More evaluation of decision procedures for modal logics

Enrico Giunchiglia
DIST, V.le Causa 13
16145 Genova, Italy

Fausto Giunchiglia
DISA, Università di Trento.
IRST, 38050 Povo,
Trento, Italy

Roberto Sebastiani
IRST, 38050 Povo,
Trento, Italy

Armando Tacchella
DIST, V.le Causa 13
16145 Genova, Italy

Abstract

This paper follows on previous papers which presented and evaluated various decision procedures for modal logics. It confirms previous experimental results in showing that SAT based decision procedures, i.e., the procedures built on top of decision procedures for propositional satisfiability, are more efficient than tableau based decision procedures. It also confirms previous evidence of an easy-hard-easy pattern in the satisfiability curve for modal K. Finally, it provides further experimental results, suggesting that SAT based decision procedures are also more efficient than the decision procedures based on Ohlbach's translation method. Our results contradict some of the results presented in previous papers.

1 INTRODUCTION

Until recently, the decision procedures for modal logics which could be found in the literature were either tableau based [1] (see, e.g., [Kripke, 1959; Fitting, 1983; Massacci, 1994]), or based on Ohlbach's translation method [Ohlbach, 1991]. Only few of these procedures were implemented, and even less were thoroughly tested and comparatively evaluated. Giunchiglia and Sebastiani [Giunchiglia and Sebastiani, 1996a] changed this situation in three ways. First, they proposed a novel approach where a decision procedure for modal logics, called KSAT, is defined in terms of a SAT decision procedure. To emphasize this fact, they called the resulting class of procedures, SAT based. Second, they provided a new test methodology which naturally extends the fixed-clause-length test method, a very popular test method for SAT [Mitchell *et al.*, 1992; Buro and Buning, 1992]. Third, they built a straightforward LISP implementation of KSAT, called KSATLISP from now on, and tested it against the tableau system KRIS [Hollunder *et al.*, 1990; Baader *et al.*, 1994]. These latter results are reported in [Giunchiglia and Sebastiani, 1996c].

Following on this work, Horrocks presented FACT, a system which further enhances KSAT's ideas by introducing several other optimizations (e.g. backjumping). According to [Horrocks, 1997], the LISP implementation of FACT further enhances KSATLISP performances. Hustadt and Schmidt, instead, make in [Hustadt and Schmidt, 1997a; Hustadt and Schmidt, 1997b] a very detailed and critical analysis of the work in [Giunchiglia and Sebastiani, 1996a; Giunchiglia and Sebastiani, 1996c]. They start by improving on Giunchiglia and Sebastiani's test methodology, thus avoiding possible situations where this method can generate trivial problems. Then, they challenge the following three claims made by Giunchiglia and Sebastiani in [Giunchiglia and Sebastiani, 1996c] (as Hustadt and Schmidt quote them in [Hustadt and Schmidt, 1997a]):

1. KSAT outperforms by orders of magnitude the previous state-of-the-art decision procedures.
2. All SAT based modal decision procedures are intrinsically bound to be more efficient than tableau based decision procedures.
3. There is partial evidence of an easy-hard-easy pattern on randomly generated modal logic formulas independent of all the parameters of evaluation considered.

Finally, they propose TA, a system based on Ohlbach's translation method, which is implemented by appropriately calling FLOTTER and the first order theorem prover SPASS [Weidenbach *et al.*, 1996]. Their analysis suggests that TA has better computational behavior than KSATLISP.

[1] We call "tableau based" any system that implements Smullyan's tableau calculus as defined in [Smullyan, 1968]. Thus, in our terminology, KRIS [Hollunder *et al.*, 1990; Baader *et al.*, 1994] and LWB [Heuerding *et al.*, 1996] are tableau based systems, whence KSAT [Giunchiglia and Sebastiani, 1996a], TA [Hustadt and Schmidt, 1997a] and also FACT [Horrocks, 1997] are not.

The goal of this paper is to provide further experimental results and use them to shed some light on the previous analysis, which, as hinted above, is contradictory in some parts. Thus, in Section 2 we describe KsatC, a C++ implementation of Ksat built starting from a pre-existing state-of-the-art SAT decision procedure. The goal of this section is twofold. First, it shows how the methodology and ideas introduced in [Giunchiglia and Sebastiani, 1996a; Giunchiglia and Sebastiani, 1996c] allows us, in practice, to develop decision procedures for modal logics starting from existing implementations of SAT decision procedures. Given the fact that this latter domain has received much more attention and it is far more developed, we consider this an important result. To build an efficient SAT procedure from scratch requires a lot of man power. Second, a comparison of KsatLisp and KsatC allows us to get an idea of how much we can gain by working on the implementation. [2] Then, in Section 3 we describe a first set of tests which allows us to confirm the three claims of ours listed above and challenged by Hustadt and Schmidt. These tests also suggest that KsatC and (contrarily to what reported in [Hustadt and Schmidt, 1997a; Hustadt and Schmidt, 1997b]) even KsatLisp are faster than TA. Finally, in Section 4 we focus on KsatC and TA. These tests confirm and make more evident the superior computational properties of KsatC over TA.

2 KsatC

We use the same conventions as [Giunchiglia and Sebastiani, 1996c; Giunchiglia and Sebastiani, 1996b]. A *modal atom* is a wff of the form $\Box\psi$, and an *atom* is either a modal atom or a propositional letter. An *assignment* is a partial function from the set of atoms to $\{True, False\}$. In this paper, all the assignments have a finite domain. This allows us to identify an assignment μ as the conjunction of the atoms true in μ and of the negations of the atoms false in μ. The idea is to see a modal wff as a propositional wff in its top-level atoms, so that μ corresponds to a standard assignment in propositional logic.

Ksat is a correct and complete decision procedure for satisfiability in the modal logic K (K-satisfiability from now on) presented and analyzed in detail in [Giunchiglia and Sebastiani, 1996a; Giunchiglia and Sebastiani, 1996c]. A high-level description of Ksat is reported in Figure 2. For any modal wff φ, $\text{Ksat}(\varphi)$ returns *True* if φ is K-satisfiable and *False* otherwise. This is accomplished by calling $\text{Ksat}_W(\varphi, \emptyset)$. Ksat_W is a variant of the Davis-Putnam-Longemann-Loveland SAT procedure [Davis and Putnam, 1960; Davis *et al.*, 1962] (DPLL from now on). The fundamental difference with DPLL is that, whenever an assignment μ satisfying φ has been found ("base" case), Ksat_W invokes $\text{Ksat}_A(\mu)$ instead of returning $True$. Basically, Ksat_W is used to generate a complete set of assignments for φ.[3] Then, the K-consistency of each assignment generated by Ksat_W is checked by Ksat_A, and this is done by recursively calling Ksat. These procedures recurse until we get to an assignment with no modal atoms.

KsatC is a C++ implementation of Ksat. KsatC is implemented on top of Böhm's C implementation of DPLL (called Dpll from now on), the winner of a 1992 SAT competition [Buro and Buning, 1992].[4] Dpll has served as the basis for the implementation of Ksat_W. The basic features of Ksat_W inherited from Dpll are:

- efficient data structures for literal assignment and wff (partial) evaluation. In Dpll (and hence in Ksat_W) literals are assigned and unassigned dynamically inside a wff simply by moving pointers (that is, without copying the wff in the stack at each recursive call (!)) in time proportional to the number of their occurrences.
- smart splitting heuristics. Similarly to other state-of-the-art satisfiability checkers (see, e.g., [Crawford and Auton, 1993]) Dpll splits on the literal occurring most often in the shortest clauses.[5]

Ksat_W has been obtained from Dpll by performing the following steps:

- The code implementing the pure literal rule has been ruled out.[6] With the pure literal rule, the set of assignments generated by Ksat_W might be incomplete, so that KsatC might conclude that the input wff is not satisfiable even if this is actually the case;
- Whenever an assignment μ satisfying φ is found, Ksat_W invokes $\text{Ksat}_A(\mu)$ (instead of printing out the assignment);

[2] To this extent, it must be pointed out that the implementation of KsatC is very similar in spirit to that of KsatLisp. (See Section 2 for the details.) We believe that the behavior of KsatC could be further improved by adding new heuristics, e.g., those implemented in Fact [Horrocks, 1997].

[3] We say that a set S of assignments is *complete* for a wff ϕ, if ϕ is logically equivalent to the disjunction of the assignments in S.

[4] The code of Dpll and a report about this competition are available at the http address `http://www.informatik.uni-koeln.de/ls_juenger/staff/boehm.html`.

[5] By contrast, in KsatLisp a copy of the wff is passed at each DPLL recursive call; literals are assigned by recursively descending the whole wff; splitting happens on atoms (and not on literals), and KsatLisp chooses the atom occurring most often.

[6] According to the pure literal rule, if a literal l occurs in φ while $\neg l$ does not, l is assigned to *True* [Davis and Putnam, 1960].

```
function  KSAT(φ)
     return  KSAT_W(φ, T);


function KSAT_W(φ, μ)
     if φ = T                                             /* base           */
          then return KSAT_A(μ);
     if φ = F                                             /* backtrack      */
          then return False;
     if {a unit clause (l) occurs in φ}                   /* unit           */
          then return KSAT_W(assign(l, φ), μ ∧ l);
     if not KSAT_A(μ)                                     /* early pruning (optional) */
          then return False;
     l := choose-literal(φ);                              /* split          */
     return  KSAT_W(assign(l, φ), μ ∧ l)   or
             KSAT_W(assign(¬l, φ), μ ∧ ¬l);


function  KSAT_A(⋀_i □α_i ∧ ⋀_j ¬□β_j ∧ γ)
     for any conjunct "¬□β_j" do
          if not KSAT(⋀_i α_i ∧ ¬β_j)
               then return False;
     return True;
```

Figure 1: The structure of KSAT.

- Many of the global variables have been made local to the procedures where they are used: this is necessary to avoid interferences between two calls $\text{KSAT}_W(\varphi, \emptyset)$ and $\text{KSAT}_W(\varphi', \emptyset)$. Analogously for the static variables.

- The (optional) step of early pruning has been included in DPLL (see Figure 2). This step is not mandatory. However, as discussed in [Giunchiglia and Sebastiani, 1996a], early pruning may detect early inconsistencies in the partial assignment so far built and therefore greatly prune the search space.

KSAT_W is interfaced with the rest of the procedure by means of a global look-up table (LUT) which associates a propositional literal B_i (resp. $\neg B_i$) to each distinct modal literal $\Box\varphi_i$ (resp. $\neg\Box\varphi_i$) occurring in φ. Each row in LUT is a pair $\langle B_i, \varphi_i^* \rangle$ (resp. $\langle \neg B_i, \neg\varphi_i^* \rangle$), where φ_i^* is the propositional wff obtained from φ_i by recursively substituting each atom $\Box\varphi_j$ with the corresponding B_j. When invoked on the input wff φ, KSATC initializes LUT and passes φ^* to KSAT. Each time it is invoked on a propositional assignment μ (which represents $\bigwedge_i \Box\alpha_i \wedge \bigwedge_j \neg\Box\beta_j \wedge \gamma$), KSAT_A retrieves from LUT the CNF wffs corresponding to the literals in μ, it merges them and invokes KSAT on the resulting CNF wffs $\wedge_i \alpha_i \wedge \neg\beta_j$'s. Notice that each retrieval requires only constant time.

3 MORE TESTING OF THREE EARLIER CLAIMS

Our goal in this section is to confirm the three claims listed in the introduction, first made in [Giunchiglia and Sebastiani, 1996c] and then challenged by Hustadt and Schmidt.

3.1 THE EVALUATION STRATEGY

We compare five systems, namely: KRIS, KSATLISP, KSATC, KSATLISP(UNSORTED), and TA.[7] KRIS is a tableau based system implemented in LISP [Hollunder *et al.*, 1990; Baader *et al.*, 1994]. Both KSATC and KSATLISPpre-sort input wffs before the main routine is invoked for the first time. This allows for renaming the permutations of a modal atom (e.g., $\Box(A_1 \vee A_2)$ and

[7] TA is available at `http://www.mpi-sb.mpg.de/` in `~hustadt/mdp/`. In particular, we used TA version 1.2, SPASS/FLOTTER version 0.55. KRIS, KSATLISP, KSATC, the random wff generator and detailed instructions about how to reproduce all the tests presented are available at `ftp://ftp.mrg.dist.unige.it/` in `pub/mrg-systems`. (As the following makes clear, KSATLISP(UNSORTED) is KSATLISP with a heuristic turned off).

$\Box(A_2 \vee A_1)$) with the same propositional letter. This step, if not performed, may dramatically affect the performance of the algorithm [Giunchiglia and Sebastiani, 1996a]. KSATLISP(UNSORTED) is KSATLISP with sorting disabled. We test KSATLISP(UNSORTED)(which, we know a priori, is much slower than KSATLISP) in order to reproduce the tests in [Hustadt and Schmidt, 1997a; Hustadt and Schmidt, 1997b]. In fact, as they explicitly state in [Hustadt and Schmidt, 1997a][page 21], Hustadt and Schmidt test KSATLISP with sorting disabled (that is, what in this paper we call KSATLISP-(UNSORTED)).

To test these systems, we use the test methodology described in [Giunchiglia and Sebastiani, 1996a; Giunchiglia and Sebastiani, 1996c] with the improvements suggested in [Hustadt and Schmidt, 1997a].[8] These improvements solve the problem — highlighted in [Hustadt and Schmidt, 1997a] — that, for some choices of the parameters' values, the generated wffs can be trivial or can be significantly simplified by a simple preprocessing step (like that, for instance, which can be optionally activated in both KSATC and KSATLISP). As a matter of fact, the samples that we used throughout our experimental analisys are highly insensitive to preprocessing (i.e. simplifying). We verified this property by running the built-in simplifier of KSATC on every sample of the PS1 test, and then comparing the original sample vs. the simplified one. The result is that there is little or no difference between them.

We can briefly overview our testing methodology as follows. We consider problem sets, i.e., sets of randomly generated 3CNF_K wffs. A 3CNF_K wff is a conjunction of 3CNF_K clauses, where each clause is a disjunctions of three literals. Modal atoms are restricted to have the form $\Box C$, where C is a 3CNF_K clause. A 3CNF_K wff is randomly generated according to the following parameters: (i) the modal depth d; (ii) the number of clauses L; (iii) the number of propositional variables N; (iv) the probability p with which an atom occurring in a clause of depth > 0 is purely propositional. Following [Hustadt and Schmidt, 1997a], we fix $p = 0$ and modify the generator to avoid multiple occurrences of the same propositional atom inside one clause.

A problem set is thus characterized by a fixed N and d: we let L vary in such a way to empirically cover the "100% satisfiable – 100% unsatisfiable" transition. Then, for each tuple of the four values in a problem set, we randomly generate 100 3CNF_K wffs. These wffs are given in input to the procedure under test. Satisfiability percentages and median CPU times are plotted against the number of clauses L. For practical reasons, a timeout mechanism stops the execution on one problem after 1000 seconds of CPU. A 1000 seconds median time thus means that more than 50% of the total number of samples has exceeded the timeout.

[8]The features and advantages of this methodology are described in [Giunchiglia and Sebastiani, 1996a].

3.2 RESULTS

We start with the problem set PS1 (called PS12 in [Hustadt and Schmidt, 1997a]) characterized by $d = 1$ and $N = 4$. Figure 2 reports the median CPU time for the five systems (left) and the median number of DPLL calls (right) for KSATLISP and KSATC. For KSATLISP and KSATC, the CPU time does not include the time needed for wff sorting. However, this overhead is negligible when compared to the time spent in the decision process. For TA, we consider only the time that SPASS takes to solve the problem. We do not take into account the time to convert modal formulas into first order formulas and the time used by FLOTTER to perform the conversion to conjunctive normal form. We also plot the percentage of satisfiable wffs to get a rough estimation of where the cross-over point of 50% of satisfiable wffs occurs.

Consider first Figure 2 left.[9] Notice the logarithmic scale on the vertical axis. As a first remark, observe the dramatic effect that disabling the sorting of the input formula has on the computational behavior of SAT-based procedures: the gap between KSATLISP-(UNSORTED) and KSATLISP grows up to more than 2 orders of magnitude (e.g., for $L = 80$). The performance gap between KSATLISP and TA is more than 1 order of magnitude for $L \geq 60$. The orders of magnitude become 2 if we consider KSATC instead of KSATLISP. Running TA on highly constrained formulas ($L = 140, 180, 200$), we observed that the median CPU time on 100 samples is always higher than 40 seconds, so we have still no evidence that TA runtime decreases when formulas become trivially unsatisfiable. If we consider KRIS, the gap with KSATLISP and KSATC reaches about 4 and 5 orders of magnitude respectively for $L = 28$. For $L = 36$ no wff is solved by KRIS within the timeout. [10] These results support Claim 1 that KSAT outperforms the other current state-of-the-art decision procedures. With respect to [Giunchiglia and Sebastiani, 1996b] we have further evidence coming from the comparison with TA. These results also show that the arguments against our empirical analysis in [Giunchiglia and Sebastiani, 1996c]

[9]The tests in Figure 2 have been performed on a `Pentium 200MHz MMX, 64MBRam` workstation on `Linux Red-Hat 2.0.30`. SPASS and KSATC are compiled by `gcc 2.7.2.1`, option `-O2`. KRIS and KSATLISP/KSATLISP-(UNSORTED) are compiled by `allegro cl 4.3` and `gcl 2.2` respectively.

[10]For each value of L the test is stopped if more that 50% samples exceed the timeout (i.e., the median value is greater than 1000 seconds). This saves a lot of testing time. Therefore, "no wff is solved within the timeout" means that the first 50%+1 samples all exceeded the timeout, while the others have not been tested.

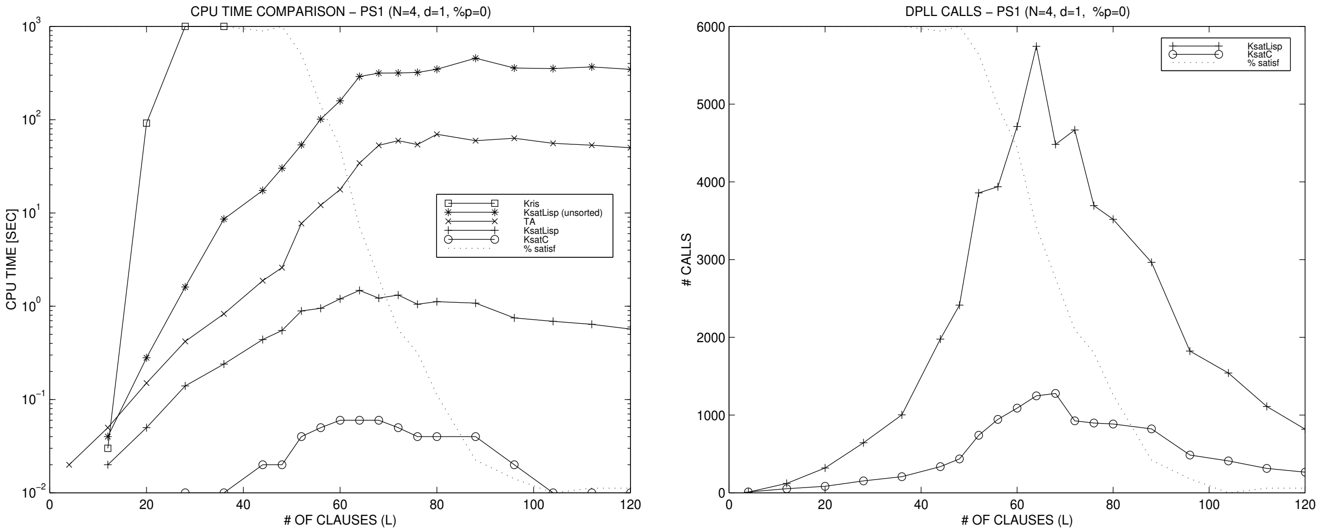

Figure 2: Left: KRIS, TA, KSATLISP(UNSORTED) KSATLISP and KSATC median CPU time, 100 samples/point. Right: KSATLISP and KSATC median search space size, 100 samples/point. Background: satisfiability percentage.

given in [Hustadt and Schmidt, 1997a; Hustadt and Schmidt, 1997b] are wrong. Even fixing the testing methodology, there is still evidence supporting Claim 1. The different results obtained in [Hustadt and Schmidt, 1997a; Hustadt and Schmidt, 1997b] are simply due to the fact that Hustadt and Schmidt tested KSATLISP(UNSORTED) and not KSATLISP, clearly underestimating the effects of this choice.

Consider now Claim 2 that SAT based decision procedures are intrinsically bound to be more efficient than tableau based decision procedures. In [Giunchiglia and Sebastiani, 1996c], (and, more in detail, in [Giunchiglia and Sebastiani, 1996b]) this strong claim was supported by some empirical results and also by a theoretical analysis. We have discussed the empirical results in the above paragraph. For what concerns the theoretical analysis given in [Giunchiglia and Sebastiani, 1996c; Giunchiglia and Sebastiani, 1996b], we still support it. The argument goes as follows. We start from the observation that SAT deciders are intrinsically superior to propositional tableaux, for

(a) they don't generate redundant assignments, and
(b) they prune branches as soon as they violate one propositional constraint.

Then, it is sufficient to notice that, as [Giunchiglia and Sebastiani, 1996c; Giunchiglia and Sebastiani, 1996b] show, this performance gap propagates and expands with the modal depth of the formulas.

Finally, consider Claim 3 about the existence of easy-hard-easy patterns. From a theoretical point of view, it is shown in [Giunchiglia and Sebastiani, 1996b] that these patterns are a consequence of property (b) above. From an empirical point of view, consider Figure 2, right. This figure reports the size of the search space for both KSATLISP and KSATC. It is easy to notice the existence of easy-hard-easy peaks centered about the 50% satisfiable point. This issue will be further discussed in Section 4.

4 KsatC VS. TA

Consider now the three problem sets PS2 ($d = 1, N = 4$), PS3 ($d = 1, N = 5$) and PS4 ($d = 1, N = 6$). PS2 and PS4 reproduce the two experiments presented in [Hustadt and Schmidt, 1997a], with a slight change in the testing methodology. In particular, the improved test methodology presented in [Hustadt and Schmidt, 1997a] and described in Section 3 is not yet optimal, as it may be the case that multiple occurrences of the same modal atom $\Box C$ (or different permutations of $\Box C$) occur in clauses of depth > 0. For instance, the modal clause $\Box(A_1 \vee A_2 \vee A_3) \vee \neg\Box(A_1 \vee A_3 \vee A_2)$ becomes a tautology after sorting. To avoid this, we have introduced a further slight improvement in the wff generator: pre-sort all modal atoms and avoid multiple occurrences of (sorted) modal atoms inside clauses. It is clear that there is no point in trying to simplify these samples.

The results of PS2, PS3, PS4 are described in Figure 3.[11] In all figures, the horizontal axis represents the number of clauses L, and each point represent the median value out of 100 values. The three columns cor-

[11] All the tests of Figure 3 have been performed with the same machine, operating system and compilers as those in Figure 2.

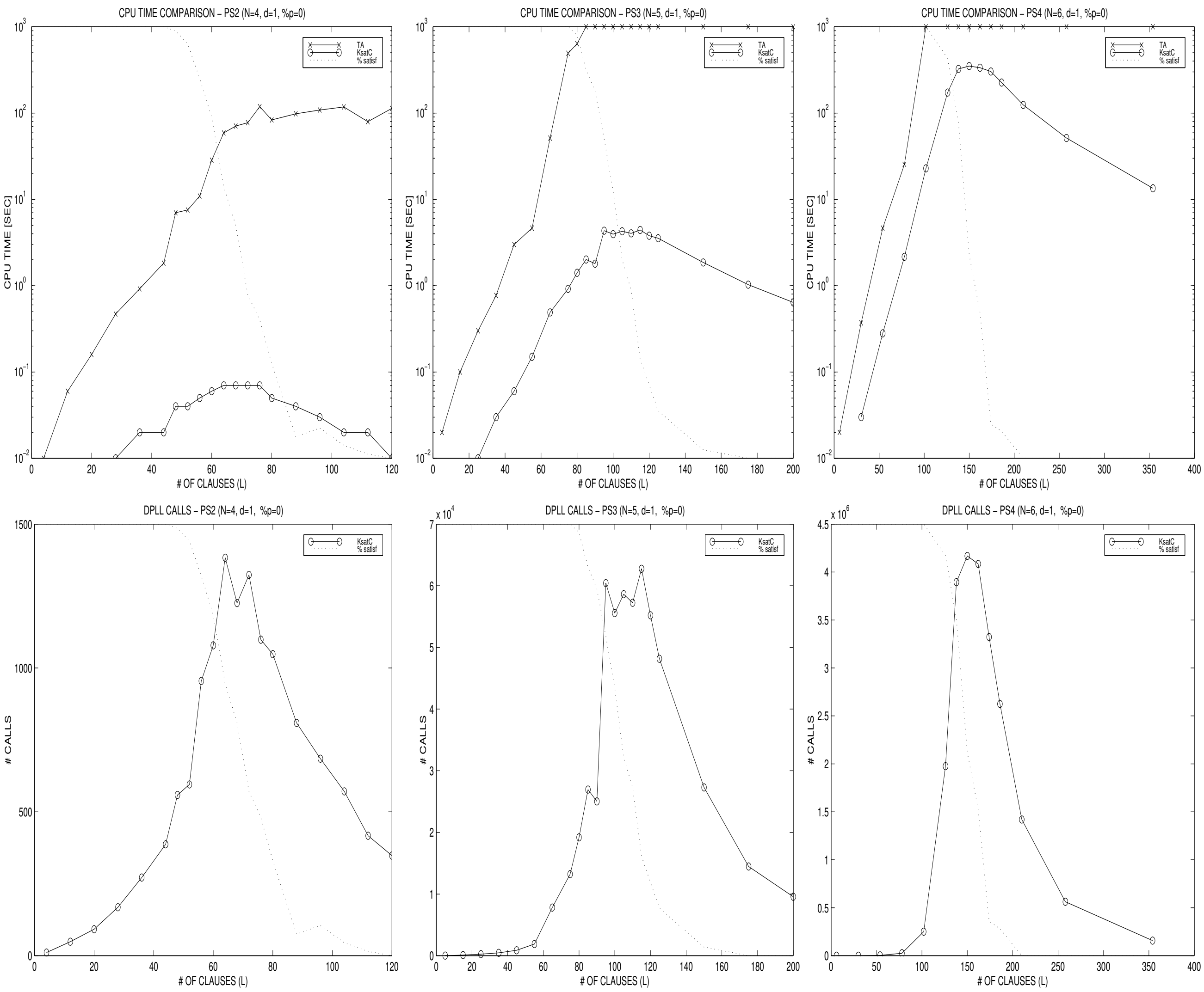

Figure 3: First row: TA versus KsatC on PS2, PS3, PS4. Second row: KsatC size of search space. Background: satisfiability percentages (100% satisfiable at end-scale).

respond to PS2, PS3 and PS4 respectively. In the first row we compare TA and KsatC CPU times, in logarithmic scale. In the second row we plot the KsatC global number of recursive DPLL calls, i.e., the size of the space effectively searched by KsatC, in linear scale. The dotted plots in the background represent the percentages of satisfiable formulas, in linear scale, with 100% satisfiable at end scale.

An eye-catching observation to the first row shows that KsatC performs better than TA in all the three tests. For PS2, the gap between TA and KsatC is of 2 (almost 3) orders of magnitude for $L = 68$ (i.e. at the cross-over point of 50% of satisfiable wffs) and goes up till 4 orders of magnitude at the right end side of the horizontal axis. For PS3 and PS4, TA median values exceed the timeout for $L = 85$ and 102 respectively, while the corresponding values of KsatC are 2.0 and 22.9 seconds. TA keeps exceeding the timeout for all the successive values. After $L = 125$ and 162 respectively, no wff is solved by TA within the timeout.

It is very important to notice the *qualitative* difference between the KsatC and TA plots: when the tests enter the "unsatisfiable" area, the KsatC curves *decrease* with L. In fact the size of the space searched (bottom row) tends to zero whenever L approaches the "100% unsatisfiable" area. [Giunchiglia and Sebastiani, 1996b] showed that this feature is a consequence of the ability of a procedure to detect constraint violations as soon as they occur: the more constrained the formula is, the more likely a branch violates a constraint, the higher the search tree is pruned. Roughly speaking, this matches the general intuition that overconstrainedness makes unsatisfiability more evident, and thus easier to detect

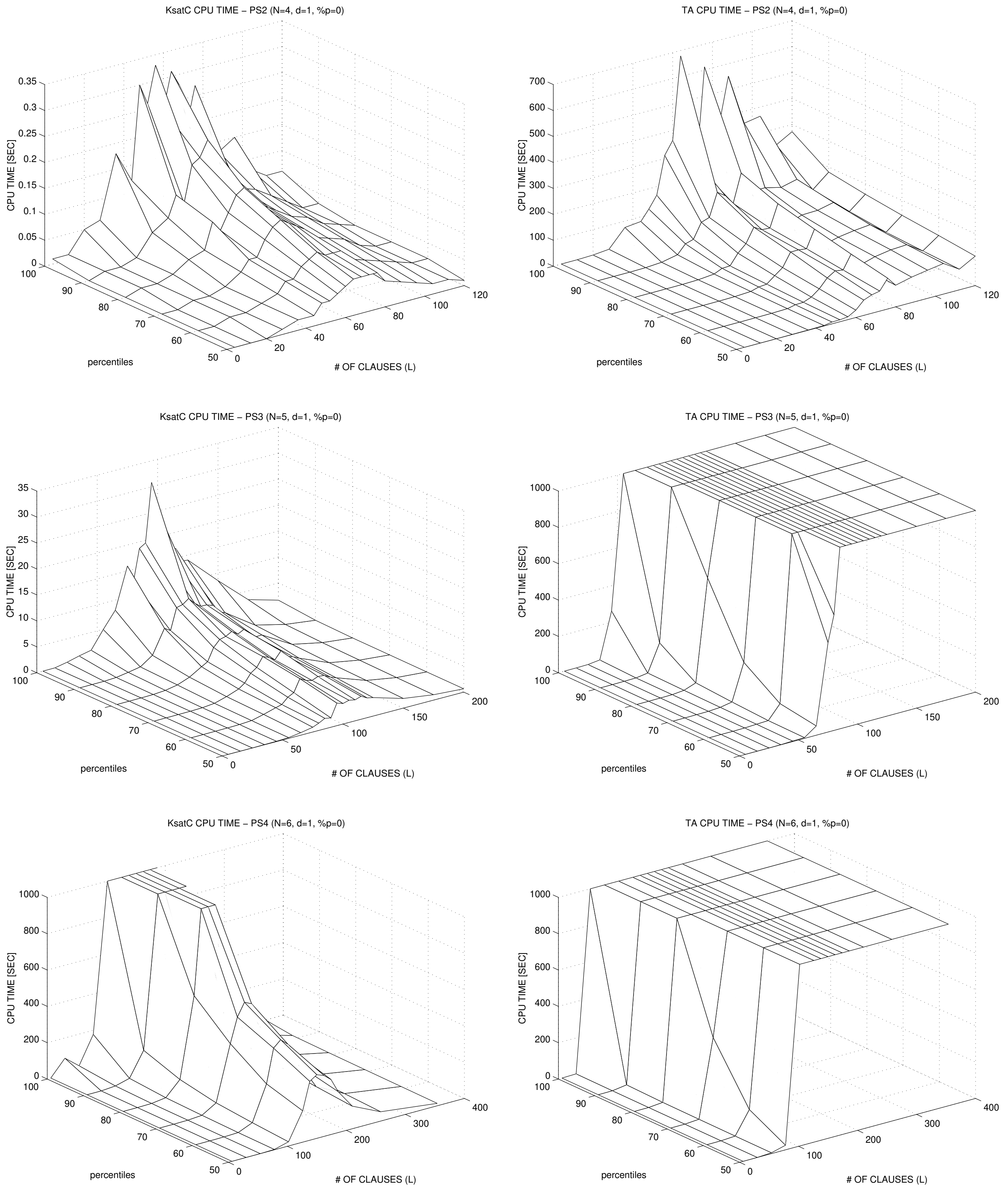

Figure 4: The percentile graphs for KsatC (left column) and TA (right column) on PS2 (top row), PS3 (middle row) and PS4 (bottom row).

(see [Williams and Hogg, 1994] for a fine-grained analysis of this point.) Similarly to tableau based systems [Giunchiglia and Sebastiani, 1996b], TA does not seem capable to take full advantage of over-constrainedness of the input wffs. For instance, in the extreme right points of some plots, TA is not able to solve one single instance within the timeout, while these instances are easy or even trivial to solve for KSATC. To support this consideration we generated a sample $L = 200, N = 5, d = 1$ and tested TA without time limits. TA came out with a response after 9534 seconds (more than two hours and a half) while KSATC took 0.88 seconds of CPU.

It is also worth noticing that the plots support the evidence of a phase transition phenomenon – i.e., steep satisfiability transition plots and easy-hard-easy hardness patterns whose peaks are located about the 50% satisfiability point – for K-satisfiability. This confirms the analogous results in [Giunchiglia and Sebastiani, 1996c; Giunchiglia and Sebastiani, 1996b].

All the above considerations are confirmed by the Q%-percentile graphs in Figure 4. Formally, the Q%-*percentile* of a set S of values is the value V such that Q% of the values in S are smaller or equal to V. The median value of a set thus corresponds to the 50% percentile of the set. Figure 4 reports the 50%, 60%, 70%, 80%, 90% and 100% percentile curves for KSATC (left column) and TA (right column) on PS2 (top row), PS3 (middle row) and PS4 (bottom row). All the Q%-percentile plotted curves show that KSATC performs better than TA (notice the different end of scale used on the vertical axis for KSATC and TA), except for those points in PS4 on which both KSATC and TA exceed the time limit. However, on the basis of the respective values of the surrounding points, we may conjecture that even for these points KSATC values are less than the respective TA values. Interestingly, for PS2, there does not seem to be much difference between the 50%-percentile and the 90%-percentile curves for TA. On PS2, TA 100%-percentile curve seems to present an easy-hard-easy pattern, but this is the only curve where this seems to happen.

To extend our comparison, we have further tested TA and KSATC on the class of wffs $\{\varphi_d^K\}_{d=1,2,\ldots}$ presented in [Halpern and Moses, 1992]. These are K-satisfiable wffs, with depth d and $2d+1$ propositional letters. This test is interesting as every Kripke structure satisfying φ_d^K has at least $2^{d+1}-1$ distinct states, while $|\varphi_d^K|$ is $O(d^2)$. From the results in [Halpern and Moses, 1992] we can reasonably assume a minimum exponential growth factor of 2^d in time for any ordinary algorithm based on Kripke semantics. We have run TA and KSATC on these formulas, for increasing values of d.

Figure 5 presents the resulting CPU times. [12] TA

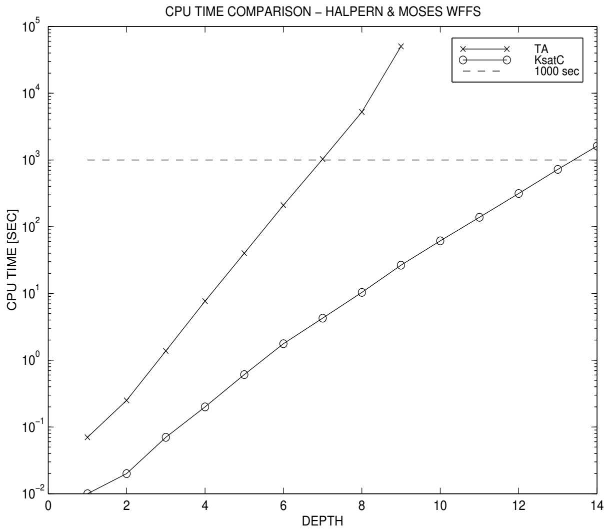

Figure 5: TA and KSATC CPU times for the class of φ_d^K formulas.

and KSATC CPU times grow approximately as $(5.0)^d$ and $(2.3)^d$, respectively, exceeding the 1000 seconds limit for $d = 7$ (1032.35 seconds) and $d = 14$ (1616.87 seconds), respectively. This confirms the performance gap highlighted above. On the same test, on the same machine, KRIS and KSATLISP grow approximatively as $(12.7)^d$ and $(2.6)^d$, respectively. They exceed the 1000 seconds limit for $d = 7$ (5085 seconds) and $d = 11$ (2541.28 seconds) respectively, as reported in [Giunchiglia and Sebastiani, 1996c].

5 CONCLUSION

Our analysis in this paper suggests that SAT based decision procedures have superior behavior than decision procedures based on the translation method. In previous papers we have presented results, confirmed by the results in this paper, which suggest that SAT based decision procedures are also more efficient than tableau based decision procedures. So far we have restricted ourselves to few basic modal logics (e.g., K, S5). An interesting open problem is whether these (substantial) computational advantages will also exist in other modal logics which are more "interesting" from an applicational point of view, e.g., CTL, LTL, Dynamic Logics.

ACKNOWLEDGEMENTS

The work described in this paper has benefitted a lot from the interactions with Ullrich Hustadt and Renate Schmidt. The results published in their papers

[12] The test in Figure 5 have been performed on a `SUN SPARC10, 32MBRam` workstation on SunOS 4.1.4. SPASS and KSATC are compiled by `gcc 2.7.2`, option `-O2`.

[Hustadt and Schmidt, 1997a; Hustadt and Schmidt, 1997b] challenged us and forced us to look deeper into the previous analisys and results. Furthermore, the help of Hustadt has been crucial in the generation of some of the results described in this paper.

This work has also benefitted of many useful discussions with Ian Horrocks, who provided very useful feedback.

The MRG group at Genova put up with many weeks of time-consuming tests.

Finally, thanks to the anonymous referees whose criticisms helped us during the writing of the revised version.

References

[Baader *et al.*, 1994] F. Baader, E. Franconi, B. Hollunder, B. Nebel, and H.J. Profitlich. An Empirical Analysis of Optimization Techniques for Terminological Representation Systems or: Making KRIS get a move on. *Applied Artificial Intelligence. Special Issue on Knowledge Base Management*, 4:109–132, 1994.

[Buro and Buning, 1992] M. Buro and H. Buning. Report on a SAT competition. Technical Report 110, University of Paderborn, Germany, November 1992.

[Crawford and Auton, 1993] J. Crawford and L. Auton. Experimental results on the crossover point in satisfiability problems. In *Proc. of the 11th National Conference on Artificial Intelligence*, pages 21–27, 1993.

[Davis and Putnam, 1960] M. Davis and H. Putnam. A computing procedure for quantification theory. *Journal of the ACM*, 7:201–215, 1960.

[Davis *et al.*, 1962] M. Davis, G. Longemann, and D. Loveland. A machine program for theorem proving. *Journal of the ACM*, 5(7), 1962.

[Fitting, 1983] M. Fitting. *Proof Methods for Modal and Intuitionistic Logics.* D. Reidel Publishg, 1983.

[Giunchiglia and Sebastiani, 1996a] F. Giunchiglia and R. Sebastiani. Building decision procedures for modal logics from propositional decision procedures - the case study of modal K. In *Proc. of the 13th Conference on Automated Deduction*, Lecture Notes in Artificial Intelligence, New Brunswick, NJ, USA, August 1996. Springer Verlag. Also DIST-Technical Report 96-0037 and IRST-Technical Report 9601-02.

[Giunchiglia and Sebastiani, 1996b] F. Giunchiglia and R. Sebastiani. Building decision procedures for modal logics from propositional decision procedures - the case study of modal K(m). Technical Report 9611-06, IRST, Trento, Italy, 1996. A shorter version will appear on *Information and Computation.*

[Giunchiglia and Sebastiani, 1996c] F. Giunchiglia and R. Sebastiani. A SAT-based decision procedure for ALC. In *Proc. of the 5th International Conference on Principles of Knowledge Representation and Reasoning - KR'96*, Cambridge, MA, USA, November 1996. Also DIST-Technical Report 9607-08 and IRST-Technical Report 9601-02.

[Halpern and Moses, 1992] J.Y. Halpern and Y. Moses. A guide to the completeness and complexity for modal logics of knowledge and belief. *Artificial Intelligence*, 54(3):319–379, 1992.

[Heuerding *et al.*, 1996] A. Heuerding, G. Jager, S. Schwendimann, and M. Seyfried. The logics workbench LWB: A snapshot. *Euromath Bulletin*, 2(1):177–186, 1996.

[Hollunder *et al.*, 1990] B. Hollunder, W. Nutt, and M. Schmidt-Schauß. Subsumption Algorithms for Concept Description Languages. In *Proc. 8th European Conference on Artificial Intelligence*, pages 348–353, 1990.

[Horrocks, 1997] Ian R. Horrocks. *Optimising Tableaux Procedures for Description Logics.* PhD thesis, Department of Computer Science, University of Manchester, 1997.

[Hustadt and Schmidt, 1997a] U. Hustadt and R.A. Schmidt. On evaluating decision procedures for modal logic. Research report MPI-I-97-2-003, Max-Planck-Institut für Informatik, Saarbrücken, Germany, February 1997.

[Hustadt and Schmidt, 1997b] U. Hustadt and R.A. Schmidt. On evaluating decision procedures for modal logic. In *Proc. of the 15th International Joint Conference on Artificial Intelligence*, 1997.

[Kripke, 1959] S. Kripke. *A completeness theorem in modal logic*, volume 24. 1959.

[Massacci, 1994] F. Massacci. Strongly analytic tableaux for normal modal logics. In *Proc. of the 12th Conference on Automated Deduction*, 1994.

[Mitchell *et al.*, 1992] D. Mitchell, B. Selman, and H. Levesque. Hard and Easy Distributions of SAT Problems. In *Proc. of the 10th National Conference on Artificial Intelligence*, pages 459–465, 1992.

[Ohlbach, 1991] H. J. Ohlbach. Semantics based translation methods for modal logics. *Journal of Logic and Computation*, 1(5):691–746, 1991.

[Smullyan, 1968] R. M. Smullyan. *First-Order Logic.* Springer-Verlag, NY, 1968.

[Weidenbach *et al.*, 1996] C. Weidenbach, B. Gaede, and G. Rock. SPASS & FLOTTER version 0.42. In M.A. McRobbie and J.K. Slaney, editors, *Proceedings of the 13th Conference on Automated Deduction (CADE-13)*, volume 1104 of *Lecture Notes in Artificial Intelligence*, pages 141–145, New Brunswick, New Jersey, USA, July/August 1996. Springer.

[Williams and Hogg, 1994]
C. P. Williams and T. Hogg. Exploiting the deep structure of constraint problems. *Artificial Intelligence*, 70:73–117, 1994.