Exploration via Random Walks in CDCL SAT Solving amid Conflict Depression

Md Solimul Chowdhury, Martin Müller, and Jia-Huai You

Department of Computing Science, University of Alberta
Edmonton, Alberta, Canada.
{mdsolimu, mmueller, jyou}@ualberta.ca

Abstract. The efficiency of Conflict Driven Clause Learning (CDCL) SAT solving depends crucially on finding conflicts at a fast rate. The development of branching heuristics such as VSIDS, CHB and LRB reflects this goal. Here we take a closer look at the way in which conflicts are generated over the course of a CDCL SAT search. Our case study with the VSIDS branching heuristic shows that conflicts are typically generated in short bursts, followed by a *conflict depression* phase when the search fails to generate any conflicts in a span of decisions. The lack of conflict indicates the ineffectiveness of the variables that are currently ranked highest by the branching heuristic for conflict generation. Based on this analysis, we propose an exploration strategy with the goal of escaping from conflict depressions more quickly. To probe the future search space, our CDCL SAT solving algorithm *expSAT* randomly samples variable selection sequences in order to learn an updated heuristic from the generated conflicts. The branching heuristic deployed in *expSAT* combines these updates with the standard VSIDS activity scores. An empirical evaluation demonstrates the performance gains with the *expSAT* approach.

1 Introduction

Modern CDCL SAT solvers have become the enabling technology for many real-world domains, such as hardware design verification [6], planning [19], and encryption [7]. The key decision-making step in a CDCL SAT solver is selecting a variable from the current set of unassigned variables using a *branching heuristic*, before making a boolean assignment to it. Variable selection has a dramatic effect on search efficiency. Popular branching heuristics include *variable state independent decaying sum* (VSIDS) [14] and its variants, *learning rate based* (LRB) [11], and *conflict history based* (CHB) [10]. These heuristics reward the variables taking place in recent conflicts. The intuition is that assignments of these variables are likely to generate further conflicts leading to learned clauses and thus pruning the search space.

The metric based on *global learning rate* (GLR) [12] measures the number of conflicts obtained per branching decision. In CDCL SAT, a single decision may generate multiple conflicts. State-of-the-art branching heuristics, such as, LRB, VSIDS or CHB have average GLR values of about 0.5 [12], i.e., average one conflict per two decisions.

In this work, we first report a study on the phenomenon of conflicts generation during CDCL search. We find that there are clear non-random patterns of bursts of conflicts

followed by longer phases of *conflict depression (CD)*. To correct the course of such a search, we propose to use exploration to combat conflict depression. We therefore propose an extension of SAT solving, called *expSAT*, which applies random walks in the context of CDCL SAT solving. In a conflict depression phase, random walks help identify more promising variables for branching. As a contrast, while exploration explores *future* search states, VSIDS relies on conflicts generated from the *past* search states. We report the following findings.

- By an empirical study of conflict depression using one of the strongest purely VSIDS-based solver, *glucoseLCM*[1], on recent SAT competition benchmarks, we show that CD phases occur at a high rate and often with long average duration.
- We formulate *expSAT*, an exploration-driven extension of VSIDS-based SAT solvers. We show a scheme of setting the exploration parameters dynamically at runtime to control when to perform exploration and how much exploration to perform.
- We perform an empirical evaluation of *expSAT* by extending two leading solvers *glucoseLCM* and *Maple_CM*[2], called *expGLCM* and *expM_CM*. On the main track benchmarks of SAT Competition-2018 (abbreviated as SAT-2018), *expGLCM* solves 9 more instances (than its baseline) and *expM_CM* solves 2 more; both achieve a lower PAR-2 score.[3] On 40 hard instances from SHA-1 preimage attack cryptographic benchmarks, *expGLCM* solves 3, as opposed to 2 by *glucoseLCM*, and reduces the average run-time by half. In comparison, the winner of SAT-2018, *MapleLCMDistChronoBT* solves only 1 instance.

2 Preliminaries

We assume familiarity with CDCL SAT solving [2]. We briefly review some concepts.

The VSIDS Heuristic: VSIDS [14] is a popular family of dynamic branching heuristics. We focus on exponential VSIDS as used in *glucoseLCM*. VSIDS maintains an *activity score* for each variable in a given SAT formula. It increases the activity score of each variable that is involved in conflict resolution by a *variable bumping factor* g^z, where $g > 1$ is a constant and z is the count of the number of conflicts in the search so far. This strongly favors variables that participated in the most recent conflicts.

The Literal Block Distance (LBD) Score and Glue Clauses: The LBD score [1] of a learned clause c is the number of distinct decision levels in c. If $\text{LBD}(c) = n$, then c contains n propagation blocks, where each block has been propagated within the same branching decision. Variables within one block are considered to be closely related, and learned clauses with lower LBD scores are likely of higher quality. A *glue clause* c is one with $\text{LBD}(c) = 2$.

Global Learning Rate (GLR): Suppose a CDCL solver takes d decisions to solve a given SAT instance $\mathcal{F}$ and generates q conflicts for these d decisions. The GLR of the

[1] glucose 4.2.1, which implements the *learned clause minimization* (LCM) technique [13] on top of glucose 4.1.

[2] Source: http://sat2018.forsyte.tuwien.ac.at/solvers/main_and_glucose_hack/

[3] Defined as the sum of all run-times for solved instances + $2 * timeout$ for unsolved instances; lowest score wins.

solver for $\mathcal{F}$ is defined as $\frac{q}{d}$. GLR measures the overall tendency of a solver for conflict generation for a given problem.

Test Environment: Our first baseline is *glucoseLCM*, one of the best CDCL SAT solvers that use VSIDS exclusively as a branching heuristic. Our second baseline is *Maple_CM*, the second runner up of the main track of SAT-2018, which uses VSIDS and LRB.

All experiments presented in this paper were run on a workstation with 48GB RAM and a processor clock speed of 2.93GHz. Two test sets were used in experiments.

(a) **Test Set 1** contains 400 instances from the main track of SAT-2018 and is run with a time limit of 5k seconds per instance.

(b) **Test Set 2** consists of 40 hard instances from SHA-1 preimage attack cryptographic benchmark, which are generated with the instance generator described in [16]. This benchmark is known to be challenging for current SAT solvers. The difficulty of instances is regulated by three parameters, *rounds*, *hash-bits*, and *message-bits*. We use the following value ranges of these parameters: rounds between 23-30, hash-bits between 66-97, and message-bits 0. We set time limit to be 36k seconds per instance.

3 Conflict Depression

Consider a run of a CDCL SAT solver $\mathcal{S}$ which makes a total of d decisions. In each decision, a variable is selected according to a branching heuristic. Each decision i ($0 < i \leq d$) leads to some number $c_i \geq 0$ of conflicts. We can represent the conflict history of decisions by a sequence $\{c_1, c_2, \ldots, c_d\}$. We define a *conflict depression (CD) phase* as a sequence of one or more consecutive decisions with no conflict. We further define the *length* of a CD phase as the number of decisions in it. For example, the conflict history of decisions $(1, 0, 0, 0, 0, 4, 2, 1, 0, 1, 0, 0)$, where a number represents the number of conflicts at that decision, contains 3 CD phases: one starting at decision c_2 with length 4, one starting at decision c_9 with length 1, and one at the end with length 2.

Decision Rates, CD Phase Rates, and FDC: Suppose $\mathcal{S}$ takes a of total d decisions, encounters u CD phases, and gives r restarts. We define *decision rate* (DR) as d/r and *CD phase rate* (CDR) as u/r. We also define the *fraction of decisions with conflicts* (FDC) as a measure related to, but different from, the global learning rate (GLR): FDC is the fraction of decisions which produce at least one conflict. This measure counts decisions with conflicts, not conflicts, which is different since some decisions cause multiple conflicts.

3.1 Conflict Depression of VSIDS in *glucoseLCM*

In this section, we study conflict depression empirically, using VSIDS as a representative CDCL branching heuristic, and *glucoseLCM* as the CDCL SAT solver. We collect the statistics for each search of an instance on DR, CDR, Average CD phase length, GLR, and FDC.

CDR and average CD Phase Length: The left plot of Figure 1 shows the decision rates (DR), CD phase rate (CDR) and average CD phase length (in log scale) for instances in Test Set 1, where the instances are sorted by average CD phase length. We

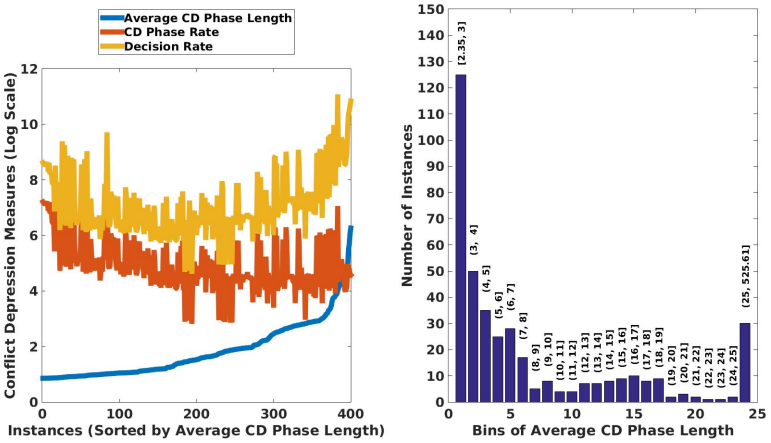

Fig. 1. Conflict Depression plots for Test Set 1 with *glucoseLCM*

observe that the average CD phase length is short for most instances, which, however, consists of multiple decisions (blue line, left plot). Furthermore, irrespective to their average CD phase length, for all-most all of the instances CD phases (orange line) occur at a high rate w.r.t. the decision rates (yellow line).

The histogram on the right side of Figure 1 shows the distribution of average length of CD phases. This average ranges from 2.35 to 525.61. Total 125 instances have very short length (at most 3, leftmost bin). The distribution is heavy tailed, with over 30 instances with average length greater than 25 (rightmost bin).

Overall, the data indicates that for *glucoseLCM* on Test Set 1, conflict depressions occur frequently and often last over multiple decisions (high average CD phase length), which is a serious problem for search efficiency.

GLR and FDC: In Table 1, column C shows that the average GLR values for all three types of problems are close to 0.5, so the number of conflicts is about half the number of decisions. In contrast, the FDC values in column D are much lower, averaging 0.2506 over all instances. Therefore, about 75% of all decisions do not produce any conflict!

Type (A)	#Instances (B)	GLR (C)	FDC (D)
Satisfiable	95	0.4915	0.2562
Unsatisfiable	97	0.4718	0.2543
Unsolved	208	0.5060	0.2543
All	400	0.4943	0.2506

Table 1. Statistics for Test Set 1 on GLR and FDC with *glucoseLCM*

To summarize, our analysis shows that the typical search behavior alternates between high-conflict bursts and longer conflict depression phases. We conjecture that the beginning of a CD phase corresponds to a new region in the search space where VSIDS scores are not a good predictor of a variable's future performance. In such a phase,

VSIDS fails to generate any conflicts. No conflict means no learned clauses, and the solver only performs truth value propagations.

4 The *expSAT* algorithm

During a CD phase, VSIDS is ineffective. Is it possible to correct the course of the search by identifying promising variables that are currently under-ranked by VSIDS? We have formulated a solver framework named *expSAT*, which performs random explorations that probe into the future search space. The goal is to find branching variables that are likely to lead to *good* conflicts from which important clauses can be learned.

Let $\mathcal{F}$ be a CNF SAT formula. In addition to $\mathcal{F}$, *expSAT* also accepts four *exploration parameters* nW, lW, p_{exp} and ω, where $1 \leq nW, lW \leq uVars(\mathcal{F})$, $0 < p_{exp}, \omega \leq 1$, with $uVars(\mathcal{F})$ be the set of unassigned variables at any given state of the search. These parameters control the exploration aspects of *expSAT*.

Given a CDCL SAT solver, *expSAT* modifies it as follows:

- Before each branching decision, if a substantially large CD phase[4] is detected then with probability p_{exp}, *expSAT* performs an *exploration episode*, consisting of a fixed number *nW* of random walks. Each walk consists of a limited number of *random steps*. Each such step consists of the uniform random selection of a currently unassigned *step variable* and assigning a boolean value to it using a standard CDCL *polarity* heuristic, followed by unit propagation (UP). A walk terminates either when a conflict occurs during UP, or after a fixed number *lW* of random steps have been taken. Figure 2 illustrates an exploration episode amid a substantially large CD phase.
- Each variable v that participates in any of the nW walks receives an exploration score $expScore$, which is the average of the *walk-scores* of v. The walk-score $ws(v)$ for v is: $\frac{\omega^d}{\text{LBD}(c)}$, if the walk in which v participates ends with a conflict, and 0 otherwise. d is the distance between the step for v and the step at which the conflict has occurred: If v was assigned at some step j during the current walk, and the conflict occurred after step $j' \geq j$, then $d = j' - j$. $0 < \omega \leq 1$ is the decay factor and $\text{LBD}(c)$ is the LBD score of the learned clause c, which is derived from the conflict that terminates the walk. We assign credit to all the step variables in a walk that ends with a conflict and give higher credit to variables closer to the conflict. This approach is patterned on reward decay in reinforcement learning [22].
- The new branching heuristic adds VSIDS score and *expScore* for each unassigned variable. An unassigned variable with maximum combined score is selected for branching.
- All other components remain the same as in the underlying CDCL SAT solver.

[4] Given a run of a solver on a given SAT instance, let R be the measure $\frac{\#decisions\_without\_conflicts}{\#decisions\_with\_conflicts}$, where *#decisions_with_conflicts* (resp. *#decisions_without_conflicts*) are the number of decisions with conflicts (resp. without conflicts) encountered by the search so far. $R + 1$ is the average number of decisions taken until one generated a conflict. In *expSAT*, we call a CD phase *substantial*, if its length is greater than *R*.

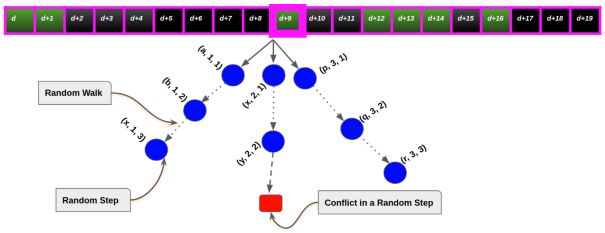

Fig. 2. The 20 adjacent cells denote 20 consecutive decisions starting from the d^{th} decision, with $d > 0$, where a green cell denotes a decision with conflicts and a black cell denotes a decision without conflicts. Say that amid a CD phase, just before taking the $(d + 9)^{th}$ decision, *expSAT* performs an exploration episode via 3 random walks each limited to 3 steps . The second walk ends after 2 steps, due to a conflict. A triplet (v, i, j) represents that the variable v is randomly chosen at the j^{th} step of the i^{th} walk.

Example: Using the three random walks of Fig. 2, we show how to compute ws and $expScore$ of variables. Only the second walk produces a conflict at the second step. Let c be the derived clause from this conflict, with LBD$(c) = m$.

The walk and exploration scores for all variables participating in the first and third random walk are 0. The variables x and y which participate in the second walk receive non-zero walk and exploration scores: $ws(y) = \frac{\omega^{2-2}}{m} = \frac{\omega^0}{m} = \frac{1}{m}$ and $ws(x) = \frac{\omega^{2-1}}{m} = \frac{\omega^1}{m}$. Since y only appears in this walk, but x appears in two walks, the exploration scores of y and x are $\frac{1}{m}$, and $(\frac{\omega}{m})/2$, respectively.

The Parameter Adaptation Algorithm: A parameter setting that is effective for one instance may not be effective for another. We use an adaptive algorithm *paramAdapt* to dynamically control when to trigger exploration episodes, and how much exploration to perform in an exploration episode. The three exploration parameters p_{exp}, nW, and lW are adapted between CDCL restarts based on the search behavior.

A parameter setting is a triple $P = (p_{exp}, nW, lW)$, which is updated at the beginning of each restart by *paramAdapt* by comparing the exploration performance of the two most recent search periods, the period between the latest two restarts and the period before it.

The search in *expSAT* starts with a default value P^{def} of P. $paramAdapt$ keeps track of the following statistics about all exploration steps within a period: the number of random steps $rSteps$, the number of conflicts c, the number of glue-clauses gc, the mean LBD value, lbd, of the learned clauses.

With fixed weights $w_1 > w_2 > w_3$, an *exploration performance metric* (EPM) is defined as $\frac{w_1 \times gc + w_2 \times c}{rSteps} + w_3 * \frac{1}{lbd}$

This performance metric rewards finding glue clauses (most important), finding any conflict (very important), and learning clauses with low LBD score (important).

At each restart, the algorithm computes a new EPM σ^{new} and compares (the comparison starts after the second restart) it with the prior one σ^{old}, and update the parameter setting P^{old} just used to get a new setting P^{new}.

- If $\sigma^{new} < \sigma^{old}$, the performance of exploration is worse than before. First, P^{new} is set to the old P^{old}, then we perform an *increment*: Randomly select a parameter $p \in P$ and increase its value by a predefined stepsize.
- If $\sigma^{new} = \sigma^{old}$, we only perform the *increment*, no reset.
- If $\sigma^{new} > \sigma^{old}$, then exploration is working better than before. We do not change P^{old} in this case.

The values of a parameter are bounded by a range. Whenever a value leaves its range, it is reset to its default value.

5 Experimental Evaluation

We have implemented *expSAT* in two systems, *glucoseLCM* and *Maple_CM*, and call the resulting solvers *expGLCM* and *expM_CM*, respectively.

Maple_CM uses a hybrid of two heuristics, *LRB* and *VSIDS*. Based on the activation of these heuristics, a run in *Maple_CM* is divided into two stages: In stage 1, which lasts for the first 2500 seconds of a run, it uses a combination of these two heuristics. Stage 2 starts after 2500 seconds, where it uses VSIDS exclusively. In *expM_CM*, we apply the *expSAT* approach only at stage 2.

We compare the performance of these systems on both Test Sets as described in Preliminaries. To set the values of the exploration parameters, we took one instance from each benchmark of SAT-2018, which gives us a small subset of instances. We did a small scale grid search with *glucoseLCM* applying a small range of parameter values to set the value of the parameters used in the experiments (shown in Appendix A).

Solver	SAT	UNSAT	Combined	Average Solve Time	PAR2 - Score
glucoseLCM	95	97	192	1018.86	2275
expGLCM	**105 (+10)**	96 (-1)	**201 (+9)**	**894.06**	**2169**
Maple_CM	128	100	228	744.49	1889
expM_CM	**130 (+2)**	100	**230 (+2)**	767.32	**1876**

Table 2. Comparison of *expGLCM* and *expM_CM* against their baselines for Test Set 1.

Results on Test Set 1 Table 2 shows the Test Set 1 results for *glucoseLCM*, *expGLCM*, *Maple_CM* and *expM_CM*.

- *expGLCM* solves 9 more instances than its baseline, with a strong performance over SAT instances (solves 10 more) and slightly weaker performance over UNSAT instances (solves 1 less).
- *Maple_CM* and *expM_CM* solves 228 and 230 instances, respectively. Among thse solved instances, 210 instances are solved in stage 1, where runs in both solver are identical (*expM_CM* does not perform exploration). In stage 2, *expM_CM* solves 2 additional instances (both SAT).

Overall, this experiment pointed out strength of the *expSAT* approach in solving SAT instances from the latest SAT competition.

Results on Test Set 2 We evaluate the performance of these two extended solvers. We set time limit to be 36k seconds per instance.

To put the hardness of these cryptographic instances in perspective, we performed experiments with the winner of the main track of SAT-2018, *MapleLCMDistChronoBT*. It solves only 1 instance out of these 40 instances.

expGLCM solves 3 satisfiable instances with an average time of 6680s, and *glucoseLCM* solves 2 satisfiable instances with an average time of 12277s. For *Maple_CM* and *expM_CM*, neither system solves any of the 40 hard instances within the 36000 seconds time limit. Clearly, in this experiment, *expGLCM* is the winner.

6 Discussion

System	avg. GLR	avg. aLBD	avg. CD Phase Length
glucoseLCM	0.49	17.72	14.03
expGLCM	0.49	**17.59**	**13.90**

Table 3. Comparison of *glucoseLCM* and *expGLCM* over performance metrics

GLR, average LBD and CD Phase Length In [12], the authors observed that better branching heuristics have higher GLR and average LBD (aLBD), on average. First two columns of Table 3 compares the average GLR and average aLBD achieved by *glucoseLCM* and *expGLCM*, respectively, over the 400 instances of Test Set 1. While average GLR value is equal in both of the systems, exploration in *expGLCM* (system with better heuristic) helps the solver to achieve slightly lower average aLBD values. Hence, our experimental data are largely consistent with the observation in [12].

Compared to its baseline, *expGLCM* achieve slightly lower CD phase length, on average (3rd Column of Table 3). Exploration in *expGLCM* helps the system to flee from the CD phases expeditiously than its baseline, on average.

A: Type	B: #Instances	C: Exploration Average		D: Search Average	
		C1: $\mathbf{GLR_E}$	C2: $\mathbf{aLBD_E}$	D1: GLR	D2: aLBD
Solved	201	0.020	**11.74**	0.47	**13.46**
Unsolved	199	**0.025**	17.24	**0.51**	21.75

Table 4. Comparison of GLR and aLBD during exploration and search between Solved and Unsolved instances

Exploration Performance vs Search Performance For a given run with an *expSAT* solver, we define *exploration GLR* (GLR_E) as the number of discovered conflicts per random step (taken during exploration) and *exploration aLBD (aLBD$_E$)* as the average LBD of the derived clauses from the conflicts (discovered during exploration). Table 4 compares the average value of exploration GLR and aLBD (Column C) with average search GLR and aLBD (Column D) for the solved and unsolved instances of Test Set 1 with *expGLCM*. For the solved instances, on average, both search and exploration find conflicts at slightly lower rate than unsolved instances (Compare C1 and D1, for solved and unsolved). However, for solved instances, for both exploration and search, higher quality clauses are derived from the discovered/generated conflicts (Compare C2 and D2, for solved and unsolved).

Hence, GLR and aLBD of exploration correlates well with the GLR and aLBD of search.

7 Related Work

Randomized exploration in SAT is used in local search methods such as GSAT [21] and WalkSAT [20]. The *Satz* algorithm [9] heuristically selects a variable x, then performs two separate unit propagations with x and $(\neg x)$ respectively, in order to evaluate the potential of x. Modern CDCL SAT solvers include exploration components such as a small amount of random variable selection [5]. Exploration can make a search process more robust by allowing an escape from *early mistakes* caused by inaccurate heuristics [23]. Examples of recently popular exploration methods in search are Monte Carlo Tree Search (MCTS) [3], and the random walk techniques used in both classical [15] and motion planning [8]. A SAT solver based on MCTS is proposed in [18]. These works have motivated our work on random exploration in CDCL SAT.

In [4], we reported experimental results for a preliminary version of the *expSAT* approach, where exploration are triggered up-to a fixed height of the search tree, with probability p_{exp}. During the search, the exploration parameters are not adapted. Here, we trigger exploration amid the onset of substantial CD phases with probability p_{exp}, where the exploration parameters are adapted in between restarts based on the metric EPM.

8 Conclusions and Future Work

In this paper, we showed that VSIDS can fall into the pathological states of conflict depressions due to the ineffectiveness of its heuristic estimation. This observation led us to develop an exploration guided CDCL SAT solver framework *expSAT*, which performs random exploration amid substantial conflict depression phases. Our empirical evaluation of the *expSAT* approach shows strong performance gain for SAT instances.

Interesting research avenues to explore further include: (a) Integrate *expSAT* to LRB and CHB based systems. (b) Study exploration as in *expSAT* to guide polarity selection, e.g., by extending the *phase-saving* [17] heuristic. (c) Develop machine learning methods to predict the onset of a long CD phase. (d) Better understand (i) the performance gap between SAT and UNSAT instances with the *expSAT* approach. (ii) the relationship between properties of conflict depressions such as length and rate on one side, and the performance of a solver on the other side. (e) Identify characteristics of SAT domains which influence the effectiveness of exploration.

Appendix A: Parameter values used in Experiments

Description	Parameters	Value
(1) Default value of Exploration Parameters	P^{def}	$(0.02, 5, 5)$
(2) Weights for Computing σ	(w_1, w_2, w_3)	$(40, 10, 3)$
(3) Step size for parameters	$(s_{nW}, s_{lW}, s_{p_{exp}})$	$(1, 1, 0.01,)$
(4) Range for number of walks per episode	$[l_{nW}, u_{nW}]$	$[1, 20]$
(5) Range for length of walk	$[l_{lW}, u_{lW}]$	$[1, 10]$
(6) Range for exploration trigger probability	$[l_{p_{exp}}, u_{p_{exp}}]$	$[0.02, 0.6]$
(7) Exponential Decay Factor	ω	0.9

References

1. Gilles Audemard and Laurent Simon. Predicting learnt clauses quality in modern SAT solvers. In *IJCAI 2009, Proceedings of the 21st International Joint Conference on Artificial Intelligence, Pasadena, California, USA, July 11-17, 2009*, pages 399–404, 2009.
2. A. Biere, M. Heule, H. van Maaren, and T. Walsh. *Handbook of Satisfiability: Volume 185 Frontiers in Artificial Intelligence and Applications*. IOS Press, Amsterdam, The Netherlands, The Netherlands, 2009.
3. Cameron Browne, Edward Jack Powley, Daniel Whitehouse, Simon M. Lucas, Peter I. Cowling, Philipp Rohlfshagen, Stephen Tavener, Diego Perez Liebana, Spyridon Samothrakis, and Simon Colton. A survey of monte carlo tree search methods. *IEEE Trans. Comput. Intellig. and AI in Games*, 4(1):1–43, 2012.
4. Md. Solimul Chowdhury, Martin Müller, and Jia-Huai You. Preliminary results on exploration-driven satisfiability solving. In *Proceedings of the AAAI 2018*, pages 8069–8070, 2018.
5. Niklas Eén and Niklas Sörensson. An extensible sat-solver. In *Theory and Applications of Satisfiability Testing, 6th International Conference, SAT 2003. Santa Margherita Ligure, Italy, May 5-8, 2003 Selected Revised Papers*, pages 502–518, 2003.
6. Aarti Gupta, Malay K. Ganai, and Chao Wang. SAT-based verification methods and applications in hardware verification. In *Proceedings of SFM 2006*, pages 108–143.
7. Frédéric Lafitte, Jorge Nakahara Jr., and Dirk Van Heule. Applications of SAT solvers in cryptanalysis: Finding weak keys and preimages. *JSAT*, 9:1–25, 2014.
8. Steven M LaValle. *Planning algorithms*. Cambridge University Press, 2006.
9. Chu Min Li and Anbulagan. Look-ahead versus look-back for satisfiability problems. In *Principles and Practice of Constraint Programming - CP97, Third International Conference, Linz, Austria, October 29 - November 1, 1997, Proceedings*, pages 341–355, 1997.
10. Jia Hui Liang, Vijay Ganesh, Pascal Poupart, and Krzysztof Czarnecki. Exponential recency weighted average branching heuristic for SAT solvers. In *Proceedings of the Thirtieth AAAI Conference on Artificial Intelligence, February 12-17, 2016, Phoenix, Arizona, USA.*, pages 3434–3440, 2016.
11. Jia Hui Liang, Vijay Ganesh, Pascal Poupart, and Krzysztof Czarnecki. Learning rate based branching heuristic for SAT solvers. In *Theory and Applications of Satisfiability Testing - SAT 2016 - 19th International Conference, Bordeaux, France, July 5-8, 2016, Proceedings*, pages 123–140, 2016.
12. Jia Hui Liang, Hari Govind V.K., Pascal Poupart, Krzysztof Czarnecki, and Vijay Ganesh. *An Empirical Study of Branching Heuristics Through the Lens of Global Learning Rate*, pages 119–135. Springer International Publishing, Cham, 2017.
13. Mao Luo, Chu-Min Li, Fan Xiao, Felip Manyà, and Zhipeng Lü. An effective learnt clause minimization approach for cdcl sat solvers. In *Proceedings of the Twenty-Sixth International Joint Conference on Artificial Intelligence, IJCAI-17*, pages 703–711, 2017.
14. Matthew W. Moskewicz, Conor F. Madigan, Ying Zhao, Lintao Zhang, and Sharad Malik. Chaff: Engineering an efficient SAT solver. In *Proceedings of the 38th Design Automation Conference, DAC 2001, Las Vegas, NV, USA, June 18-22, 2001*, pages 530–535, 2001.
15. Hootan Nakhost and Martin Müller. Monte-carlo exploration for deterministic planning. In *IJCAI 2009, Proceedings of the 21st International Joint Conference on Artificial Intelligence, Pasadena, California, USA, July 11-17, 2009*, pages 1766–1771, 2009.
16. Vegard Nossum. Instance generator for encoding preimage, second-preimage, and collision attacks on SHA-1. In *Proceedings of SAT Competition*, pages 119–120, 2013.
17. Knot Pipatsrisawat and Adnan Darwiche. A lightweight component caching scheme for satisfiability solvers. In *Proceedings of SAT 2007*, pages 294–299.

18. Alessandro Previti, Raghuram Ramanujan, Marco Schaerf, and Bart Selman. Monte-carlo style UCT search for boolean satisfiability. In *AI*IA 2011: Artificial Intelligence Around Man and Beyond - XIIth International Conference of the Italian Association for Artificial Intelligence, Palermo, Italy, September 15-17, 2011. Proceedings*, pages 177–188, 2011.
19. Jussi Rintanen. Engineering efficient planners with SAT. In *Proceedings of ECAI 2012*, pages 684–689, 2012.
20. Bart Selman, Henry A. Kautz, and Bram Cohen. Local search strategies for satisfiability testing. In *Cliques, Coloring, and Satisfiability, Proceedings of a DIMACS Workshop, New Brunswick, New Jersey, USA, October 11-13, 1993*, pages 521–532, 1993.
21. Bart Selman, Hector J. Levesque, and David G. Mitchell. A new method for solving hard satisfiability problems. In *Proceedings of the 10th National Conference on Artificial Intelligence. San Jose, CA, July 12-16, 1992.*, pages 440–446, 1992.
22. Richard S. Sutton and Andrew G. Barto. *Introduction to Reinforcement Learning*. MIT Press, Cambridge, MA, USA, 1st edition, 1998.
23. Fan Xie, Martin Müller, Robert Holte, and Tatsuya Imai. Type-based exploration with multiple search queues for satisficing planning. In *Proceedings of the Twenty-Eighth AAAI Conference on Artificial Intelligence, July 27 -31, 2014, Québec City, Québec, Canada.*, pages 2395–2402, 2014.