A Proposed SAT Algorithm

Dr. Bagais A.[1] and Abdullahi M.[2]

[1] Ahmadu Bello University, Zaria-Nigeria

Received: 11 April 2013 Accepted: 1 May 2013 Published: 15 May 2013

Abstract

This paper reviews existing SAT algorithms and proposes a new algorithm that solves the SAT problem. The proposed algorithm differs from existing algorithms in several aspects. First, the proposed algorithm does not do any backtracking during the searching process that usually consumes significant time as it is the case with other algorithms. Secondly, the searching process in the proposed algorithm is simple, easy to implement, and each step is determined instantly unlike other algorithms where decisions are made based on some heuristics or random decisions. For clauses with three literals, the upper bound for the proposed algorithm is O(1.8171n). While some researchers reported better upper bounds than this, those upper bounds depend on the nature of the clauses while our upper bound is independent of the nature of the propositional formula.

Index terms— propositional satisfiability, NP-complete, complexity, complete algorithms.

1 Introduction

ropositional satisfiability (SAT) is one of the classical problems in Computer Science. The importance of SAT comes from the fact that a large class of real-world problems can be expressed in terms of a SAT instance and that it was the first problem proven to be NP-Complete (Cook, 1971). The SAT problem has a wide range of practical real world applications (Barbour, 1992;Crawford & Baker, 1994;Devadas, 1989;Kauts & Selman, 1992;Larrabee, 1992). Many algorithms, categorized into complete and incomplete algorithms, were proposed to solve this problem efficiently over the last decades.

Complete algorithms can state whether a SAT instance is satisfiable giving the satisfying assignments or unsatisfiable giving a 'no' answer. Incomplete algorithms can only give an answer of 'yes' for satisfiable SAT instances only but cannot give an answer for unsatisfiable instances.

This paper proposes a new complete algorithm that differs from the ones in the literature in the following aspects:

? No backtracking during the searching process that usually consumes significant amount of time. ? Has a simple, deterministic and easy to implement search process, unlike other algorithms where decisions are either made randomly or based on some heuristics. The remainder of the paper is structured as follows. Section 2 describes the proposed algorithm with the aid of an example. Section 3 captures the algorithm in pseudo code while Section 4 presents the complexity analysis of the algorithm. We present related work in Section 5. Sections 6 and 7 summarize and provide references, respectively.

2 II.

3 Illustrating the Proposed Algorithm

Unlike other algorithms that make a decision on a single value (true/false) for a variable x , the proposed algorithms takes into consideration all satisfying assignments for a clause C and use them for the next clauses so that backtracking is avoided.

Consider the following formula:4 1 5 3 1 3 2 2 4 ( ) ( ) ( ) F x x x x x x x x x = ? ? ? ? ? ? ? ?

The first clause can be satisfied by any of the following assignments 1 The algorithm tries to find assignments for all variables in clause while preserving at least one of the given assignments for 1 3 , x x , or 4

x in the first clause.

In general, the process starts from the first clause 1

If the clause has k literals, then k assignments can satisfy it (as in the previous formula, the first clause has three assignments). In the next step, the set of assignments that satisfy the set of previous clause(s) are checked with all the literals of the next clause. The process continues until all the clauses in the formula are covered, after which the resulting set of assignments each satisfies the formula.

When a set of assignments from previous clause(s) is checked with the literals of the current clause, each literal may agree, disagree or be neutral to the assignment. A literal agrees with an assignment when the assignment includes the literal. A literal disagrees with an assignment when the assignment includes a negation of the literal. A literal is neutral to an

4 D

assignment when the assignment neither agrees nor disagrees with the literal.4 1 5 3 1 3 2 2 4 ( ) ( ) ( ) F x x x x x x x x x = ? ? ? ? ? ? ? ? ? Figure 1 : Assignment Production

In the first step, the satisfying assignments for the first clause are its literals. The assignments produced for the first clause are shown in the top-left rectangle in Figure **??**. Each of these assignments is checked with the literals of the second clause,1 5 2 ( ) x x x ? ?

. The assignment of 1

x disagrees with the first literal of the second clause, 1

x resulting in no assignment produced. The same assignment, 1

x is checked with the second literal, 2

x . Since this literal is neutral to 1

x , a new assignment is produced by combining 1

x and 2 x , as shown in the middle rectangle in Figure **??**. Next, 1

x is checked with 5 x , giving 5 1

x x , since 5

x is neutral to 1 x . Similarly, the assignments Note that each of these 18 assignments satisfies the given formula. Note that when an assignment agrees with the clause in consideration, the process might produce shorthand for 1 2 x x ? etc. We will illustrate this with the pair of clauses:1 2 3 1 4 5 ( ) ( ) x x x x x x ? ? ? ?

The satisfying assignments for this pair of clauses are:1 1 1 2 1 3 1 1 4 2 4 3 4 1 5 2 5 3 5 ( ) x x or x x x x x x x x x x x x x x x x x

From this group, it can be seen that the assignments } { 1 4 1 5 2 1 3 1 , , , x x x x x x x x are subsumed in the first assignment 1

x . This is because each of these assignments produces the same result as 1

x .

Thus, these assignments can be dropped to avoid redundancy. Therefore, Figure **??** can now be redrawn without the subsumed assignments as shown in Figure 2. x x x x x x x x x x x x x x x x x x x x x x x x x x x x x x x x x x x x x x x x x x x x x x x x x x x x x x x x Since the subsumed assignments are produced from clauses that have a literal in common, the proposed algorithm starts by extracting all clauses that do not share a literal. For a satisfiability formula with n literals each clause containing exactly k literals, the minimum number of clauses in which no two clauses have a common literal is 2n k4 1 3 ( ) x x x ? ? 1 5 2 ( ) x x x ? ? 3 2 4 ( ) x x x ? ?. 4 1 3 2 6 5 6 5 1 2 3 4 1 6 5 4**2 3 ( ) ( ) ( ) ( ) (**)( ) F x x x x x x x x x x x x x x x x x x = ? ? ? ? ? ? ? ? ? ? ? ? ? ? ? ? ? ?

For example, we need at least 4 clauses to have the 12 literals in the following formula. But because of the distribution of literals, we need 5 for that purpose. Therefore, the algorithm will extract the clauses that do not have common literals. There are two advantages in doing so: 1. The algorithm will save the time to check the existence of subsumed assignments which is a process that consumes an amount of time equal to the number of assignments. 2. The time complexity of the algorithm becomes easier to prove (see Section 4). m i m i k k i k k i m ? ? ? ? = ? ? ? ? ? redundant assignments.

Proof: (By induction).

5 Base Case

The base case is when i m = and the total number of redundant assignments will be0 (1) (1)**(1)**m m k k k k k ? ? = ? = ? .

Clearly, the theorem holds for i m = .

6 Inductive Hypothesis

Suppose the theorem holds for 2,3, 4,..., i p = for some clause 2 p m ? < . The total redundant assignments will be 2

; 3m p m p k k p k k p m ? ? ? ? = ? ? ? < ? . If a literal with which an assignment agrees is in1 p + clause,

then the total redundant assignments will be1 (1) 1 (1) 2(1) 2(1) 2(1) ; 2 (1) (1)**(1)**

; 3m p m p m p m p m p m p k k k k k k i k k k k k k k i m k ? ? ? ? + ? ? ? ? + ? ? =? =? = ? ? ? ? ? = ? = ? ? ? ? ?

That is, the theorem holds for 1 p + . By induction on p , the theorem is true for all values of i . x x x x x x x x x x x x x x x x x x x x x x x x 1 3 4 x x x 4 1 3 ( ) x x x ? ? 1 5 2 ( ) x x x ? ? 3 2 4 ( ) x x x ? ? (D D D D D D D D) Year 013 2 D .

7 Base Case

The base case is when i m =

and the total number of assignments will be reduced by0 1 m m k k ? = = .

Clearly, the theorem holds for i m = .

Inductive Hypothesis Suppose the theorem holds for 2,3, 4,..., i p = for some clause 2 p m ? < . The total assignments will be reduced by m p k ? . If a literal with which an assignment agrees with is in 1 p + clause, then the total assignments will be reduced by1 (1) m p m p m p k k k k ? ? ? ? + = =

. That is, the theorem holds for 1 p + . There by induction on p , the theorem is true for all values of i .

8 III.

9 The Proposed Algorithm Pseudocode

The most important step in any complete or incomplete SAT algorithm is the decision over the value of a given variable in the formula. If the decision on that variable is wrong, the algorithm will waste its time searching for a solution before it discovers that the value assigned to the variable does not lead to a satisfying assignment and consequently a backtrack is done to change that value. The problem with making a decision for a variable x using the heuristics is that they do not consider how this decision or assignment will affect other related variables that appear in the same clauses as the variable x . If the search process keeps all possible assignments that satisfy a clause before moving forward, then no backtrack is needed. Instead, these assignments can be used to determine the values of variables that satisfy the next clauses. In the case that none of the variables in the current clause agrees with all the assignments, then the formula is unsatisfiable. This leads to the main idea of the proposed algorithm for assigning values to the variables.

10 The Algorithm

11 Input: F[m]; //formula with m clauses

Output : A[k m]; //Possible assignment satisfying m clauses.

12 getDistinctClauses(F[m]); 2. For i = 1 to disticnt-clauses.length -1;//number of distinct clauses For j = 1 to k //k is the number of literals in a clause LIT[i][j] := disticntclauses[i];

End for End for

13 For i = 1 to k A[i] := LIT[1][i]; //literals of the first clause(initial set of satisfying substitutions) End for 4. For i = 2 to disticntclauses.length;//number of distinct clauses For j = 1 to k generateAssignment(LIT[i][j], A[], temp[]); //A[] contains the set of satisfying substitutions from previous clauses //temp[] contains assignments formed by combining assignments in A[] with a literal LIT[i][j] End for A[] := A[] + temp[];

End for

14 For i = distinctclauses.length + 1 to m;//number of distinct clauses For j = 1 to k //k is the number of literals in a clause LIT[i][j] := nondistinctclauses[i]; End for End for

For i = distinctclauses.length to m For j=1 to k

15 generateAssignment(LIT[i][j], A[], temp); End for removeSubsumedAssignments(tempassignments[], array-subsumed[]); A[] := A[] + temp;

End for

16 If A[] is empty

Output ”the formula is unsatisfiable”; Else Output the assignments in A[] as the satisfying assignments for the formula F.

17 Procedure getDistinctClauses(F[m])

Input: Formula with m clauses Output: arrayofdistinctclauses and arrayofnondistinccaluses distinctclauses [1] = clause [1]

18 Procedure: removeSubsumedAssignments(tempassignments[], arraysubsumed[])

Input: list of assignments containing subsumed assignments and list of assignments subsuming the subsume assignments. Output: list of assignments without subsumed assignments. n:=0; For i = 0 to tempassign-ments.length -1

For j = 0 to arraysubsumeb.length -1 where m is some number of clauses. In step four, clauses in CLS could either be:

19 If (arraysubsumed[j] is not contained in tempassignents[i]) arrayassignments[n++] = tempassignment[i] ; Endfor End-for Return arrayassignments[];

20 Procedure: generateAssignment(lit, A[], temp[]); Input: a literal in a clause and a list of assignments in A[]. Output: a list of assignments stored in temp[] produced by combining lit with A[].

1.

2n k clauses (worst case). Because of the existence of repeated literals in Case 2, Case 1 will produce the maximum number of assignments (see ??heorem 3).

We now determine the number of possible assignments, ( ) A n , in the worst case. If the clauses in CLS have conflicting literals, ( )m A n k ? .

In this case, a literal in one clause will not be combined with a literal 1

x in another clause. The number of substitutions to be eliminated is shown by Theorem 2.

To count the exact number of assignments, the principle of inclusion-exclusion is used. The principle states that the number of elements that have property 1, property 2, property 3, ?, or property n is found by the summation.1 1 2 3 1 2 3 1 1 1

(, , ,...,) ... (1) ... . For any satisfiability instance, the previous quantity cannot be found. That is because unlike the example given earlier, the arrangement of variables or literals differs from one instance to another. However, there is an arrangement that will produce the highest number of variables.n n i i j i j k n i n i j n i j k n

21 N P P P P A A A A A A A A A A

+ ? ? ? < ? ? < < ? = ? ? + ? ? ? + ? ? ? ? ? ? ? ? ? If i P is

22 b) The upper bound

At this point, we need to prove two theorems. One that states case 1 is the worst case and the other states the arrangement that will produce the highest number of assignments.

23 Theorem 3

In step 5 of the algorithm, generating assignments with the least number of clauses

24 ( ) n k

that include 2n literals is the worst case.

25 Proof

If more than 2n k clauses are needed to include the 2n literals then we must have literals that are repeated. If we have one additional clause, then there must be k literals repeated and this will make the set of assignments to be excluded more than n. Having a repeated literal means that we have three clauses of this form:1 1 2 3 1 4 5 6 7 x x x x x x x x x ? ? ? ? ? ? ? .

The two clauses that have the repeated literal 1

x will produce the unnecessary assignments. These assignments are generated when the repeated literal is combined with the (1) k ? literals of the other clause.

This means that the assignments that include{ } 1 4 1 5 1 2 1 3

, , , x x x x x x x x are unnecessary. The only useful assignment is 1x produced from 1 1 ( ) x x . This indicates that 2(1)

k ? sets of assignments should be discarded. In addition to these assignments, the two repeated literals when combined with 1

x will produce

26 D

The first three steps of the algorithm take polynomial time of number of clauses. Steps four and five are clearly the main contributors to the time complexity of the whole algorithm. These two steps rely on the number of assignments generated in each iteration of the for-loop. For step four, that number is determined by the clauses in CLS and for step five, that number is determined by the end of step four. Therefore, let us start with step four.

The proof of the principle can be found in (Rosen, 1999).

two sets of assignments of the form 1 1 ( ) x x that are also discarded from the total number of assignments when we count them using the inclusion exclusion principle. Therefore, a repeated literal will result to discard 2(1) 1 k ? + additional sets excluded.

Writing the inclusion exclusion series with n sets plus (2(1) 1) k k ? + sets is hard because there will be many possibilities for the intersection of sets. The approach to show that 2n/k is the worst case is to exclude the additional sets first from the total number of assignments and compare that with the worst case. The number of assignments of the additional sets can be counted by: (1) (2(1) 1) (,)n n k k n n k k k k k n k i i i k i A k C k k k C k k k C k k k C k k k k C k i k ? ? ? ? + + ? + + = = ? + ? ? + + ? + ? + ? ? + = ? ? + ?

Excluding this from the total assignments2 2 1 2 1 1 1 2 2 1 2**1 2( ) 1 (1) (2(1) 1) (,) ((1) (2(1) 1) (,)**

)n n k i i i k k i n k k k i i k i k i N k k C k i k N k k k C k i k + ? + + = ? + + ? = = ? ? ? + = ? ? ? + ? ? Evaluating 2**1 2( ) 1 ((1) (2(1) 1) (,)**

)k k i i k i i k k C k i k + ? = ? ? ? + ?

for values of k gives quantity less than2 1 k k ?

and result to a number of assignments less than 2n k k and excluding the n sets of the form (v -v) from N gives a value that is less than the one in the worst case.2n k k excld(n sets) 2 2 1 2**1 2( ) 1 ((1) (2(1) 1) (,)**

)n k k k i i k i k i k k k C k i k ? + + ? = > ? ? ? + ? excld(n sets) because 2n k k 2 2 1 2**1 2( ) 1 ((1) (2(1) 1) (,))**

.n k k k i i k i k i k k k C k i k ? + + ? = > ? ? ? + ?

This is for one additional clause. For i additional clauses the limit of the summation is to ik and this also will give the same result.

Theorem 3 tells us that step six will not generate assignments that are more than step five. This should make step 5 the dominant factor for time complexity.

27 Theorem 4

For the worst case, the upper bound is ((1))n k k k ? Proof

The inclusion-exclusion principle takes care of assignments that are counted more than once by considering the intersections between the n sets to be excluded as seen in the summation. Therefore, the least value of x literals:1 2 3 1 4 5 x x x x x x

The assignments that include 1

x and 1 x can never occur with assignments that include 2

x and 2 x , (1) k k ? assignments, then the number of assignments will be (1)n k k k ? . c) Related Work

Complete algorithms for SAT satisfiability problems include those algorithms that can state whether or not a SAT instance is satisfiable, giving a 'yes' answer together with a satisfying assignment or a 'no' answer as the case may be. The first complete algorithm is the Davis Putnam procedure . This procedure is based on resolution rule to eliminate variables one by one till the formula is satisfied. When a variable is eliminated in each iteration, all resolvents are added to the set of the clauses. This algorithm requires polynomial space. It handles CNF formulas and it is one of the efficient SAT algorithms. (Davis, Logemann, & Loveland, 1962) Developed a divide-and-conquer algorithm that enhances on . This improved algorithm is the main procedure for most state-of-the-art SAT solvers today.

The search space of DPLL could grow as large as 2 n which is the worst case for any complete algorithm. Due to the possibility of consuming huge amount of time, researchers have been focusing on mechanisms to reduce that and came up with more reasonable time complexities. These improvements usually come in two aspects: the decision to branch to next literal and the backtracking mechanism if a solution is not found in the chosen branch. The achievements accomplished in improving SAT algorithm in these two aspects show that the complexity could be reduced significantly.

i. Branching Decisions DPLL procedure chooses any literal for branching and goes down that region in the search space. The procedure will spend time searching for a solution and if it discovers that the branch is not successful, it backtracks to the other branch and continues searching. Choosing the next literal for branching more carefully will allow the algorithm to save time exploring a region where a satisfying assignment cannot be found at all and direct the searching to regions where a solution is likely to be found. In order to accomplish this, several heuristics have been proposed and the most effective ones can be found in (Bruni & A., 2003 ii. Backtracking Mechanisms When the algorithm fails to find an answer or an empty clause (contradiction) appears down the path of the search tree, it backtracks to a certain point and continues searching in another part of the tree. The DP backtracks to the most recently untoggled (complemented) literal and tests its complement branch. As mentioned earlier this will cost a lot of time for DP to discover that this part of the search space does not have a solution and search for a solution elsewhere. For backtracking in the DP procedure, much work has not been done as compared to branching decision. This is due to the fact that backtracking is an essential step in any algorithm to prove its completeness. Nevertheless, there are a number of proposals to improve the backtracking in the DP procedure.

28 iii. Upper Bounds

The improvements made in backtracking and branching heuristics are of practical interests. However, the experimental analysis of these improvements indicates that satisfiability could be solved in time less than 2 n . A number of people gave lower bounds for this problem but most of them rely on a certain structure or property that exists in the formula. The following are some of the achievements made to find an upper bound that is better than the trivial one.

29 a. Autarkness Principle

The first attempt to achieve a non-trivial upper bound for SAT was done by (Monien & Speckenmeyer, 1985). They introduced the notion of autarks which are partial assignments of variables. If all clauses that include the variables in the assignment are satisfied, then that assignment is an autark. They proved that the time complexity of their algorithm is When dealing with 3-SAT problem, the clauses with 2 literals help in reducing the search space. Schiermeyer was the first to make use of the number of clauses with 2 literals after the resolution step is made (Schiermeyer, 1993). He said that for the next branch, a 2-clause is chosen such that it produces at least one new 2-clause in every branch that follows. With the help n). (Kullmann, 1999) showed that the algorithm of Schiermeyer can perform better through a new concept called blocked clauses. A clause C is blocked for a literal l if every clause C' containing l has also another literal that is complemented with C. By making use of these blocked clauses, Kullmann proved that the algorithm in (Schiermeyer, 1993) [1] [2] [3]

[1]© 2013 Global Journals Inc. (US) Global Journal of Computer Science and Technology

[2]Communications of the ACM, 5, (pp. 394-397).

[3]© 2013 Global Journals Inc. (US)

4

Figure 1: 3 x and 4 x

respectively where m is the number of clauses and L is the length of the formula. An improvement was made to the second algorithm in (Hirsch, 2000) to become O(0.10299L

.1 2

).

e. Covering Codes (Danstin, et al., 2002) proposed a deterministic algorithm that is based on covering codes. This algorithm can be seen as a derandomization of (Schoning, 1999) algorithm that uses random walk model. The search space is divided into group of assignments say balls of some radius r. Each group or ball represents some assignment a and all assignments that differ with it in r variables. The algorithm checks in each ball if there is a satisfying assignment and if there is none in any ball then the formula is unsatisfied. The authors of (Danstin, et al., 2002) showed that the time complexity of this V.

.2 Conclusion and Future Work

The proposed does not require the clauses or the formula to have any specific structure to achieve a competitive upper bound which is a significant advantage over the existing algorithms in the literature where they derive their time complexity based on a property that must exist in the formula. The algorithm gives a new insight towards solving SAT. Most of the other algorithms are based on the classical rule of splitting the search space into regions and search for a solution in each one. The new perspective of the algorithm has the potential to design further effective SAT algorithms that outperforms the existing ones in theory and practice.

The implementation of the proposed algorithm will be considered in future work. The algorithm proposed here can also be improved. The time complexity of the proposed algorithm is based on preprocessing of clauses in the formula. This arrangement is so unlikely to exist in all clauses considered. That means that there exists a tighter upper bound for the algorithm but to achieve that the order in which clauses are considered should be more intelligent. To show that such an upper bound exists, many cases have to be covered and counted. Parallelisation of the proposed algorithm is also a potential future work.

[Marques-Silva and Sakallah ()] , J P Marques-Silva , K A Sakallah . *IEEE Transactions on Computers. GRASP-A Search Algorithm for Propositional Satisfiability* 1999. p. .

[Bruni ()] 'A Complete Adaptive Algorithm for Propositional Satisfiability'. R Bruni , S . Discrete Applied Mathematics 2003. p. 127.

[Davis and Putnam ()] 'A Computing Procedure for Quantification Theory'. M Davis , H Putnam . *Journal of Association for Computing Machinery* 1960. p. .

[Davis and Putnam ()] 'A Computing Procedure for Quantification Theory'. M Davis , H Putnam . *Journal of Association for Computing Machinery* 1960. p. .

[Danstin et al. ()] 'A Deterministic (2-2 / k+1)n Algorithm fork-SAT based on Local Search'. E Danstin , A Goerdt , E A Hirsch , R Kannan , J Kleinberg , C Papadimitriou . *Theoretical Computer Science* 2002. p. .

[Davis et al. ()] *A Machine Program for Theorem Proving*, M Davis , G Logemann , D Loveland . 1962.

[Schoning ()] 'A Probabilistic Algorithm for k-SAT and Constraint Satisfaction Problems'. U Schoning . *Proceedings of the 40th Annual Symposium on Foundations of Computer Science (FOCS)*, (the 40th Annual Symposium on Foundations of Computer Science (FOCS)) 1999. IEEE. p. .

[Hooker and Vinay ()] *Branching Rules for Satisfiability. GSIA Working Paper 1994-09. Pennsylvania: Graduate School of Industrial Administration*, J N Hooker , V Vinay . 1994. Pittsburgh. Carnegie-Mellon University

[Lynce and Marques-Silva ()] 'Building Stateof-The-Art SAT Solver'. I Lynce , J Marques-Silva . *Proceedings of the European Conference on Artificial Intelligence (ECAI)*, (the European Conference on Artificial Intelligence (ECAI)) 2002. p. 105.

[Pretolani ()] 'Efficient and Stability of Hypergraph SAT Algorithms'. D Pretolani . *Proceedings of DIMACS Challenge II Workshop*, (DIMACS Challenge II Workshop) 1993.

[Moskewicz et al. ()] 'Engineering an Efficient SAT Solver'. M Moskewicz , C Madigan , Y Zhao , L Zhang , S Malik . *Proceedings of the Design Automation Conference*, (the Design Automation Conference) 2001.

[Crawford and Baker ()] *Experimental Results on the Application of Satisfiability Algorithms to Scheduling Problems*, J M Crawford , A B Baker . 1994. AAAI-94.

[Stallman and Sussman ()] 'Forward Reasoning & Dependencydirected Backtracking in A System for Computeraided Circuit Analysis'. R M Stallman , G J Sussman . *Artificial Intelligence* 1977. 9 p. . (Artificial Intelligence 9)

[Li and Anbulagan ()] 'Heuristics Based on Unit Propagation for Satisfiability Problems'. C M Li , Anbulagan . *Proceedings of 15th International Joint Conference on Artificial Intelligence*, (15th International Joint Conference on Artificial IntelligenceNagoya, Japan) 1997. 1 p. .

[Freeman ()] *Improvements to Propositional Satisfiability Search Algorithms*, J W Freeman . 1995. Computer and Information Science, University of Pennsylvania (Ph. D. Dissertation. Department of)

[Kullmann ()] ‘New Methods for 3-SAT Decision and Worst Case Analysis’. O Kullmann . *Theoretical Computer Science* 1999. p. .

[Hirsch ()] ‘New Worst Case Upper Bounds for SAT’. E Hirsch . *Journal of Automated Reasoning* 2000. 24 p. .

[Devadas ()] ‘Optimal Layout via Boolean Satisfiability’. S Devadas . *Proceedings of ICCAD 89*, (ICCAD 89) 1989. p. .

[Kauts and Selman ()] ‘Planning as Satisfiability’. H Kauts , B Selman . *Proceedings of the 10th European Conference on Artificial Intelligence (ECAI 92*, (the 10th European Conference on Artificial Intelligence (ECAI 92) 1992.

[Rosen ()] K H Rosen . *Discrete Mathematics and Its Applications*, 1999. McGraw Hill. (4th Edition ed.)

[Barbour ()] ‘Solutions to The Minimization Problem of Fault-Tolerant Logic Circuits’. A E Barbour . *IEEE Transactions on Computers* 1992. 41 (4) p. .

[Schiermeyer ()] ‘Solving 3-Satisfiability in less than 1.579n’. I Schiermeyer . *Selected papers from Computer Science Logic 12*, 1993. 702 p. .

[Jeroslow and Wang ()] ‘Solving Propositional Satisfiability Problems’. R G Jeroslow , J Wang . *Annals of Mathematics & Artificial Intelligence* 1990. 1 p. .

[Monien and Speckenmeyer ()] ‘Solving Satisfiability in less than 2n steps’. B Monien , E Speckenmeyer . *Discrete Applied Mathematics* 1985. p. .

[Larrabee ()] ‘Test Pattern Generation Using Boolean Satisfiability’. T Larrabee . *IEEE Transactions Computer Aided Design* 1992. 1 p. .

[Cook ()] ‘The Complexity of Theorem Proving Procedures’. S Cook . *Proceedings of the 3rd Annual ACM Symposium on Theory of Computing*, (the 3rd Annual ACM Symposium on Theory of Computing) 1971. p. .

[Lynce and Silva ()] *The Effect of Nogood Recording in MAC-CBJ SAT Algorithms*, I Lynce , J P Silva . RT/4/2002. 2002. (Technical Report)

[Hirsch ()] ‘Two New Upper Bounds for SAT’. E Hirsch . *Proceedings of 9th Annual ACM Siam Symposium on Discrete Algorithms*, (9th Annual ACM Siam Symposium on Discrete Algorithms) 1998. p. .