An Analysis of Empirical Testing for Modal Decision Procedures

Ian Horrocks, *Department of Computer Science, University of Manchester, UK; E-mail: horrocks@cs.man.ac.uk*

Peter F. Patel-Schneider, *Bell Labs Research, Murray Hill, NJ, U.S.A.; E-mail: pfps@research.bell-labs.com*

Roberto Sebastiani, *DISA - Università di Trento, Via Inama 5, I-38100, Trento, Italy; E-mail: rseba@cs.unitn.it*

Abstract

Recent years have seen the emergence of a new generation of heavily-optimised modal decision procedures. Several systems based on such procedures are now available and have proved to be much more effective than the previous generation of modal decision procedures. As both computational complexity and algorithm complexity are generally unchanged, neither is useful in analysing and comparing these new systems and their various optimisations. Instead, empirical testing has been widely used, both for comparison and as a tool for tuning systems and identifying their strengths and weaknesses. However, the very effectiveness of the new systems has revealed serious weaknesses in existing empirical test suites and methodologies. This paper provides a detailed survey of empirical testing methodologies, analyses the current state of the art and presents new results obtained with a recently developed test method.

Keywords: empirical testing, modal decision procedures

1 Motivations for empirical testing

Heavily-optimised systems for determining satisfiability of formulae in propositional modal logics are becoming available. These systems, including DLP [Patel-Schneider1998], FaCT [Horrocks1998], KSATC [Giunchiglia *et al.*1998a], *SAT [Tacchella1999], and TA [Hustadt & Schmidt1997], have more optimisations and are much faster than the previous generation of modal decision procedures, such as leanK [Beckert & Goré1997], Logics Workbench [Heuerding *et al.*1995], and □**KE** [Pitt & Cunningham1996].

As with most theorem proving problems, neither computational complexity nor algorithm complexity is useful in determining the effectiveness of optimisations. The worst-case complexity of the problem, of course, remains unchanged. For many propositional modal logics, this complexity ranges from PSPACE-complete to EXPTIME-complete. The worst-case complexity of the algorithms in the systems also generally remains unchanged under optimisation. The worst-case complexity for most of these systems is exponential time and either polynomial or exponential space. Further, determining any useful normal-case or special-case complexity is essentially impossible.

As theoretical studies do not provide any indication of the effectiveness of the new systems and their optimisations, this has to be determined by empirical testing. In any case, empirical testing provides a number of benefits over theoretical complexity. It directly gives resource consumption, in terms of computation time and memory use. It factors in all the pieces of the system, not just the basic algorithm itself.

Empirical testing can be used not only to compare different systems, but also to tune a system with parameters that can be used to modify its performance. Moreover, it can be used to show what sort of inputs the system handles well, and what sort of inputs the system handles poorly.

In this paper we provide a detailed survey of empirical testing methodologies for modal decision procedures, including a review of previous work in the area and an analysis of the current state of the art.[1] We point out desirable and undesirable characteristics of these methodologies. We also present some new results obtained with a recently developed test method, and identify some of the remaining weaknesses in both modal decision procedures and testing methodologies.

Our goal in this paper is not to show the effectiveness of various systems and their optimisations, but is instead to provide a framework for evaluating empirical testing methodologies for modal decision procedures, to analyse these methodologies, and to give a direction for better empirical testing methodologies for modal decision procedures.

2 Evaluating Empirical Testing

2.1 Kinds of Empirical Test Sets

Several kinds of inputs can be used for empirical testing. Inputs that have been encountered in the past can be used. Variations on these past inputs, either systematic or random, can be used. It is also possible to deterministically synthesise inputs. These can be either hand-generated individual inputs, possibly parameterised, or inputs that systematically cover an area. Finally, randomly-generated inputs can be used. Each of these kinds of inputs has benefits and potential problems.

Actual past inputs provide a good mechanism for comparing the behaviour of different systems. However, as systems improve, old inputs can become so easy that they provide no guidance. Also, newer inputs will not necessarily be the same as past inputs, so using past inputs may not provide guidance for future performance. Further, there may not be enough past inputs to provide sufficient testing.

Variations on past inputs can be used to overcome some of these problems. Variations provide more inputs. If the past inputs can be modified to be larger or more difficult, then the problem of past inputs being too easy can be overcome.

Hand-generated inputs can be specifically tailored to provide good tests, at least for particular systems. In particular, if the inputs are parameterised, then they can often be made large enough to be difficult, even for newer systems. However, it can be very hard to hand-generate difficult problems, even if parameterised—a particular optimisation may make a whole parameterised set of inputs trivial. Further, hand-generation is expensive.

Systematic inputs can be very effective, provided that systematic generation is possible. However, many kinds of inputs are too large to systematically cover. Random inputs are often easy to generate, and can provide a mechanism to cover a class of inputs. Both systematic and random inputs may not be typical inputs, and so testing using these kinds of inputs may not provide useful data.

In many cases there is a lack of past inputs, not enough resources for hand-generation, and

[1] Some of the testing in this paper has been reported on in other papers [Horrocks & Patel-Schneider1999b; 1999a; Giunchiglia *et al.*1998a].

no way of systematically covering the input space, so the major mechanism for generating inputs for empirical testing has to be random generation. This is the case in propositional modal logics. The current main empirical testing methodologies involve random generation of formulae. The only other significant test consists of hand generated, parameterised formulae, but this test has become too easy for state-of-the-art systems.

2.2 *Good and Bad Empirical Testing*

The benefits of empirical testing depend on the characteristics of the inputs provided for the testing, as empirical testing only provides data on these particular inputs. If the inputs are not typical or suitable, then the results of the empirical testing will not be useful. This means that the inputs for empirical testing must be carefully chosen.

We believe that good test sets should be created according to the following key criteria.[2]

Reproducibility. Any test is not very useful if it cannot be reproduced, and varied. The test formulae or their generation function should thus be made available. Even if the test formulae are made available, the generation function should also be made available, so that variants of the test can be developed. If the actual test formulae are not made available it should be possible to *exactly* reproduce the entire test set, so all the inputs to the generation function should be disclosed, including any "random" or environmental inputs.

Representativeness. The ideal test set should represent a significant area of the whole input space, and should span the whole range of sources of difficulty. A good empirical test set should at least cover a large area of inputs. Empirical test sets that consist of only a few inputs or that concentrate on only a small area of the input space provide no information about most inputs. This can be a particular problem if the small area has a different computational complexity than the input space as a whole.

Valid vs. not-valid balance. In a good test set, valid and not-valid (or, equivalently, satisfiable and unsatisfiable) problems should be more or less equal both in number and in difficulty. In fact, solvable and unsolvable problems may present different sources of difficulty, so that a system which is good at handling one type may be not good—or not capable at all—of handling the other type. Moreover, to prevent the usage of routines/heuristics which are explicitly aimed at detecting either solvability or unsolvability, the testbed should provide no *a priori* information which could help in guessing the result —that is, *maximum uncertainty* regarding the solvability of the problems is desirable. (Notice that, in the real world, the solvability of a problem is not known a priori.)

Difficulty. A good empirical test set should provide a sufficient level of difficulty for the system(s) being tested. (Some problems should be too hard even for state-of-the-art systems, so as to be a good benchmark for forthcoming systems.) If the inputs are too easy, then the resulting resource consumption may be too small to easily measure, and the resource consumption may be dominated by start-up costs that do not grow as the difficulty of the inputs grow. Comparing absolute performances—which may depend on factors like the platform used, the quality of implementation, etc.—may be less significant than comparing how performances scale up with problems of increasing difficulty.

Termination. To be of practical use, the tests should terminate and provide information within a reasonable amount of time. If the inputs are too hard, then the system may

[2] Notice that some of the criteria are identical or similar to those suggested by Heurding and Schwendimann[1996].

not be able to provide answers within the established time. This inability of the system is of interest, but can make system comparison impossible or insignificant.

The following criteria derive from or are significant sub-cases of the main criteria above.

Parameterisation. One way of creating test sets with the appropriate features and with a large number of inputs is to have parameterised inputs with sufficient parameters and degrees of freedom to allow the inputs to range over a large portion of the input space. On the other hand, the number of parameters and their degrees of freedom should not be too large, otherwise the number of tests required to cover a significant subspace might blow up. (Ideally, the parameter set should work as much as possible as a "base" for the input space.)

Control. In particular, it is very useful to have parameters that control *monotonically* the key features of the input test set, like the average difficulty and the solvable vs. unsolvable rate. Monotonicity is a key point, as it allows for controlling one feature independently of the values of the other parameters, and for eliminating uninteresting areas of the input space.

Modal vs. propositional balance. Reasoning in modal logics involves alternating between two orthogonal search efforts: pure modal reasoning—that is, spanning the potential Kripke models—and pure propositional reasoning—that is, assigning truth values to sub-formulae within each Kripke state. A good test set should be challenging from both viewpoints.

Data organisation. The data should be summarisable—so as to make a comparison possible with a limited effort—and plottable—so as to enable the qualitative behaviour of the system(s) to be highlighted. For instance, a list of hundreds of uncorrelated numbers is not a well-organised data set, since it makes a comparison impractical, and makes it very hard to produce any qualitative information from it.

Focus on narrow problems. As an alternative to wide-ranging tests, small "ad hoc" test sets may be used for testing systems on one particular source of difficulty, or for revealing one particular possible weakness. For instance, formulae which are satisfied only by exponentially-large Kripke models (see, e.g., [Halpern & Moses1992]) might cause the system under test to blow up in space, thus revealing its non-PSPACEness.

Finally, in creating good test sets, particular care must be taken to avoid the following problems.

Redundancy. Empirical test sets must be carefully chosen so as not to include inadvertent redundancy. They should also be chosen so as not to include small sub-inputs that dictate the result of the entire input. Empirical test sets can be made irrelevant by advances in systems if the advanced systems include optimisations that identify some inherent redundancy and cause the test set to be trivially solved. Of course, a system that can detect such redundancy is better than one that cannot, but the presence of detectable redundancies can reduce test sets to triviality.

Triviality. A good test set should be flawless, that is, it should not contain significant subsets of trivial problems. This problem has claimed victims in many other areas of AI, as flaws have been detected in random test methods for propositional satisfiability [Mitchell *et al.* 1992], for constraint satisfiability problems [Achlioptas *et al.*1997], and for quantified boolean formulae [Gent & Walsh1999a].

Artificiality. A good empirical test set should correspond closely to inputs from applications. If the test set does not resemble actual inputs, then the results from the empirical testing will not necessarily correspond with the behaviour of the system in real use.

Over-size. The single problems should not be too big with respect to their difficulty, so that the resources required for parsing and data managing do not seriously influence total performance.

In general, these criteria boil down to providing a reproducible sample of an interesting portion of the input space with appropriate difficulty. This is no different from the criteria in other areas, notably propositional satisfiability testing [Gent & Walsh1993; DIM1993; Selman *et al.*1996], theorem proving [Suttner & Sutcliffe1995], CSP [DIM1993; Gent & Walsh 1999b]. However, the situation for modal decision procedures is more difficult than for the fields above, because of the greater variety in modal logics and formulae and the lesser capabilities of current modal decision procedures.

3 Systems

The systems involved in most of this testing have different characteristics, but are all based around a front-end that takes an input formula, converts it into an internal form, and uses a search engine to exhaustively search for a model of the formula. The input formula is satisfiable if such a model is found and unsatisfiable otherwise. All the systems are able to handle (at least) the propositional modal logic $\mathbf{K_{(m)}}$.

Two systems, DLP [Patel-Schneider1998] and FaCT [Horrocks1998], are based on custom-built search engines that employ tableaux techniques to search for the model. These systems both translate input formulae into an internal normal form, attempting to exploit redundancies and local analytic truth and falsity. Their search engines employ mechanisms to reduce overlapping search, cut off search branches that cannot succeed, detect forced branches, and reuse cached results from previous searching.

Both DLP and FaCT can handle logics that are supersets of $\mathbf{K_{(m)}}$. FaCT allows transitive modalities, deterministic (functional) modalities and inclusion relationships between modalities. DLP allows full propositional dynamic logic, although it has a compile-time switch to change from propositional dynamic logic to $\mathbf{S4_{(m)}}$. Most of the tests in this paper will use only DLP as it is based on the ideas developed in the FaCT system, includes most of FaCT's optimisations, and has some additional optimisations (in particular caching). If not otherwise specified, all the examples with DLP were obtained with DLP version 3.1 in its $\mathbf{S4_{(m)}}$ configuration running on a machine roughly comparable to a SPARC Ultra 1 with 256MB of main memory.

Two other systems, KSATC [Giunchiglia *et al.*1998a] and *SAT [Tacchella1999], are based on state-of-the-art propositional satisfiability testing procedures. The two systems make multiple calls to the propositional decision procedure. While KSATC's optimisations are largely those of the underlying propositional system, *SAT features many modal search pruning optimisations like modal backjumping and caching. *SAT also handles many non-normal modal logics. If not otherwise specified, all the examples with *SAT in this paper were obtained with *SAT `-e -m6`, compiled with `Linux gcc -O2` and run on a 350MHz PentiumII with 128MB of main memory.

The TA system [Hustadt & Schmidt1997] translates propositional formulae into a decidable fragment of first order logic. It uses an optimised first-order theorem prover to determine the satisfiability of the translated formulae. TA thus inherits the optimisations built into this theorem prover, but it also uses a translation that is designed to produce easier first-order formulae.

4 The Heuerding and Schwendimann Tests

As discussed in Section 2.1, it is possible to hand-generate formulae for testing modal decision procedures. In the past such formulae were difficult to analyse, but the optimised decision procedures that are now available make short work of such hand-generated formulae. For example, Heuerding and Schwendimann [1996] report that their (moderately-optimised) system, LWB, can rapidly process several previous collections, with the longest test taking under 1/10th of a second.

Such short times are not satisfactory as differences between systems may be the result of startup costs and not indicative of their behaviour on more-difficult formulae.

4.1 Rationale

To overcome the above difficulty, and also to provide more test formulae, Heuerding and Schwendimann [1996] created a suite of formulae for testing modal decision procedures. They wanted to provide a test suite that would not be quickly rendered obsolete and that would provide a comprehensive test of a modal decision procedure, so they started with a number of postulates:[3]

1. The test suite should include valid as well as invalid formulae.
2. The test suite should provide formulae of various structures.
3. Some of the formulae should be hard for future systems.
4. The validity status of the formulae should be known in advance.
5. The formulae should be resistant to simple tricks.
6. Executing the entire benchmark should not take an excessive amount of time.
7. It should be possible to summarise succinctly the results of the benchmark.

4.2 Description

To meet these postulates Heuerding and Schwendimann created classes of formulae. Each class was generated from a (relatively) simple parameterised logical formula that was either valid or invalid. Some of these formulae were made harder by hiding their structure or adding extra pieces. The parameters allow formulae of different size to be created, thus allowing for formulae of differing difficulty. The idea behind the parameter is that the difficulty of most of the problems should be exponential in the parameter. This supposed exponential increase in difficulty would make differences in the speed of the machines used to run the benchmarks relatively insignificant.

For each logic, $\mathbf{K}$, $\mathbf{KT}$, and $\mathbf{S4}$, 9 classes of formula were created, in both valid and invalid versions. For example, the branching formulae of Halpern and Moses [1992], form a formula class for all three logics. Other problem classes in the set are based on the pigeon-hole principle and a two-colouring problem on polygons.

[3] These postulates are elaborated in the paper by Heuerding and Schwendimann [1996, pp. 2–3].

	branch		*d4*		*dum*		*grz*		*lin*		*path*		*ph*		*poly*		*t4p*	
K	*p*	*n*	*p*	*n*	*p*	*n*	*p*	*n*	*p*	*n*	*p*	*n*	*p*	*n*	*p*	*n*	*p*	*n*
leanK 2.0	1	0	1	1	0	0	0	>	>	4	2	0	3	1	2	0	0	0
□KE	13	3	13	3	4	4	3	1	>	2	17	5	4	3	17	0	0	3
LWB 1.0	6	7	8	6	13	19	7	13	11	8	12	10	4	8	8	11	8	7
TA	9	9	>	18	>	>	>	>	>	>	20	20	6	9	16	17	>	19
KSAT	8	8	8	5	11	>	17	>	>	3	4	8	5	5	13	12	10	18
***SAT 1.2**	>	12	>	>	>	>	>	>	>	>	>	>	8	12	>	>	>	>
Crack 1.0	2	1	2	3	3	>	1	>	5	2	2	6	2	3	>	>	1	1
Kris	3	3	8	6	15	>	13	>	6	9	3	11	4	5	11	>	7	5
FaCT 1.2	6	4	>	8	>	>	>	>	>	>	7	6	6	7	>	>	>	>
DLP 3.1	19	13	>	>	>	>	>	>	>	>	>	>	7	9	>	>	>	>

TABLE 1. Results for $\mathbf{K}$

	45		*branch*		*dum*		*grz*		*md*		*path*		*ph*		*poly*		*t4p*	
K	*p*	*n*	*p*	*n*	*p*	*n*	*p*	*n*	*p*	*n*	*p*	*n*	*p*	*n*	*p*	*n*	*p*	*n*
leanKT 2.0	3	0	1	0	3	4	0	0	3	2	2	1	2	1	0	0	>	0
□KE	14	2	16	15	1	1	0	>	4	4	16	6	4	3	0	0	7	17
LWB 1.0	5	4	5	6	5	10	6	>	5	5	10	9	4	8	14	2	5	7
TA	17	6	13	9	17	9	>	>	16	20	>	16	5	12	>	1	11	0
KSAT	5	5	8	7	7	12	9	>	2	4	2	5	4	5	1	2	1	1
Crack 1.0	0	0	2	2	0	1	0	0	2	4	1	5	2	2	1	1	0	1
Kris	4	3	3	3	3	14	0	5	3	4	1	13	3	3	2	2	1	7
FaCT 1.2	>	>	6	4	11	>	>	>	4	5	5	3	6	7	>	7	4	2
DLP 3.1	>	>	19	12	>	>	>	>	3	>	16	14	7	>	>	12	>	>

TABLE 2. Results for $\mathbf{KT}$

The benchmark methodology was to test formulae from each class, starting with the easiest instance, until the validity status of a formula could not be correctly determined within 100 seconds. The result from this class would then be the parameter of the largest formula that could be solved within the time limit. The parameter ranges only from 1 to 21—if a system can solve all 21 instances of a class, the result is given as ">".

This benchmark suite and methodology meets several of the postulates above simply as a result of its design. The suite contains both valid and invalid formulae of various structures whose validity status is known in advance. The benchmark can be executed in a few hours at most and the results can be given in three tables each with nine rows and two columns.

4.3 Results

The benchmark suite was used in a comparison at Tableaux'98 [Balsiger & Heuerding1998]. Six entries were submitted to this comparison, giving results for a total of ten systems. Since then the benchmark has been run on several other systems, including some more-recent versions of systems included in the original test. Several results are given in Tables 1, 2, and 3. Some of these results are from the Tableaux'98 comparison, but some are more recent.

The results show that this benchmark is appropriate, perhaps even too difficult, for some of the systems. However, the heavily-optimised systems, including *SAT and DLP, are able to handle all of the instances of many of the problem classes in their areas of coverage. *SAT

	45		*branch*		*dum*		*grz*		*md*		*path*		*ph*		*poly*		*t4p*	
K	*p*	*n*	*p*	*n*	*p*	*n*	*p*	*n*	*p*	*n*	*p*	*n*	*p*	*n*	*p*	*n*	*p*	*n*
KT4	1	6	2	3	0	17	5	8	>	18	1	2	2	2	2	2	0	3
leanS4 2.0	0	0	0	0	0	0	1	1	2	2	1	0	1	0	1	1	0	0
□KE	8	0	>	>	0	>	6	4	3	3	9	6	4	3	1	>	3	1
LWB 1.0	3	5	11	7	9	>	8	7	8	6	8	6	4	8	4	9	9	12
TA	9	0	>	4	14	0	6	>	9	10	15	>	5	5	>	1	11	0
FaCT 1.2	>	>	4	4	2	>	5	4	8	4	2	1	5	4	>	2	5	3
DLP 3.1	>	>	18	12	>	>	10	>	3	>	15	15	7	>	>	>	>	>

TABLE 3. Results for **S4**

	*SAT 1.2		DLP 3.2		TA 1.4		
Test	Size	Time	Size	Time	Size	SPASS	FLOTTER
branch_p	>	0.21	19	46.06	6	51.95	13.81
branch_n	12	94.49	13	53.63	6	84.21	12.23
d4_p	>	0.06	>	0.05	15	0.64	70.47
d4_n	>	2.87	>	1.12	14	1.14	42.92
dum_p	>	0.04	>	0.02	17	3.32	61.67
dum_n	>	0.12	>	0.02	16	1.75	64.07
grz_p	>	0.04	>	0.04	>	0.35	0.16
grz_n	>	0.01	>	0.05	>	0.16	0.17
lin_p	>	0.01	>	0.03	>	1.03	8.21
lin_n	>	47.80	>	0.13	>	16.07	63.94
path_p	>	0.72	>	0.32	5	22.85	2.18
path_n	>	0.96	>	0.36	4	58.70	2.14
ph_p	8	48.54	7	10.23	6	42.19	0.97
ph_n	12	0.60	>	2.69	9	45.21	9.92
poly_p	>	1.73	>	0.11	5	2.48	51.00
poly_n	>	2.25	>	0.18	4	1.23	7.86
t4p_p	>	0.29	>	0.06	16	3.91	84.75
t4p_n	>	1.28	>	0.13	9	3.37	84.35

TABLE 4: Timing Results from [Giunchiglia *et al.*1999] for *SAT (options `-k1 -e -m6`), DLP and TA for **K**. (Courtesy of E. Giunchiglia, F. Giunchiglia and A. Tacchella.)

is able to completely solve 15 out of 18 of the **K** tests and DLP is able to completely solve 11 out of 18 of both the **KT** tests and the **S4** tests. This means that the effective number of tests is reduced considerably.

In fact, the situation is even worse than indicated by the raw results. The heavily-optimised systems can solve many of the problem classes with little or no search. This is indicated in Table 4, which gives the time taken for the most-difficult solved problems in **K** for *SAT, DLP and TA.[4] The times for TA are subdivided between FLOTTER (a pre-processor) and SPASS itself.

As the table shows, the hardest instances of many of the completely-solved formula classes

[4]These results differ slightly from the previous results for DLP and differ considerably for TA because they were performed on a different version of the systems.

Test Name	Result	Total Time	Search Time	Search Growth	Backtrack Growth	Successor Growth
branch_p	19	46.06	45.07	$O(2^n)$	0	$O(2^n)$
branch_n	13	53.63	53.50	$O(2^n)$	0	$O(2^n)$
d4_p	>	0.05	0.02	$O(n)$	0	$O(n)$
d4_n	>	1.12	1.08	$O(n^c)$	$O(n)$	$O(n)$
dum_p	>	0.02	0.01	$O(n)$	1	$O(n)$
dum_n	>	0.02	0.01	$O(n)$	0	$O(n)$
grz_p	>	0.04	0.00	$O(c)$	2	$O(c)$
grz_n	>	0.05	0.02	$O(n)$	$O(n)$	$O(n)$
lin_p	>	0.03	0.00	0	0	0
lin_n	>	0.13	0.05	$O(n)$	0	0
path_p	>	0.32	0.25	$O(n)$	0	$O(n)$
path_n	>	0.36	0.28	$O(n^c)$	0	$O(n)$
ph_p	7	10.23	10.21	$O(c^n)$	$O(c^n)$	1
ph_n	>	2.69	0.53	$O(n^c)$	0	$O(n^c)$
poly_p	>	0.11	0.04	$O(n)$	1	$O(n)$
poly_n	>	0.18	0.11	$O(n)$	0	$O(n)$
t4p_p	>	0.06	0.04	$O(n)$	$O(n)$	$O(n)$
t4p_n	>	0.13	0.10	$O(n)$	$O(n)$	$O(n)$

TABLE 5. Growth for DLP for **K**

can be solved in under one second. Allowing for larger values of the parameter would not make these tests effective. In fact, some of the problems are completely or almost-completely solved in the input normalisation phases of the systems. This is shown in Table 5, which gives (for DLP) the maximum total time and time for the search component, as well as the growth with respect to the parameter in search time, backtracks, and number of modal successors visited.[5] As can be seen, most of the tests no longer have exponential growth in the parameter. In fact, many of them have linear growth or even no growth at all.

The tests for **K** that remain hard for both *SAT and DLP are *branch_n* and *ph_p*. The first of these consists of the Halpern and Moses branching formulae [Halpern & Moses1992], which have an exponentially-large counter-model but no disjunction. The time taken to build this counter-model is what makes these formulae difficult, and systems that try to store the entire model at once will find these formulae even more difficult. The second is an instance of the Pigeon-Hole principle [Pellettier1986], which has hard propositional reasoning but essentially no modal reasoning.

4.4 Discussion

The Heuerding and Schwendimann benchmarks were designed to meet many of the criteria that we deem important. Both the formulae and their generator were published, along with the rationale behind the formulae, so the tests can be reproduced and extended. The formula classes have a built-in balance between valid and invalid formulae. The test methodology

[5] Some of the growth orders are only approximate because of the limited data.

is not computationally intensive and does provide data that can be easily summarised. The size parameter was designed to allow for a monotonic, exponential increase in the difficulty of the formulae. The formula classes attempt to cover a considerable variety of formula structures, thus providing a measure of significance to the test. The formulae are large, but not excessively so. The formulae are artificial, in that they do not correspond to any particular application, but were supposed to be representative of general formulae.

However, with the advent of heavily-optimised provers, many of the formula classes have become too easy under the initial parameter cut-off. Most of these easy formula classes have become in some sense trivial, with input normalisation of the formulae doing most or all of the work. This "triviality" has two effects. First, increasing the parameter will not significantly increase the difficulty of a formula, at least until the formulae become gigantic. Second, these classes no longer test the comparative performance of the search mechanisms, but instead only show that of a particular normalisation. Thus these formula classes, although historically interesting, are no longer good tests for state-of-the-art modal decision procedures, which all employ sufficient input normalisation to render these formula classes "trivial".

An augmented set of formula classes could be created to try to solve the problems detailed above, but it is difficult to devise by hand formulae that are resistant to the various input normalisations of the heavily-optimised systems, and it may prove impossible to devise formulae that will resist new and as yet unknown optimisations.

5 The 3CNF$_{\Box_m}$ Random Tests

Random formulae can be generated that do not have the problems of the Heuerding and Schwendimann formulae. The first random generation technique used in testing modal decision procedures, the random 3CNF$_{\Box_m}$ test methodology, was proposed by Giunchiglia & Sebastiani [1996a; 1996c]. It was conceived as a generalisation of the 3SAT test method which is widely used in propositional satisfiability [Mitchell *et al.*1992]. The method was subsequently criticised and improved by Hustadt & Schmidt [1997; 1999], who pointed out some major weaknesses of the method, and proposed some solutions. Finally Giunchiglia *et al.* [1997; 1998a] proposed the current version of the method, which embeds Hustadt & Schmidt's suggestions, plus some further improvements. (If not otherwise stated, from now on by "random 3CNF$_{\Box_m}$ formulae" we implicitly mean the formulae generated with the final version of the 3CNF$_{\Box_m}$ random generator algorithm described in [Giunchiglia *et al.*1998a].)

5.1 Description

In the 3CNF$_{\Box_m}$ test methodology, the performance of a system is evaluated on sets of randomly generated 3CNF$_{\Box_m}$ formulae. A CNF$_{\Box_m}$ formula is a conjunction of CNF$_{\Box_m}$ clauses, where each clause is a disjunction of either propositional or modal literals. A literal is either an atom or its negation. Modal atoms are formulae of the form $\Box_i C$, where C is a CNF$_{\Box_m}$ clause. For normal modal logics there is no loss in the restriction to CNF$_{\Box_m}$ formulae, as there is an equivalence between arbitrary normal modal formulae and CNF$_{\Box_m}$ formulae. [6]

A 3CNF$_{\Box_m}$ formula is a CNF$_{\Box_m}$ formula where all clauses have exactly 3 literals. The definition extends trivially to K-CNF$_{\Box_m}$ formulae, for any positive integer K. Again, there is no loss in restricting attention to 3CNF$_{\Box_m}$ formulae, as there is a satisfiability-preserving

[6] The conversion works recursively on the depth of the formula, from the leaves to the root, each time applying to sub-formulae the propositional CNF conversion and the transformation $\Box_r \bigwedge_j \bigvee_i \varphi_{ij} \Longrightarrow \bigwedge_j \Box_r \bigvee_i \varphi_{ij}$.

way of converting any modal formula into $3\mathrm{CNF}_{\Box_m}$.

In the $3\mathrm{CNF}_{\Box_m}$ test methodology, a $3\mathrm{CNF}_{\Box_m}$ formula is randomly generated according to the following parameters:

- the (maximum) modal depth d;
- the number of clauses L;
- the number of propositional variables N;
- the number of distinct box symbols m;
- the probability p of an atom occurring in a clause at depth $< d$ being purely propositional.

Notice that d represents the *maximum* depth of a $3\mathrm{CNF}_{\Box_m}$ formula: e.g., if $p = 1$, then the depth is 0, no matter the value of d.

The random $3\mathrm{CNF}_{\Box_m}$ generator works as follows:

- a $3\mathrm{CNF}_{\Box_m}$ formula of depth d is produced by randomly, and independantly,[7] generating L $3\mathrm{CNF}_{\Box_m}$ clauses of depth d, and forming their conjunction;
- a $3\mathrm{CNF}_{\Box_m}$ clause of depth d is produced by randomly generating three distinct, under commutativity of disjunction, $3\mathrm{CNF}_{\Box_m}$ atoms of depth d, negating each of them with probability 0.5, and forming their disjunction;
- a propositional atom is produced by picking randomly an element of $\{A_1, \ldots, A_N\}$;
- a $3\mathrm{CNF}_{\Box_m}$ atom of depth $d > 0$ is produced by generating a random propositional atom with probability p; and with probability $1-p$, a $3\mathrm{CNF}_{\Box_m}$ atom $\Box_r C$, where $\Box_r$ is picked randomly in $\{\Box_1, \ldots, \Box_m\}$ and C is a randomly generated $3\mathrm{CNF}_{\Box_m}$ clause of depth $d-1$.

As there are no repetitions inside a clause, the number $D(d, N)$ of possible distinct, under commutativity of disjunction,[8] $3\mathrm{CNF}_{\Box_m}$, with $m = 1$, atoms of depth d is given by the recursive equation

$$\begin{aligned} D(0,N) &= N \\ D(d,N) &= 2^3 \cdot \binom{D(d-1,N)}{3} \quad [+D(0,N) \text{ if } p \neq 0], \end{aligned} \tag{5.1}$$

That is, $D(d, N)$ grows approximately as $(2N)^{(3^d)}$.

A typical problem set is characterised by a fixed N, m, d and p: L is varied in such a way as to empirically cover the "100% satisfiable—100% unsatisfiable" transition. Then, for each tuple of the five values in a problem set, a certain number (100, 500, 1000, ...) of $3\mathrm{CNF}_{\Box_m}$ formulae are randomly generated, and the resulting formulae are given in input to the procedure under test, with a maximum time bound of, typically, 1000 seconds. The fraction of satisfiabile formulae, median/percentile values of CPU times, and median/percentile values of other parameters, e.g., number of steps, memory, etc., are plotted against the number of clauses L.

To save testing time, if significantly more than 50% of the samples seen so far exceed the time bound for a given value of L—so that the median and the other Qth percentiles for $Q \geq 50$ are very likely to exceed the bound—then the system is not run on the other samples, and the test goes on with the next L value.

---

[7] This means that clauses may be repeated in a formula.

[8] We consider two formulae that differ only in the order of disjuncts in embedded disjunctions to be the same formulae, that is $A \vee B \vee C$ is the same as $B \vee C \vee A$.

5.2 Parameters and features

In the random $3\mathrm{CNF}_{\Box_m}$ test method, each of the five parameters plays a specific role.

d represents the maximum depth of the Kripke models for the $3\mathrm{CNF}_{\Box_m}$ formulae, and thus increases exponentially the size of the potential Kripke models [Halpern1995]. Thus d allows for increasing at will the difficulty of the solution, particularly for the the modal component of reasoning. The size of generated formulae grows on average as $(3-3p)^d$.

N defines the number of propositional variables which are to be assigned within each Kripke state, and thus enlarges exponentially the space of the possible models for φ. (Typically N is very small with respect to 3SAT problems, as each propositional atom has an "implicit multiplicity" equal to the number of states of a potential Kripke model.) Thus N allows for increasing at will the difficulty of the solution, particularly for the propositional component of reasoning. N has no effect on the size of generated formulae.

L is the number of conjuncts in the formula, and thus it controls the *constrainedness* of the formula (see, e.g., [Williams & Hogg1994; Gent *et al.*1996]): the bigger L, the more likely unsatisfiable the formula. Increasing L, we pass with continuity from an initial 100% satisfiability fraction to 100% unsatisfiability. Tuning L allows for balancing the satisfiable vs. unsatisfiable ratio. Obviously the size of generated formulae grows as $O(L)$.

m represents the number of distinct accessibility relations in the potential models. In $\mathbf{K_{(m)}}$ m partitions each branch in the search tree into m independent sub-branches, each restricted to a single $\Box_r$. Therefore, for larger m the search space is more partitioned and the problem should be easier; moreover, as there is no mutual dependency between the modal satisfiability of the distinct sub-branches, for larger m the formulae are less constrained and more likely to be satisfiable. Thus m affects both difficulty and constrainedness. m has no effect on the size of generated formulae.

p represents the "propositional vs. modal" rate for the atoms. p has been introduced to unbalance the tree-structure of the random $3\mathrm{CNF}_{\Box_m}$ formulae, and allows for distributing the propositional atoms at the different depth levels. Increasing p reduces the number of modal atoms, and thus reduces the difficulty of the solution, particularly for the the modal component of reasoning. Increasing p causes a reduction in size of the formula, as the average size of formulae is $O((3-3p)^d)$.

In a $3\mathrm{CNF}_{\Box_m}$ test session the values of the parameters should be chosen as follows. First, set the values for d, m and p so as to target an area of the input space; then set N to fix the desired level of difficulty; finally, run tests for increasing values of L, so as to cover the whole transition from 100% satisfiability to 0% satisfiability.

Consider the plots in Figure 1, presenting a test for KSATC from [Giunchiglia *et al.*1998a], with d=1, N=6, p=0, m=1, and 100 samples/point. Figure 1 (left) represents the fraction of satisfiabile formulae together with the median number of recursive calls of KSATC, which measures the size of the space searched. Figure 1 (right) represents the Qth percentile CPU time. Both plots exhibit the typical easy-hard-easy pattern centred around the 50% crossover point—also called "phase transition"—which has been revealed in many NP-complete problems (see, e.g., [Mitchell *et al.*1992; Williams & Hogg1994]). This should not be a surprise, as satisfiability in $\mathbf{K_{(m)}}$ and in most modal logics is NP-complete if the modal depth is bounded [Halpern1995], which is the case of each single plot. Thus, generally speaking,

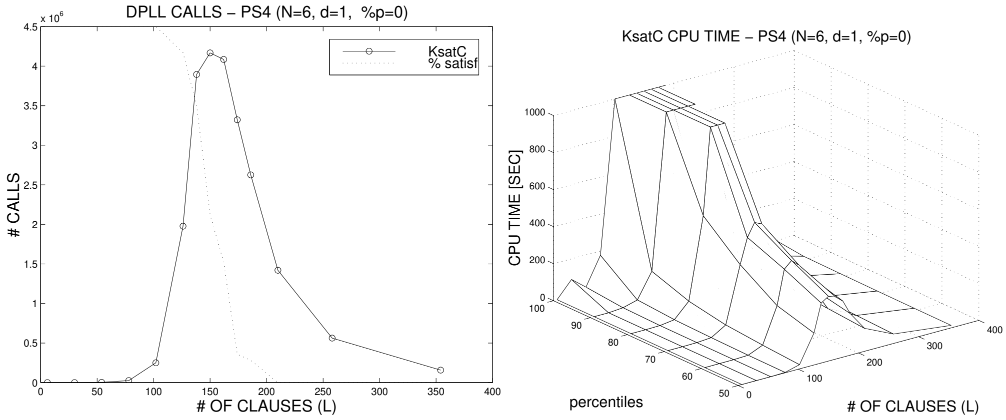

FIG. 1: KSATC on a 3CNF$_{\Box_m}$ test set (d=1, N=6, p=0, m=1, 100 samples/point). (Left): satisfiability fraction and median # of calls; (Right): Qth percentile CPU time.

by tuning N and L it is possible to generate very hard 3CNF$_{\Box_m}$ problems with the maximum uncertainty on the results.

It might sound counter-intuitive that, after the 50% cross-over point, the difficulty of the problem decreases with the size of the formula. Giunchiglia & Sebastiani [1996b] noticed that this is due to the capability of a system to backtrack at any constraint violation: the larger the value of L, the more constraint violations are detected, and the higher the search tree is pruned. As a side-effect, if a plot does not decrease after the 50% cross-over point, then this may reveal a problem of constraint violation detection in the system.

5.3 Problems

As highlighted by Hustadt & Schmidt [1997; 1999], the formulae generated by the first version of the method suffered from a couple of major drawbacks:

(Propositional) redundancy: As the intial 3CNF$_{\Box_m}$ generator did not check for repeated propositional variables inside the same clause, the random formulae generated could contain propositional tautologies. When this was the case, the size of formulae could be reduced (in some cases dramatically) by a propositional simplification step;

Trivial (un)satisfiability: For certain values of the generator's parameters, the formulae generated were "trivially (un)satisfiable", that is, they could be solved at the top level by purely propositional inference. (This definition is slightly different from the one in Hustadt and Schmidt, as explained below.)

As a solution, Hustadt & Schmidt [1999] proposed three guidelines for generating more challenging problems:

(i) Set to their smallest possible value those parameters that do not significantly influence the difficulty of the 3CNF$_{\Box_m}$ formulae. They included among them the parameters m and d, which they suggested setting to 1.

(ii) To avoid trivially (un)satisfiable formulae, set $p = 0$.

(iii) Modify the random generator so as to prevent repeated propositional variables inside the same clause.

They also improved the data analysis considering not just median values as in [Giunchiglia & Sebastiani1996a], but a range of Qth percentile values.

Redundancy was a problem due to a flaw in the first version of Giunchiglia & Sebastiani's generator, and it was easily solved. The solution (iii) completely solved propositional redundancy. Giunchiglia *et al.* [1998a] later noticed that an extra component of redundancy was due to the presence inside one clause of repeated *modal* atoms and of permutations of the same modal atom. Thus, they introduced the following variation of (iii):

(iii') Modify the random generator so as to create distinct atoms, under commutativity of disjunction, both propositional and modal, inside the same clause.

Trivial (un)satisfiability is a much more complex problem, and deserves some more discussion.[9]

5.3.1 Trivial satisfiability

A $3\mathrm{CNF}_{\Box_m}$ formula φ is trivially satisfiable iff it has at least one $\neg\Box$-free satisfying assignment. (Hustadt and Schmidt's definition is in terms of being satisfiable on a Kripke model with one world, which works out the same.) The existence of one $\neg\Box$-only clause is a sufficient condition for avoiding trivial satisfiability, as every assignment then contains at least one $\neg\Box$ literal. Each top-level literal is $\neg\Box$ with probability $(1-p)/2$; each top-level clause is $\neg\Box$-only with probability $(1-p)^3/8$; the probability of having no such clauses is thus $(1-(1-p)^3/8)^L$. As the set of trivially satisfiable formulas is a subset of the set of formulas having no $\neg\Box$-only clause, the probability of being trivially satisfiable is bounded above by a function that converges exponentially to 0 as L goes to infinity. This matches the empirical behaviour revealed in [Hustadt & Schmidt1999], where the fraction of trivially satisfiable formulae decays exponentially with L.

Trivial satisfiability is not a big problem for random $3\mathrm{CNF}_{\Box_m}$ testbeds. First, a trivially satisfiable formula is not necessarily trivial to solve. In fact, in the general case, a literal l occurring in a model μ for φ, has the same probability of being positive or negative; thus μ is $\neg\Box$-free with probability 2^{-w}, w denoting the number of modal literals in μ. On average, only a very small percentage of satisfying assignments are $\neg\Box$-free, and unless the procedure is explicitly biased to detect trivial satisfiability there is no reason to assume that the first assignment found by the procedure will be one of them. Biasing the procedure for detecting trivial satisfiability may not be a good strategy in the general case. (For instance, the suggested default settings of the *SAT system are not the best ones to detect trivial satisfiability.) In practice, most current state-of-the-art procedures are not guaranteed to solve trivially satisfiable formulae without any modal reasoning. Second, due to the exponential decrease in the fraction of trivially satisfiable formulae with L, the effects of trivial satisfiability are limited to the extreme left part of the satisfiability plots—where problems are easy and satisfiable anyway—and typically are negligible in the satisfiability transition area—which is the really interesting zone. Moreover, notice that the $(1-(1-p)^3/8)^L$ bound is pessimistic, as typically the fraction decreases much faster. This is due to the fact that for larger values of L, it is harder to get $\neg\Box$-free assignments, even if no $\neg\Box$-only clause occurs in the formula; for

[9] Some discussion about trivial solvability—although to a lower level of detail—can be found in [Giunchiglia *et al.*1998b].

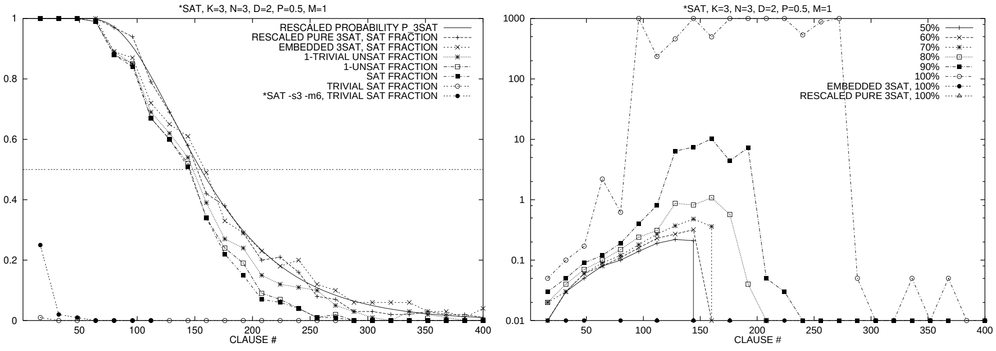

FIG. 2: *SAT on a $3\text{CNF}_{\Box_m}$ test set, (d=2, N=3, p=0.5, m=1, 100 samples/point). (Left): (un)satisfiability rates; (Right): Qth percentile CPU time (seconds), log scale.

instance, the presence of a clause like $(\neg\Box\varphi_1 \vee \neg\Box\varphi_2 \vee \neg A_1)$ forces all assignments which include A_1 to include at least one $\neg\Box$ literal.

Consider for example the plots in Figure 2 (left), which results from running *SAT with its default settings on a $3\text{CNF}_{\Box_m}$ testbed with d=2, N=3, p=0.5, m=1, 100 samples/point. In this test, *SAT found only 3 trivially satisfiable formulae for $L = 16$, and none elsewhere. Even rerunning the testbed with the *SAT option `-s3 -m6` ("while branching, choose always positive values first"), which will find more $\neg\Box$-free assignments, we obtain only 25, 2 and 1 trivially satisfiable formulae for $L = 16, 24, 32$ respectively, and none elsewhere.

5.3.2 Trivial unsatisfiability

A random $3\text{CNF}_{\Box_m}$ formula φ contains on average Lp^3 clauses that contain three propositional literals (and thus contain no boxes). The larger the value of L, the more likely it is that these clauses will be jointly unsatisfiable—thus making φ trivially unsatisfiable. (The definition of trivial unsatisfiability by Hustadt and Schmidt is precisely this, but our definition also allows unsatisfiability resulting from complementary modal literals in top-level clauses.) This explains the large number of trivially unsatisfiable formulae detected empirically by Hustadt and Schmidt [1999] in the centre and right of their plots. If $p = 0$, then φ contains no such purely propositional clauses so the problem is eliminated, at least so far as Hustadt and Schmidt's definition is concerned.

When $p > 0$, however, trivial unsatisfiability becomes a serious problem in $3\text{CNF}_{\Box_m}$ testbeds. First, a trivially unsatisfiable formula is typically exceedingly easy to solve for most state-of-the-art procedures because any reasonable $3\text{CNF}_{\Box_m}$ test set for these procedures will have only a very small number of propositional variables. In fact most procedures look for modal successors only after having found a satisfying assignment: if no such assignment exists, the formula is solved without performing any modal reasoning. Second, when p is significantly greater than zero, trivially unsatisfiable formulae may affect all the values beyond the 100% satisfiable range, including the satisfiability transition area. For instance, when $p = 0.5$, it is difficult to generate unsatisfiable $3\text{CNF}_{\Box_m}$ formulae that are not trivially unsatisfiable. A "signature" of the heavy presence of trivial unsatisfiability is what we call a "Half-Dome plot"[10] for median CPU times: the median value grows until L reaches the 50% satisfiability crossover point, then it falls abruptly down to (nearly) zero, and does not increase significantly thereafter. Analogously, the Qth percentile value grows until L reaches the (100-Q)% satisfiability crossover point, and then falls abruptly.

For example, consider Figure 2. In this figure, together with the 3CNF$_{\Box_m}$ plots, we also present the results of running $*$SAT on the conjunction of the purely propositional clauses of the formulae. We call these propositional conjunctions the embedded 3SAT sub-formulae. Moreover, as a comparison, we also run $*$SAT on a pure 3SAT testbed with $N = 3$, rescaled horizontally by a $1/p^3$ factor. (E.g., $L = 144$ in the plot means that the pure 3SAT formulas have $144 \cdot 0.5^3 = 18$ clauses.) We call these formulas the rescaled pure 3SAT formulas. As $N = 3$ means that we have only 8 distinct pure 3SAT clauses, it is possible to evaluate the SAT probability exactly, which is given by the following equation:

$$[RESCALED]\ P_{3SAT}(L) = \sum_{k=1}^{8} \binom{8}{k} (-1)^{8-k} (k/8)^{L \cdot p^3}. \tag{5.2}$$

In Figure 2 (left) we plot the fraction of satisfiable 3CNF$_{\Box_m}$ formulae, 1 minus the fraction of unsatisfiable 3CNF$_{\Box_m}$ formulae, and 1 minus the fraction of trivially unsatisfiable 3CNF$_{\Box_m}$ formulae. For the embedded 3SAT formulae and the rescaled pure 3SAT formulae, we plot only the fraction of satisfiable formulae, which in both cases is identical to 1 minus the fraction of unsatisfiable formulae. We also plotted the (rescaled) 3SAT probability of Equation 5.2. Needless to say, the two latter plots match apart for some noise. Finally, we plotted the fraction of formulae found to be trivially satisfiable by $*$SAT and by $*$SAT `-s3 -m6`, as discussed above. (Considering the well-known satisfiability transition results [Mitchell *et al.*1992; Crawford & Auton1993], for the rescaled pure 3SAT plot one might expect an average of 50% unsatisfiable formulae for $L \approx 4.28 \cdot N/p^3 = 102.72$; however, $N = 3$ is too small for applying these results, whilst it is possible to provide an exact calculation as in Equation 5.2.) Figure 2 (right) presents run times for the tests.

Several conclusions are evident from this data. Firstly, the fraction of satisfiable 3CNF$_{\Box_m}$ formulae nearly coincides with 1 minus the fraction of unsatisfiable 3CNF$_{\Box_m}$ formulae that is, very few tests exceeded the bound. Secondly, the fraction of unsatisfiable 3CNF$_{\Box_m}$ formulae and the fraction of trivially unsatisfiable 3CNF$_{\Box_m}$ formulae are very near, that is, most unsatisfiable formulae are also trivially unsatisfiable. (In our testing experience, unsatisfiable 3CNF$_{\Box_m}$ formulae that are not trivially unsatisfiable are rare in testbeds with $p = 0.5$.) Thirdly, the fraction of unsatisfiable embedded 3SAT formulae is very close to the fraction of trivially unsatisfiable 3CNF$_{\Box_m}$ formulae, that is, most trivially unsatisfiable formulae are such because the embedded 3SAT component is unsatisfiable. Fourthly, the fraction of unsatisfiable embedded 3SAT formulae is very close to the fraction of unsatisfiable rescaled pure 3SAT formulae.

In Figure 2 (right), the plots for the embedded 3SAT and for the rescaled pure 3SAT testbeds cannot be distinguished from the X axis—that is, CPU times are always smaller than 0.01 seconds. The median CPU time plot for the 3CNF$_{\Box_m}$ testbed grows until L reaches the 50% satisfiability crossover point, where it falls down abruptly; the remainder of the plot cannot be distinguished from the X axis. The Qth percentile values behave analogously with respect to the (100-Q)% satisfiability crossover point.

This behaviour can be explained as follows. If we plot the fraction of satisfiable formulae

[10] The name refers to the shape of a famous mountain in Yosemite, California.

for the embedded 3SAT formulae, we obtain a satisfiability transition similar to the one of the rescaled pure 3SAT, with a 50%-satisfiable crossover point for $L \approx 150$. The slight differences between the two plots are due to the fact that the length of the embedded 3SAT formulae is not fixed with L, as $p^3 \cdot L$ is only an average value. As N is generally rather small, the embedded 3SAT formula is mostly unsatisfiable with a very low value of L, making the entire formula trivially unsatisfiable. Thus the embedded 3SAT transition dominates the whole satisfiability plot. The first effect is that the transition is expected to nearly coincide with the embedded 3SAT transition. The second effect is that nearly all unsatisfiable formulae are trivially solvable. This means that the CPU times are entirely dominated by the values required to solve satisfiable formulae, which typically grow with L. Immediately after the (100-Q)% satisfiability crossover point, the easiest Q% of samples are nearly all trivially unsatisfiable, so that the Qth percentile value falls down abruptly to a negligible value.

The small difference between the fraction of the embedded 3SAT formulae that are unsatisfiable and the fraction of the $3\text{CNF}_{\Box_m}$ formulae that are trivially unsatisfiable can be explained as follows. When the embedded 3SAT formula is "nearly unsatisfiable", that is, it has only one or very few models, some other clauses may contribute to cause trivial unsatisfiability. For instance, consider $\varphi = \varphi_1 \wedge (C_1 \vee \Box\psi) \wedge (C_2 \vee \neg\Box\psi)$, where C_1 and C_2 are propositional sub-clauses, and the embedded 3SAT formula, φ^*, is "nearly unsatisfiable". If the few models of φ^* each violate both C_1 and C_2, then the only valid propositional assignment is $\varphi^* \wedge (\Box\psi) \wedge (\neg\Box\psi)$. If $\Box\psi$ is treated as an atom (which is the case in most optimised systems), then φ is trivially unsatisfiable, even though φ^* is satisfiable. With $p = 0.5$, an average of $3/8$ clauses have exactly one modal literal. With $N = 3$ and $d = 2$, the probability that two such clauses have mutually contradictory modal literals $\Box\psi$ and $\neg\Box\psi$ is not negligible. As before, with $p = 0$ no "nearly unsatisfiable" embedded 3SAT formula occurs.

5.3.3 The $p = 0,\ d = 1,\ m = 1$ solution

Unfortunately the guidelines indicated by Hustadt & Schmidt [1999], and described above, are not a panacea, as they introduce new problems.

Consider the $d = 1$ guideline. First, in $\mathbf{K_{(m)}}$ the $3\text{CNF}_{\Box_m}$ class represents only the class of formulae of depth d, as there is no way to reduce the depth of formulae. Therefore, if $d = 1$ the input subspace sampled is not very representative. Moreover, as shown in [Halpern1995], a formula φ which is satisfiable in $\mathbf{K_{(m)}}$ (and also in $\mathbf{KT_{(m)}}$, $\mathbf{K}45_{\mathbf{(m)}}$, $\mathbf{KD}45_{\mathbf{(m)}}$ and $\mathbf{S5_{(m)}}$) has a tree-like Kripke model whose number of states is smaller than $|\varphi|^{depth(\varphi)}$, where $|\varphi|$ and $depth(\varphi)$ are respectively the size and the modal depth of φ. As a consequence, satisfiable $3\text{CNF}_{\Box_m}$ formulae with $d = 1$ have very small models, so that they are not very challenging from the viewpoint of pure modal reasoning, regardless of the values chosen for the other parameters. More generally, when bounding the modal depth, the satisfiability problems for the logics above decays from PSPACE-complete to NP-complete [Halpern1995].

Consider the $p = 0$ guideline. If $p = 0$, then the random $3\text{CNF}_{\Box_m}$ formulae are complete ternary trees where propositional atoms occur only at the maximum depth level. Such formulae can hardly be considered as a representative sample of the modal input space. Moreover, they are even less representative if used as a testbed for most modal logics different from $\mathbf{K}$. In fact, in most modal logics, restricting the occurrence of propositional variables to the maximum depth level hinders a relevant source of reasoning due to the interaction between variables occurring at different depth levels. For instance, the assignment $\{A_1, \Box_1\varphi\}$ is sat-

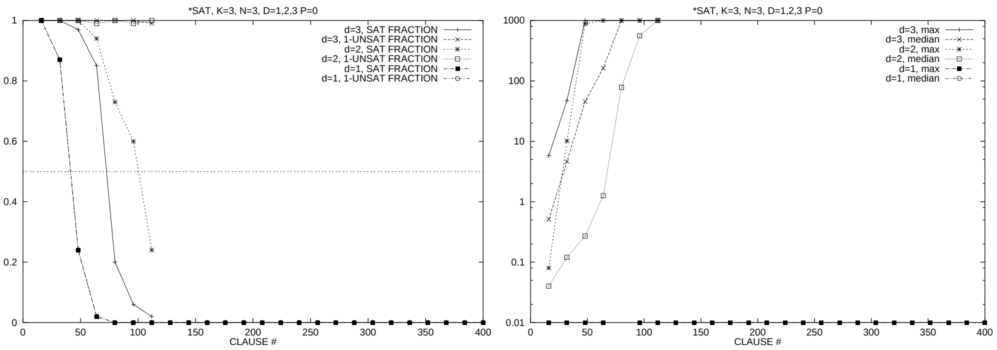

FIG. 3: A 3CNF$_{\Box_m}$ random test set (d=1,2,3, N=3, p=0, m=1, 100 samples/point). (Left): (un)satisfiability rates; (Right): median and max CPU times (seconds) for *SAT, log scale.

isfiable in $\mathbf{K_{(m)}}$ but may be not in $\mathbf{KT_{(m)}}$ if A_1 occurs in φ (e.g., if $\varphi = \neg A_1$).

Finally, consider the $d = 1$ and $p = 0$ guidelines together. In [Hustadt & Schmidt1999] the $d = 1$ statement derived from the results of an experiment with $N = 3$, $p = 0.5$, $m = 1$, $d = 2,\ 3,\ 4,\ 5$ where the complexity did not seem to increase significantly with d, and the growth was smaller than the increase in size. Unfortunately, this experiment was strongly influenced by the $p = 0.5$ choice. With $p = 0$, it turns out that the overall difficulty grows dramatically with d. Consider the plots in Figure 3, which have been obtained by running *SAT on three test sets, with $d = 1$, 2 and 3, $N = 3$, $p = 0$, m =1, and 100 samples/point. Figure 3 (left) shows both the fraction of formulae found to be satisfiabile and 1 minus the fraction of formulae found to be unsatisfiable. Figure 3 (right) shows the median and max CPU times. For $d = 1$, all formulae are either found to be satisfiabile or found to be unsatisfiable—that is, no sample exceeded the timeout—and the satisfiability fraction decreases very fast, reaching 100% unsatisfiability for $L < 100$; the median and max CPU times are so small that cannot be distinguished from the X axis. This should not be a surprise: with $N = 3$, $p = 0$ and $d = 1$, there are only 8 distinct modal atoms, so that the test bed is only a little harder than a 3SAT testbed with $N = 8 + 3$ variables. For $d = 2, 3$ things change dramatically. Both median and maximum CPU times rapidly reach the timeout. As a consequence, the fraction of formulae found to satisfiable plus the fraction of formulae found to be unsatisfiable add to much less than 1, as most problems exceed the timeout. The real satisfiability fraction is somewhere between them. For instance, with $L = 80$ more than 50% of the $d = 3$ samples have exceeded the timeout, while for $L = 120$ more than 50% of the $d = 2$ samples have exceeded the timeout. Running *SAT for $d = 2$ and $L = 400$, no sample was solved within the timeout. As a consequence, for $p = 0$ and $d > 1$, the test sets are mostly out of the reach of *SAT, and no information about the satisfiability transition can be provided. The situation is no better with other current decision procedures such as DLP. To our knowledge, no system so far has been able to fully plot the satisfiability transition for $d = 2$, $N = 3$, $p = 0$, $m = 1$, as in the satisfiability transition area most solution times exceed the timeout.

We believe that such behaviour should be expected. As for satisfiability fraction, the number of possible distinct atoms $N(d, N)$ in equation (5.1) grows exponentially with d, decreasing the probability of conflicts between modal literals. As a consequence, with L fixed, the

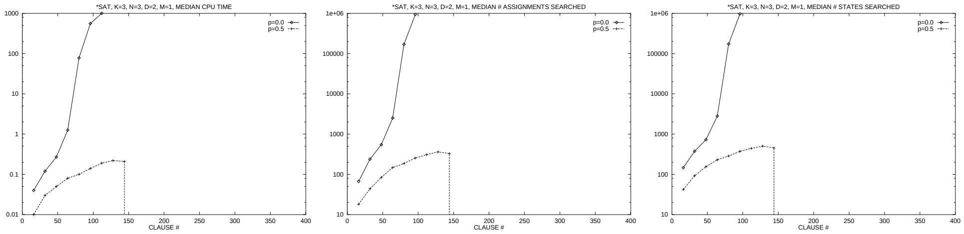

FIG. 4: Two 3CNF$_{\Box_m}$ random test set, $d = 2$, $N = 3$, $p = 0.0$ and 0.5, $m = 1$, 100 samples/point. (Left): median CPU time; (Centre): median # of assignments found; (Right): median # of states explored.

fraction of unsatisfiable formulae decreases very rapidly with d, causing a relative increase in the number of clauses L necessary to reach the 50% cross-over point, that is, a relative shift to the right of the transition area. As for difficulty, the size of the Kripke models for φ is up to $|\varphi|^{depth(\varphi)}$ [Halpern1995]. Thus, if L is fixed and d grows, it is reasonable to expect that the effort required to search for such exponentially-big models grows at least exponentially with d. Notice that, if φ is a random 3CNF$_{\Box_m}$ formula with $p = 0$, then $|\varphi|$ also grows as $O(3^d)$.

This prompts the question as to why the same behaviour is not observed with $p = 0.5$. As for satisfiability, the effects of increasing d are counteracted by the embedded 3SAT component, which forces trivial unsatisfiability with very low values of L, preventing the shifting of the satisfiability transition described above. As for difficulty, first, $|\varphi|$ is much smaller, as only about a $(1 - p)^d$ fraction of the branches of the formula tree of φ actually reach depth d. Moreover, the high percentage of propositional literals reduces dramatically the number of assignments found, as it gets much harder for most candidate assignments to avoid containing propositional contradictions like A_j, $\neg A_j$. Finally, the modal literals are only a subset of each assignment found, so that the number of states to be explored for every assignment is reduced. Figure 4 shows both the median CPU times, the median number of assignments found and the median number of states explored by $*$SAT for $d = 2$, $N = 3$, $m = 1$, with both $p = 0.0$ and 0.5. It can be seen that all three values are drastically reduced by setting $p = 0.5$.

5.4 Discussion

Even with the above problems, the random 3CNF$_{\Box_m}$ test methodology produces a good empirical test for many purposes. The generators are available and their "randomness" can be controlled by setting the seed of their random number generator to reproduce test sets if the actual formulae are not available. The rationale behind the methodology has been extensively discussed. The formulae generated, although large, are not too large, and many very difficult (relative to size) formulae are generated if appropriate parameter values are chosen. The test methodology produces a balance between satisfiable and unsatisfiable formulae. With the recent improvements, the problems of redundancy and triviality are much reduced.

The test generator is highly parameterised, perhaps too highly, but by concentrating on a subsection of the test space interesting tests can be generated. The biggest problem with the

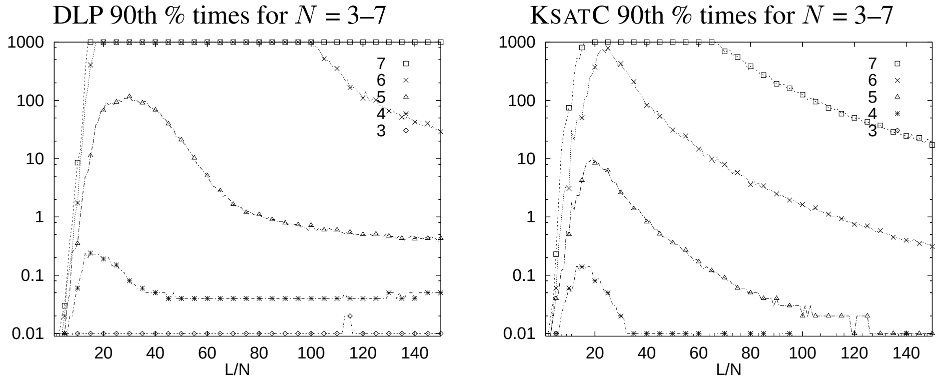

FIG. 5. Results for $m = 1$, $d = 1$, and $p = 0.0$

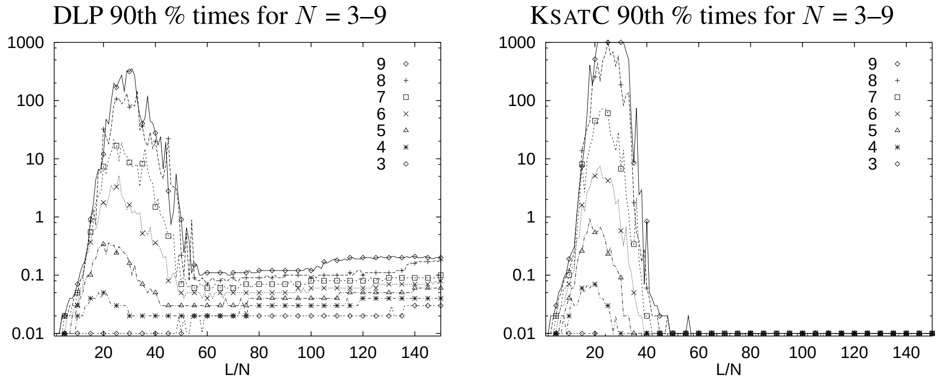

FIG. 6. Results for $m = 1$, $d = 1$, and $p = 0.5$

test methodology is that the maximum modal depth of formulae is fixed, thus reducing the inherent difficulty of the problem from PSPACE-complete to NP-complete. However, even at modal depths 1 and 2, difficult tests can easily be devised.

One disadvantage with random tests is that they take much longer to perform (for interesting hard problems) because a large number of formulae have to be generated and tested at each data point for the results to be reliable. However, probably the biggest drawback with the random $3\text{CNF}_{\Box_m}$ test methodology is that, because of their low modal depths, the formulae generated are very artificial. This is not really a problem with the test methodology *per se*, but is instead due to the combination of the test methodology and the capabilities of current decision procedures.

One benefit of the random $3\text{CNF}_{\Box_m}$ test methodology is that it can show the changing relative behaviour of several systems as the various parameters change. For example, several qualitative differences between DLP and KSATC can be discerned from the tests shown in Figures 5, 6, and 7 which give 90th percentile results for several tests.

These results illustrate a number of differences between the two decision procedures, which can be traced back to characteristics of the system. For example, KSATC uses an underlying satisfiability engine with very efficient data structures whereas DLP does not have

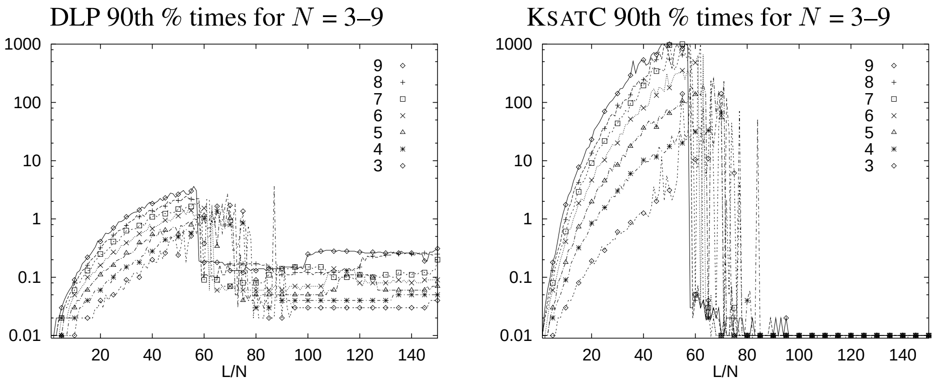

FIG. 7. Results for $m = 1$, $d = 2$ and $p = 0.5$.

as highly optimised data structures. This difference in data structures shows up in different run times for larger formulae (larger values of L/N) with non-zero p (Figures 6 and 7) where formulae are mostly trivially unsatisfiable, but where DLP takes some amount of time just to traverse its data structures.

KSATC uses an aggressive look-ahead technique that investigates modal successors very early on in the search space. This technique is good when the problems are over-constrained, resulting in better performance in particular for $d = 1$ and $p = 0$ (Figure 5), and also in narrower peaks for $d = 1$ and $p = 0.5$ (Figure 6). However, when there are significant numbers of modal successors that are satisfiable, this early investigation, and the necessary reinvestigation when more information is known, becomes a serious liability, as shown for $d = 2$ and $p = 0.5$ (Figure 7).

It is this sort of comparative analysis that is most useful to the understanding of how various algorithms behave and how they can be improved.

6 New Random Empirical Testing

Since 1998 some new forms of random testing—both variants of the $3\mathrm{CNF}_{\Box_m}$ method and completely new ones—have been investigated and/or proposed. In the following section we briefly outline and review the most significant of these.

6.1 Using modalised atoms

Recently Massacci [1999] proposed a "$\mathbf{K}$-modalised" variant of the $3\mathrm{CNF}_{\Box_m}$ method borrowing an idea from [Halpern1995]: within each $3\mathrm{CNF}_{\Box_m}$ formula φ, substitute each occurrence of each propositional variable A_i with the corresponding modal expression $\neg\Box(A_0 \vee \Box^i\neg A_0)$. (A similar encoding was proposed for $\mathbf{S4}$.) The encoding preserves satisfiability in $\mathbf{K}$, and the resulting formula φ' has only one propositional variable A_0 and depth $d + N + 1$. Moreover, φ' is relatively bigger than φ, as the global number of propositional literals is doubled and, for each propositional atom, one "$\vee$" and an average of $N/2 + 1$ "$\Box$"s are added. Thus, we would expect $\mathbf{K}$-modalised $3\mathrm{CNF}_{\Box_m}$ formulae to be much harder than their corresponding $3\mathrm{CNF}_{\Box_m}$ ones, especially for low d's and high N's.

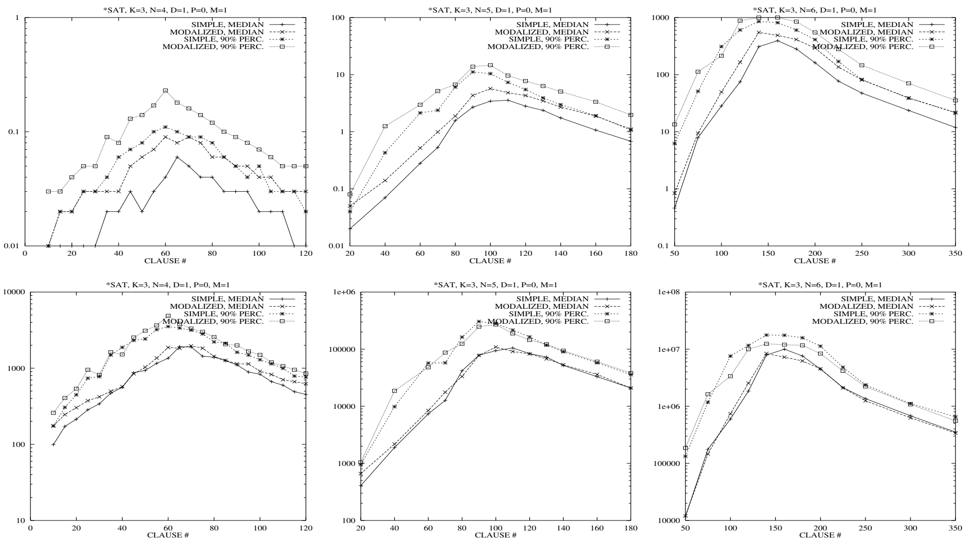

FIG. 8: *SAT on $3\mathrm{CNF}_{\Box_m}$ and $\mathbf{K}$-modalised $3\mathrm{CNF}_{\Box_m}$ test sets, (N=4,5 and 6, d=1, p=0, m=1, 100 samples/point). (Top): median and 90% percentile CPU time (seconds). (Bottom): median and 90% size of the space searched.

Figure 8 shows the results of running *SAT on the testbeds in [Hustadt & Schmidt1999; Giunchiglia *et al.*1998a]—i.e., with d=1, p=0, m=1, N=4,5,6 and 100 samples/point—using both $3\mathrm{CNF}_{\Box_m}$ formulae and their $\mathbf{K}$-modalised counterparts.The top row represents median and 90% percentile CPU time; the bottom row represents median and 90% percentile size of the space searched, that is, the number of single truth-value assignments performed by *SAT. All curves present the usual easy-hard-easy pattern. From the first row, we notice that the $\mathbf{K}$-modalised samples in general require a longer CPU time to solve. Nevertheless, the gap never exceeds a 2-3 factor, which is well justified by the increase in size of the input formulae. (Similarly, the CPU time gap between the $\mathbf{K}$-modalised and non-modalised $3\mathrm{CNF}_{\Box_m}$ tests presented in [Massacci1999] never exceeds a 2-3 factor.) Moreover, from the second row, we notice that there is no significant difference in the size of the space effectively explored.

These results may be explained as follows. As far as basic propositional reasoning is concerned, modalisation introduces no difference, as *SAT considers boxed formulae as propositional atoms. In the simple $3\mathrm{CNF}_{\Box_m}$ case, *SAT takes one shot to determine the satisfiability of a purely propositional assignment like

$$\{A_{i_1}, ..., A_{i_n}, \neg A_{j_1}, ..., \neg A_{j_m}\}. \tag{6.1}$$

In the $\mathbf{K}$-modalised case, the assignment corresponding to (6.1) is

$$\{\neg\Box(A_0 \vee \Box^{i_1}\neg A_0), ..., \neg\Box(A_0 \vee \Box^{i_n}\neg A_0), \Box(A_0 \vee \Box^{j_1}\neg A_0), ..., \Box(A_0 \vee \Box^{j_m}\neg A_0)\}. \tag{6.2}$$

Although (6.2) is satisfiable in $\mathbf{K}$ [Halpern1995], checking its satisfiability requires determining the satisfiability of the sub-formulae

$$\neg A_0 \wedge \neg\Box^i\neg A_0 \wedge (A_0 \vee \Box^{j_1}\neg A_0) \wedge ... \wedge (A_0 \vee \Box^{j_m}\neg A_0), \tag{6.3}$$

for every negated boxed atom $\neg\Box(A_0 \vee \Box^i \neg A_0)$ in (6.2). However, this step is deterministic. First, all the A_0 disjuncts are wiped off by unit-propagating the first $\neg A_0$:

$$\neg\Box^i \neg A_0 \wedge \Box^{j_1} \neg A_0 \wedge \ldots \wedge \Box^{j_m} \neg A_0. \tag{6.4}$$

As (6.4) and all its modal successors contain only one negated box, *SAT solves (6.4) deterministically by generating a linear concatenation of i states, without performing any search within each of them. An analogous behaviour is to be expected, e.g., from DLP and FaCT.

As far as we can see from the experiments, despite the relative increase in depth and size of the formulae, **K**-modalisation does not seem to produce testbeds which are significantly more challenging than standard 3CNF$_{\Box_m}$ ones. For instance, in the tests in Figure 8, modalisation simply introduces an overhead due to an extra amount of deterministic steps, without increasing significantly the size of the search space.

6.2 Re-interpreting p: the New_3CNF$_{\Box_m}$ test method

To overcome the problems of the 3CNF$_{\Box_m}$ generator due to the embedded SAT component φ^*, we propose a new variant of the random generator of [Giunchiglia *et al.*1998a], called New_3CNF$_{\Box_m}$. The difference relies on a different interpretation of the p parameter. In the 3CNF$_{\Box_m}$ generator, p is interpreted as "*the probability of an atom being propositional*". In the New_3CNF$_{\Box_m}$ generator, p is interpreted as "*the proportion of propositional atoms in a clause*", in the sense that

- if $p = k/3,\ k \in \{0, 1, 2, 3\}$, then the proportion is interpreted in the obvious way, that is, "*exactly k propositional literals and* $3 - k$ *modal literals*".
- otherwise, the residual part is interpreted as a probability, that is, "*exactly* $\lfloor 3p \rfloor$ *propositional literals,* $3 - \lceil 3p \rceil$ *modal literals, and the last literal is propositional with probability* $3p - \lfloor 3p \rfloor$", where $\lfloor x \rfloor =_{def} max\{n \in \mathcal{N} | n \leq x\}$ and $\lceil x \rceil =_{def} min\{n \in \mathcal{N} | n \geq x\}$.

The first case is a sub-case of the second: if $p = k/3$, then $\lfloor 3p \rfloor = \lceil 3p \rceil = 3p = k$. The definition trivially extends to K-CNF$_{\Box_m}$ formulae by substituting 3 with K. A random New_3CNF$_{\Box_m}$ clause is thus generated in the following way (and then sorted):

1. generate randomly $\lfloor 3p \rfloor$ distinct propositional literals;
2. generate randomly $3 - \lceil 3p \rceil$ distinct New_3CNF$_{\Box_m}$ literals;
3. flip a coin: with probability $3p - \lfloor 3p \rfloor$, generate randomly a fresh propositional literal; otherwise, generate randomly a fresh New_3CNF$_{\Box_m}$ literal ("fresh" here means "not already present in the clause").

For instance, if $p = 1/3$, then the clause contains 1 propositional and 2 modal literals; if $p = 0.5$, then it contains 1 propositional and 1 modal literal, and the other is propositional with probability 0.5; if $p = 0.6$, then it contains 1 propositional and 1 modal literals, and the other is propositional with probability 0.8, as $3 \cdot 0.6 - \lfloor 3 \cdot 0.6 \rfloor = 1.8 - 1 = 0.8$.

As with the 3CNF$_{\Box_m}$ case, a New_3CNF$_{\Box_m}$ clause contains an average of $3p$ propositional literals. However, if $p < 2/3$, then no purely propositional clause can be generated. This prevents a random New_3CNF$_{\Box_m}$ formula φ from containing any embedded 3SAT sub-formulae φ^*, and thus eliminates the main source of trivial unsatisfiability while preserving the benefits of setting $p > 0$.

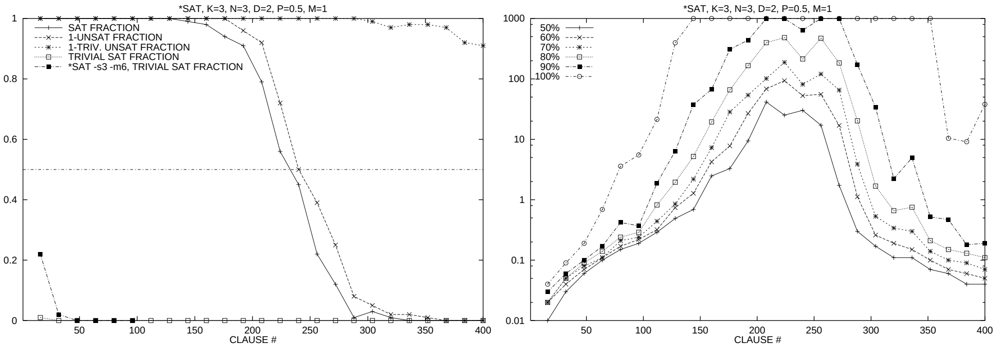

FIG. 9: *SAT on a New_3CNF$_{\Box_m}$ test set, (d=2, N=3, p=0.5, m=1, 100 samples/point). (Left): (un)satisfiability fractions; (Right): Qth percentile CPU time (seconds), log scale.

Figure 9 shows plotted the results of running *SAT on a random New_3CNF$_{\Box_m}$ test set, with $d = 2$, $N = 3$, $p = 0.5$, $m = 1$ and 100 samples/point. Figure 9 (left) shows the fraction of formulae found to be satisfiable, 1 minus the fraction of formulae found to be unsatisfiable, 1 minus the fraction of formulae found to be trivially unsatisfiable, the fraction of formulae found to be trivially satisfiable by *SAT and by *SAT `-s3 -m6`. First, the satisfiable and unsatisfiable fractions add to close to 1 but not exactly to 1, that is, a few tests exceeded the bound. Secondly, there are very few trivially unsatisfiable formulae, but there are some. The trivial unsatisfiability in these formulae is caused by complementary modal atoms in top-level clauses. Thirdly, the fraction of unsatisfiable formulae is much greater than the fraction of trivially unsatisfiable formulae, that is, very few unsatisfiable formulae are also trivially unsatisfiable, and nearly all of these are located in the 100% unsatisfiable zone. With its default settings, *SAT found only 1 trivially satisfiable formula for $L = 16$, and none elsewhere. As before, rerunning the testbed with the *SAT option `-s3 -m6` we obtained 22 and 2 trivially satisfiable formulae for $L = 16$ and 24 respectively, and none elsewhere. In Figure 9 (right), the median CPU time and the other Qth percentile plots reveal a typical easy-hard-easy pattern centred in the satisfiability transition area, growing up to the 50% cross-over point, and then decreasing gently.

Now compare the plots with those of the analogous 3CNF$_{\Box_m}$ test set in Figure 2 (the parameters' values and the system tested are the same). Firstly, the satisfiability transition here is relatively shifted to the right. The New_3CNF$_{\Box_m}$ satisfiability plot is no longer dominated by the embedded 3SAT component, and thus it is free to follow its "natural" course. Secondly, the New_3CNF$_{\Box_m}$ CPU time plots are slightly harder than the 3CNF$_{\Box_m}$ plots in the satisfiable area, becoming dramatically harder as the fraction of trivially unsatisfiable 3CNF$_{\Box_m}$ formulae increases. In fact, the effect is due to the Qth percentile values for the 3CNF$_{\Box_m}$ test set becoming dramatically lowered by the increasing percentage of trivially unsatisfiable formulae, the solution times for which are negligible. This does not happen with the New_3CNF$_{\Box_m}$ test set, where the percentage of trivially unsatisfiable formulae is negligible and hard unsatisfiable formulae are generated. Finally, the New_3CNF$_{\Box_m}$ CPU time plots describe easy-hard-easy patterns which decrease gently with L, instead of falling down abruptly. The New_3CNF$_{\Box_m}$ plots are no longer dominated by the trivially unsatisfiable formulae, and thus they are free to follow their "natural" easy-hard-easy pattern induced by the increase in the constrainedness.

On the whole, the New_3CNF$_{\Box_m}$ test method with $p \leq 2/3$ solves the problems related to trivial unsatisfiability in the 3CNF$_{\Box_m}$ method. This is done without imposing the $p = 0$ and $d = 1$ restrictions, which introduce a new problems of their own. Work is still ongoing to plot full diagrams for $d > 2$. Generating full New_3CNF$_{\Box_m}$ plots for bigger values of d and N may be a challenge for both state-of-the art and future systems.

6.3 Random QBF tests

A common criticism of 3CNF$_{\Box_m}$ testbeds comes from the consideration that for many modal logics the class of formulae with bounded depth is in NP [Halpern1995], so that 3CNF$_{\Box_m}$ testbeds with bounded d have only NP complexity [Massacci1999]. To overcome this problem, Massacci proposes a completely new kind of random empirical benchmark, which was used in the TANCS'99 system performance comparison. In this benchmark, random QBF formulae are generated according to the method described by Cadoli *et al.* [1998], and then converted into modal logic by using a variant of the conversion by Halpern & Moses [1992]. The converted modal formulae are satisfiable iff the QBF formulae are true.

Random QBF formulae are generated with alternation depth D and at most V variables at each alternation. The matrix is a random propositional CNF formula with C clauses of length K, with some constraints on the number of universally and existentially quantified variables within each clause. (This avoids the problem of trivial unsolvability for random QBF formulae highlighted by Gent & Walsh [1999a].) For instance, a random QBF formula with $D = 3$, $V = 2$ looks like:

$$\forall v_{32} v_{31}.\exists v_{22} v_{21}.\forall v_{12} v_{11}.\exists v_{02} v_{01}.\psi[v_{32}, ..., v_{01}]. \tag{6.5}$$

In this section, ψ is a random QBF formula with parameters C, V and D, while U and E denote the total number of universally and existentially quantified variables respectively. Clearly, both U and E are $O(D \cdot V)$. Moreover, φ is the modal formula resulting from Halpern & Moses' **K** conversion, so both the depth and the the number of propositional variables of φ are also $O(D \cdot V)$.

A from-scratch empirical evaluation of the random QBF test method would require an effort exceeding the proposed scope (and length) of this paper. Thus we will restrict our analysis to some basic considerations.

As with 3CNF$_{\Box_m}$, the results of each QBF-based testbed are easy to reproduce as the generator's code and all the parameters' values are publicly available. By increasing V and D the difficulty of the generated problems scales up, while C allows for tuning the sat-versus-unsat rate of the formula. Random QBF plots, with fixed D and V, also present easy-hard-easy patterns centred in the solvability transition areas [Cadoli *et al.*1998; Gent & Walsh1999a]. Moreover, with respect to 3CNF$_{\Box_m}$ formulae, random QBF formulae allow for generating 50%-solvable formulae with higher modal depth that are still within the reach of current state-of-the-art deciders.

From a purely theoretical viewpoint, it is claimed that, unlike CNF$_{\Box_m}$ formulas, modal-encoded QBF formulas can capture the problems in Σ^P_D, as QBF formulas with bounded D and unbounded V are in Σ^P_D, while CNF$_{\Box_m}$ formulas with bounded d and unbounded N are "stuck at NP" [Massacci1999]. We notice that this statement is rather misleading. Massacci treats the alternation depth D and the number of variables within each alternation V as the "QBF-equivalent" of the modal depth d and the number of propositional variables

N respectively – they are the same parameters in the random generator. We remark that the "QBF-analogous" role of modal depth is played instead by the total number of universally quantified variables $U \approx V \cdot D/2$. In fact, similarly to modal $\mathbf{K_{(m)}}$ with bounded depth, the class of random QBF formulae with bounded U is in NP, as it is possible to "guess" a tree-like witness with $O(U \cdot 2^U)$ nodes.[11] Moreover, as in the case of 3CNF$_{\Box_m}$ formulae with bounded d and N, if D and V are bounded—which is the case of every finite-size test set—then the random QBF problems are not only in NP, but even in P.

Another problem with modal-encoded QBF formulae is that they are rather artificial, as their potential Kripke structures are restricted to those having the very regular structure imposed by the QBF and/or binary search trees. As far as representativity is concerned, modal-encoded QBF formulae have a very peculiar modal structure, so that they can hardly be considered as a representative sample of the input space.

Finally, a serious problem with random modal-encoded QBF formulas is size. Initial versions of the translation method produced test sets in the 1GB range. The problem with such very large formulae is that they may be only stressing the data-storage and retrieval portion of the provers; e.g., running DLP on these formulae resulted in a 1000s timeout without any significant search. Even the current versions produce very large modal formulae, mostly to constrain the Kripke structures.

7 Discussion

The current situation in empirical testing of modal decision procedures is not completely satisfactory. There are a number of available test sets and methodologies that can be used to examine modal decision procedures, each of which has certain benefits and certain flaws.

The Heuerding and Schwendimann test suite provides a number of interesting inputs. The input formula classes are different from each other, but there is only a small number of classes and they do not cover all kinds of input formula. The input formulae are parameterised, potentially providing a good range of difficulty. However, current systems incorporate preprocessing steps that reduce many of the test formula classes to trivial formulae even before search starts, dramatically reducing the difficulty of the class.

The standard 3CNF$_{\Box_m}$ random test methodology provides a means for easily generating very hard problems. It has some problems with trivial (un)satisfiability, but these problems are not severe. Disguising the 3CNF$_{\Box_m}$ formulae by modalising the propositional atoms does not make the tests appreciably better. Our new random test methodology may help alleviate these problems, but more testing is required.

The new QBF random testing, when compared with New_3CNF$_{\Box_m}$ formulae, allows for generating 50%-solvable formulae with higher modal depth which are still within the reach of current state-of-the-art deciders. On the other hand, modal-encoded QBF formulae are rather artificial, as their potential models are restricted to those having a very regular structure.

Unfortunately, current systems can only handle very small values for some of the parameters at interesting points in the parameter space of the 3CNF$_{\Box_m}$ random test methodology. This makes it very hard to investigate the behaviour of systems over an interesting range of the parameter. Further, there are four active parameters, which makes it hard to cover large sections of the input space.

However, even with the above-mentioned flaws, current empirical test methodologies provide evidence of great strides forward in the performance of modal decision procedures. The

[11] More precisely $O(E \cdot 2^U)$ nodes, but E is $O(U)$ in the class of formulae considered.

current fastest systems, including DLP and *SAT, can quickly solve problems that were impossible to solve just a couple of years ago. The Heuerding and Schwendimann test suite shows the importance of pre-processing to remove as many redundancies as possible. The standard $3\text{CNF}_{\Box_m}$ test suites show the effect, both positive and negative, of certain strategies built into these systems.

It is interesting to compare the situation in empirical testing of modal decision procedures with that for propositional satisfiability [Gent & Walsh1993; Crawford & Auton1996; Selman *et al.*1996], where a random-3CNF methodology is also used. With propositional formulae, the methodology can be tuned to provide problems of appropriate difficulty, and captures the hardest formulae of a particular size. There are only two parameters, which allows for easy coverage of large sections of the input space. Current systems can process reasonably large inputs, but it is easy to generate formulae that are hard or even impossible to solve. Achieving a similarly satisfactory situation in the empirical testing of modal decision procedures will require advances in both the decision procedures and the testing methodology.

Acknowledgments

The third author is indebted to Fausto Giunchiglia and Marco Roveri from ITC-IRST, for providing support and feedback, and to Luciano Tubaro from the Maths Department of University of Trento, for finding out Equation 5.2.

References

[Achlioptas *et al.* 1997] Achlioptas, Kirousis, Kranakis, Krizanc, Molloy, and Stamatiou. Random constraint satisfaction: A more accurate picture. In *ICCP: International Conference on Constraint Programming (CP), LNCS*, 1997.

[Balsiger & Heuerding 1998] P. Balsiger and A. Heuerding. Comparison of theorem provers for modal logics — introduction and summary. In H. de Swart, editor, *Automated Reasoning with Analytic Tableaux and Related Methods: International Conference Tableaux'98*, number 1397 in Lecture Notes in Artificial Intelligence, pages 25–26, Berlin, May 1998. Springer-Verlag.

[Beckert & Goré 1997] B. Beckert and R. Goré. Free variable tableaux for propositional modal logics. In *Automated Reasoning with Analytic Tableaux and Related Methods: International Conference Tableaux'97*, number 1227 in Lecture Notes in Artificial Intelligence, pages 91–106, Berlin, 1997. Springer-Verlag.

[Cadoli *et al.* 1998] M. Cadoli, A. Giovanardi, and M. Schaerf. An algorithm to evaluate quantified Boolean formulae. In *Proceedings of the 15th National Conference on Artificial Intelligence (AAAI-98)*, pages 262–267. AAAI Press, 1998.

[Crawford & 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.

[Crawford & Auton 1996] J. M. Crawford and L. D. Auton. Experimental results on the crossover point in random 3-sat. *Artificial Intelligence*, 81(1-2):31–57, 1996.

[DIM 1993] DIMACS. *The Second DIMACS International Algorithm Implementation Challenge*, Rutgers University, USA, 1993.

[Gent *et al.* 1996] I. P. Gent, E. MacIntyre, P. Prosser, and T. Walsh. The constrainedness of search. In *Proceedings of the Thirteenth National Conference on Artificial Intelligence and the Eighth Innovative Applications of Artificial Intelligence Conference*, pages 246–252, Menlo Park, August 4–8 1996. AAAI Press / MIT Press.

[Gent & Walsh 1993] I. P. Gent and T. Walsh. An empirical analysis of search in GSAT. *J. of Artificial Inteligence Research*, 1:47–57, 1993.

[Gent & Walsh 1999a] I. Gent and T. Walsh. Beyond NP: the QSAT phase transition. In *Proc. of the 16th National Conference on Artificial Intelligence*. AAAI Press, 1999.

[Gent & Walsh 1999b] I. Gent and T. Walsh. CSPLib: a benchmark library for constraints. Technical report, APES-

09-1999, 1999. Available from http://csplib.cs.strath.ac.uk/. A shorter version appears in the Proceedings of the 5th International Conference on Principles and Practices of Constraint Programming (CP-99).

[Giunchiglia *et al.* 1997] F. Giunchiglia, M. Roveri, and R. Sebastiani. A new method for testing decision procedures in modal logics. In W. McCune, editor, *Proceedings of the 14th International Conference on Automated deduction*, volume 1249 of *LNAI*, pages 264–267, Berlin, July 13–17 1997. Springer.

[Giunchiglia *et al.* 1998a] E. Giunchiglia, F. Giunchiglia, R. Sebastiani, and A. Tacchella. More evaluation of decision procedures for modal logics. In A. G. Cohn, L. Schubert, and S. C. Shapiro, editors, *Principles of Knowledge Representation and Reasoning: Proceedings of the Sixth International Conference (KR'98)*, pages 626–635. Morgan Kaufmann Publishers, San Francisco, California, June 1998.

[Giunchiglia *et al.* 1998b] E. Giunchiglia, F. Giunchiglia, R. Sebastiani, and A. Tacchella. SAT vs. Translation based decision procedures for modal logics: a comparative evaluation. Technical Report 9812-65, IRST, Trento, Italy, December 1998. To appear on *Journal of Applied Non Classical Logics.*

[Giunchiglia *et al.* 1999] E. Giunchiglia, F. Giunchiglia, and A. Tacchella. SAT Based Decision Procedures for Classical Modal Logics. *Journal of Automated Reasoning. Special Issue: Satisfiability at the start of the year 2000 (SAT 2000)*, 1999. To appear.

[Giunchiglia & 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.

[Giunchiglia & Sebastiani 1996b] F. Giunchiglia and R. Sebastiani. Building decision procedures for modal logics from propositional decision procedures - the case study of modal K(m). To appear in "Information and Computation". A longer version is published as ITC-IRST techrep 9611-06, November 1996.

[Giunchiglia & 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.

[Halpern 1995] 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.

[Halpern & Moses 1992] J. 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.* 1995] A. Heuerding, G. Jäger, S. Schwendimann, and M. Seyfreid. Propositional logics on the computer. In P. Baumgartner, R. Hähnle, and J. Posegga, editors, *Automated Reasoning with Analytic Tableaux and Related Methods: International Conference Tableaux'95*, number 918 in Lecture Notes in Artificial Intelligence, pages 308–319, Berlin, 1995. Springer-Verlag.

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

[Horrocks 1998] 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.

[Horrocks & Patel-Schneider 1999a] I. Horrocks and P. F. Patel-Schneider. Generating hard modal problems for modal decision procedures. In *Proceedings of the First Methods for Modalities Workshop*, May 1999.

[Horrocks & Patel-Schneider 1999b] I. Horrocks and P. F. Patel-Schneider. Optimising description logic subsumption. *J. of Logic and Computation*, 9(3):267–293, 1999.

[Hustadt & Schmidt 1997] U. Hustadt and R. A. Schmidt. On evaluating decision procedures for modal logic. In *Proceedings of the 15th International Joint Conference on Artificial Intelligence (IJCAI-97)*, volume 1, pages 202–207, 1997.

[Hustadt & Schmidt 1999] U. Hustadt and R. A. Schmidt. An empirical analysis of modal theorem provers. *Journal of Applied Non-Classical Logics*, 9(4), 1999.

[Massacci 1999] F. Massacci. Design and Results of Tableaux-99 Non-Classical (Modal) System Competition. In *Automated Reasoning with Analytic Tableaux and Related Methods: International Conference (Tableaux'99)*, 1999.

[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.

[Patel-Schneider 1998] P. F. Patel-Schneider. DLP system description. In E. Franconi, G. D. Giacomo, R. M. MacGregor, W. Nutt, C. A. Welty, and F. Sebastiani, editors, *Collected Papers from the International Description Logics Workshop (DL'98)*, pages 87–89, 1998. Available as CEUR-WS/Vol-11 from http://SunSITE.Informatik.RWTH-Aachen.DE/Publications/CEUR-WS.

[Pellettier 1986] F. J. Pellettier. Seventy-Five Problems for Testing Authomatic Theorem Provers. *Journal of Automated Reasoning*, 2:191–216, 1986.

[Pitt & Cunningham 1996] J. Pitt and J. Cunningham. Distributed modal theorem proving with KE. In P. Minglioli, U. Moscato, D. Mindici, and M. Ornaghi, editors, *Automated Reasoning with Analytic Tableaux and Related Methods: International Conference Tableaux'96*, number 1071 in Lecture Notes in Artificial Intelligence, pages 160–176, Berlin, 1996. Springer-Verlag.

[Selman *et al.* 1996] B. Selman, D. G. Mitchell, and H. J. Levesque. Generating hard satisfiability problems. *Artificial Intelligence*, 81:17–29, 1996.

[Suttner & Sutcliffe 1995] C. B. Suttner and G. Sutcliffe. The TPTP Problem Library. Technical Report TR 95/6, James Cook University, Australia, August 1995.

[Tacchella 1999] A. Tacchella. *SAT system description. In P. Lambrix, A. Borgida, M. Lenzerini, R. Möller, and P. Patel-Schneider, editors, *Proceedings of the 1999 International Workshop on Description Logics (DL'99)*, pages 142–144, 1999. Available as CEUR-WS/Vol-22 from http://SunSITE.Informatik.RWTH-Aachen.DE/Publications/CEUR-WS.

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