VOL. 16, NO. 20, OCTOBER 2021 ISSN 1819-6608
ARPN Journal of Engineering and Applied Sciences

www.arpnjournals.com

EXPLORING A NOVEL STRATEGY FOR SAT

Edgardo S. Barraza Verdesoto[1], Edwin Rivas Trujillo[2] and Henry Montaña Quintero[2]
[1]Universidad Fundación Tecnológica Autónoma de Bogotá, Colombia
[2]Universidad Distrital Francisco José de Caldas Bogotá, Colombia
E-Mail: hmontanaq@udistrital.edu.co

ABSTRACT

The SAT problem is a very important topic in the computer science, its condition of NP problem has been a focus of studies and approaches for resolving it in a low computational cost as well in time as in space, e.g., the International SAT competition that started in 1992 and remain in force until the date. Its importance stems from the fact that resolves decision problems that can be very complex because of its number of restrictions. This paper presents a novel strategy based on clusters of literals that conforms the nodes of a dynamic tree that resolves the SAT problem, the implementation of which includes the clause as the cluster and a heuristic function to select it. In addition, the implementation is compared against recognized solvers that has been winners of competitions of SAT solvers.

Keywords: SAT, heuristic function, clusters, literal, clause.

1. INTRODUCTION

Satisfiability (SAT) [1] is a problem that search to give a logic answer to a formula expressed in Conjunctive Normal Form (CNF). This paper introduces a strategy to calculate satisfiability by using a novel technique which uses the clause as the unit to resolve it.

Typical algorithms that resolve SAT repeatedly use the procedure named *unit propagation* over the formula *CNF* [2]. Informally, this operation assigns a logical value to a literal which decreases the number of literals in the clauses; the routine stops when reaches satisfiability or unsatisfiability. On the other hand, when the procedure leads to unsatisfiability a reverse action, named backtracking, applies the logical value contrary to the chosen for the last literal used. DPLL algorithms [3] [4] [5] [6] are solvers based on this technique. SAT solvers have applications in several fields such as Data Mining [7], Big Data [8] [9] or electronic applications [10].

This paper proposes to use the clauses as a cluster of literals to evaluate by reducing the scope generated by the *CNF*. The strategy is to select the best clause based on weights calculated by a heuristic function in each stage of the algorithm, that require it; The literals inside of the selected clause will take the value "true" one by one for the evaluation of the *CNF*. In the backtracking process, the failed literal will take the value "false" and join with the next literal valued "true"; This process involves *unit propagation* and *pruning* at the same time. The implementation of this method is simple and have the same or better performance than algorithms highly recognized

The document has the following organization, first of all, an overview about the *SAT* solvers does. Second, the algorithm is mathematically formalized. Third, an implementation is proposed for performing analysis and benchmarking against various popular and recognized algorithms. Finally, the conclusions are presented.

2. OVERVIEW ABOUT THE SAT AND SOLVERS

SAT is a set of decision problems whose instances are Boolean expressions written in a *CNF* format which should answer the question: ¿Is there any logic assignment that can make true the expression? *CNF* is a logical expression with elements named clauses joined with the and-logical operator. The clauses contain several logical variables named literals grouped by means the or-logical operator. The maximum number of literals that a clause contains determine the name of the *SAT* problem, e.g., if this amount is 2 then the *SAT* problem is named 2-SAT. Additionally, if a literal represents a proposition, then the *SAT* problem will belong to the propositional satisfiability. *SAT* was the first problem declared *NP-Complete* [11], its computational complexity increases exponentially when new literals are added within the clauses. There are *2-SAT* instances that can be solved in polynomial time, but problems with more literals are still difficult to solve.

There are 2 types of algorithms to solve *SAT* instances whose based on the Davis-Putnam algorithm, and other on stochastic methods. The former group carries out a deterministic search; among those well recognized are Chaff [12] and GRASP [13]. The stochastic algorithms are non-deterministic; Hence, the answer may not find it; WalkSAT [14] [15] is an example of them. Modern solvers have been substantially improved due to the integration with techniques such as conflict analysis, learning, and non-chronological backtracking.

3. AN ALGORITHM BASED ON CLAUSES

The *unit-propagation* technique simulates a unit-clause by making that its unique literal takes a logic value (true/false). The *backtracking* procedure is the process that made to the literal chosen takes the opposite logic value when the formula become inconsistent. These two procedures are the basis of the *DPLL algorithms* [3] [4] [5] [12] [13] [14] [15], and their aim is to prune the tree of possibilities at each step in an efficient manner. Although this type of algorithm uses a clause as its main component, it contains a sole literal which is the pivot of the process;

therefore, the literals are the foundation of these algorithms.

This paper proposes a type of algorithm based on groups of literals and suggests to the clause, as well, shows that this strategy can be equally effective and efficient as the typical algorithms. The process starts by selecting the best clause and making it the node-root of the search tree for generating a good balancing. In addition, modal techniques in the middle of the operation make the prune of the tree.

The following are some definitions about concepts under supposing that exist a problem given and structured as a formula *CNF*. In this proposal, CNF and literal maintain their definitions, but the definition of clause changes a little.

Definition 1 (Set of literals $\mathcal{L}$). Let us define a set of *literals* which will be named $\mathcal{L}$ and will contain all literals.

Definition 2 (Clause $\mathbb{C}$). Let us define a clause as a subset of literals which will be named $\mathbb{C}$. Every element belonging to $\mathbb{C}$ is a literal that belongs to $\mathcal{L}$.

Definition 2 (Set K). Let us define the set of all clauses which will be named **K**.

Definition 3 (Formula CNF). Let us define a subset of clauses which will be named **CNF** and is contained in K.

Note: There is a function that maps a subset of clauses to the set of all *CNF* possible that will not be defined because of this is not relevant for the algorithm.

3.1 Weights

The selection procedure of a clause requires to compare them; Hence, every clause must have a value that stands for its weight in the *CNF*. The algorithm computes the weight of the clauses at begin and when any clause loses a literal. The computation must be simple because of the algorithm repeats this method several times. The next paragraphs expose an adequate technique to these requirements.

Definition 4 (Function $\mathcal{L}_w$). Let us define an injective function which will be named $\mathcal{L}_w$ and informally *the weight of ℓ respect to a CNF*. The function maps each literal that belongs to a CNF to a natural number ($\mathbb{N}$). The function works as follows, given a literal ℓ belong to a CNF:

a. To choose the contrary literal $\bar{\ell}$.

b. To calculate the number of clauses inside CNF that contain $\bar{\ell}$. The number obtained will be named ℓ_w.

The pair resulting will be (ℓ, ℓ_w). The target of this function is to determine the number of clauses that are affected if the literal ℓ is assigned with the value "true".

Definition 5 (Function $\mathbb{C}_w$). Let us define an injective function which will be named $\mathbb{C}_w$ and informally *the weight of c respect to a CNF*. The function maps each clause that belongs to a CNF to a natural number ($\mathbb{N}$). The function works as follows, given a clause c belongs to a CNF:

- To compute the minimum ℓ_w calculated for each literal inside of c. The number obtained will be named c_w.

The pair resulting will be (c, c_w). This result will be analyzed later.

3.2 Selecting the Best Clause

The search tree has clauses as their nodes; these contain two or more literals that generate the possibilities of truth for the clause. The selection of the clauses that conform the tree is dynamic; hence, it is necessary to compare their weights continuously.

Definition 5 (Maximal set $\mathcal{H}_c$). Let us define a set named $\mathcal{H}_c \subseteq K$ such that $\mathcal{H}_c$ is the maximal set of K respect to the range of $\mathbb{C}_w$. The set contains all clauses with the largest c_w.

Algorithm 1 (Selecting the clause).

1. To Read the *CNF*.
2. To Apply the function $\mathcal{L}_w$ to all literals.
3. To Apply the function $\mathbb{C}_w$ to all clauses.
4. To find the maximal set $\mathcal{H}_c$
5. Select whatever clause of $\mathcal{H}_c$

It is important to note that in each step of the SAT evaluation process, the *CNF* changes and the functions $\mathcal{L}_w$ and $\mathbb{C}_w$ must be calculated dynamically generating a new maximal set, hence, the implementation of this algorithm and its data structures must be efficient.

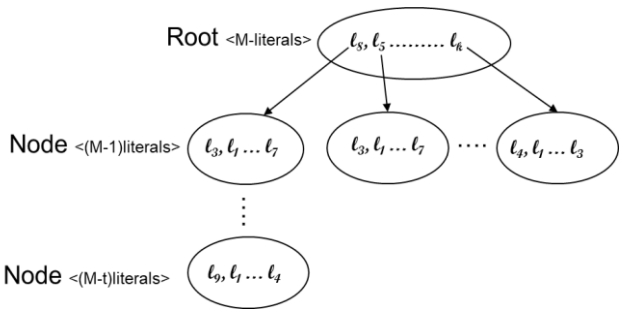


Figure-1. Tree of clauses by E. Source: Authors

Figure-1 presents the resulting tree when Algorithm 1 is executed. The exploration of the truth values through the literals in each node creates a new node dynamically.

3.3 Node Preprocessing

The clauses chosen contains literals evaluated such as a search tree by following the Algorithm 2.

Algorithm 2

1. To apply the *Algorithm 1*
2. Node preprocessing: To transform the node-clause selected in a chain of expressions.
3. To assign the value "true" to the first/next expression in the clause selected.
4. If *SAT* then stop.
5. Else Go to step 1

The step 2 shows a preprocessing inside of the clause which transforms its contents. The idea underlying in this routine is to acknowledge and learn that the logical assignment made in the previous literal or expression failed and this is not desirable that occurs again. The clauses with two literals have a specific attention; the following theorems explain it.

Theorem 1. The expression in a clause-node with two literals $(A \vee B)$ can be replaced by $\left((\bar{B} \wedge A) \vee B\right)$.

Proof

If B generates inconsistency with the assignation "true", then the literal A is the sole possibility to reach successful from this node. The expression $\bar{B} \rightarrow A$ represents this reasoning which is equivalent to $(B \vee A)$; The same way for A. Therefore, if added a clause such as $(A \vee \bar{A})$ or $(B \vee \bar{B})$ to the current *CNF*, the outcomes of the evaluation hold:

$(B \vee \bar{B}) \wedge (A \vee B)$
$(B \wedge A) \vee (B \wedge B) \vee (\bar{B} \wedge A) \vee (\bar{B} \wedge B)$ by contradiction and simplification
$(\bar{B} \wedge A) \vee [(B \wedge A) \vee B]$ by reordering and simplification
$(\bar{B} \wedge A) \vee B$
Q.E.D.

Theorem 2. The expression of a clause-node with more than two literals $(A_0, A_1, \dots, A_k)$ can be replaced by $(A_0, [\bar{A}_0 \cdot A_1], \dots, [\bar{A}_0 \cdot \bar{A}_1 \dots . \bar{A}_{k-1} \cdot A_k])$. The symbol "," represents the logical operator OR and "·" the logical operator AND.

Proof

If the literals of a node-clause have an order instituted by its weighs, then the literals must be evaluated in this specific order (c*hain*). Hence, if the clause is a chain, then it can apply the Theorem 1 as follow:

Step 0 $(A_0,\bar{A}_0) \wedge (A_0, A_1, \dots, A_k) \equiv (A_0, [\bar{A}_0 \cdot A_1], \dots, [\bar{A}_0 \cdot A_k])$
Step 1.
If A_0 is successful then finish
Else (*Theorem 1 over* A_1*)*
$(A_1, \bar{A}_1) \wedge ([\bar{A}_0 \cdot A_1], [\bar{A}_0 \cdot A_2], \dots, [\bar{A}_0 \cdot A_k])$
$\equiv ([\bar{A}_0 \cdot A_1], [\bar{A}_0 \cdot \bar{A}_1 \cdot A_1], \dots, [\bar{A}_0 \cdot \bar{A}_1 \cdot A_k])$
Step k-1.
If A_{k-2} is successful then finish
Else (*Theorem 1 over* A_{k-1}*)*
$(A_{k-1}, \bar{A}_{k-1}) \wedge ([\bar{A}_0 \cdot \bar{A}_1 \cdots A_{k-1}], [\bar{A}_0 \cdot \bar{A}_1 \cdots \bar{A}_{k-2} \cdot A_k])$
$\equiv ([\bar{A}_0 \cdot \bar{A}_1 \cdots A_{k-1}], [\bar{A}_0 \cdot \bar{A}_1 \cdots \bar{A}_{k-2} \cdot \bar{A}_{k-1} \cdot A_k])$
Step k.
If A_{k-1} is successful then finish
Else
$([\bar{A}_0 \cdot \bar{A}_1 \cdots \bar{A}_{k-2} \cdot \bar{A}_{k-1} \cdot A_k])$

Because of the node-clause is a *chain*, all results generated at each step can be linked in a sole node, as follow:

$(A_0, [\bar{A}_0 \cdot A_1], \dots, [\bar{A}_0 \cdot \bar{A}_1 \dots . \bar{A}_{k-1} \cdot A_k])$
Q.E.D.

For better understanding, the following Tables 1 and 2, and Figures 2 and 3 exhibit the procedures described in the las section with a 3-CNF SAT. The Algorithm 2 is a loop that dynamically creates the branches of the tree. The first step is to apply the Algorithm 1 (see Table-1).

Table-1. Algorithm 1 applied to a CNF. Source: Authors

Clause	Variables	Weight		Select
		Literals	Clause	
C1	*(~p, ~t, ~s)*	*~p=1, ~t=2, ~s=2*	*1*	
C2	*(~p, q, s)*	*~p=1, q=2, s=1*	*1*	
C3	*(p, ~t, r)*	*p=2, ~t=2, r=1*	*1*	
C4	***(~q, t, ~r)***	***~q=2, t=3, ~r=2***	***2***	*****
C5	*(~q, ~t, s)*	*~q=2, ~t=2, s=1*	*1*	
C6	*(q, t, r)*	*q=2, t=3, r=1*	*1*	

The clause C4 is the clause selected as the node-root, then, its literals conform a *chain* according to the Theorem 2 (see Figure-2).

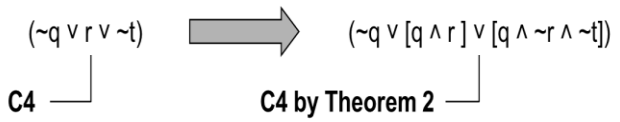


Figure-2. To create the node-clause according to the Theorem 2. Source: Authors

The following step is to evaluate the *CNF* with the current expression in the node-clause, this modifies the original *CNF* and the Algorithm 2 returns to the begin. If there are clause with two literals, then, the *Algorithm 1* applies as in Table-2.

Table-2. Algorithm 1 applied to the new CNF. Source: Authors

Clause	Variables	Weight		Select
		Literals	Clause	
C1	(~p,~t, ~s)	~p=1, ~t=1, ~s=1	1	
C2	*(~p, s)*	*~p=1, s=1*	***1***	*****
C3	(p, ~t, r)	p=2, ~t=1, r=0	0	
C6	*(t, r)*	*t=2, r=0*	*0*	

The tree expansion is shown in Figure-3, here is applied the Theorem 1. This process continues such as a search tree with an *undo* procedure as part of it.

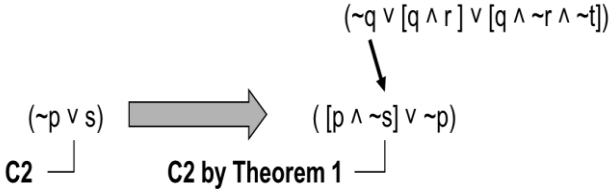
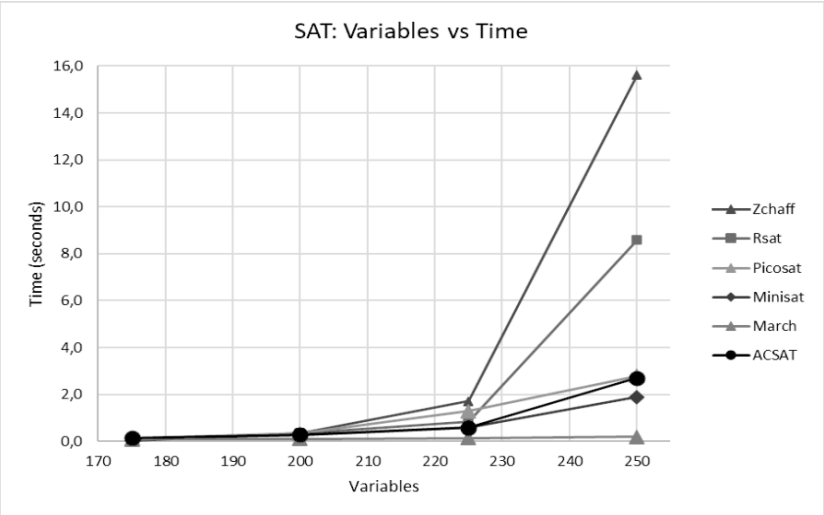


Figure-3. The new node-clause according to the Theorem 1. Source: Authors

4. IMPLEMENTATION AND ANALYSIS

The algorithm implementation (ACSAT) used the C/C++ language and the testing ran on a computer very

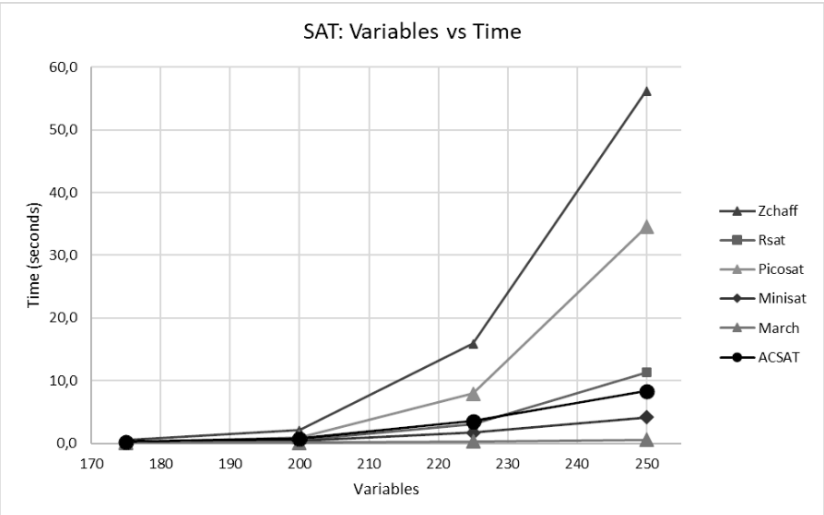
simple with two cores. The benchmark compared *ACSAT* against relevant algorithms [16] such as Zchaff [12], Picosat [17], Minisat [18], Rsat [19], and March [20], in *UNSAT* and *SAT* modalities. Figures 4 and 5 show the results.



		Average-time (seconds)					
Num-Clauses	Num-Vars	Zchaff	Rsat	Picosat	Minisat	March	ACSAT
753	175	0,0	0,1	0,1	0,1	0,1	0,1
860	200	0,4	0,4	0,3	0,3	0,1	0,3
960	225	1,7	0,9	1,3	0,6	0,2	0,6
1065	250	15,6	8,6	2,8	1,9	0,2	2,7

Figure-4. Comparison among Algorithms for SAT by E. Source: Authors

The benchmark for *SAT* (Figure-4) shows that ACSAT has a response near to Picosat which is a good algorithm to resolve this kind of problems. On the other hand, Figure-5 shows the benchmark for *UNSAT* problems, also, with good performance.



		Average-time (seconds)					
Num-Clauses	Num-Vars	Zchaff	Rsat	Picosat	Minisat	March	ACSAT
753	175	0,6	0,2	0,2	0,2	0,1	0,3
860	200	2,2	0,6	1,0	0,5	0,2	0,8
960	225	15,9	3,1	8,0	1,7	0,4	3,6
1065	250	56,2	11,3	34,6	4,2	0,6	8,3

Figure-5. Comparison among Algorithms for UNSAT. Source: Authors

5. CONCLUSIONS

The strategy proposed in this paper founds on literals groups such as the clause; this allows visualize properties and techniques more sophisticated addressed to the relation between literals and clauses, and, of course, to prune of the search tree.

Algorithms such as Zchaff founds their performance in the unit-clause concept (*unit propagation*), and hence, in the set of literals of the *CNF* without establishing the relationship between them; this obligates to the algorithm uses techniques such as the learning and the non-chronological backtracking, among others.

To use weights for the literals and clauses and it's dynamically assessment is a powerful tool to select an adequate candidate to be a node in the search tree as the performance graphs shows. A good way to improve this algorithm could be to find better heuristic functions that establish closer relationships between literals and clauses.
Finally, it is significant to mention that the benchmark was against mature algorithms which have been winners of competition in several SAT modalities. Consequently, the ACSAT algorithm, in this early implementation, takes in a behavior that is near to these algorithms; this creates good expectations about implementations more refined.

This paper introduces a strategy to resolve SAT based on clusters. The implementation built showed a good performance respect to others solvers. SAT problems with a large amount of restrictions are the adequate context for this strategy because of the algorithm founds its performance in a heuristic function that makes relationships between groups of literals.

REFERENCES

[1] M. Järvisalo, D. Le Berre, O. Roussel & L. Simon. 2012. The international SAT solver competitions. Ai Magazine. 33(1): 89-92.

[2] M. Davis and H. Putnam. 1960. A Computing Procedure for Quantification Theory. Journal of the Association for Computing Machinery. 7: 201-215.

[3] M. Davis, G. Logemann and D. Loveland. 1962. A Machine Program for Theorem Proving. Communications of the ACM. 5(7): 394-397.

[4] M. R. Krom. 1967. The Decision Problem for a Class of First-Order Formulas in which all Disjunctions are Binary. Zeitschrift für Mathematische Logik und Grundlagen der Mathematik. 13(1-2): 15-20.

[5] B. Aspvall, M. F. Plass and R. E. Tarjan. 1979. A linear-time algorithm for testing the truth of certain quantified boolean formulas. Information Processing Letters. 8(3): 121-123.

[6] R. Brummayer, F. Lonsing and A. Biere. 2010. Automated Testing and Debugging of SAT and QBF Solvers. Theory and Applications of Satisfiability Testing, Lecture Notes in Computer Science. Vol. 6175.

[7] A. Hidouri, S. Jabbour, B. Raddaoui & B. B. Yaghlane. 2020. A SAT-Based Approach for Mining High Utility Itemsets from Transaction Databases. Proceeding of the International Conference on Big Data Analytics and Knowledge Discovery, Bratislava, Slovakia, Big Data Analytics and Knowledge Discovery. pp. 91-106.

[8] H. Hong, L. Khan, A. Gbadebo, Z. Shaohua & W. Yong. 2018. A Complex Task Scheduling Scheme for Big Data Platforms Based on Boolean Satisfiability Problem. 2018 IEEE International Conference on Information Reuse and Integration (IRI), Salt Lake City, UT. pp. 170-177.

[9] H. Huang, L. Khan & S. Zhou. 2020. Classified enhancement model for big data storage reliability based on Boolean satisfiability problem. Cluster Comput. 23: 483-492.

[10] V. G. Bogdanova & S. A. Gorsky. 2018. Scalable parallel solver of Boolean satisfiability problems. 2018 41st International Convention on Information and Communication Technology, Electronics and Microelectronics (MIPRO). IEEE.

[11] S. Cook. 1971. The Complexity of Theorem Proving Procedures. Proceeding of the 3rd Ann. ACM Symp. On Theory of Computing, Association for Computing Machinery. pp. 151-158.

[12] M. Moskewicz, C. Madigan, y. Zhao, L. Zhang and S. Malik. 2001. Chaff: engineering an efficient SAT solver. Proceeding of the 38th ACM/IEEE Design Automation Conference, Las Vegas, Nevada. pp. 530-535.

[13] J. Marques-Silva and K. Sakallah. 1999. Grasp: A Search Algorithm for Propositional Satisfiability. IEEE Transactions on Computers. 48(5): 506-521.

[14] S. Liu. 2015. An Efficient Implementation for WalkSAT. https://arxiv.org/abs/1510.07217.

[15] R. H. Russel. 2019. A Probabilistic Approach to Satisfiability of Propositional Logic Formulae. https://arxiv.org/abs/1912.02150v1.

[16] D. Kroening & O. Strichman. 2016. Decision procedures. An algorithmic point of view. Texts in

Theoretical Computer Science, An EATCS Series, Berlin, Springer, 2nd edition.

[17] A. Biere, PicoSAT. 2008. Essentials. Journal on Satisfiability. Boolean Modeling and Computation (JSAT), Delft University. 4: 75-97.

[18] N. Eén and N. Sörensson. 2003. An Extensible SAT-solver. Proceeding of the International Conference on Theory and Applications of Satisfiability Testing, Lecture Notes in Computer Science. 2919: 502-518.

[19] K. Pipatsrisawat and A. Darwiche. 2006. RSat 1.03: SAT Solver Description. Technical report D152 Automated Reasoning Group, Computer Science Department, University of California, Los Angeles.

[20] M. Heule, M. Dufour, J. van Zwieten and H. van Maaren. 2005. March_eq: Implementing Additional Reasoning into an Efficient Look-Ahead SAT Solver. Theory and Applications of Satisfiability Testing, Lecture Notes in Computer Science. 3542: 345-359.