in conjunctive normal form. This amounts mostly to sorting disjuncts and then checking for syntactic identity.

2.3. Differences between Dlp and KsatC

Dlp is tableaux-based, in that it allows assignment to any subformula, even though it performs semantic branching instead of syntactic branching. Because Dlp is tableaux-based, it does not need formulae to be in conjunctive normal form—its normalisation steps do not increase the size of input formulae. KsatC is DPLL-based, in that it assigns only to propositional variables and modal formulae. KsatC requires its input formulae to be in a conjunctive normal form.

Dlp includes a large collection of optimisations, including input formulae normalisation, semantic branching, unit resolution, backjumping between modal nodes, caching of modal node status, and heuristic optimisations. KsatC employs a fast DPLL procedure, and thus incorporates most of its optimisations inside each modal node, augmenting these with a modal lookahead optimisation.

Dlp is implemented in a functional programming style. Its data structures and low-level algorithms are only moderately optimised. For example, Dlp uses a binary tree with functional updates to store the collection of assignments for a node, resulting in changes to the assignments taking time logarithmic in the number of assignments and updates to the status of formulae taking time linear in the number of formulae in the node label. On the other hand, KsatC is written in an imperative programming style, with heavily optimised data structures and low-level algorithms. In KsatC updates to assignments take constant time and updates to the status of formulae take time linear in the number of occurrences of the assigned propositional variable.

We expected that these differences between Dlp and KsatC would result in considerable differences between their performance on various test suites.

3. Previous Testing Methodology

Variants and precursors of DLP and KSATC have been tested on various test suites, including the test suite for the Tableaux'98 propositional modal logic comparison [22] and various collections of random formulae.

Randomly generated modal formulae were were first used by Giunchiglia and Sebastiani [19, 20] to test KSAT_0 and KSAT, two predecessors of KSATC that use the same DPLL-based methodology, but not the same optimised implementation of DPLL. Giunchiglia and Sebastiani took their cue from the testing of non-modal propositional satisfiability decision procedures on randomly-generated 3CNF formulae [11, 38]. Their generator generates formulae in $3\text{CNF}_{\mathbf{K_{(m)}}}$.

A $3\text{CNF}_{\mathbf{K_{(m)}}}$ formula is a conjunction of $3\text{CNF}_{\mathbf{K_{(m)}}}$ clauses. A $3\text{CNF}_{\mathbf{K_{(m)}}}$ clause is a disjunction of three $3\text{CNF}_{\mathbf{K_{(m)}}}$ literals, i.e., $3\text{CNF}_{\mathbf{K_{(m)}}}$ atoms or their negations. A $3\text{CNF}_{\mathbf{K_{(m)}}}$ atom is either a propositional variable or a formula of the form $\Box_r\phi$, where ϕ is a $3\text{CNF}_{\mathbf{K_{(m)}}}$ clause. $3\text{CNF}_{\mathbf{K_{(m)}}}$ formulae capture all $\mathbf{K_{(m)}}$ formulae, as there is a satisfiability preserving transformation from $\mathbf{K_{(m)}}$ into $3\text{CNF}_{\mathbf{K_{(m)}}}$.

The generator of Giunchiglia and Sebastiani has several parameters:

- the number of clauses in the main formula, L.
- the number of propositional variables, N.
- the number of modalities, m.
- the probability that an atom is propositional, p.
- the maximum modal depth, d.

A propositional variable is said to have modal depth 0, the modal depth of a propositional formula is the maximum modal depth of its components, and the modal depth of a modal atom $\Box_r\phi$ is one more than the modal depth of ϕ. Regardless of the value of p, the generator will not choose a modal atom if so choosing would result in an overall formula with depth greater than d.

The generator is similar to the one widely used for 3CNF formulae [38]: the first two parameters (L and N) correspond to two of the 3CNF parameters, and in both generators propositional variables (and modal subformulae in the $3\text{CNF}_{\mathbf{K_{(m)}}}$ generator) are negated with probability 0.5.

Giunchiglia and Sebastiani proceeded by fixing N, m, p, and d. They then generated random formulae for various values of L, and used the resulting formulae to test both their own systems and older decision procedures. Their results showed that their systems, KSAT_0 and KSAT,

were much more effective than previous decision procedures for $\mathbf{K_{(m)}}$. They also showed that multiple modalities, i.e., $m \neq 1$, were easier than single modalities. Subsequent testing has generally been restricted to a single modality.

Unfortunately, the generator of Giunchiglia and Sebastiani suffers from a serious flaw. Instead of selecting different atoms when generating a $3\text{CNF}_{\mathbf{K_{(m)}}}$ clause, their generator allows for repeated $3\text{CNF}_{\mathbf{K_{(m)}}}$ atoms to occur in a $3\text{CNF}_{\mathbf{K_{(m)}}}$ clause. If the atoms are given the same negation status, then the size of the clause is effectively reduced, changing the character of the generated formula. If the atoms are given different negation status the situation can be even worse: tautological or contradictory clauses can be generated, such as $\neg\Box_1(p \vee \neg p \vee q)$. A contradictory clause, even nested deep in a modal subformula, can result in the entire formula being contradictory from just a single one of its clauses. When such formulae are used as input to a decision procedure that includes lexical normalisation, such as DLP, many of the formulae are determined to be satisfiable or unsatisfiable just by lexical normalisation.

Hustadt and Schmidt [31, 32] modified the generator of Giunchiglia and Sebastiani to eliminate some of these problems. The generator of Hustadt and Schmidt ensures that propositional variables only occur once in a $3\text{CNF}_{\mathbf{K_{(m)}}}$ clause. In this way they eliminated many of the problematic formulae generated by Giunchiglia and Sebastiani. With this new generator, and turning off the normalisations of KSAT, they determined that KSAT is much less dominant over previous decision procedures and that their system, TA, is faster than KSAT. This result is somewhat controversial in that it can be claimed on the one hand that normalisation is an integral part of the KSAT system, while on the other hand it can be seen as extraneous to the underlying decision procedure.

We used the improved generator of Hustadt and Schmidt to test the performance of several systems [28, 26, 29], including a previous version of DLP as well as KSAT, as part of a larger series of tests of various modal decision procedures. We concluded that DLP was faster than KSAT, although the speed difference was least on the random formulae.

Giunchiglia, Giunchiglia, Sebastiani, and Tacchella [18] took the improved generator of Hustadt and Schmidt and further improved it. The generator of Hustadt and Schmidt only removes repetitions of propositional variables from clauses. Even if the depth parameter d is 1, this will not remove all repetitions from within clauses, as modal atoms can reoccur. This happens because the number of propositional variables is very small in current tests, and the number of different modal atoms of

depth 1 is thus quite small: with 4 propositional variables there are only 32 different depth-1 modal atoms [24]. As the testing often involves top-level formulae with hundreds of clauses there is a significant chance that a top-level formula includes clauses that have repetitions or are tautologies or contradictions. The presence of such clauses can significantly change the expected difficulty of a formula. In our previous testing we found that with formulae from the Hustadt and Schmidt generator, as well as from the earlier Giunchiglia and Sebastiani generator, lexical normalisation was the most important optimisation in DLP.

Giunchiglia *et al* modified the Hustadt and Schmidt generator so that it does not generate repeated modal formulae and tested their improved system, KSATC, on generated formulae. They found that their improved system (with normalisation turned on) was much faster than TA, often by orders of magnitude.

This later testing suffers from flaws of a methodological nature. Hustadt and Schmidt observed that some of the formulae become unsatisfiable simply from top-level propositional interaction[1], and can be determined to be unsatisfiable without investigating any modal successors. They proposed that this factor be eliminated by ensuring that propositional letters only occur at the maximum modal depth, and that this be achieved by setting the parameter p to 0. Their own testing was of this form, as has been most subsequent testing.

Although it is true that formulae that are propositionally unsatisfiable are not very hard, it is not true that test suites that contain significant numbers of propositionally unsatisfiable formulae are uninteresting. First, even formulae with propositional variables only occurring within modal formulae can be determined to be unsatisfiable by DLP without examining any modal successors, because the modal subformulae may repeat in different clauses. Second, restricting propositional variables to only occur at the deepest level eliminates many kinds of formulae. As different systems may perform well (or badly) on different sorts of formulae, it is not a good idea to so severely restrict the sorts of formulae considered.

On the basis of Giunchiglia and Sebastiani's results for formulae of varying modal depth, Hustadt and Schmidt concluded that, relative to the size of formulae, the hardest problems were generated when the modal depth was 1, and they thus fixed d at 1 in all their subsequent experiments. Unfortunately, this result is partly an artifact of Giunchiglia and Sebastiani's generator in that the probability of gen-

[1] They call these formulae *trivially unsatisfiable*, although they are different from the problems called trivially unsatisfiable in random constraint satisfaction problems and random QSAT problems [15].

erating formulae amenable to the normalisation optimisation increases with increasing modal depth.

4. Our Testing Methodology

To show the importance of testing with different sorts of formulae, and also to evaluate the performance of DLP and KSATC, the best-performing modal decision procedures, we have developed a new random formula generator and performed several experiments employing it. These experiments extend the range of hard problems for propositional modal logics and show the effects of the differing optimisations.

Our generator is similar to the twice-improved generator of Giunchiglia *et al* [18], but modified so that it is able to generate less uniform formulae, e.g., by employing different probabilities for negating propositional variables and modal formulae. The modified generator generates $\mathrm{CNF}_{\mathbf{K}_{(\mathbf{m})}}$ formulae employing the following parameters:

- the number of clauses in the main formula, L.
- the number of propositional variables, N.
- the number of modalities, m.
- the maximum modal depth, d.
- the minimum and maximum number of clauses in a disjunct, c_{min}, c_{max}, with equal probability for each number in the range.
- the probability that an atom is propositional, p (except at depth d where all atoms are propositional).
- the probability that a modal formula is negated, n_m.
- the probability that a propositional variable is negated, n_p.

When generating the atoms in a clause, our generator employs a scheme that is equivalent to picking formulae from a population, without repetition, so that the modal structure of a formula is defined by p. The generator does *not* ensure that the $3\mathrm{CNF}_{\mathbf{K}_{(\mathbf{m})}}$ clauses in the main conjunctive formula are distinct, effectively picking clauses from a population, but in this case with the possibility of repetition. Because the generator is often employed for small N but large L, there is the distinct possibility that there would be too few clauses of a certain form, such as purely propositional clauses, to allow for distinct clauses while retaining the correct probabilities for modal structures.

We restricted the testing as follows. We used only 1 modality ($m = 1$) because, as discovered earlier, multiple modalities generally cause the problems to become easier. We used only $3\text{CNF}_{\mathbf{K}_{(\mathbf{m})}}$ formulae ($c_{min} = c_{max} = 3$). We set the probability of negated propositional variables to 0.5 ($n_p = 0.5$).

However, we varied all the other parameters, not just L and N as in recent testing. We varied the propositional probability (p), using 0.5 and 0.4, as well as 0.0. We varied the maximum modal depth (d), using 2 as well as 1. We varied the probability of negated modal formulae (n_m), using 0.1, as well as 0.5. We varied the number of propositional variables (N) from 3 to 9, ranging the number of clauses from N to $150N$ in increments of N, and generating 100 formulae at each data point. We also performed a separate series of tests with $d = 100$ and p between 0.9 and 0.7. The parameters used in the various tests are summarised in Figure 1.

Both DLP and KSATC were tested using the same seed for the random number generator in the formula generator. We did not see any disagreements between DLP and KSATC on any of the formulae although we did not make a systematic search for disagreements.

We would have liked to try all combinations of the parameters above. However, this would have taken much too long, so we had to pick various combinations of the parameters and various ranges of N. Moreover, we used a timeout of 1,000 seconds of CPU time, and terminated processing on a data point as soon as it became evident that the median time would exceed the timeout. All times reported are CPU times, provided from DLP and KSATC, exclusive of any time required to read, translate, or preprocess the formulae: the extra time just confuses the results, and would include different sorts of processing for the two systems.

4.1. TESTING DETAILS

The February 1999 version (version 3.2) of DLP was used for the tests. This version is very close to previous versions of DLP; the only differences are some optimisations to make processing of large formulae more efficient. This version of DLP is available at `http://www.bell-labs.com/user/pfps/dlp`. DLP was run under Standard ML of New Jersey, version 110.0.3.

A single setting of the options for DLP was used throughout the tests. All the optimisations mentioned above were turned on, and the heuristics were set to select a formulae from among the backjumping-oldest clauses using a MOMS-style heuristic. The code to handle the transitive closure of modalities was turned off.

For the tests KSATC was compiled under g++, version 2.7.2.3, with optimisation -O2, as in the supplied makefile. The version of KSATC used for previous testing was used in these tests. It is available at `ftp://ftp.mrg.dist.unige.it/pub/mrg-systems/KR98-sources/KSat-source/KSatC`.

Our testing was performed on two classes of machines. Most testing was performed on upgraded SPARCstation 20s with two 150 MHz Ross RT626 CPUs, roughly equivalent to a 140M SPARC Ultra-1, and 128M of main memory. Testing for propositional probability 0.0 was performed on a SPARC Ultra-2 with two 296 MHz UltraSPARC-II CPUs and 256M of main memory. The testing machines were otherwise very lightly loaded. Tests that were rerun showed almost exactly the same timing; in all examined cases the difference was less than the maximum of 0.02 seconds or 1 percent of the run time. Using a different seed for the random number generator *did* result in different timing results, but the qualitative and comparative results were unchanged in all examined cases.

The formula generator was coded in Standard ML of New Jersey. Each test started with a known random seed, which was not reset during the run. The random number generator was used only to generate formulae, so that the tests would be reproducible. If a test was terminated prematurely, it was restarted by reading the results of the previous run of the test, generating random formulae without testing them for each test performed by the previous run, and then continuing in the normal fashion.

For the DLP tests, the formulae were generated and tested in the same process. Each test was terminated as soon as the time limit of 1,000 CPU seconds was exceeded, as calculated by the built-in timing facilities of Standard ML of New Jersey. The times reported include any garbage collection that happened during the test. For the KSATC tests, the formulae were written in a format acceptable to KSATC and then read in and tested by a different process. This separate process was limited to 1200 seconds of real time; if this time limit was exceeded, the test was deemed to have take at least 1000 seconds of CPU time. This extra limit was needed, as KSATC sometimes did not terminate gracefully. As the testing systems were very lightly loaded, it is extremely unlikely that any results would have been different without this real time limit.

The results of our testing are presented in two kinds of graph. The first and most common kind of graph plots run time against the ratio of clauses to number of propositional variables (L/N) for a single test, i.e., various values of N, and fixed values of maximum modal depth (d), propositional probability (p), and modal negation probability (n_m).

Tests	N	m	d	c_{min}	c_{max}	p	n_m	n_p
1	3–9	1	2	3	3	0.5	0.5	0.5
2	3–9	1	1	3	3	0.5	0.5	0.5
3	3–7	1	1	3	3	0.0	0.5	0.5
4	3–8	1	1	3	3	0.4	0.5	0.5
5	3–7	1	2	3	3	0.5	0.1	0.5
6	3–8	1	2	3	3	0.4	0.5	0.5
7	3–6	1	100	3	3	0.9–0.7	0.5	0.5

Figure 1. Test parameter settings

Each graph shows, on a log scale, either the median, the 90th percentile, or the worst case run time for either DLP or KSATC. The Pth percentile is the maximum time taken by the fastest $P\%$ of the tests, so the 50th percentile would be equivalent to the median.

The second kind of graph is three-dimensional and plots, again on a log scale, the fiftieth through one hundredth percentile run times against L/N for a single test and a single value of N. Overlaid on these plots are dotted surfaces that show the ratio of satisfiable to total solvable formulae, as determined by DLP or KSATC, provided that the program could solve at least half of the formulae at that point.

5. Test Results

The first series of tests that we performed was for maximum modal depth 2, propositional probability 0.5, and modal negation probability 0.5. We ran tests with 3 through 9 propositional variables. The tests with 3, 4, and 5 variables correspond to some of the original tests performed by Giunchiglia and Sebastiani [20], while those with 3, 4, 5, and 8 variables correspond to some of the tests performed by Hustadt and Schmidt, although in both cases the improved formulae generator now eliminates local redundancies, tautologies, and contradictions.

The results for DLP and KSATC are given in Figure 2. These formulae are not all easy, and there is a general easy-hard-easy behaviour for both DLP and KSATC. However, the shape of the curves is very different from the narrow, pronounced peak in propositional satisfiability. Instead, there is a general "half-dome" shape to the curves. This shape is caused by two factors. First, there are many modal successors

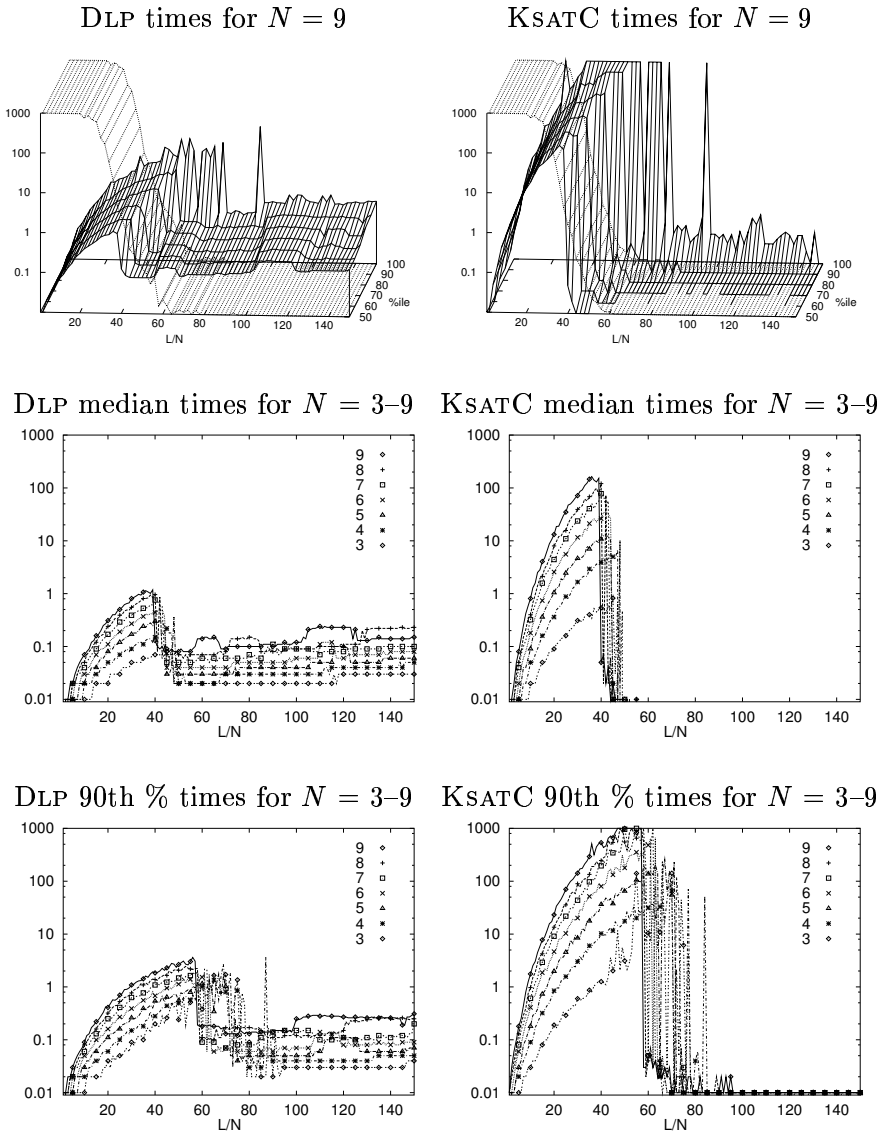

Figure 2. Results for test 1 ($d = 2$, $p = 0.5$, and $n_m = 0.5$)

and if the formula is satisfiable they all have to be investigated. If the successors are all or almost all easily determined to be satisfiable then they contribute a polynomial growth in difficulty as the formula grows in size, resulting in the left-hand-side of the half-dome. Second, at some point enough formulae become trivially unsatisfiable, and thus easy for both DLP and KSATC, resulting in the steep "cliff". This point

varies somewhat with the number of variables and there may be some "ringing", particularly at the higher percentiles.

The only major exception to this smooth half-dome shape occurs with small numbers of propositional variables, most prominently at $N = 3$. Here for L/N in the range 50–70 there are some formulae that are harder than would be indicated by half-dome-shaped successor evaluation. This behaviour shows up most clearly at the higher percentiles. The hard formulae are those where there are a significant number of modal successors whose satisfiability status is hard to determine, resulting in backtracking and a peak in the difficulty. These extra-hard formulae are highlighted in Figure 3.

DLP generally performs better than KSATC for these tests, being often two orders of magnitude faster than KSATC. However, DLP is not always faster than KSATC.

In the trivially-unsatisfiable region KSATC is faster than DLP (less than 0.01 seconds compared with about 0.1 seconds). This is because DLP's data structures and low-level algorithms are not heavily optimised, and it takes several hundredths of a second just to go through these large formulae. The DPLL algorithm in KSAT can perform single passes over a large set of clauses in a very short time, and KSAT can thus process large trivially unsatisfiable formulae very quickly.

For data points in the L/N range from about 20–50, where all or most of the successors are satisfiable, DLP is uniformly faster than KSATC. Much of this difference is due to KSATC investigating modal successors every time a new branching variable is chosen. Modal successors do not produce much, if any, unsatisfiability in these tests, so the investigation of modal successors does not cut off search. Worse, KSATC has to investigate some modal successors very often, as each time a new box formula is assigned true, every modal successor has to be re-examined.

Even outside the above ranges (of L/N values), DLP consistently performs better than KSATC for *satisfiable* formulae. Although there may be some benefit in examining modal successors early for these formulae, this benefit would occur only when a large number of box formulae have been assigned true. Because of the repeated work performed by KSATC on such successors, it does not appear that the benefits are realised.

This series of tests shows that it is not a good idea to restrict testing to depth 1 and propositional probability 0, as this would ignore at least one area where DLP significantly outperforms KSATC.

The second series of tests that we performed was for maximum modal depth 1, propositional probability 0.5, and modal negation prob-

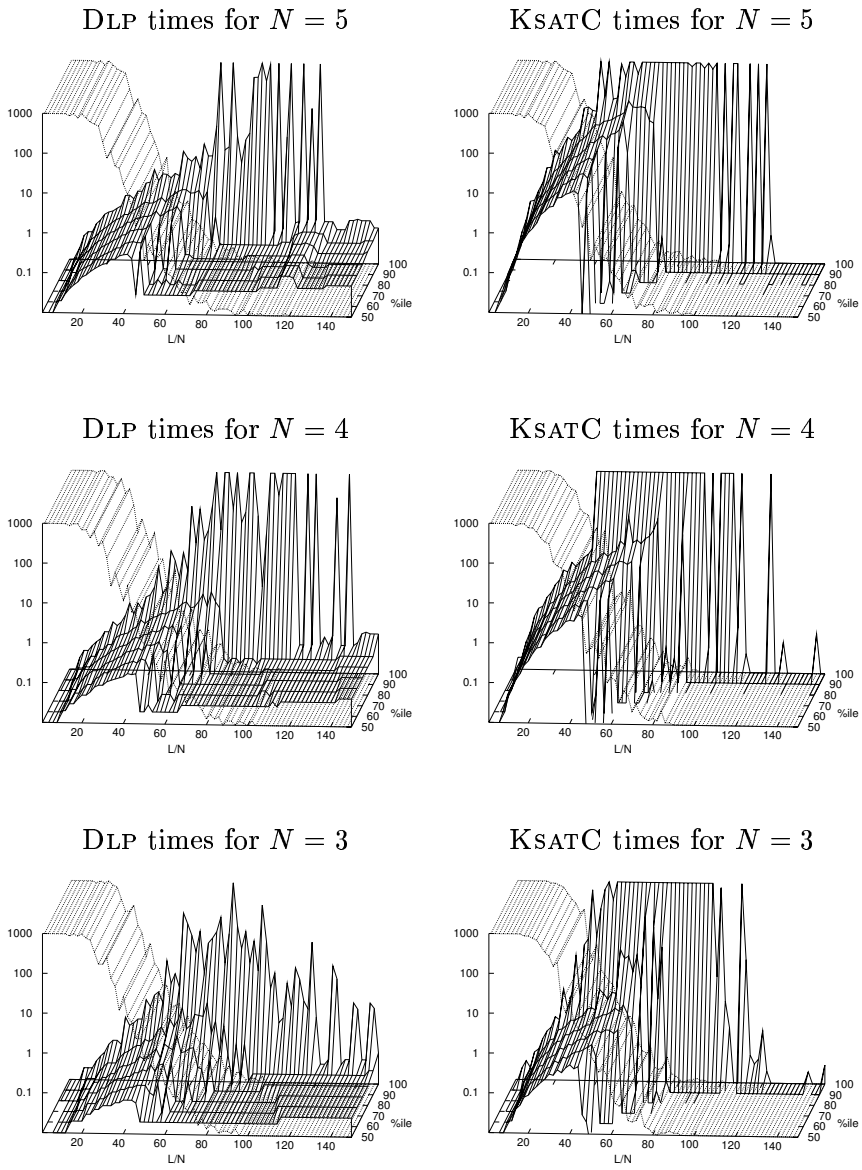

Figure 3. Detailed results for test 1 ($d = 2$, $p = 0.5$, and $n_m = 0.5$)

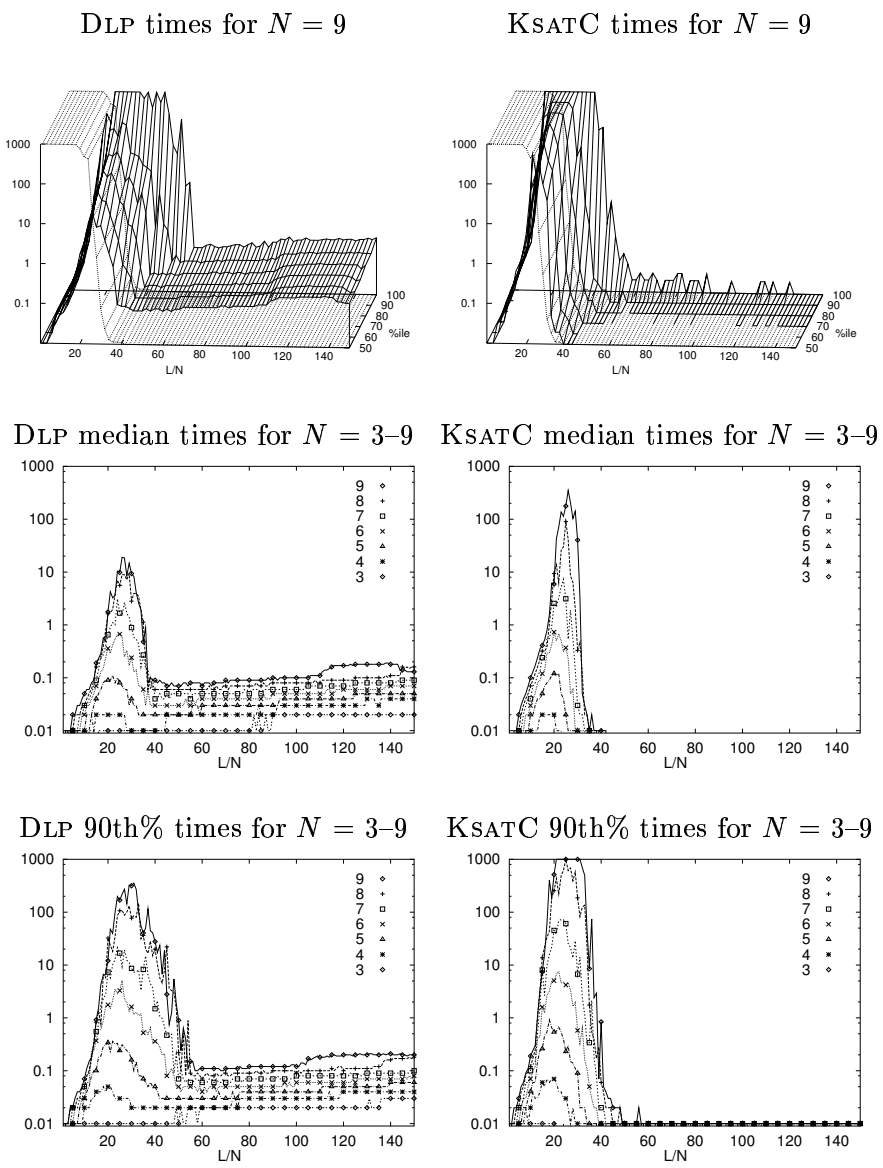

Figure 4. Results for test 2 ($d = 1$, $p = 0.5$, and $n_m = 0.5$)

ability 0.5. We again ran tests with 3 through 9 propositional variables. These tests do not correspond to any previous tests, as all previous testing with propositional probability different from 0.0 was performed with maximum modal depth greater than 1.

The results of the tests for DLP and KSATC are given in Figure 4. In these tests there is a more distinct easy-hard-easy behaviour with

a pronounced peak from both DLP and KSATC although the peak is quite wide by propositional satisfiability standards.

Again, DLP generally performs better than KSATC, but the difference is less: only slightly more than one order of magnitude. Also, there is a new area where KSATC outperforms DLP.

In these tests, when the formulae are mostly unsatisfiable but not trivially unsatisfiable, the modal successors do provide an important source of the unsatisfiability. KSATC's early investigation of the modal successors can thus be used to cut off search. This shows up in the results for values of L/N around about 40, where KSATC is considerably faster than DLP.

As pointed out by Hustadt and Schmidt, the tests with maximum modal depth 1 are in some sense *harder* than the tests with maximum modal depth 2. This is true almost uniformly for DLP for larger numbers of propositional variables, except where the low-level overhead is most prominent. It is also true in some sense for KSATC, as with a given number of propositional variables the hardest median depth-1 problems are harder for KSATC than the hardest depth-2 problems. If the difference in average formula size between the two sets of formulae is taken into consideration, the depth-1 formulae are decidedly harder.

This being the case, why should the depth-2 formulae be investigated at all? First, it is not appropriate to limit investigations to just some small group of formula unless the excluded formulae can be transformed into the investigated ones. This is certainly not the case here, as a limitation to a modal depth of 1 removes from consideration the vast majority of modal formulae. Second, the hardest formulae may change with different systems. Although this is not the case here, at least for DLP, there are significant differences in the results for the groups of formulae. If attention had been paid to only one group, these differences would not have been seen.

But why is modal depth 1 "harder" than modal depth 2? Well, both DLP and KSATC include the MOMS heuristic. Because there are more modal atoms than propositional variables, and because modal atoms and propositional variables are picked with the same probability (except at the deepest depth), the atom that has the highest MOMS score will almost certainly be a propositional atom. Therefore both DLP and KSATC will end up with assignments at the root node that are heavily weighted to giving values to propositional variables. The only modal atoms that will be given assignments are those that are required to make all the clauses evaluate to true.

So how many modal atoms will be given assignments? Clauses with three propositional literals will generate no assignments to modal atoms.

Clauses with two propositional literals will, on average, have one of their propositional literals assigned to true in three out of four cases, and thus will produce an assignment to a modal atom in only one out of four cases. Clauses with one propositional literal will, on average, have it assigned to true in one out of two cases, and thus will produce an assignment to a modal atom in only one out of two cases. Clauses with no propositional literals will give rise to an assignment to a modal atom in every case.

If we make the assumption that modal atoms occur only once[2], then for propositional probability 0.5 about

$$\frac{3}{8}\frac{1}{4} + \frac{3}{8}\frac{1}{2} + \frac{1}{8}1$$

or 13/32 of the clauses will give rise to an assignment to a modal atom. About half of these assignments will result in a box formula and about half in a diamond formula. Therefore at the next depth there will only be about 13/64 or 1/5 as many clauses as at the current depth.

For maximum modal depth of 1, this means that a top-level formula with about $30N$ clauses will give rise to modal successors with about $6N$ clauses. Such clauses will be propositional, and because clauses can occur multiple times at the top level, this is just about at the hardest L/N point for propositional formulae. This is also near the point where the strictly propositional clauses in the formula as a whole (1/8 of the total clauses) are providing a non-trivial problem at that level. The compounding of the two sources of hardness results in very hard problems.

As the number of clauses increases, the sub-problems generated become more unsatisfiable, so early detection of this unsatisfiability becomes more beneficial. This is precisely where KSATC outperforms DLP.

For maximum modal depth of 2, a top-level formula with about $40N$ clauses will give rise to modal successors with about $8N$ clauses. Such clauses will have maximum modal depth 1, and give rise to problems similar to (but not exactly the same as) the maximum modal depth 1 problems generated by the formula generator. With only $8N$ clauses, the chance of such problems being satisfiable is almost exactly 1.0, and both DLP and KSATC are able to find a satisfying assignment in very little time and with very little search. Therefore, the time taken here is due to the very large number of modal successors investigated rather than to their individual hardness. As KSATC goes through this search

[2] This is not a valid assumption for small numbers of propositional variables and large numbers of clauses, but it is not too far off for the analysis here, except for very small numbers of propositional variables where the analysis breaks down somewhat.

multiple times for each top-level assignment, it takes longer on some depth 2 problems. As DLP does this investigation only once for each top-level assignment, and thus often only once in total, it is much faster on these sorts of problems.

Top-level formulae with more clauses are usually trivially unsatisfiable, and thus do not give rise to any modal processing. Those top-level formulae that are not trivially unsatisifiable, however, can be difficult, as their modal successors may have enough clauses to be moderately difficult in their own right. This is the cause of the occasional difficult problems for $L/N > 80$.

Our third series of tests has maximum modal depth 1, propositional probability 0.0, and modal negation probability 0.5. We ran tests with 3 through 7 propositional variables. These are similar to some of the tests performed by Hustadt and Schmidt and by Giunchiglia *et al* [18].

The results of the tests for DLP and KSATC are given in Figure 5. The qualitative behaviour of both DLP and KSATC is yet again different. There is an easy-hard-easy pattern, but the hard section is spread over most of the L/N values and the formulae only gradually become easier.

In these tests KSATC performs much better than DLP. This is to be expected, as almost all of the unsatisfiability comes from the modal successors, which are all easy to investigate, because there are a small number of propositional variables and the maximum modal depth is 1. Because KSATC aggressively investigates the modal successors it can significantly reduce search in the root node, and for these formulae the advantage of this reduction outweighs the disadvantage of potentially repeated investigations of each successor. DLP, on the other hand, does not investigate modal successors until it has discovered a complete top-level assignment. It will therefore generate many more top-level assignments and have to examine the modal successors, although in many cases these will be nearly the same. DLP's caching of modal results does not help very much as there are so many different modal successors that can be generated.

We next ran a series of tests with an "intermediate" value for the propositional probability to see where the cross-over between DLP and KSATC lies. These tests have maximum modal depth 1, propositional probability 0.4, and modal negation probability 0.5.

The results of the tests for DLP and KSATC are given in Figure 6. Here the two systems are very close in performance, with KSATC being slightly faster and having a smaller range of L/N in which it finds hard problems. With a propositional probability of 0.4, the benefits of

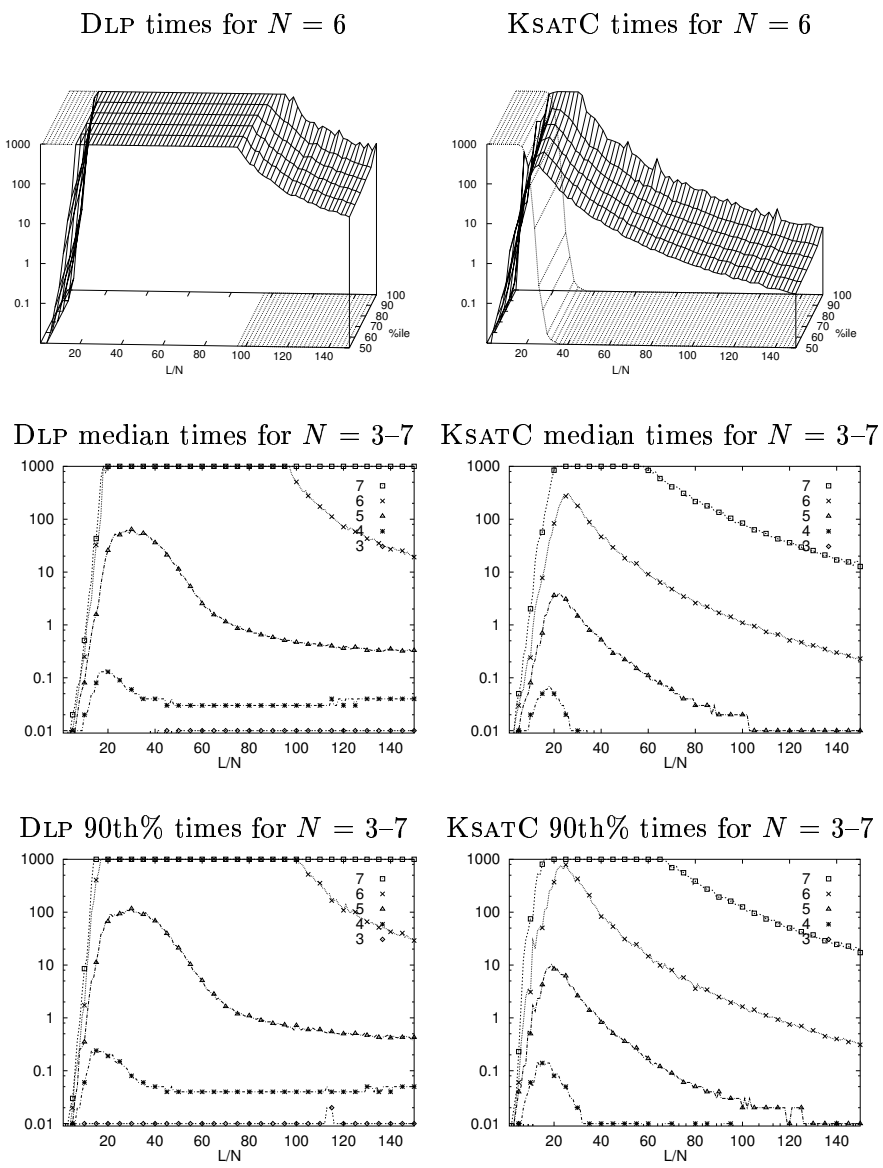

Figure 5. Results for test 3 ($d = 1$, $p = 0.0$, and $n_m = 0.5$)

KSATC's aggressive modal investigation are starting to outweigh its detriments, especially in the slightly overconstrained area.

The qualitative behaviour of both systems is intermediate between the two previous tests, with a pronounced peak in difficulty but one quite a bit wider than the peak for $p = 0.5$.

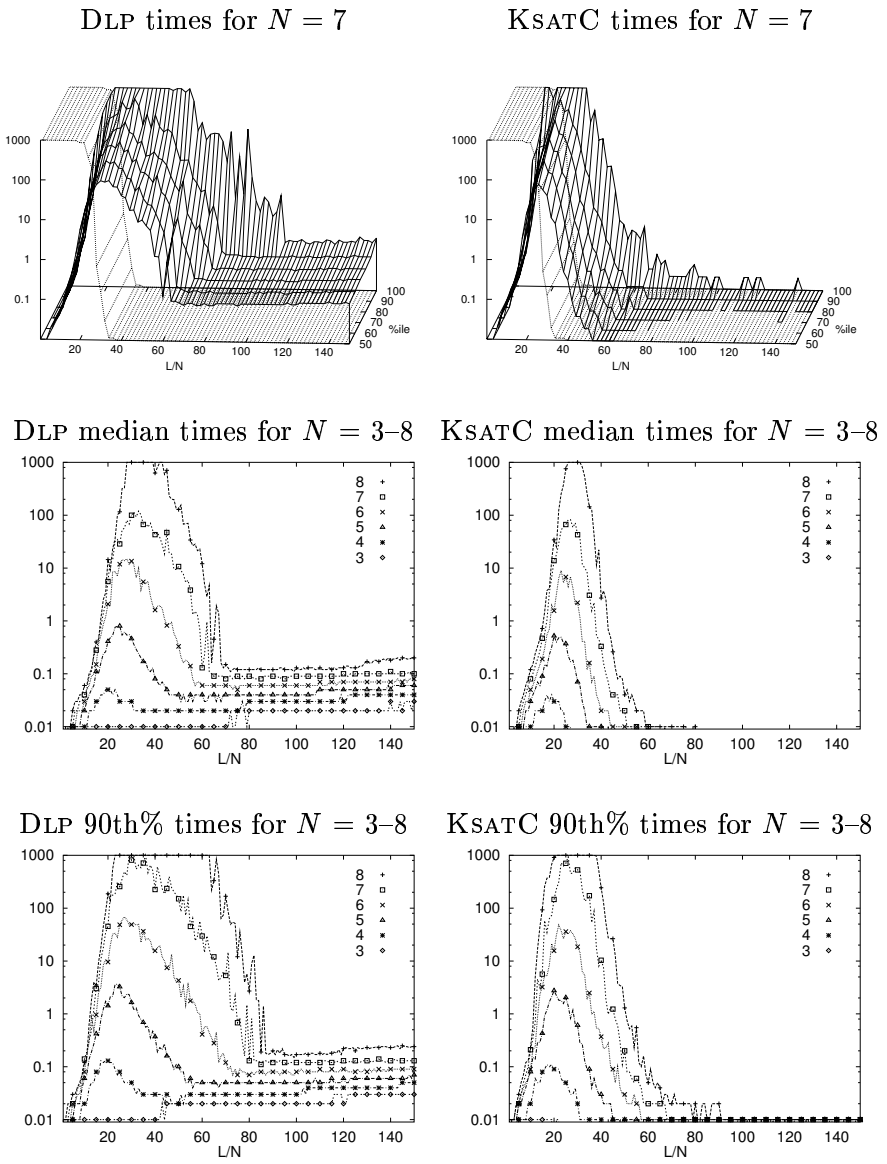

Figure 6. Results for test 4 ($d = 1$, $p = 0.4$, and $n_m = 0.5$)

So with a maximum modal depth of 1, DLP performs better than KSATC when the propositional probability is high and worse when the propositional probability is low. When the propositional probability is low the modal successors have a high chance of being overconstrained and KSATC's early investigation of the incomplete successors will find unsatisfiability early, thus cutting off considerable amounts of search.

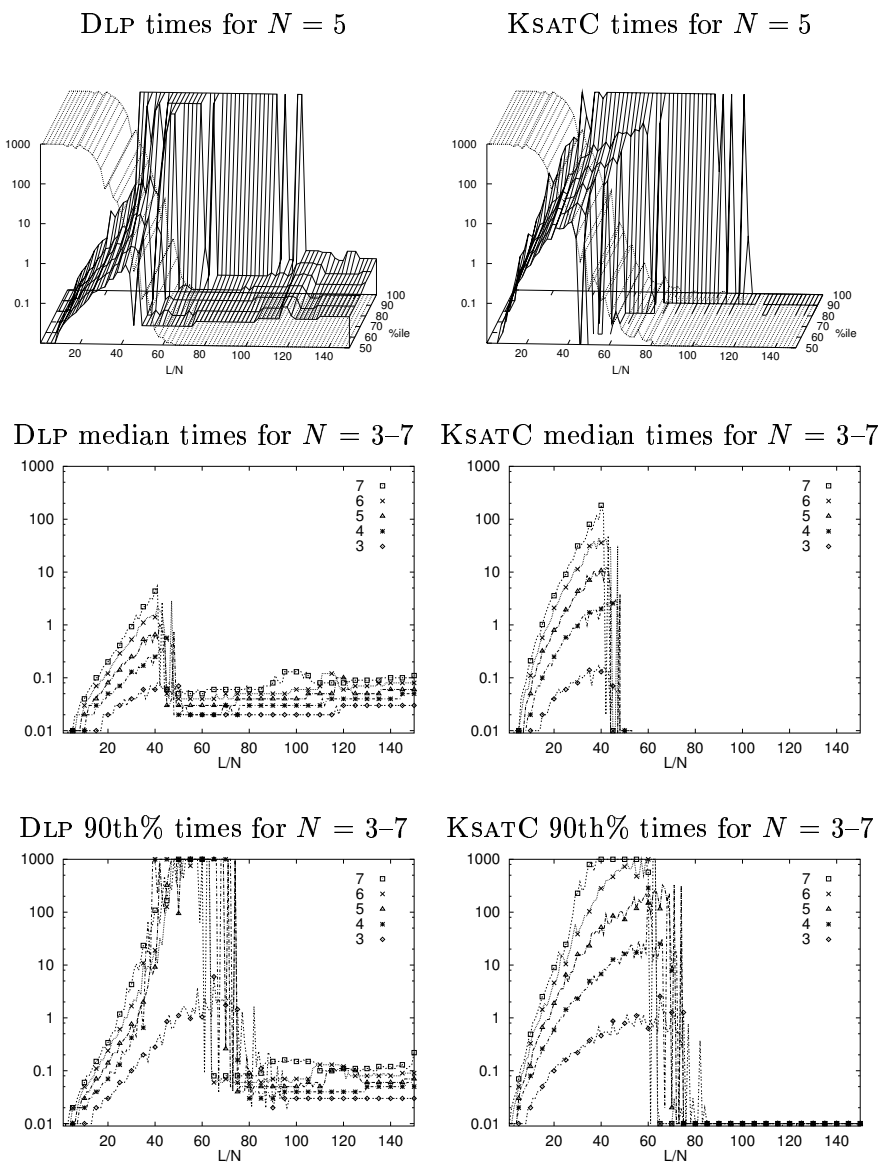

Figure 7. Results for test 5 ($d = 2$, $p = 0.5$, and $n_m = 0.1$)

When the propositional probability is high the modal successors have a high chance of being satisfiable, or only being unsatisfiable when all information is known about them. Here, most or all of KSATC's early investigation of incomplete successors will be wasted effort.

Our fifth set of tests further investigated formulae with a maximum modal depth of 2. There are two ways to make the tests with maximum modal depth of 2 harder. The first approach is to reduce the propositional probability. The second approach to make the tests harder is to have fewer but larger modal successors by reducing the probability of negating a modal atom, which, in this generator, are always box formulae. This approach has the advantage that the change does not affect the size of the resulting formulae, whereas reducing the propositional probability considerably increases formula size.

We performed a series of tests with maximum modal depth 2, propositional probability 0.5, and modal negation probability 0.1. The results for DLP and KSATC are given in Figure 7. In these tests DLP has by far the better median performance, by a couple of orders of magnitude. However, DLP is slower on the harder formulae in this series. This is because the harder formulae provide places for KSATC's early investigation of modal successors to cut off search, whereas DLP has to investigate many more choice points before determining that the modal successors block any solution.

We also performed testing with maximum modal depth 2 and propositional probability 0.4. These results of these tests are given in Figure 8. Again DLP is generally faster than KSATC, often by several orders of magnitude. The only place where DLP is slower than KSATC is in the trivially unsatisfiable region.

KSATC's early investigation of modal successors performs very poorly in these tests, as indicated by the relative height of the half-dome section of the tests. Even where there are hard modal successors, and early investigation might provide some early cutoffs, KSATC performs worse than DLP. This is probably due to the backjumping optimisation in DLP.

These tests have some of the half-dome behaviour indicative of many but easy modal successors. However, there is even more indication of backtracking behaviour, most prominent for small numbers of propositional variables, but still evident with 8 propositional variables. Plots detailing this behaviour are given in Figure 9.

The plots show that, for DLP, the tests with smaller numbers of propositional variables are harder than the test with 8 propositional variables. This unusual behaviour occurs because of conditions in the modal successors for very small numbers of propositional variables having to do with the few modal formulae of depth 1 and the presence of conjunctions supplied by the top-level diamond formula. These conditions give rise to modal subproblems that are more difficult than they

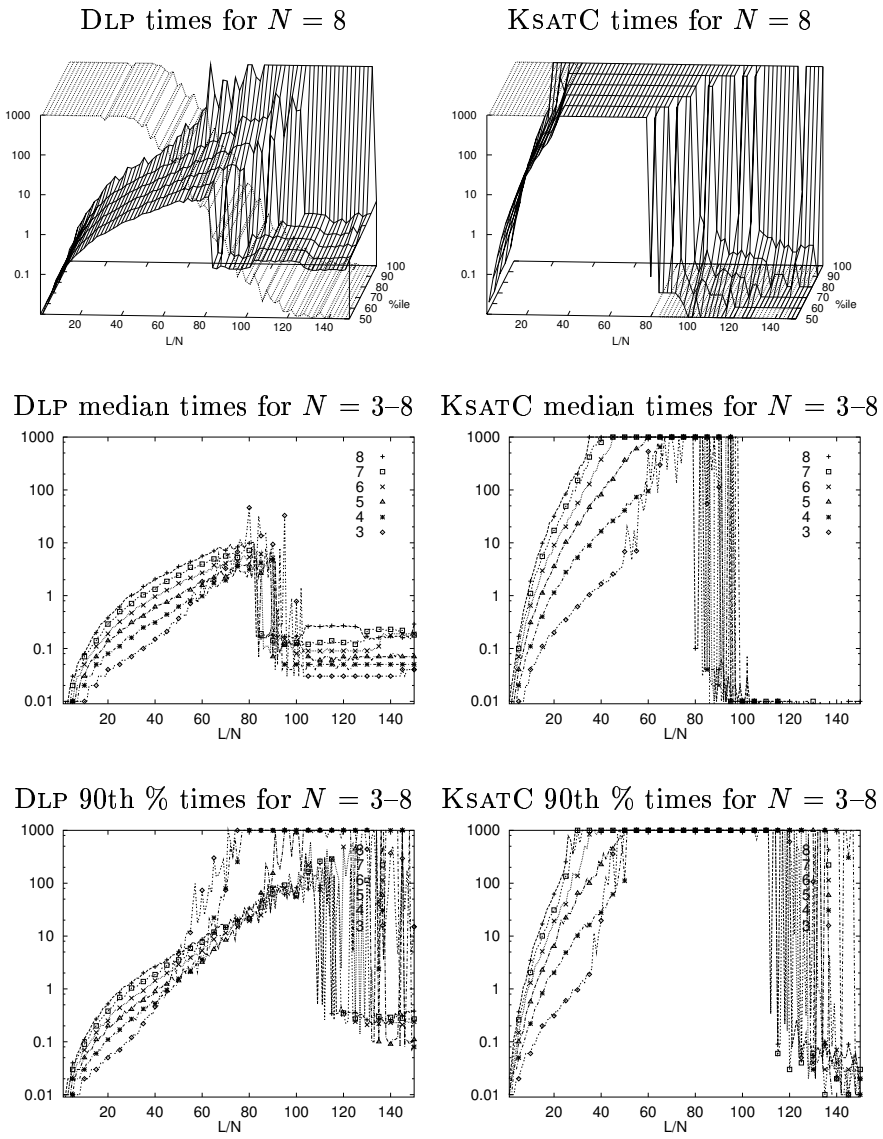

Figure 8. Results for test 6 ($d = 2$, $p = 0.4$, and $n_m = 0.5$)

would otherwise be. KSATC experiences similar hardness effects but they are masked by its repeated examination of the modal successors.

For our last set of tests we investigated a very different kind of formulae. We wanted to see what would happen if there were no "hard stop" to the modal structure, and thus create formulae where the modal subformulae looked just like the main formula, except smaller. One way

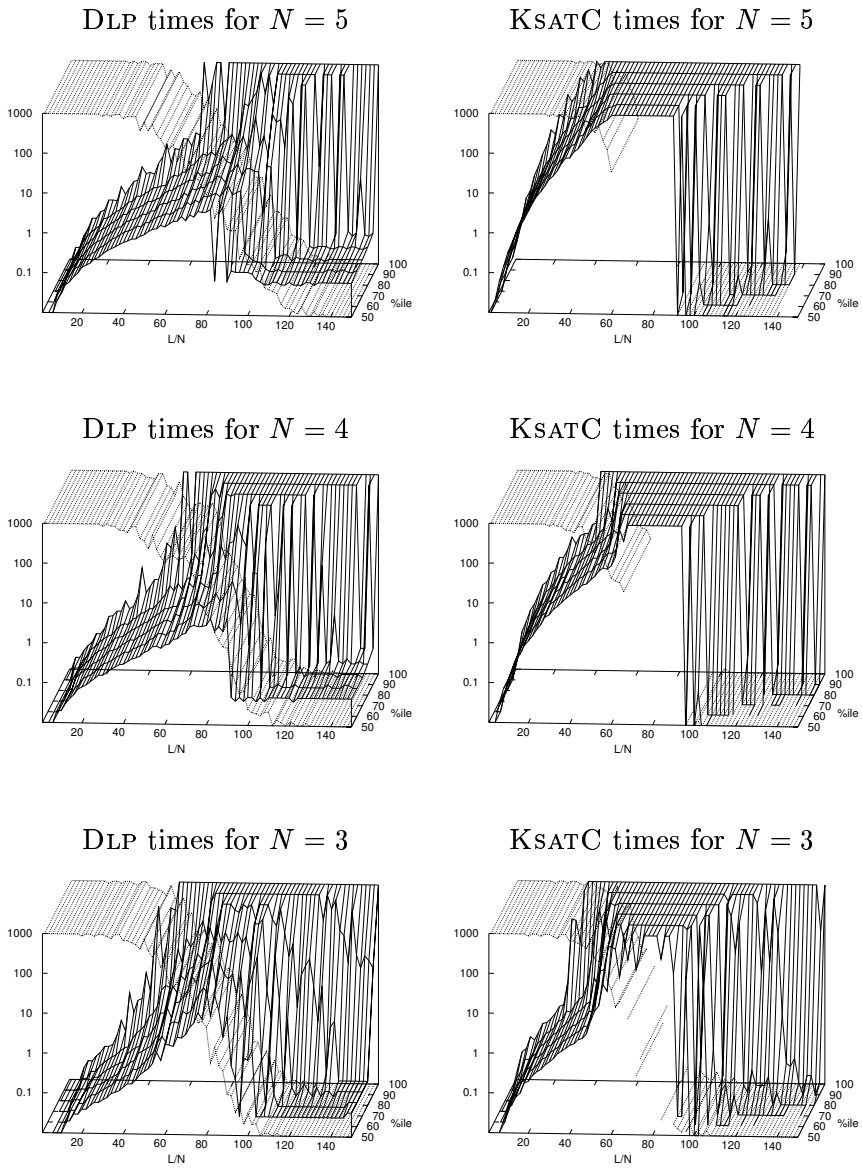

Figure 9. Detailed results for test 6 ($d = 2$, $p = 0.4$, and $n_m = 0.5$)

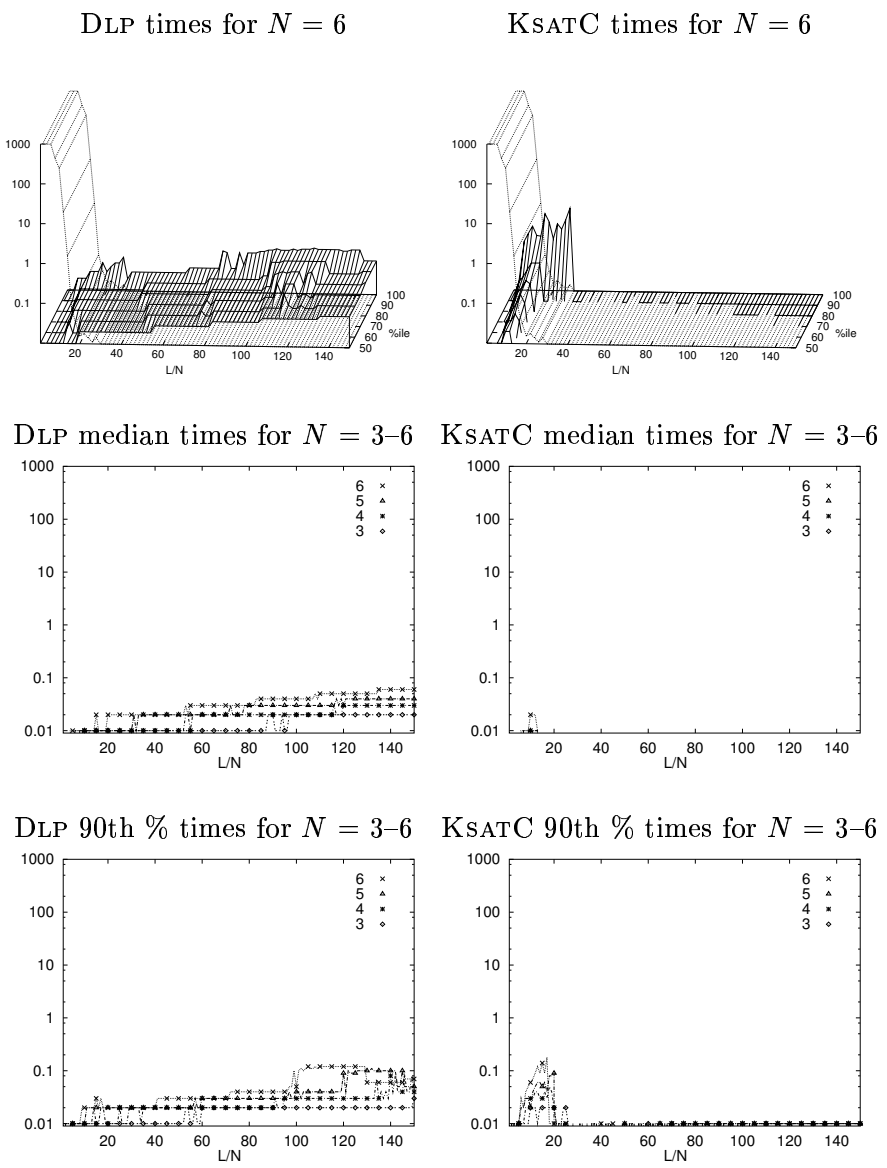

Figure 10. Results for test 7, part 1 ($d = 100$, $p = 0.75$, and $n_m = 0.5$)

of creating such formulae would be to have no maximum modal depth. We approximated formulae with no maximum modal depth by setting the maximum modal depth parameter, d, to 100. We ran tests with the propositional probability, p, ranging from 0.9 to 0.7 and the number of propositional variables, N ranging from 3 to 6.

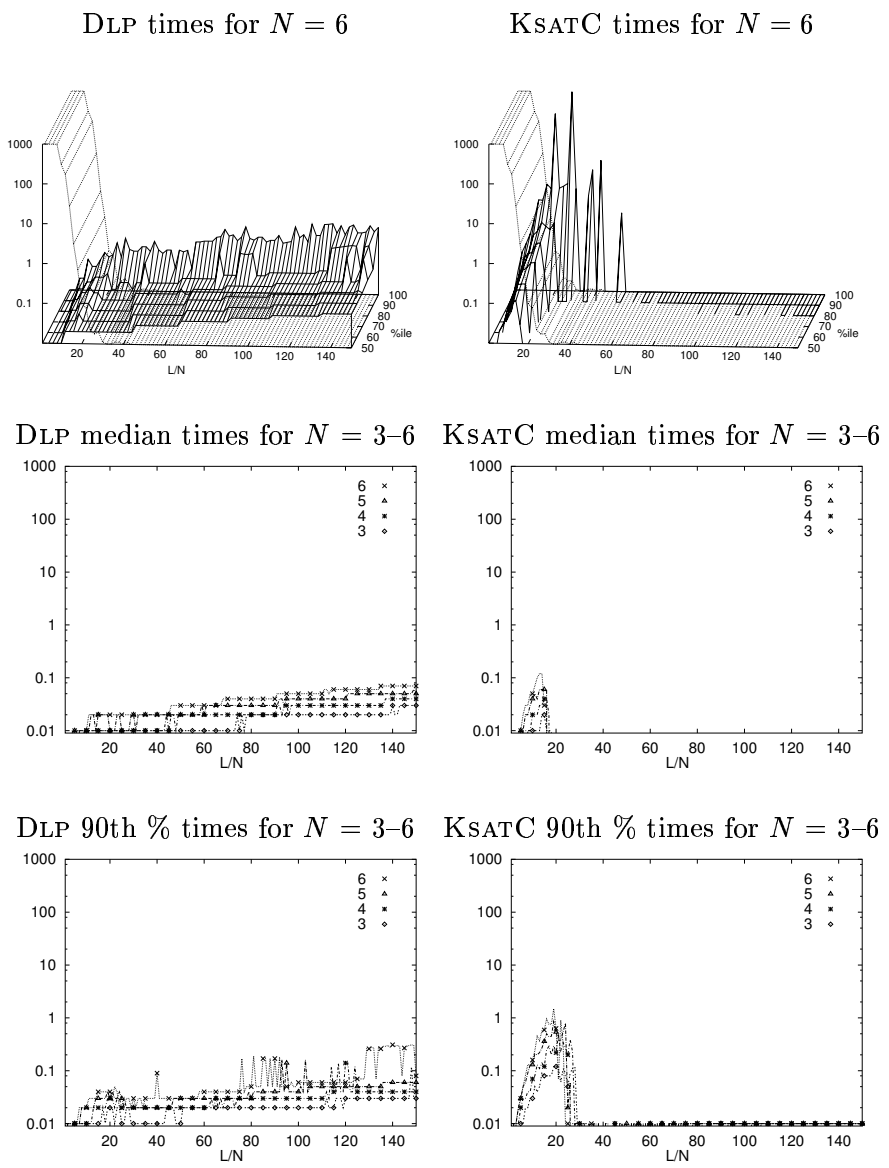

Figure 11. Results for test 7, part 2 ($d = 100$, $p = 0.7$, and $n_m = 0.5$)

In these tests the propositional probability cannot be lowered much beyond 0.7 or else the generated formulae become too large to store. Even with a propositional probability of 0.7 the formulae generated are very large, as at each level the expected number of clauses shrinks by only about 5 percent. (However, the number of clauses in the modal

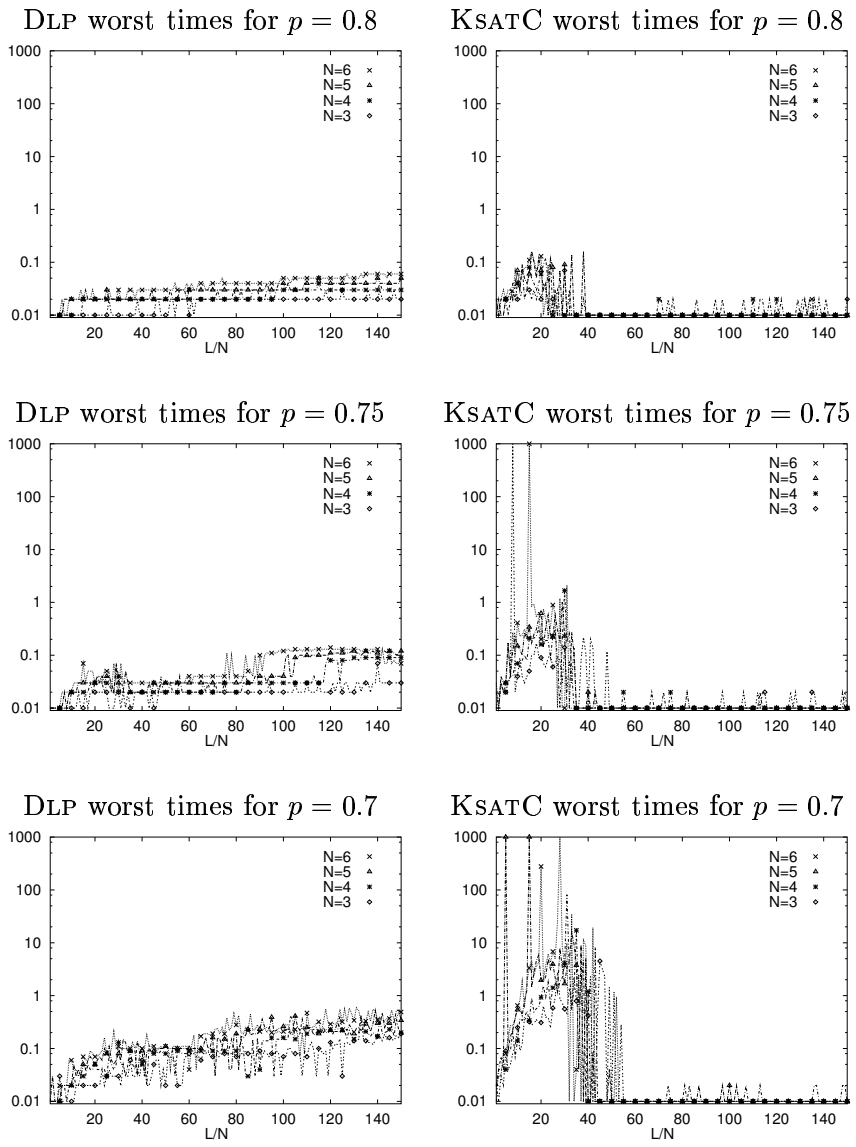

Figure 12. Worst-case for test 7 ($d = 100$; $p = 0.8$, 0.75, 0.7; $n_m = 0.5$)

successors shrinks by about 50 percent, because only half of the modal clauses are box formulae.)

The results for DLP and KSATC for propositional probabilities of 0.75 and 0.7 are given in Figures 10 and 11. It is obvious that these formulae are not nearly as difficult as some of the earlier formulae. DLP found all the formulae extremely easy—the processing required for the formulae that required the most search (for L/N from around 30 to 50)

was less than the processing required for the largest formulae, which are trivially unsatisfiable and require almost no search.

KSATC found most of the formulae easy, although there is a small harder section for L/N around 10–30, resulting in behaviour qualitatively different from that of DLP. In this section there were even a few formulae that KSATC found very hard, as shown in the top graphs of Figures 10 and 11. To better illustrate this effect, the results for the hardest formulae at each data point for propositional probabilities of 0.8, 0.75, and 0.7 are given in Figure 12. As these results show, for propositional probability 0.7 KSATC usually took between 0.5 and 10 seconds to solve the hardest formulae at data points with L/N between 15 and 50. However, in some cases it took around 100 seconds, and in a very few cases it took longer than 1,000 seconds. The results for propositional probability 0.75 are similar, except that the usual worst-case was slightly under 0.1 seconds, and only a couple of formulae took a very long time.

These harder formulae are those where KSATC is investigating many modal successors multiple times. On the other hand, DLP found all these formula very easy, with the hardest of the hardest taking only around 0.1 seconds, except for the very large formulae where DLP's processing overhead pushed the processing time up to 0.5 seconds.

If the size of the formulae are taken into consideration, these formulae are much easier than the previous ones, even though KSATC finds some of them very hard. However, they do provide another useful data point indicating that concentration on one kind of formula does not present a complete performance picture.

6. Conclusion

Our experiments with DLP and KSATC show that they are both effective decision procedures. Both systems are vastly better than previous approaches. Neither system dominates the other: DLP is better when there are complex modal subproblems, or these modal subproblems are mostly satisfiable; KSATC is better for big, simple problems, and where the modal subproblems provide early cutoffs.

Both systems still have weaknesses. DLP suffers from a mostly-functional implementation style; KSATC suffers from the divorce between its propositional and modal components. While it would be relatively easy to reimplement DLP with better data structures and low-level algorithms, we believe that the problem with the KSATC approach is more fundamental as its use of a pre-existing propositional reasoner makes it extremely hard to employ any optimisation techniques, such

as backjumping, that cut across the separate modal and propositional reasoning steps. This is because adding such optimisations to KSATC would involve significant modifications of its propositional reasoner which is not created by the authors of KSATC.

Further, the early investigation of modal successors, which is a major optimisation in KSATC, is also a source of inconsistent performance. This early investigation can be very effective if the modal successors can quickly shown to be unsatisfiable before all the propositional reasoning is performed. However, it can also result in much extra work if the modal successors are mostly satisfiable or only unsatisfiable when all information is known about them. In the experiment with maximum modal depth 100, the extra work caused by this optimisation produces a "false" peak in difficulty, which is completely absent in DLP.

Moreover, extending KSATC to deal with more expressive logics may be problematical as the loop-checking required to guarantee termination, and other processing required for the more-expressive logics, may be difficult to interface with the propositional reasoner. Tableaux systems on the other hand can easily be extended to deal with a wide range of expressive logics [1, 2, 4, 16, 30]: DLP, for example, already deals with a superset of propositional dynamic logic. However, in KSATC's favour is the fact that it can take advantage of most advances in propositional satisfiability testing simply by substituting Böhm's DPLL implementation with a more efficient SAT procedure and making the few changes required to eliminate optimisations that are not valid in a modal setting;[3] this would be more difficult in DLP where the propositional and modal components are tightly integrated.

Our experiments also show that concentrating on one set of tests is not appropriate, and can give misleading results if used as a methodology for comparing satisfiability testers. The two systems tested here behaved differently on the different sets of tests, and concentrating on one set of tests would have masked the benefits of certain optimisations and the drawbacks of others. Moreover, the increased complexity of propositional modal formulae, and the interaction between propositional and modal reasoning, gives rise to results that would have been difficult to predict without detailed analysis of both formulae and algorithms.

[3] Some propositional reasoners may, of course, be implemented in a fashion that makes the changes required for the KSATC approach difficult or impossible. However, the changes are not extensive and should be possible for most propositional reasoners. They are certainly much less involved than the modifications required to augment the control structure of a propositional reasoner to incorporate an optimisation like backjumping across modal nodes.

Our experiments generally show an easy-hard-easy distribution, as in propositional satisfiability. However, there is no hard peak evidenced in many of the experiments, and the different experiments produce qualitatively different results. The differences are partly due to the greater numbers of parameters in the modal case; with more ways of setting the parameters there are more kinds of behaviour possible. The lack of a hard peak may also be partly due to the inadequacies of the provers being tested, and there may be optimisations yet undiscovered that would restrict the hard problems to a small range of parameter values. The lack of a hard peak may also be partly due to the hardness of the problem: current provers cannot reach beyond small numbers of propositional variables, so asymptotic behaviour may not yet be evidenced.

There are other parameter settings that should be investigated. We have only reported results for modal depths 1 and 2 with a clause length of 3, and results for modal depth 100. We have mostly used a modal negation probability of 0.5. Changing any of these would result in different kinds of tests.

We are experimenting with new approaches for building modal decision procedures as part of a continuing effort to build effective systems for expressive description logics. Our newest system, in its early stages of development, employs an advanced form of backtracking, called dynamic backtracking [17]. It also performs early analysis of modal successors but does not throw away the successful successor nodes, retaining them instead for the duration of the proof. In this way we hope to further improve the performance of propositional modal satisfiability testing procedures.

References

1. Baader, F.: 1991, 'Augmenting Concept Languages by Transitive Closure of Roles: An Alternative to Terminological Cycles'. In: *Proceedings of the 12th International Joint Conference on Artificial Intelligence (IJCAI-91).* pp. 446–451.
2. Baader, F. and P. Hanschke: 1991, 'A Scheme for Integrating Concrete Domains into Concept Languages'. In: *Proceedings of the 12th International Joint Conference on Artificial Intelligence (IJCAI-91).* pp. 452–457.
3. Baader, F. and B. Hollunder: 1991, 'A Terminological Knowledge Representation System with Complete Inference Algorithms'. In: *Processing declarative knowledge: International workshop PDK'91.* Berlin, pp. 67–86.
4. Baader, F. and U. Sattler: 1996, 'Number Restrictions on Complex Roles in Description Logics'. In: L. C. Aiello, J. Doyle, and S. C. Shapiro (eds.): *Principles of Knowledge Representation and Reasoning: Proceedings of the Fifth International Conference (KR'96).* pp. 328–339.

5. Baker, A. B.: 1995, 'Intelligent Backtracking on Constraint Satisfaction Problems: Experimental and Theoretical Results'. Ph.D. thesis, University of Oregon.
6. Balsiger, P. and A. Heuerding: 1998, 'Comparison of Theorem Provers for Modal Logics — Introduction and Summary'. In: H. de Swart (ed.): *Automated Reasoning with Analytic Tableaux and Related Methods: International Conference Tableaux'98*. pp. 25–26.
7. Buchheit, M., F. M. Donini, and A. Schaerf: 1993, 'Decidable Reasoning in Terminological Knowledge Representation Systems'. *Journal of Artificial Intelligence Research* **1**, 109–138.
8. Buro, M. and H. Buning: 1992, 'Report on a SAT competition'. Technical Report 110, University of Paderborn, Germany.
9. Davis, M., G. Logemann, and D. Loveland: 1962, 'A machine program for theorem proving'. *Communications of the ACM* **5**, 394–397.
10. Donini, F., G. D. Giacomo, and F. Massacci: 1996, 'EXPTIME Tableaux for $\mathcal{ALC}$'. In: L. Padgham, E. Franconi, M. Gehrke, D. L. McGuinness, and P. F. Patel-Schneider (eds.): *Collected Papers from the International Description Logics Workshop (DL'96)*. pp. 107–110.
11. Franco, J. and M. Paull: 1983, 'Probabilistic analysis of the Davis-Putnam procedure for solving the satisfiability problem'. *Discrete Applied Mathematics* **5**, 77–87.
12. Franconi, E., G. D. Giacomo, R. M. MacGregor, W. Nutt, C. A. Welty, and F. Sebastiani (eds.): 1998, 'Collected Papers from the International Description Logics Workshop (DL'98)'.
13. Freeman, J. W.: 1995, 'Improvements to propositional satisfiability search algorithms'. Ph.D. thesis, Department of Computer and Information Science, University of Pennsylvania, Philadelphia, PA, USA.
14. Freeman, J. W.: 1996, 'Hard random 3-SAT problems and the Davis-Putnam procedure'. *Artificial Intelligence* **81**, 183–198.
15. Gent, I. P. and T. Walsh: 1999, 'Beyond NP: the QSAT phase transition'. In: *Proceedings of AAAI'99*. Orlando, Florida.
16. Giacomo, G. D. and F. Massacci: 1996, 'Tableaux and algorithms for propositional dynamic logic with converse'. in [34], pp. 613–628.
17. Ginsberg, M. L.: 1993, 'Dynamic Backtracking'. *Journal of Artificial Intelligence Research* **1**, 25–46.
18. Giunchiglia, E., F. Giunchiglia, R. Sebastiani, and A. Tacchella: 1998, 'More evaluation of decision procedures for modal logics'. In: A. G. Cohn, L. Schubert, and S. C. Shapiro (eds.): *Principles of Knowledge Representation and Reasoning: Proceedings of the Sixth International Conference (KR'98)*. pp. 626–635.
19. Giunchiglia, F. and R. Sebastiani: 1996a, 'Building decision procedures for modal logics from propositional decision procedures—the case study of modal K'. in [34], pp. 583–597.
20. Giunchiglia, F. and R. Sebastiani: 1996b, 'A SAT-based decision procedure for $\mathcal{ALC}$'. In: L. C. Aiello, J. Doyle, and S. C. Shapiro (eds.): *Principles of Knowledge Representation and Reasoning: Proceedings of the Fifth International Conference (KR'96)*. pp. 304–314.
21. Haarslev, V., R. Möller, and A.-Y. Turhan: 1998, 'Implementing an $\mathcal{ALCRP}(\mathcal{D})$ ABox Reasoner – Progress Report'. in [12], pp. 82–86.

22. Heuerding, A. and S. Schwendimann: 1996, 'A benchmark method for the propositional modal logics K, KT, and S4'. Technical report IAM-96-015, University of Bern, Switzerland.
23. Hollunder, B. and W. Nutt: 1990, 'Subsumption Algorithms for Concept Languages'. Research Report RR-90-04, Deutsches Forschungszentrum für Künstliche Intelligenz GmbH (DFKI).
24. Horrocks, I.: 1997, 'Optimising Tableaux Decision Procedures for Description Logics'. Ph.D. thesis, University of Manchester.
25. Horrocks, I.: 1998, 'Using an Expressive Description Logic: FaCT or Fiction?'. In: A. G. Cohn, L. Schubert, and S. C. Shapiro (eds.): *Principles of Knowledge Representation and Reasoning: Proceedings of the Sixth International Conference (KR'98)*. pp. 636–647.
26. Horrocks, I. and P. F. Patel-Schneider: 1998a, 'Comparing Subsumption Optimizations'. in [12], pp. 90–94.
27. Horrocks, I. and P. F. Patel-Schneider: 1998b, 'FaCT and DLP'. In: H. de Swart (ed.): *Automated Reasoning with Analytic Tableaux and Related Methods: International Conference Tableaux'98*. pp. 27–30.
28. Horrocks, I. and P. F. Patel-Schneider: 1998c, 'Optimising Propositional Modal Satisfiability for Description Logic Subsumption'. In: *International Conference AISC'98*.
29. Horrocks, I. and P. F. Patel-Schneider: 1999, 'Optimising Description Logic Subsumption'. *Journal of Logic and Computation* **9**(3), 267–293.
30. Horrocks, I. and U. Sattler: 1999, 'A Description Logic with Transitive and Inverse Roles and Role Hierarchies'. *Journal of Logic and Computation* **9**(3), 385–410.
31. Hustadt, U. and R. A. Schmidt: 1997a, 'On evaluating decision procedures for modal logic'. Technical Report MPI-I-97-2-003, Max-Planck-Institut Für Informatik, Im Stadtwald, D 66123 Saarbrücken, Germany.
32. Hustadt, U. and R. A. Schmidt: 1997b, 'On evaluating decision procedures for modal logic'. In: *Proceedings of the 15th International Joint Conference on Artificial Intelligence (IJCAI-97)*, Vol. 1. pp. 202–207.
33. Jeroslow, R. and J. Wang: 1990, 'Solving propositional satisfiability problems'. *Annals of Mathematics and Artificial Intelligence* **1**, 167–187.
34. McRobbie, M. and J. Slaney (eds.): 1996, 'Proceedings of the Thirteenth International Conference on Automated Deduction (CADE-13)', No. 1104 in Lecture Notes in Artificial Intelligence. Springer-Verlag.
35. Oppacher, F. and E. Suen: 1988, 'HARP: A Tableau-Based Theorem Prover'. *Journal of Automated Reasoning* **4**, 69–100.
36. Patel-Schneider, P. F.: 1998, 'DLP System Description'. in [12], pp. 87–89.
37. Sattler, U.: 1996, 'A concept language extended with different kinds of transitive roles'. In: G. Görz and S. Hölldobler (eds.): *20. Deutsche Jahrestagung für Künstliche Intelligenz*. pp. 333–345.
38. Selman, B., D. G. Mitchell, and H. J. Levesque: 1996, 'Generating hard satisfiability problems'. *Artificial Intelligence* **81**, 17–29.
39. Vellino, A.: 1989, 'The Complexity of Automated Reasoning'. Ph.D. thesis, University of Toronto.
40. Weidenbach, C., B. Gaede, and G. Rock: 1996, 'SPASS & FLOTTER version 0.42.'. in [34], pp. 141–145.

Journal of Automated Reasoning, **28**:2, February 2002, pages 173–204.