

Boosting complete techniques thanks to local search methods

Bertrand MAZURE, Lakhdar SAÏS and Éric GRÉGOIRE

CRIL – Université d'Artois, rue de l'Université SP 16, F-62307 LENS Cédex, France

E-mail: {mazure,sais,gregoire}@cril.univ-artois.fr

In this paper, an efficient heuristic allowing one to localize inconsistent kernels in propositional knowledge-bases is described. Then, it is shown that local search techniques can boost the performance of logically complete methods for SAT. More precisely, local search techniques can be used to guide the branching strategy of logically complete Davis and Putnam's like techniques, giving rise to significant performance improvements, in particular when addressing locally inconsistent problems. Moreover, this approach appears very competitive in the context of consistent SAT instances, too.

Keywords: SAT, NP-completeness, local search methods, logical inconsistency, logical completeness

1. Introduction

A fundamental problem in logical knowledge-based representation and automated theorem proving systems lies in the difficulty of checking and handling forms of inconsistency. In particular, any local contradiction in a standard logical knowledge-based system makes it wholly inconsistent: any piece of information (and its contrary) can be inferred from it under sound and complete standard rules of deduction.

At the same time, checking in the general propositional case whether a formula is satisfiable or not -namely, the SAT problem-, can be extremely time-consuming. Indeed, although theoretical and experimental results show good average-case performance for several classes of SAT instances (see e.g. [10]), SAT is NP-complete [3].

Recently, there has been a renewal of interest in understanding the nature of the difficulty of SAT (see e.g. [2] and [9]). At the same time, several authors

have proposed new -but amazingly simple and efficient- algorithms allowing for a breakthrough in the class of computer- solvable consistent SAT instances (see e.g. [17], [18], [7] and [14]). However, since these algorithms are based on local search techniques, they are logically incomplete in that they cannot be used directly to prove in a definitive manner that a formula is inconsistent.

Actually, the most efficient complete techniques (like those derived from Davis and Putnam's one -in short DP- [6]) that are used to prove that a formula is inconsistent exhibit a somewhat limited practical scope with respect to really large and hard unsatisfiable SAT instances. Moreover, we cannot hope for such logically complete techniques to exhibit a polynomial behaviour in all situations unless $P = NP$.

In this respect, the contribution of this paper is twofold. The behaviour of local search algorithms has been analysed in the context of inconsistent SAT instances. Some recurrent phenomena have been pointed out when locally inconsistent problems were considered, i.e. problems whose inconsistency can be related to one of their subparts. In this context, two main results have been derived.

- First, an efficient heuristic has been discovered, allowing one to localize the inconsistent kernels in many locally inconsistent problems. This result is clearly of prime importance since inconsistent knowledge in real-life applications is often based on contradictions that are just local.
- This heuristic can boost the performance of complete techniques, in particular when dealing with large inconsistent SAT instances. More precisely, by combining this heuristic with the power of local search methods and with the completeness of standard SAT techniques, very good results are obtained with respect to classes of both consistent and inconsistent hard SAT instances.

The paper is organized as follows. First, some technical background about SAT and local search methods is briefly recalled. Then, it is shown that these search methods can help us to localize the inconsistent kernels of large SAT instances. Families of algorithms combining logically complete techniques with local search methods are then proposed. The experimental performance of a basic combination schema is illustrated on hard problems taken from standard benchmarks [7]. The scope of these results is then discussed before further promising paths of research are motivated.

2. Technical Background

SAT consists in checking the satisfiability of a boolean formula in conjunctive normal form (CNF). A CNF formula is a set (interpreted as a conjunction) of clauses, where a clause is a disjunction of literals. A literal is a positive or negated propositional variable.

An interpretation of a boolean formula is an assignment of truth values to its variables. A model is an interpretation that satisfies the formula. Accordingly, SAT consists in finding a model of a CNF formula when such a model does exist or in proving that such a model does not exist.

Recently, there has been a renewal of interest in designing efficient methods for hard SAT instances. On the one hand, several authors have improved logically complete techniques like DP. However, these techniques remain of a somewhat limited practical scope with respect to really large and hard SAT instances. In the sequel, we shall refer to improved versions of DP: namely, a version making use of the FFIS heuristics (i.e. First-Fail In Shortened Clauses) by [16] and C-SAT [8], which appeared to be the most efficient complete technique for SAT at the last DIMACS challenge [7].

On the other hand, logically incomplete techniques based on local search have been shown particularly efficient in proving large and hard consistent problems. Let us now briefly recall one of these methods, namely Selman et al.'s GSAT algorithm [17], [18]. This algorithm performs a greedy local search for a satisfying assignment of a set of propositional clauses. The algorithm starts with a randomly generated truth assignment. It then changes (“flips”) the assignment of the variable that leads to the largest increase in the total number of satisfied clauses. Such flips are repeated until either a model is found or a preset maximum number of flips (MAX-FLIPS) is reached. This process is repeated as needed up to a maximum of MAX-TRIES times.

In the sequel, we shall consider two more recent variants of GSAT:

- First, the Random Walk Strategy GSAT [18], which outperforms basic GSAT procedures. This variant of GSAT selects the variable to be flipped in the following way: it either picks with probability p a variable occurring in some unsatisfied clause or follows, with probability $1-p$, the standard GSAT scheme, i.e. makes the best possible local move.
- Second, TSAT [14], which departs from basic GSAT by making an optimized use of a tabu list of variables in order to avoid recurrent flips and thus escape

Algorithm 1. GSAT: basic version**Procedure** *GSAT***Input** : a set of clauses *S*, MAX-FLIPS, and MAX-TRIES**Output** : a satisfying truth assignment of *S*, if found**Begin** **for** *i* := 1 **to** MAX-TRIES *I* := a randomly generated truth assignment; **for** *j* := 1 **to** MAX-FLIPS **if** *I* satisfies *S* **then return** *I*; *x* := a propositional variable such that a change in its truth
 assignment gives the largest increase (possibly negative)
 in the number of clauses of *S* that are satisfied by *I*; *I* := *I* with the truth assignment of *x* reversed; **end for**; **end for**; **return** “no satisfying assignment found”;**End.**

from local minima. More precisely, TSAT keeps a fixed length -chronologically-ordered FIFO- list of flipped variables and prevents any of the variables in the list from being flipped again during a given amount of time. TSAT proves very competitive in most situations [14].

These very simple logically incomplete algorithms, which belong to the local search procedures family, are surprisingly efficient in demonstrating that CNF formulas are consistent.

3. Using GSAT-like Techniques to Detect and Locate Local Inconsistencies

In this section, it is shown that GSAT-like techniques can be used to localize inconsistent kernels of SAT instances, although the scope of such logically incomplete algorithms was generally expected to concern consistent problems, only.

The following test has been repeated very extensively and the following results have been obtained extremely often.

When TSAT (or any other GSAT-like algorithm) is run on a SAT instance, the following phenomenon can be observed when the algorithm fails to prove that it is consistent. A trace of TSAT is recorded: for each clause, taking each flip as a step of time, the number of times during which this clause is falsified is updated. A similar trace is recorded for each literal occurring in the SAT instance, counting the number of times it appears in the falsified clauses. Intuitively, it seemed to us that the most often falsified clauses should normally belong to an inconsistent kernel of the SAT instance if this instance is actually inconsistent. Likewise, it seemed to us that the literals that exhibit the highest scores should also take part in this kernel. Actually, these hypotheses prove experimentally correct extremely often.

This phenomenon can be summarized as follows. When GSAT-like algorithms are run on a locally inconsistent SAT instance, then the above counters allow us to split the SAT problem into two parts: a consistent one and an unsatisfiable one. For example, the clausal representation of the well-known inconsistent pigeons-holes problems [19] (8 pigeons; 56 variables and 204 clauses) has been mixed with the 8-queens problems (64 variables and 736 clauses), each problem making use of its own variables. The representation of each problem contains two kinds of clauses: namely, the positive ones (i.e. made of positive literals, only) and the binary negative ones (i.e. made of two negative literals). For example, the basic representation of the pigeons-holes problem asserts, on the one hand, that each pigeon should be in one hole (by means of positive clauses), and, on the other hand, that two pigeons cannot share a hole (using negative clauses). The number of times the different clauses have been falsified is shown in Figure 1. A significant gap can be seen between the scores of the pigeons-holes clauses and the scores of the 8-queens ones. Actually, the two parts of the mixed problem are clearly identified.

Moreover, in this specific case, for each part of the clausal representation (for example, the positive clauses of the pigeons-holes problem), the concerned clauses exhibit extremely close scores. In the diagram, the medium score are mentioned (actually, the four mentioned scores belong to [17189..18003], [639..828], [182..230] and [1..38], respectively for TSAT with $\text{MAX-TRIES} = 20$ and $\text{MAX-FLIPS} = (\text{number of variables})^2$. The span of these intervals shortens when the computing resources given to TSAT are increased). These close scores show us

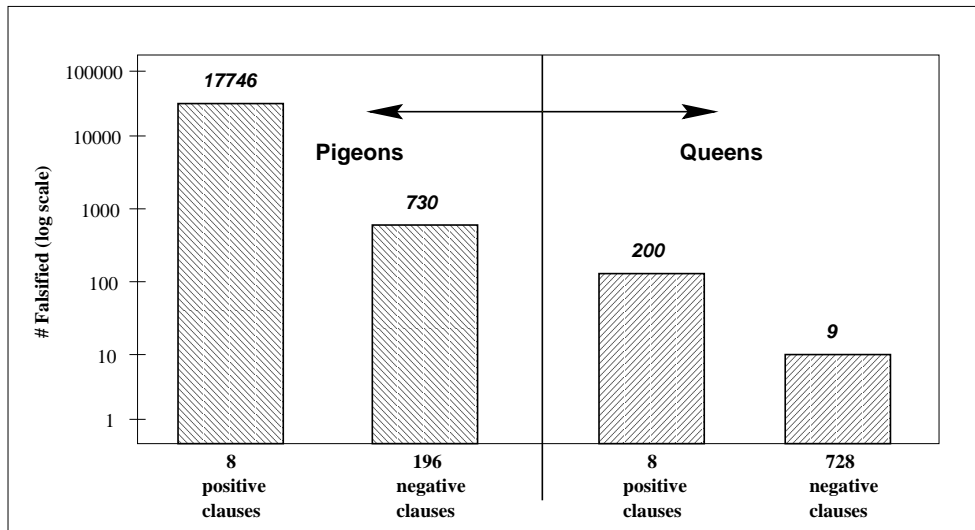


Figure 1. 8-Pigeons-holes + 8-Queens problems

that TSAT also exhibits the complete symmetry of each part of the representation of the pigeons-holes and queens problems. Actually, the trace of GSAT-related algorithms allows us to detect symmetries in SAT instances.

For less symmetrical and non-symmetrical problems, a significant gap between the scores of two parts of the clausal representation is still obtained, differentiating a probable inconsistent kernel from the remaining part of the problem. Let us stress that this phenomena also appear when both the consistent and the inconsistent parts of the SAT instance share the same variables.

Strangely enough, it appears thus that the trace of GSAT-like algorithms delivers the probable inconsistent kernel of locally inconsistent problems and sometimes allows us to detect and locate the presence of symmetries in SAT instances.

4. A Direct Approach: Focus on the Kernel

The most straightforward use of the above discovered feature to solve locally inconsistent problems is the following one. Use GSAT-like algorithms to circumscribe the probable inconsistent kernel. Then, apply complete techniques to this kernel to prove its unsatisfiability and thus, consequently, the inconsistency of the global problem. The scores delivered by the GSAT-like algorithm can be used to guide the branching strategy performed by the complete technique, by selecting

the literals with the highest scores first.

Obviously, this simple schema can only be used when the discovered probable inconsistent kernel is of manageable size and when the gap between the score of the clauses of the kernel and the score of the remaining clauses is large enough. Also, the clauses can be sorted according to their decreasing scores; incremental complete techniques can be applied on them until unsatisfiability is proved. In this respect, clauses outside the discovered kernel could also be taken into account. A somewhat similar approach has been defined independently in [5].

5. A Basic Combination Schema

Let us now describe a basic algorithm using GSAT-like procedures to guide the branching strategy of logically-complete algorithms based on DP.

TSAT [14] is run to deliver the next literal to be assigned by DP. This literal is selected as the one with the highest score as explained above. Such an approach can be seen as using the trace of TSAT as an heuristic for selecting the next literal to be assigned by DP, and a way to extend the partial assignment made by DP towards a model of the SAT instance when this instance is satisfiable. Each time DP needs to select the next variable to be considered, such a call to TSAT can be performed with respect to the remaining part of the SAT instance. This algorithm is given in Algorithm 2.

Algorithm 2. DP + TSAT: basic version

Procedure *DP + TSAT*

Input : a set of clauses *S*

Output : a satisfying truth assignment of *S*, if found
or a definitive statement that *S* is inconsistent

Begin

Unit_propagate(*S*);

if the empty clause is generated **then return** (false);

else if all variables are assigned **then return** (true)

else begin

if TSAT(*S*) succeeds **then return** (true)

else begin

p := the most often falsified literal during TSAT;

return (DP+TSAT(*S*∧*p*) ∨ DP+TSAT(*S*∧¬*p*));

```
end;  
end;  
End
```

6. Experimental Results¹

In this section, experimental results about the application of the above new logically complete algorithm to many classes of problems are presented. Let us stress that our goal in conducting these tests was simply to check the feasibility of using GSAT-like techniques to guide DP. In this respect, a basic form of combination has been tested, which can be improved in many directions as this will be described in the next section.

First, DP+TSAT has been run and compared with DP + FFIS and C-SAT (with default option) on various large inconsistent problems from the DIMACS benchmarks [7]. This benchmark consists of various SAT instances: problems from real-life applications, academic and random ones, many of them being out of reach of the most efficient techniques, in particular complete ones. In Table 1, a significant sample of our extensive experimentations is given, showing the obtained dramatical performance improvement, in particular for classes of inconsistent SAT instances. Also, very good results are obtained for consistent instances; indeed, DP+TSAT is as efficient as local search techniques since DP+TSAT begins with a call to them. Extensive results on DIMACS benchmarks are given in the appendix.

Let us comment a few examples from Table 1.

The BF1355-638 problem (2177 variables and 4768 clauses) is an actual problem from circuit fault analysis proposed by Allen Van Gelder and Yumi Tsuji. C-SAT and DP+FFIS failed to prove that this problem is inconsistent, within 73H and 17H CPU time, respectively. On the other hand, DP+TSAT

¹ All the algorithms mentioned in this paper are implemented in a common platform, available from the authors, written in C under Linux 1.1.53 (except that we reused C-SAT, the original DIMACS'93 implementation by Dubois). We have thus implemented the TSAT, DP+TSAT and DP+FFIS procedures, this latter one being at least as efficient as Rauzy's original one. All experimentations have been conducted on 133 Pentium PCs.

Table 1
DIMACS problems²

Instances	Sat	Size		Inc. Ker. ³		C-Sat time	DP+FFIS			DP+TSAT		
		Var.	Cla.	Var.	Cla.		assign.	choices	time	assign.	choices	time
<u>AIM series:</u>												
1_6-no-3	No	100	160	51	57	13s32	3E+07	2E+06	214s71	178	16	0s26
1_6-yes1-2	Yes	100	160	***	***	0s00	495858	30052	4s39	77	6	0s08
2_0-no-1	No	100	200	18	19	313s24	4E+07	2E+06	349s52	46	5	0s10
2_0-yes1-1	Yes	100	200	***	***	8s54	706388	31274	7s41	81	8	0s15
1_6-no-1	No	200	320	52	55	20116s30	***	***	>8h	240	16	0s58
1_6-yes1-3	Yes	200	320	***	***	>7h	***	***	>9h	232	11	0s32
2_0-no-3	No	200	400	35	37	>20h	***	***	>15h	120	10	0s42
2_0-yes1-1	Yes	200	400	***	***	>8h	2E+09	7E+07	21859s45	291	27	1s21
1_6-no-1	No	50	80	20	22	0s19	12072	895	0s09	72	8	0s06
1_6-yes1-1	Yes	50	80	***	***	0s07	1540	84	0s01	37	6	0s05
2_0-no-1	No	50	100	21	22	0s30	54014	2759	0s43	52	5	0s05
2_0-yes1-1	Yes	50	100	***	***	0s19	2878	176	0s03	11	3	0s03
<u>BF series:</u>												
0432-007	No	1040	3668	674	1252	463s67	9E+08	6E+06	19553s44	115766	870	85s25
1355-075	No	2180	6778	82	185	88035s05	317628	2047	18s88	4602	28	26s23
1355-638	No	2177	4768	83	154	>73h	***	***	>17h	6192	32	32s57
2670-001	No	1393	3434	79	139	11s45	***	***	>25h	490692	4822	519s40
<u>SSA series:</u>												
0432-003	No	435	1027	306	320	1s70	133794	1570	1s79	1338	16	0s80
2670-130	No	1359	3321	552	669	2053s72	***	***	>33h	2E+07	79426	8040s64
2670-141	No	986	2315	579	1247	3689s65	3E+08	2E+06	6350s77	1E+07	92421	6639s44
7552-038	Yes	1501	3575	***	***	195s75	***	***	>13h	29	1	0s34
7552-158	Yes	1363	3034	***	***	97s59	1639	78	0s19	12	1	0s29
7552-159	Yes	1363	3032	***	***	98s82	1557	84	0s21	12	1	0s25
7552-160	Yes	1391	3126	***	***	159s75	1457	76	0s18	1	1	0s30

takes 32 sec. only.

The AIM200-1_6-yes1-3 problem (200 variables and 320 clauses) by [13] is a 3-SAT instance. All AIM problems exhibit exactly one model when satisfiable. DP+TSAT proves within 0.32 sec. that the above mentioned AIM problem is satisfiable while DP + FFIS and C-SAT gave up after 9 and 7H, respectively.

The AIM200-2_0-no-3 problem (200 variables and 400 clauses) by [13] is an inconsistent 3-SAT instance. DP+TSAT takes 0.42 sec. to prove that this problem is inconsistent, whereas we gave up with DP+FFIS and C-SAT after 15 and 20 H, respectively.

² In the table, “> n H” means that we gave up after the problem had not been solved within n hours of CPU time.

³ “Inc. Ker.” in the table means “Inconsistent Kernel”.

7. Natural Optimizations

Although the above mentioned results are extremely positive, it should be clear that the tests were only conducted for checking the feasibility of mixing local search techniques with logically complete methods. We did not try to optimize the way this mixing is performed. Further significant performance improvements can be expected by fine-tuning the parameters of the involved local search techniques and by defining an optimal balance between the time spent by DP and by the local search techniques. More precisely, the following natural optimizations can be envisioned.

- In DP+TSAT, the literal that exhibits the highest score in the trace of TSAT was selected to guide DP. In this respect, a call to TSAT is performed each time DP needs to select a literal to assign. At the opposite, just one call to TSAT can be made and decreasing scores of literals can be used to guide DP at each step. Between these two extreme points of view, an optimal attitude must be found, limiting the number of calls to GSAT-like algorithms. This optimal balance should depend on both the form of the trace given by this GSAT-like algorithm and on the point reached in the search tree by DP.
- A second possible optimization lies in the fine-tuning of the local search parameters with respect to the remaining problem left by DP.

8. Actual Scope of These Techniques

Clearly, the results presented in this paper concern consistent and locally inconsistent problems, i.e. problems whose inconsistency can be related to one of their subparts. Let us define the dual concepts of local and global inconsistency for SAT instances.

Definition 1.

A SAT instance S is *globally inconsistent*
iff
 S is inconsistent and $\forall S' \subset S : S'$ is consistent.

When an inconsistent SAT instance S is not globally inconsistent, it is said to be locally inconsistent.

In this paper, locally inconsistent SAT instances have been considered. Indeed, such a form of inconsistency is of prime importance in actual applications. Very often, inconsistency is due to the accidental presence of a few pieces of contradictory information about a given subject. Clearly, several measures of locality for inconsistency can be defined, making use of e.g. the (size of the inconsistent kernel)/(size of the SAT instance) ratio. We are currently analyzing how the form and properties of the trace of local search techniques can be experimentally related to graded notions of locality for inconsistency. Let us stress that the techniques presented in this paper do not require this ratio to be negligible, as the ratio of the combination of the pigeons-holes and 8-queens problems illustrates it. Also, the size of the kernel need not be small.

In Table 1 and in the Appendix, we give the size of (one of) the globally inconsistent kernels for each inconsistent problem that DP+TSAT managed to solve. To obtain this information, first we used the trace of TSAT to get a first inconsistent kernel using a dichotomy-like approach. Then, using complete methods, we reduced the kernel to a subpart that we proved to be globally inconsistent. Let us stress that most often this resulting kernel proved to be close to the initial one, showing once again the relevance of the heuristic described in this paper.

As expected, DP+TSAT managed to prove inconsistent problems, where one globally inconsistent kernel is of moderate size. Let us note that sometimes DP+FFIS and C-SAT did not manage to solve them: see for example the AIM-200-2_0-no-3 where the size of one globally inconsistent kernel is just made of 37 clauses referring to 35 variables. Interestingly enough, DP+TSAT also gave rise to performance improvement with respect to problems involving a large globally inconsistent kernel: see for example the bf0432-007 problem that DP+TSAT proved to be inconsistent, addressing a globally inconsistent kernel made of 1252 clauses making use of 674 variables.

Obviously, large globally inconsistent problems are the most difficult to handle; they are often generated in an artificial manner since they are scarce in real life applications (see e.g. the pigeons-holes problem [4], Tseitin and Urqhart's formulas [19], etc.). Really significant progress in dealing with large globally inconsistent problems requires a better understanding of their nature and properties. However, the results presented in this paper also apply to globally inconsistent problems; at least to some extent, as we have illustrated it earlier when we discussed the size of the discovered inconsistent kernels of DIMACS benchmarks. Also, when DP+TSAT (which is once again a rough non-optimized combination

schema) did not give rise to better CPU-time on globally inconsistent problems like Dubois' ones, it allowed smaller search trees to be generated. In the same vein, we hope that some progress could be made with respect to hard random problems. In particular, we hope that some little progress could be obtained in the treatment of inconsistent K-SAT instances at the transition phase in the fixed-length clause model [9], at least, as far as locally inconsistent instances are still actually under consideration. On the other hand, we are very optimistic in making good progress in solving inconsistent random K-SAT instances at the right of the transition phase.

9. Conclusion

The contribution of this paper is twofold. On the one hand, an efficient heuristic-based technique has been proposed, allowing one to detect and locate local inconsistencies in sets of propositional clauses. We think that such a technique should be useful with respect to many computer science domains. For example, this should make the handling of local inconsistencies in logical knowledge bases possible. On the other hand, using local search techniques, the feasibility of boosting complete techniques to prove SAT instances has been demonstrated, in particular large locally inconsistent ones that were out of reach of previous approaches. Moreover, the technique presented in this paper also appears competitive for solving classes of consistent SAT instances. Additionally, further possible optimizations that could lead to additional significant performance improvements have been discussed.

Acknowledgements

This work has been supported by the Ganyèmède II project of the Contat de Plan Etat/Nord-Pas-de-Calais, by the MESR (Ministère de l'Enseignement Supérieur et de la Recherche) and by the IUT de Lens. We express our gratitude to our colleague J.-L. Coquidé for his help in providing us with access to computing facilities at the IUT de Lens, allowing our extensive tests to be performed.

References

- [1] P. Cheeseman and B. Kanefsky and W.M. Taylor, Where the Really Hard Problems are, in: *Proc. IJCAI-91*, pp. 163-169, 1991.

- [2] V. Chvátal and E. Szemerédi, Many Hard Examples for Resolution, in: *Journ. of the ACM*, vol. 33, no. 4, pp. 759-768, 1988.
- [3] S. Cook, The Complexity of Theorem-Proving Procedures, in: *Proc. 3rd Ann. ACM Symp. on Theory of Computing*, pp. 151-158, 1971.
- [4] S. Cook, A Short Proof of the Pigeon Hole Principle Using Extended Resolution, in: *SIGACT News*, vol. 8, pp. 28-32, 1976.
- [5] J.M. Crawford, Solving Satisfiability Problems Using a Combination of Systematic and Local Search, in: *Working notes of the DIMACS Workshop on Maximum Clique, Graph Coloring, and Satisfiability*, 1993.
- [6] M. Davis, H. Putnam, A Computing Procedure for Quantification Theory, in: *Journ. of the ACM*, vol. 7, pp. 201-215, 1960.
- [7] DIMACS, Second challenge organized by the Center for Discrete Mathematics and Computer Science of Rutgers University in 1993 (The benchmarks used in our tests can be obtained by anonymous ftp from Rutgers University DIMACS Center: <ftp://dimacs.rutgers.edu/pub/challenge/sat/benchmarks/cnf>).
- [8] O. Dubois and P. André and Y. Boufkhad and J. Carlier, SAT versus UNSAT, in: *Working notes of the DIMACS Workshop on Maximum Clique, Graph Coloring, and Satisfiability*, 1993.
- [9] O. Dubois and J. Carlier, Probabilistic Approach to the Satisfiability Problem, in: *Theoretical Computer Science*, vol. 81, pp. 65-75, 1991.
- [10] J. Franco and M. Paull, Probabilistic Analysis of the Davis and Putnam Procedure for Solving the Satisfiability Problem, in: *Discrete Applied Math.*, vol. 5, pp. 77-87, 1983.
- [11] I.P. Gent and T. Walsh, Towards an Understanding of Hill-climbing Procedures for SAT, in: *Proc. AAAI-93*, pp.28-33, 1993.
- [12] I.P. Gent and T. Walsh, The SAT Phase Transition, in: *Proc. ECAI-94*, pp. 105-109, 1994.
- [13] K. Iwama and E. Miyano, Test-Case Generation with Proved Securities, in: *Working notes of the DIMACS Workshop on Maximum Clique, Graph Coloring, and Satisfiability*, 1993.
- [14] B. Mazure and L. Saïs and É. Grégoire, Tabu Search for SAT, in: *Proc. AAAI-97*, pp. 281-285, July 1997. (A preliminary version appeared as: TWSAT: a New Local Search Algorithm for SAT. Performance and Analysis, in: *CP95 Workshop on Studying and Solving Really Hard Problems*, Cassis (France), September 1995.)
- [15] D. Mitchell and B. Selman and H. Levesque, Hard and Easy Distributions of SAT Problems, in: *Proc. AAAI-92*, pp. 459-465, 1992.
- [16] A. Rauzy, On the Complexity of the Davis and Putnam's Procedure on Some Polynomial Sub-Classes of SAT, in: *LaBRI Technical Report 806-94*, Université de Bordeaux 1, 1994.
- [17] B. Selman and H. Levesque and D. Mitchell, A New Method for Solving Hard Satisfiability Problems, in: *Proc. AAAI-92*, pp. 440-446, 1992.
- [18] B. Selman and H.A. Kautz and B. Cohen, Local Search Strategies for Satisfiability Testing, in: *Working notes of the DIMACS Workshop on Maximum Clique, Graph Coloring, and Satisfiability*, 1993.
- [19] G.S. Tseitin, On the Complexity of Derivations in Propositional Calculus, in: Slisenko A.O. (ed.), *Structures in Constructive Mathematics, Part II*, pp. 115-125, 1968.

Appendix

In the next table:

- “Inc. Ker.” means “Inconsistent Kernel”.
- “> n H” means that we gave up after the problem had not been solved within n hours of CPU time,
- “***” in:
 - “Inc. Ker.” column means that the instance is satisfiable,
 - other columns means that the method fails to solve the instance,
- “???” in Inc. Ker. column means that DP+TSAT fails to detect an inconsistent kernel.

Table 2
DIMACS problems

Instances	Sat	Size		Inc. Ker.		C-Sat time	DP+FFIS			DP+TSAT		
		Var.	Cla.	Var.	Cla.		assign.	choices	time	assign.	choices	time
<u>Dubois series:</u>												
dubois10	No	30	80	30	80	0s22	20564	2063	0s15	8728	641	1s30
dubois11	No	33	88	33	88	0s44	41044	4111	0s31	21800	1691	3s34
dubois12	No	36	96	36	96	0s87	82004	8207	0s59	25228	1835	4s48
dubois13	No	39	104	39	104	1s79	163924	16399	1s18	47236	3091	8s69
dubois14	No	42	112	42	112	3s67	327764	32783	2s49	99888	7693	14s39
dubois15	No	45	120	45	120	7s39	655444	65551	5s15	147180	9507	26s45
dubois16	No	48	128	48	128	15s22	1E+06	131087	10s09	303108	20733	65s97
dubois17	No	51	136	51	136	31s70	3E+06	262159	19s69	348068	23167	76s86
<u>SSA series:</u>												
0432-003	No	435	1027	306	320	1s70	133794	1570	1s79	1338	16	0s80
2670-130	No	1359	3321	552	669	2053s72	***	***	>33h	2E+07	79426	8040s64
2670-141	No	986	2315	579	1247	3689s65	3E+08	2E+06	6350s77	1E+07	92421	6639s44
6288-047	No	10410	34238	???	???	>24h	***	***	>24h	***	***	>24h
7552-038	Yes	1501	3575	***	***	195s75	***	***	>13h	29	1	0s34
7552-158	Yes	1363	3034	***	***	97s59	1639	78	0s19	12	1	0s29
7552-159	Yes	1363	3032	***	***	98s82	1557	84	0s21	12	1	0s25
7552-160	Yes	1391	3126	***	***	159s75	1457	76	0s18	1	1	0s30
<u>BF series:</u>												
0432-007	No	1040	3668	674	1252	463s67	9E+08	6E+06	19553s44	115766	870	85s25
1355-075	No	2180	6778	82	185	88035s05	317628	2047	18s88	4602	28	26s23

Instances	Sat	Size		Inc. Ker.		C-Sat time	DP+FFIS			DP+TSAT		
		Var.	Cla.	Var.	Cla.		assign.	choices	time	assign.	choices	time
1355-638	No	2177	4768	83	154	>73h	***	***	>17h	6192	32	32s57
2670-001	No	1393	3434	79	139	11s45	***	***	>25h	490692	48220	519s4
<i>AIM series (100 variables):</i>												
1.6-no-1	No	100	160	43	47	0s89	7E+06	323296	50s30	174	13	0s22
1.6-no-2	No	100	160	46	52	0s09	3E+06	167445	23s38	178	19	0s34
1.6-no-3	No	100	160	51	57	13s32	3E+07	2E+06	214s71	178	16	0s26
1.6-no-4	No	100	160	43	48	0s05	1E+07	728908	97s59	126	14	0s25
1.6-yes1-1	Yes	100	160	***	***	0s04	135966	6663	1s09	138	8	0s12
1.6-yes1-2	Yes	100	160	***	***	0s00	495858	30052	4s39	77	6	0s08
1.6-yes1-3	Yes	100	160	***	***	0s04	604	34	0s01	112	12	0s18
1.6-yes1-4	Yes	100	160	***	***	0s04	159646	7707	1s46	124	6	0s08
2.0-no-1	No	100	200	18	19	313s24	4E+07	2E+06	349s52	46	5	0s10
2.0-no-2	No	100	200	35	39	72s19	3E+07	1E+06	294s09	108	9	0s20
2.0-no-3	No	100	200	25	27	274s14	9E+06	394649	80s77	68	6	0s12
2.0-no-4	No	100	200	26	31	0s05	3E+07	1E+06	233s29	80	9	0s19
2.0-yes1-1	Yes	100	200	***	***	8s54	706388	31274	7s41	81	8	0s15
2.0-yes1-2	Yes	100	200	***	***	1s85	238794	10305	2s79	91	10	0s20
2.0-yes1-3	Yes	100	200	***	***	5s19	138	10	0s00	84	6	0s12
2.0-yes1-4	Yes	100	200	***	***	0s70	270	18	0s00	163	9	0s13
3.4-yes1-1	Yes	100	340	***	***	0s10	996	30	0s02	1	2	0s08
3.4-yes1-2	Yes	100	340	***	***	0s15	2366	74	0s06	0	1	0s00
3.4-yes1-3	Yes	100	340	***	***	0s10	5484	193	0s14	0	1	0s01
3.4-yes1-4	Yes	100	340	***	***	0s10	196	17	0s01	0	1	0s02
6.0-yes1-1	Yes	100	600	***	***	0s15	100	4	0s01	0	1	0s01
6.0-yes1-2	Yes	100	600	***	***	0s27	248	7	0s01	0	1	0s00
6.0-yes1-3	Yes	100	600	***	***	0s12	224	11	0s01	0	1	0s01
6.0-yes1-4	Yes	100	600	***	***	0s12	212	11	0s01	0	1	0s00
<i>AIM series (200 variables):</i>												
1.6-no-1	No	200	320	52	55	20116s30	***	***	>8h	240	16	0s58
1.6-no-2	No	200	320	77	80	0s17	***	***	>15h	262	24	0s88
1.6-no-3	No	200	320	77	83	0s70	***	***	>8h	302	33	1s15
1.6-no-4	No	200	320	44	46	0s04	***	***	>17h	184	16	0s61
1.6-yes1-1	Yes	200	320	***	***	>12h	210	10	0s01	222	10	0s25
1.6-yes1-2	Yes	200	320	***	***	0s04	682	34	0s02	240	14	0s44
1.6-yes1-3	Yes	200	320	***	***	>7h	***	***	>9h	232	11	0s32
1.6-yes1-4	Yes	200	320	***	***	0s04	6E+08	3E+07	5567s85	244	13	0s42
2.0-no-1	No	200	400	49	53	>7h	***	***	>15h	136	11	0s47
2.0-no-2	No	200	400	46	50	>20h	***	***	>8h	186	15	0s67
2.0-no-3	No	200	400	35	37	>20h	***	***	>15h	120	10	0s42
2.0-no-4	No	200	400	36	42	0s07	***	***	>8h	144	13	0s55
2.0-yes1-1	Yes	200	400	***	***	>8h	2E+09	7E+07	21859s45	291	27	1s21
2.0-yes1-2	Yes	200	400	***	***	8273s44	2E+08	7E+06	2809s87	290	29	1s18
2.0-yes1-3	Yes	200	400	***	***	1197s97	4960	217	0s09	444	20	0s98

Instances	Sat	Size		Inc. Ker.		C-Sat time	DP+FFIS			DP+TSAT		
		Var.	Cla.	Var.	Cla.		assign.	choices	time	assign.	choices	time
2_0-yes1-4	Yes	200	400	***	***	54296s32	272472	11783	4s79	319	27	1s07
3_4-yes1-1	Yes	200	680	***	***	0s52	210956	5409	6s88	0	1	0s06
3_4-yes1-2	Yes	200	680	***	***	0s29	8896	272	0s37	0	1	0s07
3_4-yes1-3	Yes	200	680	***	***	0s70	1302	36	0s05	2	3	0s22
3_4-yes1-4	Yes	200	680	***	***	0s35	267786	6270	8s44	0	1	0s05
6_0-yes1-1	Yes	200	1200	***	***	0s57	6110	75	0s25	0	1	0s01
6_0-yes1-2	Yes	200	1200	***	***	0s22	10284	134	0s47	0	1	0s01
6_0-yes1-3	Yes	200	1200	***	***	0s54	16704	214	0s72	0	1	0s01
6_0-yes1-4	Yes	200	1200	***	***	0s34	16412	232	0s72	0	1	0s04
<i>AIM series (50 variables):</i>												
1_6-no-1	No	50	80	20	22	0s19	12072	895	0s09	72	8	0s06
1_6-no-2	No	50	80	28	32	0s02	11408	782	0s10	86	9	0s07
1_6-no-3	No	50	80	28	31	0s07	54610	4525	0s41	92	12	0s09
1_6-no-4	No	50	80	18	20	0s02	7230	447	0s05	56	7	0s06
1_6-yes1-1	Yes	50	80	***	***	0s07	1540	84	0s01	37	6	0s05
1_6-yes1-2	Yes	50	80	***	***	0s02	5674	384	0s05	50	3	0s02
1_6-yes1-3	Yes	50	80	***	***	0s04	50	5	0s01	50	4	0s04
1_6-yes1-4	Yes	50	80	***	***	0s02	70	5	0s01	1	2	0s01
2_0-no-1	No	50	100	21	22	0s30	54014	2759	0s43	52	5	0s05
2_0-no-2	No	50	100	28	30	0s07	19342	974	0s17	80	7	0s07
2_0-no-3	No	50	100	22	28	0s22	15254	814	0s13	76	6	0s06
2_0-no-4	No	50	100	18	21	0s02	30034	1645	0s24	60	6	0s06
2_0-yes1-1	Yes	50	100	***	***	0s19	2878	176	0s03	11	3	0s03
2_0-yes1-2	Yes	50	100	***	***	0s07	432	29	0s01	0	1	0s00
2_0-yes1-3	Yes	50	100	***	***	0s10	7616	446	0s07	12	3	0s02
2_0-yes1-4	Yes	50	100	***	***	0s04	92	8	0s00	0	1	0s00
3_4-yes1-1	Yes	50	170	***	***	0s04	496	20	0s01	0	1	0s00
3_4-yes1-2	Yes	50	170	***	***	0s07	406	13	0s01	0	1	0s00
3_4-yes1-3	Yes	50	170	***	***	0s02	378	16	0s01	0	1	0s01
3_4-yes1-4	Yes	50	170	***	***	0s05	282	13	0s00	0	1	0s00
6_0-yes1-1	Yes	50	300	***	***	0s09	124	4	0s01	0	1	0s00
6_0-yes1-2	Yes	50	300	***	***	0s07	282	8	0s01	0	1	0s01
6_0-yes1-3	Yes	50	300	***	***	0s09	82	4	0s01	0	1	0s00
6_0-yes1-4	Yes	50	300	***	***	0s07	76	4	0s01	0	1	0s00