This paper appears in the *Proceedings of the Fourth International Symposium on Artificial Intelligence and Mathematics (AI/MATH-96)*, Marina Marriot, Fort Lauderdale, Florida, USA, 1996.

Detecting Logical Inconsistencies *
(Extended Abstract)

Mazure Bertrand
CRIL
Université d'Artois
rue de l'Université SP 16
F-62307 Lens Cedex
France
mazure@lens.lifl.fr

Saïs Lakhdar
CRIL - IUT de Lens
Université d'Artois
rue de l'Université SP 16
F-62307 Lens Cedex
France
sais@lens.lifl.fr

Grégoire Eric
CRIL
Université d'Artois
rue de l'Université SP 16
F-62307 Lens Cedex
France
gregoire@lens.lifl.fr

Abstract

In this paper, a practical approach to detect inconsistent kernels in propositional knowledge bases is presented. First, local search methods are shown to provide one with a powerful heuristic to localize such kernels. Then, based on this finding, a new -surprisingly efficient- logically complete method of checking propositional satisfiability is developed.

Keywords

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

* This work has been supported by the Ganymède II project of the Contrat de Plan Etat/Nord-Pas-de-Calais, by the PRC-GDR Intelligence Artifielle, by the MESR (Ministère de l'Enseignement Supérieur et de la Recherche and by the IUT de Lens.

1 Introduction

Recently, there has been significant progress in dealing with SAT, i.e. checking the satisfiability of a boolean formula. On the one hand, a better understanding of the nature of the difficulty of this NP-complete problem has been obtained thanks mainly to statistical studies and the discovery of threshold phenomena (see [13, 2, 7]). On the other hand, new algorithms extending the class of computer-solvable *consistent* SAT instances have been proposed (see e.g. [15, 16, 5, 12]). However, since these algorithms are based on local search techniques, they cannot be used to prove in a definitive manner that a formula is inconsistent and are thus logically uncomplete.

Actually, the most efficient complete techniques (like those derived from Davis and Putnam's one -in short DP- [4, 6]) that are used to prove that a formula is inconsistent exhibit a limited practical scope because they cannot solve 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.

Indeed, there is no universal efficient way to check inconsistencies in the general propositional case. This is a basic and recurrent drawback of logic-based representation and automated reasoning.

In this context, the contribution of this paper is twofold. The behaviour of local search algorithms is analyzed when applied to hard and large inconsistent SAT instances. Some recurrent phenomena are discovered when locally inconsistent problems are under consideration. This allows the following two results to be derived.

First, an efficient heuristic is discovered, allowing one to localize the inconsistent kernels in many problems. This result is clearly of prime importance since inconsistent knowledge in real-life applications is often based on contradictions that are just local and since any local contradiction in a standard logical knowlege-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.

Taking advantage of this heuristic, it is possible to boost complete techniques for inconsistent large SAT instances. More precisely, by combining this heuristic with the power of local search methods and with the completeness of standard SAT techniques, a real breakthrough is obtained with respect to the class of large and hard computer-tractable (consistent and inconsistent) SAT instances.

Accordingly, the paper is organized as follows. First, some technical background about SAT and local search methods are recalled. Then, these search methods are shown helpful in localizing inconsistent kernels. 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 many hard problems taken from standard benchmarks [5]. Further promising paths of research are then motivated.

2 Technical Background

SAT consists in checking the satisfiability of a boolean formula in conjunctive normal form (CNF). Let us recall here that any propositional formula can be translated thanks to a linear time algorithm in CNF, equivalent with respect to SAT. 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 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 (e.g. CSAT [6]). However, these techniques remain of a limited practical scope since they do not allow one to manage large and hard SAT instances. In the sequel, we shall refer to improved versions of DP based on the R. E. Jeroslow and J. Wang's heuristic [11], i.e. the First-Fail in Short Clauses (see e.g. [6]), and its variant called FFIS (i.e. First-Fail In Shortened Clauses) by [14].

On the other hand, logically uncomplete 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 [15, 16]. This algorithm performs a greedy local search

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 in the number of clauses*[1] *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

Figure 1: GSAT Algorithm: basic version

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, two more recent variants of GSAT will be considered:

- First, Random Walk Strategy GSAT [16], 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, TWSAT (as Taboo Walk Strategy for SAT) [12], which departs

[1] this number can be negative

from basic GSAT by making a *systematic* use of a taboo list of variables in order to avoid recurrent flips and thus escape local minima. More precisely, TWSAT 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. TWSAT outperforms Random Walk GSAT in most situations (see [12]).

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

3 Local vs. Gobal Inconsistencies

First, let us define 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.

Definition 2

A SAT instance S is *locally inconsistent*
iff
S is inconsistent and $\exists S' \subset S : S'$ is inconsistent

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. In this paper, we focus on locally inconsistent SAT instances. 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. Obviously, 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 [3], Tseitin and Urqhart's formulas [17], etc.). Nevertheless, since globally inconsistent problems can be seen as parts of bigger local ones, results that will be presented here also apply, to some extent, to globally inconsistent ones.

4 Using GSAT-like Techniques to Localize Inconsistent Kernels

In this section, a somewhat surprising finding is presented: GSAT-like techniques can be used to localize inconsistent kernels of SAT instances, although the scope of such logically uncomplete algorithms was normally expected to concern consistent problems, only.

The following test has been repeated very extensively, giving rise to the same result.

TWSAT (or another GSAT-like algorithm) is run on a SAT instance. The following phenomenon is encountered when the algorithm fails to prove that the instance is consistent. TWSAT is traced and, 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 also done for each literal occurring in the SAT instance, counting the number of times it has appeared 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, this hypothesis proved to be 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, we have mixed the clausal representations of the well-known inconsistent pigeons-holes problems (8 pigeons; 56 variables and 204 clauses) with the 8-queens problems (64 variables and 736 clauses) (each problem making use of its own variables). The representation of each such problem contains two kinds of clauses: namely, the positive ones (i.e. made of positive literals, only) and the binary negative ones (made of two negative

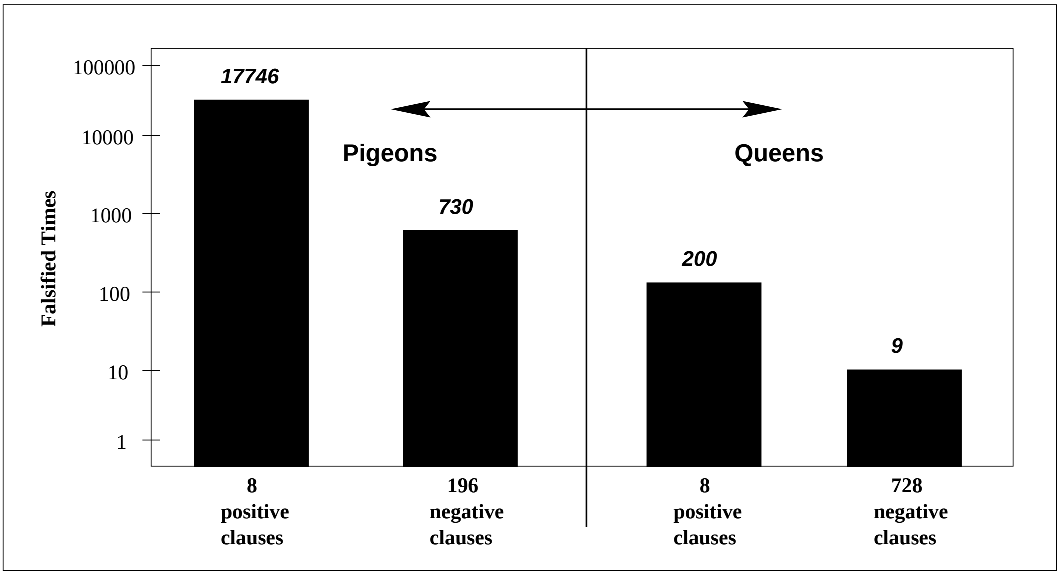

Figure 2: 8-Pigeons-holes + 8-Queens problems

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). In Figure 2, the number of times the different clauses have been falsisfied is given. A huge gap can be observed between the scores of the pigeons-holes clauses and the scores of the 8-queens ones. The two parts of the mixed problem are actually clearly identified.

Moreover, in this specific case, for each part of the clausal representation (for example, the positive clauses of the pigeon hole), the concerned clauses exhibit extremely close scores. In the diagram, the medium scores are mentioned (actually, the four mentioned scores belong to [17189..18003], [639..828], [182..230] and [1..38], respectively for TWSAT with MAX-TRIES = 20 and MAX-FLIPS = (number of variables)^2 . The span of these intervals shortens when the computing resources given to TWSAT are increased). These close scores show us that TWSAT 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 can allow us to detect the probable existence of 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. We are currently analyzing how the form and properties of this trace can be experimentally related to *graded* notions of locality for inconsistency.

5 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 detect 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 search performed by the complete technique.

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, we can sort the clauses according to their decreasing scores and use incremental complete techniques on them until unsatisfiability is proved. In this respect, clauses outside the discovered kernel could also be taken into account.

6 A Basic Combination Schema

Let us now describe a basic algorithm using GSAT-like procedures to guide logically-complete algorithms based on DP. TWSAT [12] is applied to deliver the next literal to be assigned the truth-value true 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 TWSAT as an heuristic in order to select the next literal to be assigned true by DP, and a way to extend the partial assign-

```
Procedure DP + TWSAT
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 TWSAT( S ) succeeds then return (true)
               else begin
                      p := the most often falsified literal
                           during TWSAT search;
                      return (DP+TWSAT( S∧p) ∨
                                DP+TWSAT(S∧¬p));
                    end;
             end;
End
```

Figure 3: DP + TWSAT: basic version

ment 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 TWSAT can be performed with respect to the remaining part of the SAT instance. This algorithm is given in Figure 3.

7 Experimental Results [2]

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 the intended goal in conducting these tests was simply to check

[2]all the algorithms mentioned in this paper are implemented in a common platform written in C under Linux 1.1.53, available from the authors by anonymous ftp at ftp.lifl.fr cd pub/projects/SAT. All experimentations have been conducted on i486 DX2 66 PCs.

DIMACS Instances	Sat.	DP+ffis Assign.	DP+ffis Choices	DP+ffis Time	DP+TWSAT Assign.	DP+TWSAT Choices	DP+TWSAT Time
aim-50-1_6-yes1-2	Yes	8980	586	0s29	17	4	0s08
aim-50-1_6-no-3	No	54614	4527	1s56	92	12	0s32
aim-50-2_0-yes1-3	Yes	7616	446	0s28	23	3	0s09
aim-50-2_0-no-3	No	54014	2759	1s84	80	8	0s25
aim-100-1_6-yes1-2	Yes	495858	30052	16s86	127	10	0s43
aim-100-1_6-no-3	No	67786124	3854371	2221s17	176	17	0s78
aim-100-2_0-yes1-2	Yes	361660	18153	20s40	234	14	0s95
aim-100-2_0-no-3	No	38503642	1725544	1527s69	46	5	0s27
aim-200-1_6-yes1-2	Yes	*	*	*	216	10	0s98
aim-200-1_6-no-3	No	*	*	*	244	17	1s92
aim-200-2_0-yes1-2	Yes	*	*	*	299	27	3s59
aim-200-2_0-no-3	No	*	*	*	176	18	1s74
bf0432-007	No	938037830	6208925	86086s93	153760	1079	615s09
bf1355-075	No	317628	2047	68s97	13606	67	169s87
bf1355-638	No	*	*	*	54890	259	584s23
bf2670-001	No	*	*	*	275844	66869	39126s23
ssa0432-003	No	133794	1570	7s96	2944	33	5s79
ssa2670-130	No	*	*	*	14188726	66869	39126s23
ssa2670-141	No	294818640	2312147	27814s59	14845888	99794	25471s94
ssa7552-159	Yes	1557	84	0s82	12	1	0s83
ssa7552-160	Yes	1457	76	0s72	1	1	1s20

Table 1: DIMACS problems [3]

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, the basic combination algorithm has been applied to various large inconsistent problems from the DIMACS benchmarks [5] and its performance has been compared with DP + FFIS one. This benchmark consists of various SAT instances: problems from real-life applications, academic and random ones, most of them being out of reach of the most efficient techniques. In Table 1, a significant sample of the extensive experimentations are given, showing the dramatical performance improvement obtained in this way. In particular, as we hoped, a breakthrough in the class of inconsistent SAT instances is obtained. Also, GSAT-like algorithms are outperformed for consistent instances that were out of reach for these local search methods. For consistent problems in the practical scope of local search methods, the approach presented here is at least as efficient as these techniques since it involves applying them in a first step.

[3] In the table, the symbol * means that the problem has not been solved within 30 hours

Let us comment a few examples from Table 1.

- The BF0432-007 problem (1040 variables and 3668 clauses) in the table is an actual problem from circuit fault analysis proposed by Allen Van Gelder and Yumi Tsuji. To show that this problem is inconsistent, DP+FFIS takes about 24H whereas the basic combination algorithm takes 10 minutes only.

- The AIM200-1_6-yes1-4 problem (200 variables and 320 clauses) by [10] is a 3-SAT instance. All AIM problems exhibit exactly one model when satisfiable. They seem to be out of reach of local search methods. The basic combination algorithm proves in about 1 second that the above mentioned AIM problem is satisfiable while DP + FFIS takes about 7H.

- The AIM100-1_6-no-3 problem (100 variables and 160 clauses) by [10] is an inconsistent 3-SAT instance. DP + FFIS takes 37 minutes whereas the basic combination algorithm takes 0.78 second.

Moreover, good results have been obtained with respect to the number of *performed assignments* for the global and symmetrical Dubois's inconsistent problems belonging to the family of Tseitin and Urqhart's formulas.

8 Natural Optimizations

Although the above mentioned results are extremely positive, it should be clear that these 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. We expect further significant performance improvements by fine-tuning the parameters of the involved local search techniques and by defining an optimal balance between the time spent by the basic complete method and by the local search techniques. More precisely, we suggest the following natural optimizations, as far as DP is concerned.

- In the above algorithm, only one literal with the highest score in the trace of the GSAT-like algorithm is selected to guide DP. In this respect,

of CPU time.

a call to TWSAT is performed each time DP needs to select a literal to assign. At the opposite, just one call to TWSAT could be made and the decreasing scores of literals 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.

9 Conclusions and Perspectives

The contribution of this paper is twofold.

- On the one hand, an efficient heuristic-based technique allowing one to detect and locate inconsistent kernels in sets of propositional clauses has been proposed. We think that such a technique should be useful with respect to many computer science domains. For example, this should make inconsistency handling possible in logical knowlege bases.

- On the other hand, using local search techniques, the feasibility of boosting complete techniques to prove SAT instances has been shown, in particular large locally inconsistent ones that were far out of reach of previous approaches. Moreover, the best current techniques seem to be outperformed for some classes of consistent SAT instances, as well. Additionally, it has been indicated how further optimizations leading to additional significant performance improvements could be reached.

Although this study is motivated by the treatment of real-life applications, we think 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 [7], at least, as far as locally inconsistent instances are still actually under consideration. On the other hand, we are very optimistic about making good progress in solving inconsistent random K-SAT instances at the right of the transition phase.

Clearly, the results presented here suggest the need for a deeper study of the nature of GSAT-like local search techniques. In particular, the usefulness of tracing GSAT with respect to the number of times a literal belongs to the unsatisfied clauses has been shown. An interesting result would be a clear analytical explanation for this phenomenon. Perhaps, other useful information can be derived and exploited from the trace of GSAT-like techniques: for instance, information about the structure of the problem (like the way this trace seemed to confirm the existence of symmetries in the pigeons-holes problem).

Obviously enough, practical progress brought in the treatment of inconsistency will hopefully open new perspectives in automated deduction, using more efficient refutation techniques. We are also exploring how the results presented in this paper for SAT could be applied to other NP-complete problems.

References

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

[2] V. Chvátal, E. Szemerédi (1988). Many Hard Examples for Resolution. *Journ. of the ACM*, vol. 33, no. 4, pp. 759-768.

[3] S. Cook (1976). A Short Proof of the Pigeon Hole Principle Using Extended Resolution. *SIGACT News*, vol. 8, pp. 28-32.

[4] M. Davis, H. Putnam (1960). A Computing Procedure for Quantification Theory. *Journ. of the ACM*, vol. 7, pp. 201-215.

[5] DIMACS (1993). Second challenge organized by the Center for Discrete Mathematics and Computer Science of Rutgers University. (The benchmarks used in our tests can be obtained by anonymous ftp from Rutgers University Dimacs Center: ftp dimacs.rutgers.edu cd pub/challenge/sat/benchmarks/cnf)

[6] O. Dubois, P. André, Y. Boufkhad, J. Carlier (1993). SAT versus UNSAT. *Journ. of the Am. Mathematical Society* (submitted). also in *Proc. of the Second DIMACS Challenge.*

[7] O. Dubois, J. Carlier (1991). Probabilistic Approach to the Satisfiability Problem. *Theoretical Computer Science*, vol. 81, pp. 65-75.

[8] J. Franco, M. Paull (1983). Probabilistic Analysis of the Davis and Putnam Procedure for Solving the Satisfiability Problem. *Discrete Applied Math.*, vol. 5, pp. 77-87.

[9] I.P. Gent, T. Walsh (1994). The SAT Phase Transition. *Proc. ECAI-94*, pp. 105-109.

[10] K. Iwama, E. Miyano (1993). Test-Case Generation with Proved Securities. *Proc. 1993 DIMACS Workshop on Maximum Clique, Graph Coloring, and Satisfiability.*

[11] R.E. Jeroslow, J. Wang (1990). Solving Propositional Satisfiability Problems. *Annals of Math. and Art. Int.*, vol. 1, pp. 167-187.

[12] B. Mazure B., L. Saïs, E. Grégoire (1995). TWSAT: a New Local Search Algorithm for SAT. Performance and Analysis. *Proc. CP-95 Workshop on Studying and Solving Really Hard Problems*, pp. 127-130.

[13] D. Mitchell, B. Selman, H. Levesque (1992). Hard and Easy Distributions of SAT Problems. *Proc. AAAI-92*, pp. 459-465.

[14] A. Rauzy (1994). On the Complexity of the Davis and Putnam's Procedure on Some Polynomial Sub-Classes of SAT. *LaBRI Technical Report 806-94*, Université de Bordeaux 1.

[15] B. Selman, H. Levesque, D. Mitchell (1992). A New Method for Solving Hard Satisfiability Problems. *Proc. AAAI-92*, pp. 440-446.

[16] B. Selman, H.A. Kautz, B. Cohen (1993). Local Search Strategies for Satisfiability Testing. *Proc. 1993 DIMACS Workshop on Maximum Clique, Graph Coloring, and Satisfiability.*

[17] G.S. Tseitin (1968). On the Complexity of Derivations in Propositional Calculus. in: Slisenko A.O. (ed.), *Structures in Constructive Mathematics*, Part II, pp. 115-125.