SAT vs. Translation Based decision procedures for modal logics: a comparative evaluation

Enrico Giunchiglia
DIST, V.le Causa 13
16145 Genova, Italy
enrico@mrg.dist.unige.it

Fausto Giunchiglia
DISA, Università di Trento.
ITC-IRST, 38050 Povo,
Trento, Italy
fausto@irst.itc.it

Roberto Sebastiani
DISA - Universita‘ di Trento,
38100 Trento, Italy
rseba@cs.unitn.it

Armando Tacchella
DIST, V.le Causa 13
16145 Genova, Italy
tac@mrg.dist.unige.it

ABSTRACT. This paper follows on previous papers which present and evaluate various decision procedures for modal logics. We consider new test sets and systems that have been recently proposed in the literature. This new experimental analysis confirm 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. They also confirm previous evidence of an easy-hard-easy pattern in the satisfiability curve for modal K. Finally, on these tests, SAT based decision procedures are also more efficient than the recently proposed decision procedures based on translation methods. These results contradict some of the claims presented in previous papers by other authors.

1 Introduction

Until recently, the decision procedures for modal logics which could be found in the literature were either tableau based (see, e.g., [Fit83, Mas94]), or based on a reduction to first order logic (see, e.g., [vB84, Ohl91]). Only few of these procedures were implemented, and even less were thoroughly tested and comparatively evaluated. Giunchiglia and Sebastiani [GS96a] 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 "SAT based" the resulting class of procedures. Second, they provided a new test methodology which naturally extends the fixed-clause-length test method, a very popular test method for SAT [MSL92, BB92]. Third, they built a straightforward LISP implementation of KSAT, called KSATLISP from now on, and tested it against the tableau system KRIS [BH91, BFH+94]. The latter results are reported in [GS96c]. It worths mentioning that, since [GS96a], the idea of building decision procedures for modal and description logics on top of sophisticated propositional procedures has been adopted and extended also by others (see, e.g., [Hor98a, Hor98b, PS98]).

Later on, Hustadt and Schmidt [HS98, HS97a] pointed out some problems with the tests presented in [GS96a, GS96c]. In these papers, the authors (*i*) present their own translation based procedure TA and implement it; (*ii*) show that some of the tests used by Giunchiglia and Sebastiani privileged KSATLISP more than the other systems; (*iii*) propose a way to correct the test methodology; and (*iv*) perform some experimental analysis. The results of the tests, lead the authors to claim the computational superiority of TA w.r.t. other systems, including KSATLISP.

In this paper, following on Hustadt and Schmidt's suggestions about Giunchiglia and Sebastiani's tests, we further test various systems, including KSATLISP and TA. Our results contradict the conclusions in [HS97a, HS98], at the same time confirming the results in [GS96a, GS96c]. More precisely, on these tests, KSATLISP has better computational properties than TA.

The paper is structured as follows. In Section 2 we introduce some basic notions and notation that is used throughout the whole paper. In Section 3 we present the most important approaches for what concerns decision procedures for modal logics, while in Section 4 we critically discuss some of the previous work. In Section 5 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 [GS96a, GS96c] 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. Second, a comparison of KSATLISP and KSATC allows us to get an idea of how much we can gain by working on the underlying SAT-procedure.Then, in Section 6 we consider the tests suggested by Hustadt and Schmidt [HS97a, HS98]. On these tests KSATC and (contrarily to what reported in [HS98, HS97a]) even KSATLISP are faster than TA. Finally, in Section 7 we focus on KSATC and TA. Also on these final tests KSATC has better computational properties than TA.

This paper builds on and extends [GGST98].

2 Logic preliminaries and notation

We restrict ourselves to the modal logic K. The set of *modal formulas* is constructed through finitely many applications of the unary operators $\neg$, $\Box$ and the binary operators $\wedge$, $\vee$ to the logical symbols T, F (representing truth and falsity respectively) and a given set of propositional variables. We also write "$\varphi_1 \supset \varphi_2$" and "$\varphi_1 \equiv \varphi_2$" as abbreviations for "$\neg\varphi_1 \vee \varphi_2$" and "$(\neg\varphi_1 \vee \varphi_2) \wedge (\varphi_1 \vee \neg\varphi_2)$" respectively. In the following, we use $A_1, A_2, \ldots$ for propositional variables, and $\varphi, \varphi_1, \varphi_2, \ldots$ for arbitrary modal formulas. A *Kripke structure* is a triple $M = \langle \mathcal{U}, \pi, \mathcal{R} \rangle$, where $\mathcal{U}$ is a set (the set of *states*), π is a function from the set of propositional atoms times the set of states into the truth values for propositional logic (i.e. $\pi : \mathcal{A} \times \mathcal{U} \longmapsto \{True, False\}$), and $\mathcal{R}$ is a binary relation on the set of states $\mathcal{U}$. We call a *situation* any pair M, u such that $u \in \mathcal{U}$. The binary relation $\models$ between a modal formula φ and a situation M, u is defined as follows:

$$
\begin{array}{lcl}
M, u \models A_i & \Longleftrightarrow & \pi(A_i, u) = True; \\
M, u \models (\varphi_1 \wedge \varphi_2) & \Longleftrightarrow & M, u \models \varphi_1 \textit{ and } M, u \models \varphi_2; \\
M, u \models (\varphi_1 \vee \varphi_2) & \Longleftrightarrow & M, u \models \varphi_1 \textit{ or } M, u \models \varphi_2; \\
M, u \models \neg\varphi & \Longleftrightarrow & M, u \not\models \varphi; \\
M, u \models \Box\varphi & \Longleftrightarrow & M, u' \models \varphi \text{ for every } u' \in \mathcal{U} \text{ such that } u, u' \in \mathcal{R};
\end{array}
$$

"$M, u \models \varphi$" should be read as "M, u *satisfies* φ". We say that M *satisfies* φ iff M, u satisfies φ for some $u \in \mathcal{U}$. We say that a formula φ is *satisfiable* if and only if there exists a Kripke structure M which satisfies φ. For a more detailed description on K – including axiomatic characterization, decidability and complexity results – see, e.g., [Che80, HM92]. We only recall that the problem of determining the satisfiability of a formula in K is decidable and PSPACE-complete [Lad77, HM92].

In the rest of the paper, the following terminology and notation is extensively used. A *modal atom* is a formula of the form $\Box\varphi$, and an *atom* is either a modal atom or a propositional letter. A *literal* is either an atom or its negation. An *assignment* is a partial function from the set of atoms to {*True, False*}. Notationally, assignments are denoted with the Greek letters $\mu, \mu_1, \mu_2, \ldots$, and are identified with the set of literals which are true in μ. Given a formula φ,

- $|\varphi|$ is the *size* of φ, i.e. the number of subformulas of φ.
- *depth(φ)* is the *depth* of φ, i.e. the maximum number of nested modal operators in φ.
- $atoms(\varphi, d)$ is the number of distinct atoms occurring in φ at depth d, while $atoms(\varphi)$ is the sum over d of $atoms(\varphi, d)$, i.e.

$$atoms(\varphi) = \sum_{d \leq depth(\varphi)} atoms(\varphi, d).$$

Trivially $atoms(\varphi) \leq |\varphi|$. To simplify the description, we assume that formulas are built only by applying $\vee, \wedge$ to propositional and modal literals like $A_i, \neg A_i, \Box\varphi, \neg\Box\varphi$. Therefore, we write $\neg\varphi$ (where φ is an arbitrary formula) as an abbreviation for the formula obtained from φ by recursively applying the rules mapping $\neg(\varphi_1 \wedge \varphi_2)$, $\neg(\varphi_1 \vee \varphi_2)$ and $\neg\neg\varphi'$ to $(\neg\varphi_1 \vee \neg\varphi_2)$, $(\neg\varphi_1 \wedge \neg\varphi_2)$ and φ' respectively.

3 The different approaches to modal reasoning

We call "tableau based" any system that implements Smullyan's tableau calculus as defined in [Smu68]. We call "translation based" any system in which modal formulas are translated into First Order Logic (FOL) and hence given in input to a classical theorem prover. We call "SAT based" any system that uses a SAT decision procedure to handle the embedded propositional reasoning. Notice that some authors (e.g., Ian Horrocks [Hor98a, Hor98b]) use the word "tableau" to denote both tableau and SAT based procedures. We distinguish the two approaches because, though apparently similar, they perform a very different form of reasoning and thus have search spaces of completely different shape (see [GS96b, GS96c]).

3.1 The tableau based approach

Tableau based procedures basically consist of a control strategy applied on top of a tableau framework. By tableau framework we denote any refutation formal system extending Smullyan's propositional tableau with rules handling the $\Box$-operator. Thus, for instance, in our terminology KRIS [BH91, BFH$^+$94], CRACK [BFT95] and LWB [HJSS96] are tableau based systems.[1]

For instance, in the labeled tableau framework for modal K described in [Fit83], branches are represented as sets of labeled formulas $s : \psi$, where s is the state in which the formula ψ has to be satisfiable. At the first step the root $1 : \varphi$ is created, φ being the modal formula to be proved (un)satisfiable. At the i-th step, a branch is expanded by applying to a chosen formula the rule corresponding to its main connective. The rules are the following:

$$(\wedge)\ \frac{u : (\varphi_1 \wedge \varphi_2)}{u : \varphi_1, u : \varphi_2} \qquad (\vee)\ \frac{u : (\varphi_1 \vee \varphi_2)}{u : \varphi_1 \qquad u : \varphi_2}\ ,$$

$$(\neg\Box)\ \frac{u : \neg\Box\varphi}{u' : \neg\varphi} \qquad (\Box)\ \frac{u : \Box\varphi}{u'' : \varphi}$$

We call the first two rules *propositional rules* and the latter two *modal rules*. The modal rules are constrained by the following applicability conditions:

- **$\neg\Box$-rule**: u' is a new state (u' is said to be *directly accessible* from u).
- **$\Box$-rule**: u'' is an existing state accessible from u.

Any application of the $\vee$-rule splits the branch into two sub-branches. A branch is closed when a formula ψ and its negation $\neg\psi$ occur in it. The procedure stops when all branches are closed (φ unsatisfiable) or no more rule is applicable (φ satisfiable).

[1] As Smullyan's tableau calculus is substantially a cut-free Gentzen sequent calculus looked upside-down, we have included LWB among tableau based systems.

```
function K-Consistent(φ)
    return K-Tableau({φ});

function K-Tableau(Γ)
    if A_i ∈ Γ and ¬A_i ∈ Γ                        /* branch closed */
        then return False;
    if (φ1 ∧ φ2) ∈ Γ                               /* ∧-elimination */
        then return K-Tableau(Γ ∪ {φ1, φ2}\{(φ1 ∧ φ2)});
    if (φ1 ∨ φ2) ∈ Γ                               /* ∨-elimination */
        then return   K-Tableau(Γ ∪ {φ1}\{(φ1 ∨ φ2)})  or
                      K-Tableau(Γ ∪ {φ2}\{(φ1 ∨ φ2)});
    for any ¬□β_j ∈ Γ do                           /* branch expanded */
        if not K-Tableau({¬β_j} ∪ ⋃_{□α_i∈Γ} {α_i})
            then return False;
    return True;
```

Figure 1: An example of a tableau based procedure for modal K.

For some modal logics it is possible to drop labels by using alternative sets of non-labeled modal rules [Fit83]. For instance in modal K it is possible to use unlabeled formulas and update branches according to the propositional rules

$$(\wedge)\ \frac{\Gamma, \varphi_1 \wedge \varphi_2}{\Gamma, \varphi_1, \varphi_2} \qquad\qquad (\vee)\ \frac{\Gamma, \varphi_1 \vee \varphi_2}{\Gamma, \varphi_1 \qquad \Gamma, \varphi_2}$$

where Γ is an arbitrary set of formulas; and the modal rule

$$(\Box/\neg\Box)\ \frac{\Gamma}{\alpha_1 \wedge \ldots \wedge \alpha_m \wedge \neg\beta_j}$$

In this last rule, Γ is a set of literals which includes $\neg\Box\beta_j$ and whose only modal atoms are $\Box\alpha_1, \ldots, \Box\alpha_m$.

This describes the tableau based decision procedure of Figure 1, which is the restriction to modal K of the basic version of the KRIS procedure described in [BH91]. Tableau based formalisms for many modal logics are described, e.g., in [Fit83, Mas94].

3.2 The translation based approach

Translation based procedures translate modal formulas into first-order logic and then perform conventional first-order theorem proving. Thus, TA [HS97b, HS97a, HS98] is a translation based system.

The "optimized functional translation" performed by TA is a mapping from modal K to path logic, a monadic fragment of first-order logic with one function symbol $\circ$. More in detail, given a modal formula φ, it generates the first order formula $\Phi' = cnf(\neg\Upsilon(\forall\ x\ \pi_f(\neg\varphi, x)))$, where:

- π_f is defined inductively as follows

$$\begin{array}{lcl} \pi_f(A_i, s) & = & P_i(s), \\ \pi_f(\varphi_1 \wedge \varphi_2, s) & = & \pi_f(\varphi_1, s) \wedge \pi_f(\varphi_2, s), \\ \pi_f(\varphi_1 \vee \varphi_2, s) & = & \pi_f(\varphi_1, s) \vee \pi_f(\varphi_2, s), \\ \pi_f(\neg\varphi, s) & = & \neg\pi_f(\varphi, s), \\ \pi_f(\Box\varphi, s) & = & def(s) \supset \forall\ x\ \pi_f(\varphi, s \circ x). \end{array}$$

  In the above definition, s is a term representing a Kripke state; $\circ$ is a binary function symbol: $s_1 \circ s_2$ intuitively means "s_2 is accessible from s_1"; P_i is a unary predicate uniquely associated to A_i: $P_i(s)$ intuitively means "A_i holds in the state s"; def is a unary predicate: $def(s)$ intuitively means "s has at least one successor";

- Υ denotes the quantifier exchange operator which moves inwards the existential quantifiers over universal quantifiers by applying recursively the rule
$$\exists x_1.\forall x_2.\psi \qquad \Longrightarrow \qquad \forall x_2.\exists x_1.\psi;$$

- cnf denotes a conversion in first-order conjunctive normal form, including Skolemization.

Φ' is satisfiable in FOL if and only if φ is satisfiable in K. Furthermore, [Sch98b] proves that any resolution procedure with condensing is a decision procedure for such encoded formulas. TA [HS97a, HS98] uses SPASS together with FLOTTER [WGR96] as first-order decision procedure.

Similar transformations are possible for other modal logics (e.g. KT, KD4, S4 and S5). For the logic K, one further transformation is performed over Φ' where $P_i(x \circ s_1 \circ \ldots \circ s_d)$ becomes $P_i^d(s_1, \ldots, s_d)$, P_i^d being a predicate of arity d (x is removed as it is shared by all predicates). We denote by Φ the result of this transformation. See [Sch98b, HS98] for more details.

Intuitively, P_i^d represents the propositional atom A_i occurring in φ under the scope of d boxes, while the term d-tuple $\underline{s} = s_1, \ldots, s_d$ represents a certain state w occurring at depth d in the searched Kripke model. (More precisely, it represents a *path* of depth d from the root to the state w in the tree-like structure of the model; the root is omitted.) Thus, the intended meaning of $P_i^d(\underline{s})$ is "A_i holds in the state w". The first-order theorem prover searches for an Herbrand model (if any) for Φ, which mirrors the corresponding Kripke model for φ.

3.3 The SAT based approach

SAT based procedures are built on top of decision procedures for propositional satisfiability (SAT), which handle the propositional component embedded in modal reasoning. Thus, in our terminology, KSATLISP [GS96a], FACT [Hor98b, Hor98a], DLP [PS98] are SAT based systems.

From a computational viewpoint, the idea of the SAT based approach is to see a modal formula as a propositional formula in its top-level atoms, so that an assignment μ (as defined in Section 2) corresponds to a standard assignment in classical logic. The purely modal component of reasoning can be handled by standard tableau rules. This allows us to build SAT based formalisms from tableau based ones [SG97]. To this extent, a SAT based formalism for K can be seen in terms of the rules:

$$(SAT)\,\frac{\varphi}{\mu_1 \quad \mu_2 \quad \ldots \quad \mu_n} \qquad (\Box/\neg\Box)\,\frac{\mu}{\alpha_1 \wedge \ldots \wedge \alpha_m \wedge \neg\beta_j}$$

where:

- in the SAT-rule, $\{\mu_1, \ldots, \mu_n\}$ is a *complete* set of finite assignments for φ (i.e. $\bigvee_{i=1}^{n} \bigwedge_{\psi\in\mu_i} \psi$ is propositionally equivalent to φ).
- in the $\Box/\neg\Box$-rule, $\neg\Box\beta_j \in \mu$, and $\Box\alpha_1, \ldots, \Box\alpha_m$ are the modal atoms in μ.

For efficiency reasons, it is important to guarantee that

- the set $\{\mu_1, \ldots, \mu_n\}$ of assignments is non redundant, i.e. that $\{\mu_1, \ldots, \mu_n\} \setminus \{\mu_i\}$ is not complete for each $1 \le i \le n$, and
- each assignment μ_i in $\{\mu_1, \ldots, \mu_n\}$ is not trivially simplifyable, e.g. that no two modal atoms $\Box\varphi_1$ and $\Box\varphi_2$ occur in μ_i with $\varphi_1 = \varphi_2$ or $\varphi_1 = T$, even after some simple rewriting of φ_1.

The first requirement can been obtained, e.g., by implementing the procedures on top of the Davis-Putnam-Longemann-Loveland procedure [DP60, DLL62] (DPLL from now on). The second requirement can be obtained by simplifying and (internally) sorting modal atoms, according to some order on the set of sub-formulas.

The result is the SAT based decision procedure of Figure 2, which is the basic version of the procedure KSAT [GS96a, GS96c] (except for the optional step "early pruning", which will be discussed later). In the Figure,

- *choose-literal*(φ) returns a literal in φ chosen according to some heuristic criterion, and
- *assign*(l, φ) (i) substitute l [resp. $\neg l$] with T [resp. F] in φ, and (ii) recursively evaluates the result according to the standard propositional simplification rules ($\varphi \wedge T \Rightarrow \varphi$, $\varphi \wedge F \Rightarrow F$, $\varphi \vee T \Rightarrow T$, $\varphi \vee F \Rightarrow \varphi$).

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

function KSAT_W(φ, μ)
    if φ = T                                /* base                     */
        then return KSAT_A(μ);
    if φ = F                                /* backtrack                */
        then return False;
    if {φ = (l ∧ ψ)}                        /* 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(μ)
    for any ¬□β_j ∈ μ  do
        if not KSAT(⋀_{□α_i ∈ μ} α_i ∧ ¬β_j)
            then return False;
    return True;
```

Figure 2: An example of a SAT based procedure for modal K.

KSAT_W implements the SAT-rule and is a non-CNF variant of the DPLL procedure. KSAT_A checks recursively the K-consistency of each assignment generated by KSAT_W. Improvements to this procedure are possible, see e.g. [GS96a, GS96c].

Notice that, given the SAT-rule, the problem of determining the consistency of a formula in a logic $\mathcal{L}$ (different from K) is reduced to the problem of determining the $\mathcal{L}$-consistency of an assignment. Thus, [SG97] presents a general way to encode tableau based formalisms for many modal logics into corresponding SAT based procedures. In [Tac97], SAT based procedures for various non-normal modal logics are presented. Finally, Horrocks [Hor98b, Hor98a] with FACT, and Patel-Schneider [PS98] with DLP, improved on KSATLISP enriching the underlying SAT procedure in various ways (e.g., backjumping and caching) and extending it to many terminological logics.

4 A critical analysis of the previous evaluations

In this section we first present the 3CNF_K test methodology, proposed in [GS96a] to evaluate the performances of decision procedures for modal logics. Then, we critically discuss some of the tests performed by Giunchiglia and Sebastiani in [GS96a, GS96c]; and by Hustadt and Schmidt in [HS97a, HS98].

4.1 The 3CNF_K test methodology

According to the 3CNF_K test methodology, the performance of a system is evaluated on problem sets, i.e., sets of randomly generated 3CNF_K formulas. A 3CNF_K formula is a conjunction of 3CNF_K clauses, where each clause is a disjunction of three literals. Finally, modal atoms are restricted to have the form $\Box C$, where C is a 3CNF_K clause (see [GS96a, GS96c] for a more detailed presentation and its advantages).

The main advantage of this test methodology is that a 3CNF_K formula can be 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 at depth $< d$ is purely propositional. (The parameter p has been introduced to unbalance the tree-structure of the random 3CNF_K formulas.)

Usually, (see, e.g., [GS96a, GS96c, HS97a, HS98]) a problem set is characterized by a fixed N, d and p: L is varied 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, 100 3CNF_K formulas are randomly generated, and the resulting formulas are given in input to the procedure under test. Satisfiability percentages and the mean/median of the CPU times are plotted against the number of clauses L. For practical reasons, a timeout mechanism usually stops the execution of one problem after a given amount of time (usually after 1000 seconds of CPU).[2]

4.2 Giunchiglia and Sebastiani's tests

In [GS96a, GS96c] Giunchiglia and Sebastiani (i) define various versions of SAT based procedures for modal K, (ii) introduce the 3CNF_K test method, and (iii) use this methodology to test their SAT based procedures against the tableau systems Tableau and Kris. The criticisms by Hustadt and Schmidt to the tests in [GS96a, GS96c] are based on the observation that the 3CNF_K formulas used in these testing sessions suffer from some major drawbacks. In particular:

[2]It may be objected that varying L while keeping fixed d has the effect that all the tests in a problem set belong to the same complexity class, i.e., NP [Hal95]. Nevertheless, as we will see, (i) all the systems may show very different computational behaviors on different samples of a same problem set, and (ii) this method allows for the generation of really hard instances for current solvers. Finally, in Section 7, we present a set of tests in which the depth is varied.

(i) as the $3\mathrm{CNF}_K$ generator does not check for repeated propositional variables inside the same clause, the random formulas generated can contain logical garbage (e.g., tautologies). When this is the case, their size can be (dramatically) reduced by a propositional simplification step, like the one implemented in the preprocessor of KSATLISP. Therefore in many tests of [GS96c] (the kernel of) KSATLISP is given in input over-simplified formulas;

(ii) for certain values of the generator's parameters, the formulas generated were "trivially (un)satisfiable", that is, they can be solved by pure propositional inference. Therefore in many tests of [GS96c], KSATLISP is able to easily solve the formulas due to its ability to detect propositional (in)consistencies.

As observed in [HS98, HS97a], in order to completely solve these problems, it is sufficient to

(i') modify the $3\mathrm{CNF}_K$ random generator so that it prevents repeated propositional variables inside the same clause, and

(ii') set $p = 0$.

While the problem (i) and the proposed solution (i') are straightforward to understand, the problem (ii) and the proposed solution (ii') deserve a more fine-grained discussion. If $p \neq 0$, then on average a p fraction of all top-level literals is purely propositional, so that:

- a p^3 fraction of all top-level clauses is purely propositional. This means that a random $3\mathrm{CNF}_K$ formula φ contains an embedded 3CNF propositional wff ϕ^* of average length $p^3 \cdot L$. The bigger is L, the more likely unsatisfiable is ϕ^*, the more likely trivially unsatisfiable is φ.[3] This explains the big amount of trivially unsatisfiable formulas detected empirically in [HS97b, HS98] in the center and right of the plots. If $p = 0$, then φ contains no embedded propositional formula ϕ^*, so that the problem is eliminated.

- each top-level literal has the form $\neg\Box\psi$ with probability $(1-p)/2$; each top-level clause consists only of negative modal literals with probability $(1-p)^3/8$; the probability of having at least one such clauses is thus $1-(1-(1-p)^3/8)^L$. An assignment μ containing no negative modal literals is trivially K-satisfiable, as it is satisfied by Kripke structures whose set of states is a singleton. A sufficient condition for not having trivial satisfiability is thus having at least one clause made only of negative modal literals. Therefore, the probability this condition is verified is

[3]Considering the well-know satisfiability transition results (see, e.g., [MSL92, CA93]), one would expect an average of 50% trivially unsatisfiable formulas for $L/N \approx 4.28/p^3$. Anyway, a number of propositional variables $N < 10$ is too small for applying the results in [MSL92, CA93].

very low for small L's and converges exponentially to 1 with increasing L. This explains the big amount of trivially satisfiable formulas detected empirically in [HS97b, HS98] in the very left region of each plot. If $p = 0$, the probability above converges to 1 rather quickly.

4.3 Hustadt and Schmidt's tests

In [HS97b, HS98], the authors (i) present the authors' own translation based procedure TA, (ii) point out the defects of the tests in [GS96a, GS96c], (iii) suggest the solutions described in the previous subsection, and (iv) present an empirical comparison among the systems KRIS, LWB, KSATLISP and TA, whose result lead the authors to claim the computational superiority of TA. Our criticisms to the tests in [HS97b, HS98] are based on the following four observations.

First, when testing KSATLISP, Hustadt and Schmidt disable the sorting of input formulas (this is explicitly written in [HS98][Section 6]). As it is clearly stated in the description of the KSAT algorithm ([GS96a], Section 2.2.), KSAT pre-sorts input formulas before the main routine is invoked for the first time. This is a key step in the algorithm itself, since it prevents KSAT_W from generating trivially simplifyable assignments. (Notice that pre-sorting is a key step also in DLP [PS98] and FACT [Hor98a].) To understand this better, consider the KSAT algorithm in Figure 2, and let φ be a formula in which two distinct permutations of the same modal atom, e.g., $\varphi_1 = \Box(A_1 \vee \neg A_2 \vee A_3)$ and $\varphi_2 = \Box(\neg A_2 \vee A_3 \vee A_1)$, occur. If pre-sorting is not performed, KSAT_W may invoke KSAT_A on equivalent assignments, like $\{\varphi_1\} \cup \mu$ and $\{\varphi_2\} \cup \mu$; or, more generally, on $\{\varphi_1\} \cup \mu$ and $\{\varphi_2\} \cup \mu'$, where $\mu \subseteq \mu'$; or on intrinsically unsatisfiable assignments like $\{\varphi_1, \neg\varphi_2\} \cup \mu$. As a matter of fact, sorting can reduce $atoms(\varphi)$ (up to a 3! factor in depth 1 random 3CNF_K formulas) and consequently it can reduce the number of assignments generated, which is $2^{O(atoms(\varphi))}$ [SM97]. Therefore, what Hustadt and Schmidt have tested in [HS97b, HS97a, HS98] is not KSATLISP, but a weakened version of KSATLISP, that we will call KSATLISP(UNSORTED).

Second, as noticed in the previous section, problems (i) and (ii) affect dramatically the adequateness of the original version of the 3CNF_K test method; while, on the other hand, the suggestions (i') and (ii') reveal to be very effective to overcome this problem. The obvious consequence is that, to generate significant testbeds, (i') and (ii') are to be adopted when using the 3CNF_K method. Surprisingly, in [HS97b, HS98] this is done only for the tests at the very end of the papers: many tests are performed with the non-improved generator, and many conclusions of the paper are based on the results of such tests. For example, one such test with $N = 3$, $p = 0.5$ and $d = 2, 3, 4, 5$, ([HS98], Figures 19,20,21)[4] lead the authors to conclude that the parameter d does not

[4] In [GS96c, HS97b], an additional parameter m representing the number of modalities in the formula is considered. Increasing m while leaving the other parameters fixed, makes the problem sets easier [GS96c]. Thus, in the tests that we consider m is set to 1.

seriously affect the hardness of the generated formulas, and thus they suggest to put $d = 1$. Our empirical experience instead reveals that, adopting (i') and (ii'), 3CNF_K tests with $d > 1$ become dramatically hard, probably out of the reach of most currently available procedures. For example, there are classes of formulas which are satisfiable only in Kripke structures whose set of states is exponential in the depth of the formula [HM92]. As a consequence, the performances of any procedure on such formulas should exponentially decrease if the depth is increased. Another such test with $N = 5$, $p = 0.5$ and $d = 2$, ([HS98], Figure 7) lead the authors to conclude that sorting has no significant effect for KSATLISP. (Notice that for this test, according to Figure 6 in [HS98], preprocessing causes an average 50% reduction in size of the formulas.) As we will see in Section 6, this is not true for all parameter settings (and in particular, it is not the case for some of the settings used by Hustadt and Schmidt). In order to understand why sorting has no effects for such parameter settings, we have rerun this test, and have noticed that

- for $L/N \geq 20$, more than 50% samples are trivially unsatisfiable, so that the influence of sorting is null, and
- for $L/N < 20$, the parameter values chosen are such that the average number of repeated occurrences of the same sorted modal atom of depth 2 inside the formulas is very low, so that the effect of sorting becomes negligible.[5]

Third, we notice that all the tests in [HS97b, HS98] are performed by adding the KSATLISP preprocessor to KRIS and LWB (called KRIS* and LWB*: no need to add it to TA, as FLOTTER already embeds a very sophisticated simplifier). From a technical viewpoint, without (i') and (ii'), this means giving in input the over-simplified and often trivially (un)satisfiable formulas to all procedures instead of making such formulas non-simplifyable and non-trivial. From a methodological viewpoint, we remark that what the authors actually tested —KSATLISP(UNSORTED), KRIS* and LWB*— are *not* KSATLISP, KRIS and LWB. We believe that the correct thing to do would have been to test KSATLISP, KRIS, LWB and TA with formulas which are insensitive to sorting and preprocessing, that is, 3CNF_K generated following the suggestions (i') and (ii').

Finally, we notice that the quality of the implementation of the procedures tested in [HS97a, HS98] is rather different. KRIS is a LISP implementation of the algorithm in [BH91, BFH+94]; KSATLISP is a straightforward LISP implementation of the KSAT algorithm described in [GS96c]; LWB is a

[5]The number $N(d)$ of possible distinct sorted top-level atoms in a 3CNF_K formula of depth d (without repetitions of atoms inside a clause) is given by the recursive equation

$$\begin{array}{rcl} N(0) & = & N \\ N(d) & = & N(0) + 2^3 \cdot \binom{N(d-1)}{3}. \end{array}$$

If repetitions are allowed, the number is even bigger.

well-engineered C implementation of the procedure described in [HJSS96]; TA uses SPASS (together with FLOTTER) [WGR96], a highly-engineered theorem prover written in C, and one of the fastest first order theorem provers currently available.[6] The effect of improving the quality of the implementation of the different procedures needs investigating, and it will be discussed in the next sessions.

5 KSATC

To address the problem related to the different levels of engineering of systems highlighted in Section 4.3, we have built KSATC, 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 [BB92].[7] 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 formula (partial) evaluation. In DPLL (and hence in KSAT_W) literals are assigned and unassigned dynamically inside a formula simply by moving pointers (that is, without copying the formula 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., [CA93]) DPLL splits on the literal occurring most often in the shortest clauses.

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

KSAT_W has been obtained from DPLL by performing the following steps:

- The code implementing the pure literal rule has been ruled out. (According to the pure literal rule, if a literal l occurs in φ while $\neg l$ does not, l is substituted with T [DP60].) With the pure literal rule, the set of assignments generated by KSAT_W might be incomplete, so that KSATC might conclude that the input formula 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);

[6] See `http://www.cs.jcu.edu.au/~tptp/CASC-15/` for a comparison of SPASS performances with respect to other state-of-the-art systems participating at the "ATP System Competition (CASC-15)", held at CADE-15 on Tuesday, 7th July, 1998.

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

- The (optional) step of early pruning has been included in DPLL (see Figure 2). This step is not mandatory. However, as discussed in [GS96a], 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 formula obtained from φ_i by recursively substituting each atom $\Box\varphi_j$ with the corresponding B_j. When invoked on the input formula φ, KSATC initializes LUT and passes φ^* to KSAT. Each time it is invoked on a propositional assignment μ, KSAT_A retrieves from LUT the formulas corresponding to the modal literals in μ, it conjoins them appropriately and invokes KSAT on the resulting formulas. Notice that each retrieval requires only constant time.

6 A first experimental analysis

We compare five systems, namely: KRIS, KSATLISP, KSATLISP(UNSORTED), KSATC, and TA.[8] Notice that our main interest is comparing KSATC and TA. KRIS, KSATLISP and KSATLISP(UNSORTED) are tested only to reproduce the tests in [HS97a, HS98]. Furthermore, the comparison of KSATC vs. KSATLISP allows us to get an estimate of the benefits gained by improving the underlying SAT strategy and implementation.

To test these systems, we use the 3CNF_K methodology described in [GS96a, GS96c] with the improvements (i') and (ii') of Section 4.2. As a consequence, the samples that we have used throughout our experimental analysis are (almost) insensitive to preprocessing (i.e. simplifying), and are not trivially (un)satisfiable. We have verified the first property by running the built-in simplifier of KSATC on each sample, and then comparing the original sample vs. the simplified one. The result is that there is little or no difference between them.

Consider the problem set PS1 (called PS12 in [HS98]) characterized by $d = 1$ and $N = 4$. Figure 3 reports the median CPU time for the five systems (left). It also reports the median number of consistency checks (right) for KSATLISP and KSATC(with reference to Figure 2, the number of consistency checks is the number of times the procedure KSAT is invoked). For Figure 3 left, notice the logarithmic scale on the vertical axis. For KSATLISP and KSATC, the CPU time does not include the time needed for formula 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 needed to convert modal formulas into first

[8] TA is available at `http://www.mpi-sb.mpg.de/` in `~hustadt/mdp/`. In particular, we have used TA version 1.2, SPASS/FLOTTER version 0.55. KRIS, KSATLISP, KSATC, the random formula 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`.

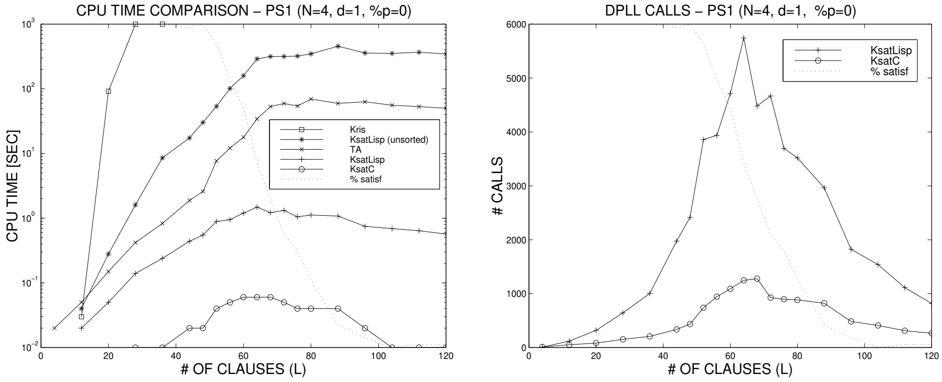

Figure 3: Left: KRIS, TA, KSATLISP(UNSORTED), KSATLISP and KSATC median CPU time, 100 samples/point. Right: KSATLISP and KSATC median number of consistency checks, 100 samples/point. Background: satisfiability percentage.

order formulas and the time used by FLOTTER to perform the conversion to conjunctive normal form. We also plot the percentage of satisfiable formulas to get a rough estimation of where the cross-over point of 50% of satisfiable formulas occurs.[9]

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: as it can be seen from Figure 3 left, the gap between KSATLISP(UNSORTED) and KSATLISP grows up to more than 2 orders of magnitude (e.g., for $L = 80$). This is not surprising, since sorting reduces the number of atoms in the input formula, and thus it may dramatically affect the computation time of the system (which grows up exponentially with such number). If we consider TA CPU times, we see that TA performs better that KSATLISP(UNSORTED) (and this agrees to the plots in Figure 23 of [HS98]), but worse than KSATLISP and KSATC. 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). Thus, Hustadt and Schmidt's claim that TA has better performance than the other systems they considered, does not seem to be justified. Even fixing Giunchiglia and Sebastiani's test methodology according to their guidelines, KSATLISP performs better than TA. The different results obtained in [HS98, HS97a] are simply due to the fact that Hustadt and Schmidt tested KSATLISP(UNSORTED) and not KSATLISP, clearly underestimating the effects of disabling the sorting of input formulas.

[9] The tests in Figure 3 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.

Considering KRIS CPU times, the gap between KRIS and KSATLISP [resp. KSATC] reaches about 4 [resp. 5] orders of magnitude for $L = 28$. For $L = 36$ no formula is solved by KRIS within the timeout.[10] As it is discussed in [GS96c, GS96b], the inefficiency of tableau based procedures compared to SAT based procedures is the result of these two facts:

1. they generate "redundant" branches, and
2. they do not prune branches as soon as they violate some propositional constraint.

Let us now consider the difference between KSATLISP and KSATC. From an algorithmic perspective, the two procedures differ only for the splitting strategy. As we said in the previous section, while KSATC splits on the literal occurring most often in the *shortest* clauses, KSATLISP splits on the atom occurring most often in the input formula. This different strategy has important effects, as it can been seen comparing the number of consistency checks performed by the two procedures (Figure 3, right). KSATLISP does always perform more consistency checks than KSATC, and the difference is biggest when $L = 64$. Interestingly, KSATLISP has its maximum value for $L = 64$ while KSATC has is maximum value for $L = 68$. In between these two values, there is the crossover point of the 50% of satisfiable formulas. If we consider the respective CPU times (Figure 3, left), we see that the gap between the two systems is not maximum for $L = 64$, but it increases with the size of the formula, reaching almost two orders of magnitude at the right end side of the plot. This is not surprising, and it mainly depends on the different quality of the two implementations. More precisely, it depends on the different computation times taken by the two procedures to assign literals (constant for KSATC, and linear in the size of the formula for KSATLISP, see the previous section).

Finally, consider the overall behavior of the different procedures. As it can be easily seen from the right plot in Figure 3, for SAT-based procedures there is an easy-hard-easy peak centered about the 50% satisfiable point. For KSATC and KSATLISP, this behavior can be also observed looking at the left plot. The same phenomenon is not observable for the other procedures. In particular, running TA on highly constrained formulas ($L = 140, 180, 200$), we observe that the median CPU time on 100 samples is always higher than 40 seconds, so we still have no evidence that TA runtime decreases when formulas become trivially unsatisfiable.

7 KSATC vs. TA

In this section we focus on KSATC and TA. We first report some experimental analysis, then we discuss the possible reasons for their different behaviors.

[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 formula is solved within the timeout" means that the first 50%+1 samples all exceeded the timeout, while the others have not been tested.

7.1 The experimental analysis

The 3CNF_K test methodology presented in [HS98] and used in the previous section is not yet optimal. In fact, 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. This strategy may produce, for instance, the clause $\Box(A_1 \vee A_2 \vee A_3) \vee \neg\Box(A_1 \vee A_3 \vee A_2)$, which becomes a tautology after sorting. To avoid this, we have introduced a further slight improvement in the formula generator: it pre-sorts all modal atoms and avoids multiple occurrences of (sorted) modal atoms inside clauses. It is clear that there is no point in trying to simplify these samples.

Using this 3CNF_K test methodology, we now consider the three problem sets PS2 ($d = 1, N = 4$), PS3 ($d = 1, N = 5$) and PS4 ($d = 1, N = 6$). PS2 and PS4 reproduce two experiments presented in [HS98] (with the change described above). The results of PS2, PS3, PS4 are described in Figure 4.[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 rows correspond to PS2, PS3 and PS4 respectively. In the first column we compare TA and KSATC CPU times, in logarithmic scale. In the second column we plot the KSATC global number of consistency checks in linear scale. The dotted plots in the background represent the percentages of satisfiable formulas, in linear scale, with 100% satisfiable at the upper end of the scale.

An eye-catching observation to the first column 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 formulas) 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 $L = 102$ 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 formula 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 number of consistency checks (right column) tends to zero whenever L approaches the "100% unsatisfiable" area. [GS96b] 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 over-constrainedness makes unsatisfiability more evident, and thus easier to detect (see [WH94] for a fine-grained analysis of this point.) Similarly to tableau based systems [GS96b], TA does not seem capable to take full advantage of over-constrainedness of the input formulas. For instance, in the extreme right

[11] All the tests of Figure 4 have been performed with the same machine, operating system and compilers as those in Figure 3.

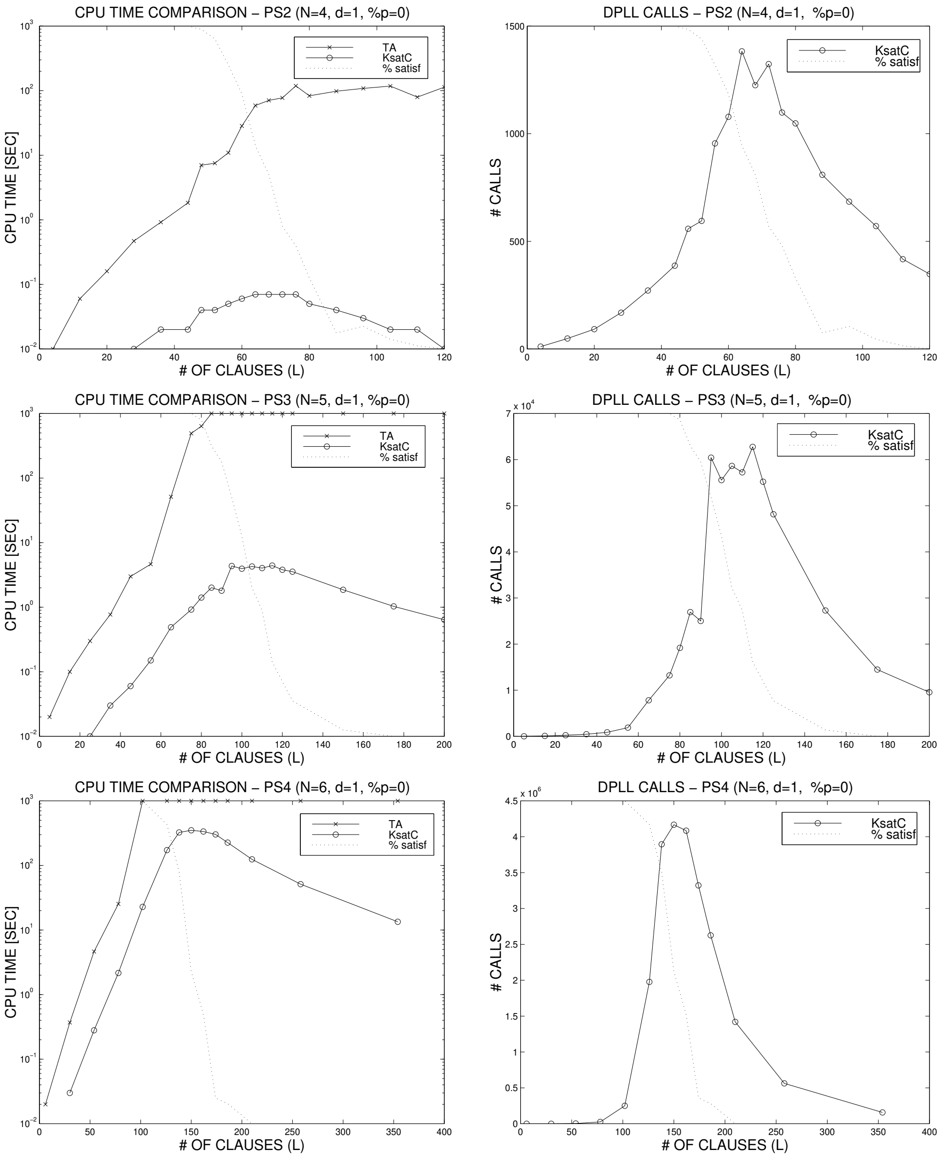

Figure 4: TA versus KsatC. CPU times (left), and KsatC number of consistency checks (right). Background: satisfiability percentages (100% satisfiable at end-scale). PS2 (first row), PS3 (second row), PS4 (last row).

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

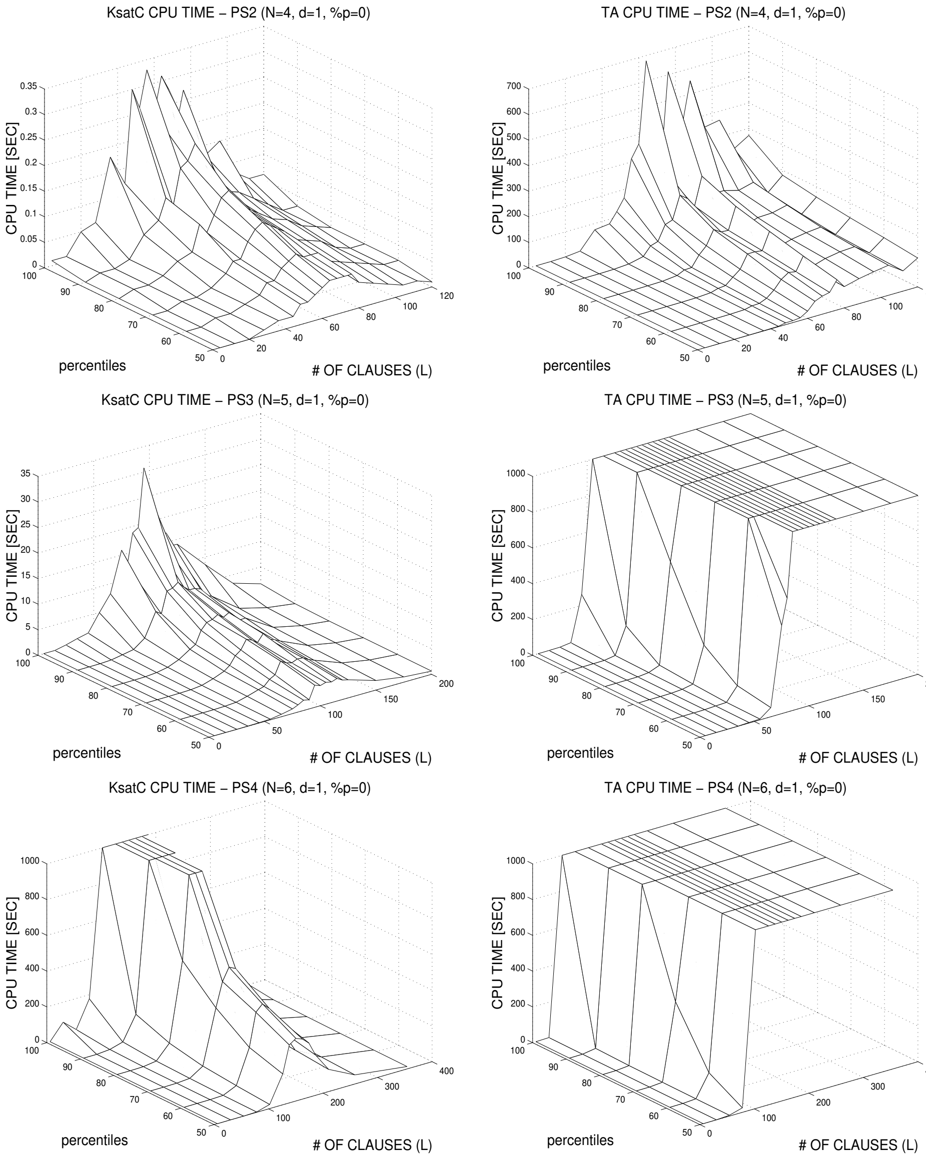

Figure 5: The percentile graphs for KsatC (left column) and TA (right column) on PS2 (top row), PS3 (middle row) and PS4 (bottom row).

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 [GS96c, GS96b].

All the above considerations are confirmed by the Q%-percentile graphs in Figure 5. 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 5 reports the 50%, 60%, 70%, 80%, 90% and 100% percentile curves of KSATC (left column) and TA (right column) CPU times when run 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 values of the end-scale used on the vertical axis for KSATC and TA: 0.35 vs. 700 and 36 vs. 1000 in the first and second row respectively), 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 formulas $\{\varphi_d^K\}_{d=0,1,2,\ldots}$ presented in [HM92]. Formally, φ_d^K is defined to be

$$A_0 \wedge \neg A_1 \wedge \bigwedge_{i=0}^{d} \Box^i(\mathit{depth} \wedge \mathit{determined} \wedge \mathit{branching})$$

where ($B_1, B_2, \ldots$ are propositional variables)

- $\Box^0\psi$ is an abbreviation for ψ, and $\Box^{i+1}\psi$ is an abbreviation for $\Box\Box^i\psi$,
- *depth* is an abbreviation for

$$\bigwedge_{i=1}^{d+1}(A_i \supset A_{i-1}),$$

- *determined* is an abbreviation for

$$\bigwedge_{i=1}^{d}(A_i \supset ((B_i \supset \Box(A_i \supset B_i)) \wedge (\neg B_i \supset \Box(A_i \supset \neg B_i)))),$$

- *branching* is an abbreviation for

$$\bigwedge_{i=0}^{d-1}((A_i \wedge \neg A_{i+1}) \supset (\neg\Box\neg(A_{i+1} \wedge \neg A_{i+2} \wedge B_{i+1}) \wedge \neg\Box\neg(A_{i+1} \wedge \neg A_{i+2} \wedge \neg B_{i+1}))).$$

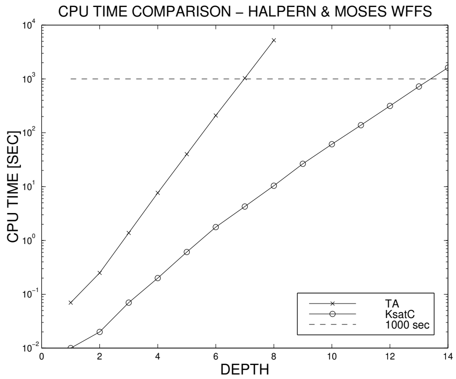

Figure 6: TA and KSATC CPU times for the class of φ_d^K formulas.

These are K-satisfiable formulas. However, every Kripke structure satisfying φ_d^K has at least 2^d states, while $|\varphi_d^K|$ is $O(d^2)$. From the results in [HM92] 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 6 presents the resulting CPU times.[12] TA 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 [GS96c].

7.2 A discussion

In this section we provide some hints for a theoretical explanation of the performance gap between KSAT and TA. We look at formulas which are satisfiable in modal K, and we start from the consideration that some of them – like, e.g., $\{\varphi_d^K\}_{d=1,2,\ldots}$ in Section 7 — have only Kripke models with exponentially many states (more precisely, $O(|\varphi|^{depth(\varphi)})$ states [HM92, Hal95]). Then we focus on two main aspects of modal reasoning:

- **branching:** which are the sources of branching for a given procedure? How many are there?
- **model spanning:** how does the procedure span the (exponentially big) model it is looking for?

[12]The test in Figure 6 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`.

and we suggest how the different answers can affect the efficiency of the two different approaches.

7.2.1 Branching

KSAT and the SAT based procedures branch "semantically" on *truth values* of the distinct modal atoms: when a modal literal l is selected (e.g., l is either $\Box\varphi_1$ or $\neg\Box\varphi_i$), then it is first considered the case where all the top-level instances of l are set to $True$, then the case where all the top-level instances of l are set to $False$. (The tableau based procedures instead branch "syntactically" on *disjunctions*: when $(\varphi_1 \vee \varphi_2)$ is selected, φ_1 is added first; if this fails, φ_2 is added in place of φ_1.) Therefore in the SAT based approach a formula φ offers $O(atoms(\varphi))$ sources of branching. Thus, we can infer an upper bound of $2^{O(atoms(\varphi))}$ for the size of the search space.

A similar analysis for TA and the translation based procedures requires a clearer understanding of what these procedures branch on. When a resolution theorem prover is run over Φ, the generic resolution step between two clauses can be represented as follows:

$$C' \vee P_i^d(\underline{t'}),\ C'' \vee \neg P_i^d(\underline{t''}) \quad \Longrightarrow \quad C'\sigma \vee C''\sigma \tag{1}$$

where C', C'' are (sub)clauses and σ is the selected unifier for $\underline{t'}$, $\underline{t''}$. The resulting clauses can then be reduced according to various simplification techniques like, e.g., condensing [Joy76]. Step (1) can be decomposed into two substeps:

- apply the unifier σ to both clauses:

$$C' \vee P_i^d(\underline{t'}),\ C'' \vee \neg P_i^d(\underline{t''}) \quad \Longrightarrow \quad C'\sigma \vee P_i^d(\underline{t'})\sigma,\ C''\sigma \vee \neg P_i^d(\underline{t''})\sigma \tag{2}$$

- apply a propositional resolution step to the unified clauses:

$$C'\sigma \vee P_i^d(\underline{t'})\sigma,\ C''\sigma \vee \neg P_i^d(\underline{t''})\sigma \quad \Longrightarrow \quad C'\sigma \vee C''\sigma \tag{3}$$

As $P_i^d(\underline{t'})\sigma = P_i^d(\underline{t''})\sigma$, the propositional resolution step (3) is a form of "local" semantic branching applied to $P_i^d(\underline{t'})\sigma$. In fact, if we set $P_i^d(\underline{t'})\sigma$ to $False$ and apply unit propagation, we get the left disjunct $C'\sigma$; if we set $P_i^d(\underline{t'})\sigma$ to $True$ and apply unit propagation, we get the right disjunct $C''\sigma$. If $\underline{t'}\sigma$ is ground, then it represents some state u of the searched Kripke model for φ, so that assigning $P_i^d(\underline{t'})\sigma$ to $True$ [resp. $False$] is equivalent to state that the corresponding atom A_i holds [resp. does not hold] in u. Thus, translation based procedures branch semantically on the truth values of pairs $\langle propositional\_variable, state\rangle$ resembling respectively the cases when the propositional variable is false and true in the state.

Now consider a formula φ satisfiable only in Kripke models with $O(|\varphi|^{depth(\varphi)})$ states. The converter maps all the states of a Kripke model of φ into distinct ground terms in the corresponding Herbrand model of the resulting formula

Φ. As a consequence, Φ has only Herbrand models with exponentially many ground terms. Consequently, the number of ground atoms $P_i^d(\underline{t'})\sigma$ in any Herbrand model of Φ is $O(|\varphi|^{depth(\varphi)})$. Therefore in the translation based approach a formula φ offers $O(|\varphi|^{depth(\varphi)})$ sources of branching. (Notice that here we do not count all the intermediate resolution steps on non-ground atoms.). Thus we can infer only an upper bound of $2^{O(|\varphi|^{depth(\varphi)})}$ for the size of the search space of a generic resolution theorem prover. We are not aware of any smaller bound in the literature.[13]

Renate Schmidt [Sch98a] has remarked that for a theorem prover using ordered resolution, condensing and a particular form of renaming, the search space is $2^{O(|pred(\Phi)|\cdot depth(\varphi))}$ at most, where $|pred(\Phi)|$ is the number of predicate symbols P_i^d in Φ (that is, the number of propositional atoms in φ times $depth(\varphi)$), and that a document containing a proof of such a result is forthcoming. This holds for logics K and KD. We postpone thus any further investigation on the topic to the reading of such a document.

7.2.2 Spanning a model

The problem of K-satisfiability is in PSPACE, as the (possibly exponentially big) Kripke models have a tree-like structure which can be spanned with strategies requiring only polynomial amounts of memory (see, e.g., [Lad77, HM92]).

KSAT and the SAT based procedures —and most tableau based procedures as well— take advantage of the tree-like structure of the model, spanning it with a depth-first strategy. Moreover, they do not keep track explicitly of the states spanned. Thus they need keeping in memory only a polynomial amount of information. (See [SM97] for a more fine-grained explanation.)[14]

There is an issue of whether TA and the translation based procedures run in polynomial space too. The converter maps all the states of a Kripke model of φ into distinct ground terms in the corresponding Herbrand model of Φ; this means that, if φ has only Kripke models with exponentially many states, then Φ has only Herbrand models with exponentially many ground terms (and thus, exponentially many ground atoms). Unlike KSAT and SAT based procedures, a general purpose FOL theorem prover has no special-purpose strategy which takes advantage of the tree-like structure of the original Kripke model and spans the exponentially big Herbrand model in such a way to require only polynomial amounts of memory. It may be the case that a translation based procedure may not run in polynomial space. To test this fact, in Figure 7 we plot the amount of memory required from TA to solve the $\{\varphi_d^K\}_{d=1,2,...}$ formulas of Section 7.

[13] For the decision problem in monadic predicate calculus *without functions* Lewis [Lew80, DL83] proves both lower and upper bounds of the form $NTIME(c^{n/log\ n})$, which means "solvable by a *non-deterministic* Turing machine in time $c^{n/log\ n}$", n being the number of bits of the input string. Such a class is "roughly one exponential up from NP" [DL83], and it is much higher in the complexity hierarchy than PSPACE.

[14] As proposed, e.g., in [PS98], both tableau based and SAT based procedures can be enhanced by some form of caching of intermediate search nodes. In this case, of course, the procedures might loose PSPACE-Ness.

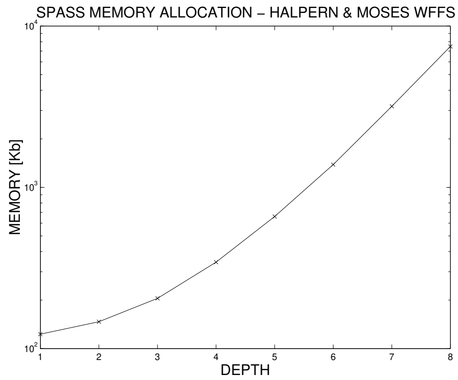

Figure 7: Amount of memory required by TA for the class of φ_d^K formulas.

(The scale of the vertical axis is logarithmic.) We notice that the graph grows exponentially with d.

8 Conclusion

We consider the tests suggested by Hustadt and Schmidt. If evaluated on these tests, SAT based decision procedures have superior behavior than decision procedures based on the translation method. This contradicts what is reported in [HS97a, HS98]. 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 some basic modal logics (e.g. E, K [Che80]). An interesting open problem is whether these (substantial) computational advantages will also exist for other interesting modal logics like CTL or LTL (see, e.g., [Eme90] for a description of these logics).

Acknowledgements

The work described in this paper has benefitted from the interactions with Ullrich Hustadt and Renate Schmidt. The results published in their papers [HS97b, HS97a, HS98] challenged us and forced us to look deeper into the previous analysis and results, bringing to new and more interesting results. The technical help of Ullrich Hustadt has been crucial in the generation of some of the empirical results described in this paper. Some discussion with Renate Schmidt improved our understanding of translation methods. This work has also benefitted of useful discussions with Franz Baader, Enrico Franconi, Harald Ganzinger, Ian Horrocks and Peter Patel-Schneider. Finally, we thank the anonymous reviewers for the many fruitful comments, and the MRG group at Genova for putting up with many weeks of time-consuming tests.

References

[BB92] M. Buro and H. Buning. Report on a SAT competition. Technical Report 110, University of Paderborn, Germany, November 1992.

[BFH+94] 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.

[BFT95] P. Bresciani, E. Franconi, and S. Tessaris. Implementing and testing expressive Description Logics: a preliminary report. In *Proc. International Workshop on Description Logics*, Rome, Italy, 1995.

[BH91] F. Baader and B. Hollunder. A Terminological Knowledge Representation System with Complete Inference Algorithms. In *Proceedings of the First International Workshop on Processing Declarative Knowledge*, volume 572 of *Lecture Notes in Computer Science*, pages 67–85, Kaiserslautern (Germany), 1991. Springer–Verlag.

[CA93] 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.

[Che80] B. F. Chellas. *Modal Logic – an Introduction*. Cambridge University Press, 1980.

[DL83] L. Denenberg and H. R. Lewis. Logical syntax and computational complexity. In *Computation and Proof Theory: proceedings of the Logic Colloquium*, Lecture Notes in Mathematics, 1104, Aachen, 1983. Springer Verlag.

[DLL62] M. Davis, G. Longemann, and D. Loveland. A machine program for theorem proving. *Journal of the ACM*, 5(7), 1962.

[DP60] M. Davis and H. Putnam. A computing procedure for quantification theory. *Journal of the ACM*, 7:201–215, 1960.

[dS98] H. de Swart, editor. *Automated Reasoning with Analytic Tableaux and Related Methods: International Conference Tableaux'98*, number 1397 in Lecture Notes in Artificial Intelligence. Springer-Verlag, May 1998.

[Eme90] E.A. Emerson. Temporal and Modal Logic. In J. van Leeuwen, editor, *Handbook of Theoretical Computer Science*, volume B, pages 995–1072. Elsevier Science Publisher B.V., 1990.

[FGM+98] E. Franconi, G. De Giacomo, R. M. MacGregor, W. Nutt, C. A. Welty, and F. Sebastiani, editors. *Collected Papers from the International Description Logics Workshop (DL'98).* CEUR, May 1998.

[Fit83] M. Fitting. *Proof Methods for Modal and Intuitionistic Logics.* D. Reidel Publishg, 1983.

[GGST98] E. Giunchiglia, F. Giunchiglia, R. Sebastiani, and A. Tacchella. More evaluation of decision procedures for modal logics. In *Sixth International Conference on Principles of Knowledge Representation and Reasoning (KR'98)*, 1998.

[GS96a] 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.

[GS96b] F. Giunchiglia and R. Sebastiani. Building decision procedures for modal logics from propositional decision procedures - the case study of modal K(m). 1996. To appear on *Information and Computation.* A longer version is IRST-Technical Report 9611-06.

[GS96c] 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.

[Hal95] J. Y. Halpern. The effect of bounding the number of primitive propositions and the depth of nesting on the complexity of modal logic. *Artificial Intelligence*, 75(3):361–372, 1995.

[HJSS96] A. Heuerding, G. Jager, S. Schwendimann, and M. Seyfried. The Logics Workbench LWB: A Snapshot. *Euromath Bulletin*, 2(1):177–186, 1996.

[HM92] 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.

[Hor98a] I. Horrocks. The FaCT system. In de Swart [dS98], pages 307–312.

[Hor98b] I. Horrocks. Using an expressive description logic: FaCT or fiction? In *Sixth International Conference on Principles of Knowledge Representation and Reasoning (KR'98)*, pages 636–647, 1998.

[HS97a] 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.

[HS97b] 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.

[HS98] U. Hustadt and R. A. Schmidt. An empirical analysis of modal theorem provers. To appear in *Journal of Applied Non-Classical Logics*, July 1998.

[Joy76] W.H. Joyner. Resolution Startegies as Decision Procedures. *Journal of the ACM*, 23(3):398–417, 1976.

[Lad77] R. Ladner. The computational complexity of provability in systems of modal propositional logic. *SIAM J. Comp.*, 6(3):467–480, 1977.

[Lew80] H. R. Lewis. Complexity Results for Classes of Quantificational Formulas. *Journal of Computer and System Sciences*, 21:317–353, 1980.

[Mas94] F. Massacci. Strongly analytic tableaux for normal modal logics. In *Proc. of the 12th Conference on Automated Deduction*, 1994.

[MSL92] 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.

[Ohl91] H. J. Ohlbach. Semantics based translation methods for modal logics. *Journal of Logic and Computation*, 1(5):691–746, 1991.

[PS98] P. F. Patel-Schneider. DLP system description. In Franconi et al. [FGM+98], pages 87–89.

[Sch98a] R. A. Schmidt, August 1998. Personal communication.

[Sch98b] R. A. Schmidt. Decidability by resolution for propositional modal logics. To appear in *Journal of Automated Reasoning*, June 1998.

[SG97] R. Sebastiani and F. Giunchiglia. From Tableau-based to SAT-based procedures - preliminary report. Technical Report 9711-14, IRST, Trento, Italy, November 1997.

[SM97] R. Sebastiani and D. McAllester. New upper bounds for satisfiability in modal logics - the case-study of modal K. Technical Report 9710-15, IRST, Trento, Italy, October 1997.

[Smu68] R. M. Smullyan. *First-Order Logic*. Springer-Verlag, NY, 1968.

[Tac97] A. Tacchella. Procedure di decisione per logiche modali classiche (decision procedures for classical modal logics). Master's thesis, DIST, Università di Genova, Italy, 1997. In italian.

[vB84] Johan van Benthem. Correspondence theory. In D. Gabbay and F. Guenthner, editors, *Handbook of Philosophical Logic, Volume II: Extensions of Classical Logic*, volume 165 of *Synthese Library*, chapter II.4, pages 167–247. D. Reidel Publ. Co., Dordrecht, 1984.

[WGR96] 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.

[WH94] C. P. Williams and T. Hogg. Exploiting the deep structure of constraint problems. *Artificial Intelligence*, 70:73–117, 1994.