Guiding CDCL SAT Search via Random Exploration amid Conflict Depression

Md Solimul Chowdhury, Martin Müller, 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. State-of-the-art CDCL branching heuristics such as VSIDS, CHB and LRB conform to this goal. We take a closer look at the way in which conflicts are generated over the course of a CDCL SAT search. Our study of the VSIDS branching heuristic shows that conflicts are typically generated in short bursts, followed by what we call a *conflict depression* phase in which the search fails to generate any conflicts in a span of decisions. The lack of conflict indicates that the variables that are currently ranked highest by the branching heuristic fail to generate conflicts. Based on this analysis, we propose an exploration strategy, called *expSAT*, which randomly samples variable selection sequences in order to learn an updated heuristic from the generated conflicts. The goal is to escape from conflict depressions expeditiously. The branching heuristic deployed in *expSAT* combines these updates with the standard VSIDS activity scores. An extensive empirical evaluation with four state-of-the-art CDCL SAT solvers demonstrates good-to-strong performance gains with the *expSAT* approach.

Introduction

Modern CDCL SAT solvers have become the enabling technology for many real-world problems, such as hardware design verification (Gupta, Ganai, and Wang 2006), classical planning (Rintanen 2012) and encryption (Massacci and Marraro 2000). 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 VSIDS (Moskewicz et al. 2001) and its variants, LRB (Liang et al. 2016b) and CHB (Liang et al. 2016a). These heuristics reward variables involved in recent conflicts. The intuition is that assignments of these variables are likely to generate further conflicts, leading to useful learned clauses and thus pruning the search space.

Global Learning Rate (GLR) (Liang et al. 2017) measures the number of conflicts obtained per branching decision. In CDCL, 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, i.e., average one conflict per two decisions (Liang et al. 2017).

In this work, we first perform a study of conflict generation during CDCL search. We find that there are clear non-random patterns of short bursts of conflicts, called *conflict burst (CB)*, followed by longer phases of what we call *conflict depression (CD)*, in which the search fails to generate any conflicts in a span of decisions. To correct the course of such a search, we propose to use exploration to combat conflict depression. We therefore design a new SAT solver extension, called *expSAT*, which applies random walks in the context of CDCL SAT solving. In a conflict depression phase, random walks help to identify more promising variables for branching. Note that exploration visits possible *future* search states, while the standard CDCL branching heuristics rely on conflicts generated from *past* search states.

The contributions of this paper are:

- An empirical study of the pathological state of conflict depression, using one of the strongest purely VSIDS-based solvers, glucoseLCM[1] (gLCM), on recent SAT competition benchmarks, shows that CD phases occur at a high rate and often with long average duration.
- A formulation of *expSAT* for an exploration-driven extension of VSIDS-based SAT solvers. *expSAT* performs random exploration when a substantial CD phase is detected. The goal is a swift escape from conflict depression.
- An empirical evaluation of *expSAT* implemented on top of four state-of-the-art SAT solvers, gLCM, MapleCOMSPS (MplCOMSPS), Maple_CM (MplCM) and MapleLCMDist_ChronoBT (MplCBT). On the benchmarks of main track of SAT Competitions 2017 and 2018 (SAT-2017 and SAT-2018), all four *expSAT* extensions solve more instances and achieve lower PAR-2 score[2] than their respective baselines. The best performing *expSAT* solver solves 16 more instances than its baseline, which is a strong performance gain. On 52 hard instances from SAT-Coin cryptographic benchmarks, most of our *expSAT* extensions show strong gains over their respective baselines.

[1] glucose 4.2.1, which implements the Learned Clause Minimization (LCM) (Luo et al. 2017) technique on top of glucose 4.1.

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

- An analysis of the experimental results shows that our results are consistent with two standard performance metrics, GLR and average LBD (Liang et al. 2017). In addition, exploration reduces average length of CD phases.
- An algorithm to update exploration parameters during the search and an experimental comparison of this adaptive version of *expSAT* with the non-adaptive one[3].

Preliminaries

We assume familiarity with SAT solving (Biere et al. 2009). Here we briefly review the most relevant concepts.

VSIDS Heuristic: VSIDS (Moskewicz et al. 2001) is a popular family of dynamic branching heuristics. We focus on exponential VSIDS as used in gLCM. VSIDS maintains an *activity score* for each variable in the given 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.

Literal Block Distance (LBD): The LBD score (Audemard and Simon 2009) of a learned clause is the number of distinct decision levels in it. If this score is n, then the clause contains n propagation blocks, where each block has been propagated within the same branching decision. As variables in a block are considered to be related, learned clauses with a lower LBD score are likely of higher quality. Especially, when LBD score is 2, they are known to be *glue clauses*.

Global Learning Rate (GLR): Suppose a CDCL solver takes d decisions to solve a given formula $\mathcal{F}$ and generates q conflicts. The GLR of the solver for $\mathcal{F}$ is defined as $\frac{q}{d}$. GLR measures the overall ability of a solver to generate conflict for a given problem (Liang et al. 2017).

Software, Hardware and Test Environment In this work, we adopt four baseline solvers: gLCM[4], MplCOM-SPS[5] (winner of SAT-2016), MplCM[4] (second runner up of SAT-2018) and MplCBT[4] (winner of SAT-2018). While gLCM uses only VSIDS as its branching heuristic, the other three combine VSIDS with other heuristics.

All experiments presented in this paper were run on a workstation with 64GB RAM and a processor clock speed of 2.4 GHz. Two test sets were used in experiments.

(a) **Test Set 1** contains 750 instances from the main track of SAT-2017 (350) and 2018 (400) and is run with a time limit of 5000 seconds per instance.

(b) **Test Set 2** consists of 52 hard instances from SATCoin (Bit Coin Mining) cryptographic benchmark, which are generated with the instance generator from (Manthey and Heusser 2018). We generated these instances by varying the *range* parameter, which determines the difficulty of a SATCoin instance. For experiments, we set the time limit to be 36,000 seconds per instance.

[3]Source code of the *expSAT* solvers and 52 SATCoin instances are available at: `https://figshare.com/articles/expSAT_Solvers_Instances/10324172`.

[4]Source: `http://sat2018.forsyte.tuwien.ac.at/solvers/main\_and\_glucose\_hack/`

[5]Source: `https://sites.google.com/a/gsd.uwaterloo.ca/maplesat/`

Conflict Depression and Conflict Bursts

Consider a run of a CDCL SAT solver Ψ 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. Let us represent the conflict history of the search by the sequence of c_i and define a *conflict depression (CD) phase* as a sequence of one or more consecutive decisions with no conflict. Let us define a *conflict burst (CB) phase* as a sequence of one or more consecutive decisions with at least one conflict.

Let us further define the *length* of a CD and CB phase as the number of decisions in it E.g., 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 2 with length 4, one starting at decision 9 with length 1, and one at the end with length 2. It also contains 3 CB phases at decision 1 with length 1, at decision 6 with length 3 and at decision 9 with length 1.

DR, CDR, FDC, FDOC and FDMC: Suppose the solver Ψ takes a total of d decisions, encounters u CD phases and gives r restarts. We define the *Decision Rate (DR)* as d/r and *CD phase Rate (CDR)* as u/r.

We also define *Fraction of Decisions with Conflicts (FDC)* as the measure of the fraction of decisions which produce at least one conflict. This measure is related to but different from GLR. It counts decision with conflicts, not conflicts. We further partition FDC into FDOC+FDMC, where FDOC and FDMC are *Fraction of Decisions with One Conflict (FDOC)* and *Fraction of Decisions with Multiple Conflicts (FDMC)*, respectively.

Conflict Depression in gLCM

We now study conflict depression empirically, using VSIDS as a representative CDCL branching heuristic, and gLCM as the underlying SAT solver. We collect the statistics for each search of an instance on DR, CDR, Average CD phase length, GLR, FDC, FDOC and FDMC.

CDR and Average CD Phase Length The left plot of Fig. 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 observe that the average CD phase length is short for most instances, but still consists of multiple decisions (blue). Furthermore, irrespective of their average CD phase length, for all-most all instances CD phases (orange) occur at a high rate given the decision rates (yellow).

The histogram on the right side of Fig. 1 shows the distribution of average length of CD phases. This average ranges from 2.09 to 1402.30. 263 instances have a very short length (at most 3). The distribution is heavy-tailed, with 69 instances of average length greater than 25 (rightmost bin).

Figure 1: CD plots for Test Set 1 with gLCM

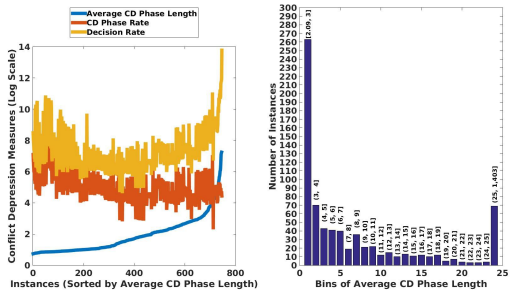


Table 1: Average PR for CD and CB Phases

1: Type	2: #Inst	3: Propagation Rate	
		3.1 CD Phase	3.2 CB Phase
SAT	177	153.43	**1560.40**
UNSAT	195	404.40	**3445.40**
Unsolved	378	173.18	**1718.51**
Combined	750	229.51	**2136.73**

Overall, the data indicates that for gLCM on Test Set 1, conflict depressions occur frequently and often last over multiple decisions (high average CD phase length).

Propagation Depression Amid a CD Phase During a CD phase, VSIDS scores are not a good predictor of a variable's future performance, since branching decisions fail to produce any conflict and perform only truth value propagations. Are there any differences in the pattern of unit propagations between CD and CB phases?

We define the *Propagation Rate (PR)* as the number of propagations per decision. Table 1 compares the average PR values for Test Set 1 over the decisions in CD and CB phases. On average, PR values during a CD phase are almost 10 times lower than CB phases. Clearly, this result demonstrates that during a CD phase, VSIDS branching decisions go through propagation depression as well.

Conflict Bursts in gLCM

How long are the CB phases compared to CD phases? For the Test Set 1 instances, average value of CB and CD length are 1.67 and 20.63, respectively. Thus, on average, shorter CB phases are followed by much longer CD phases.

Bursts of Conflict Generation Table 2 shows the average values of GLR, FDC, FDOC and FDMC for Test Set 1. Column 3 shows the average GLR values for all three types of problems to be close to 0.5. In contrast, the average FDC values in column 4 are much lower, averaging 0.2507 over all instances. Therefore, on average, about 75% of all the decisions do not produce any conflict and only 25% of all the decisions produce at least one conflict. Further, the majority of the conflict producing decisions produce more than 1 conflict. This is evident in the average FDMC value (0.1535), which is 61% of the total conflict producing decisions (0.2507).

Table 2: Average values of GLR, FDC, FDOC and FDMC

1: Type	2: #Inst	3: GLR	4. FDC	5: FDOC	6: FDMC
SAT	177	0.4644	0.2394	0.0980	0.1414
UNSAT	195	0.5070	0.2492	0.0927	0.1565
Unsolved	378	0.5099	0.2568	0.0992	0.1576
Combined	750	0.4984	0.2507	0.0972	0.1535

As a summary, we have the following conclusions.

- The typical search behavior contains shorter CB phases, which is followed by longer CD phases, where the search does not find any conflicts.
- During a CD phase, the search goes through propagation depression as well.
- The shorter CB phases are conflict intense, i.e., within a few decisions, many conflicts are generated.

Exploration Guided VSIDS

Is it possible to correct the course of the search in a CD phase by identifying promising variables that are currently underranked by VSIDS? In this work, we address this question by formulating a solver framework, called *expSAT*, which performs random explorations that probe into the future search space. The goal is to discover branching variables that are likely to lead to conflicts from which clauses are learned.

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

(i) Before each branching decision, if a *substantial* CD phase 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 an unassigned *step variable*, 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. After each walk, the search state is restored and the next walk begins. Fig. 2 illustrates an exploration episode with 3 walks and a maximum of 3 random steps per walk.

(ii) An *exploration score* is computed for each step variable.

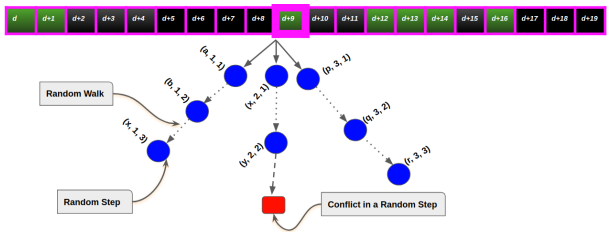
(iii) In the CDCL search, branching variables are chosen that maximize the `expVSIDS` heuristic, which combines the VSIDS activity score of a variable and its exploration score. Ties are broken randomly.

(iv) All other elements, such as unit propagation, conflict analysis, restarts, and backjumping, remain the same as in the underlying CDCL SAT solver.

Algorithm Details

Input and Parameters Given a SAT formula $\mathcal{F}$, let $uVars(\mathcal{F})$ and $assign(\mathcal{F})$ be the set of currently unassigned variables in $\mathcal{F}$ and the current partial assignment, respectively. The input to *expSAT* consists of $\mathcal{F}$ and four exploration parameters nW, lW, p_{exp}, ω, where $1 \leq nW$, $lW \leq uVars(\mathcal{F})$, $0 < p_{exp}, \omega \leq 1$. All these parameters are explained above, except ω, which we explain below. When a random walk ends in a conflict after a series of random steps, some combination of the assigned variables has caused the conflict. In *expSAT*, we assign the most credit to

Figure 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.



the most recently assigned variable, and exponentially decay the credit for the variables assigned earlier in the walk, by a factor of ω per decision step. This approach is patterned on reward decay in reinforcement learning (Sutton and Barto 1998).

Exploration, Random Walks and Steps An exploration episode performs nW walks, each containing a maximum of lW random steps. Each such step consists of two parts:

- Choose a step variable $v \in$ *uVars*$(\mathcal{F})$ uniformly at random, and assign a boolean value to v using a standard CDCL *polarity* heuristic.
- Run unit propagation (UP) after the assignment of v. Any conflict ends the walk immediately.

Algorithm 1 for *expSAT* is based on standard CDCL except lines 3-7. Line 3 checks whether an exploration episode should take place - it is triggered with probability p_{exp} within a substantial CD phase. Line 5 starts an exploration episode, and line 7 selects a branching variable with maximum `expVSIDS` score.

Detection of Substantial CD Phases The overhead of exploration must be balanced against its benefits. We perform exploration episodes with probability p_{exp}, by tracking the ratio $R = \frac{\#decisions\_without\_conflicts}{\#decisions\_with\_conflicts}$ of the search so far. $R + 1$ is the average number of decisions taken until one generates a conflict. A CD phase is said to be *substantial* if the current number of consecutive decisions without conflict since the last conflict generating decision is at least R.

Exploration Episodes (EEs) and Scores In an EE, the *exploration score* of a decision variable v, denoted *expScore(v)*, is the average of the *walk scores*, *ws*(v), of all random walks within the same episode in which v was one of the randomly chosen decision variables. The value of *ws*(v) is computed as:

Algorithm 1: Exploration Based CDCL Solver : *expSAT*

Input: A CNF SAT formula: $\mathcal{F}$
Exploration Parameters: nW, lW, p_{exp}, ω
Output: Satisfiability of $\mathcal{F}$

1 Preprocess $\mathcal{F}$ and return result if it is solved;
2 **while** *true* **do**
3 $explore \leftarrow$substantial_CDPhase() **AND** random()$> p_{exp}$;
4 **if** *explore* **then**
5 explorationEpisode$(nW, lW, p_{exp}, \omega)$;
6 **end**
7 decideBranchingVariable();
8 **while** $true$ **do**
9 Perform Unit Propagation;
10 Break, if no new deductions are made;
11 Return SAT, if no more unassigned variables;
12 If a conflict is found, perform conflict analysis;
13 Obtain a learned clause cl and a backtrack level $blevel$;
14 Return UNSAT, if the $blevel$ is 0;
15 Perform non-chronological backtracking to $blevel$;
16 Assign the asserting literal from the learned clause cl
17 **end**
18 **end**

- $ws(v) = 0$, if (i) the walk ended without a conflict, or (ii) the walk ended with a conflict and $lbd(c)$, the LBD score of the clause c derived from the current conflict, is greater than *avgLBD*, the average LBD of learned clauses by the search (i.e., the quality of the derived clasue c is below the search average).
- Otherwise, $ws(v) = \frac{\omega^d}{lbd(c)}$, with decay factor ω, and $d \geq 0$ the *decision distance* between variable v and the conflict which ended the current walk: If v was assigned at some step j during the current walk, and the conflict occurred after step $j' \geq j$, then $d = j' - j$. The values of *ws* and *expScore* are always in the interval $[0, 1)$.

Example: Using the three random walks of Fig. 1, we show how to compute ws and $expScore$ of variables. Only the second walk produces a conflict. Let c be the derived clause from this conflict, with $lbd(c) = m <$ *avgLBD*.

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

Decide Branching Variable The branching variable is chosen by maximizing the combined VSIDS + exploration

score as shown in Algorithm 2. To make both scores comparable, following VSIDS, the exploration score is scaled by the factor g^z.

Algorithm 2: Decide the Branching Variable

Input: None

foreach $v \in uVars(\mathcal{F})$ **do**
  expScore(v) $\leftarrow$ computeExpEpisodeScore(v);
  combinedScore(v) $\leftarrow$ $VSIDS(v) + g^z * expScore(v)$;
end
$v^* \leftarrow argmax_{v \in uVars(\mathcal{F})}$ combinedScore(v);
$assign(\mathcal{F}) \leftarrow assign(\mathcal{F}) \cup makeAssignment(v^*)$

Experiments

We implemented *expSAT* in four systems gLCM, MplCOMSPS, MplCM and MplCBT, and call the resulting solvers eGLCM, eMplCOMSPS, eMplCM and eMplCBT, respectively. While gLCM uses only VSIDS, MplCOMSPS and MplCM apply a combination of two heuristics, LRB and VSIDS. In addition, MplCBT also employs a third heuristic called Dist (Xiao et al. 2017). Based on the activation of these heuristics, a run in Maple (Mpl) based systems is divided into two phases: *phase 1*, which lasts for the first 2500 seconds of a run and uses a combination of these heuristics, and *phase 2*, which starts after 2500 seconds and uses VSIDS exclusively. For the Maple based systems, we apply the *expSAT* approach only to phase 2.

We compare the performance of these systems on Test Sets 1 and 2. To set the values of the exploration parameters, we performed a small scale grid search with eGLCM: we took one instance at random out of each benchmark from SAT-2018, which gave a subset of 23 instances. We run eGLCM on this subset for small parameters ranges, lW and nW in [4,5,6] and p_{exp} in [0.01,0.02,0.03]. From this grid search, we chose our default parameter setting (mW, mS, p_{exp})=$(5, 5, 0.02)$. We set the value of the exponential decay parameter ω to 0.9 based on intuition. These are the values used in the experiments.

Comparison on Test Set 1 Table 3 shows results for Test Set 1 for four *expSAT* extensions and their baseline solvers. Overall, each *expSAT* extension solves more instances and has lower (better) PAR-2 score than its respective baseline.

For each of the Maple based system, for a given instance, runs with a baseline and its *expSAT* extension are identical in phase 1. For these systems, only instances solved in phase 2 show the impact of the *expSAT* approach. For each Maple based *expSAT* solver, all the additional instances are solved in phase 2.

The best performing system eMplCOMSPS solves 16 more instances than its baseline. eMplCBT solves 9 more instances than its baseline MplCBT. eMplCM solves only 1 more instance than its baseline MplCM. eGLCM solves 7 more instances than its baseline gLCM, where most of the improvements comes from solving 8 additional SAT instances in the SAT-2018 benchmarks.

Table 3: Comparison between solvers on Test Set 1

Systems	2017			2018			Combined	
	SAT	UNSAT	Total	SAT	UNSAT	Total	Total	PAR-2
gLCM	82	**98 (+3)**	**180 (+1)**	95	97	192	372	4133
eGLCM	**84 (+2)**	95	179	**103 (+8)**	97	**200(+8)**	**379(+7)**	**4068**
MplCOMSPS	104	98	202	116	94	210	412	3701
eMplCOMSPS	**106 (+2)**	**101 (+3)**	**207(+5)**	**125 (+9)**	**96 (+2)**	**221(+11)**	**428(+16)**	**3442**
MplCM	103	111	214	128	100	228	442	3456
eMplCM	**104 (+1)**	111	**215(+1)**	128	100	228	**443(+1)**	**3445**
MplCBT	97	110	207	133	102	235	442	3498
eMplCBT	**98 (+1)**	**113 (+3)**	**211(+4)**	**138 (+5)**	102	**240(+5)**	**451(+9)**	**3400**

Figure 3: Solve time comparison for Test Set 1

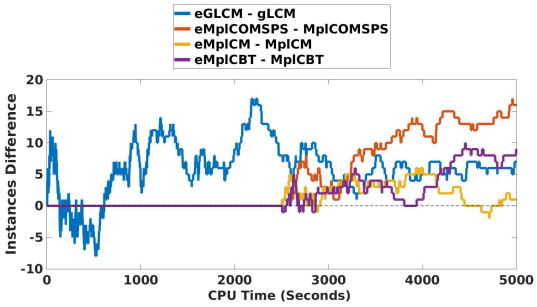


Fig. 3 compares the solving speed of eGLCM (blue line), eMplCOMSPS (red line), eMplCM (yellow line) and eMplCBT (purple line) against their baselines. This figure plots the difference in the number of instances solved as a function of time. At phase 1, there is no difference for the Maple based *expSAT* solvers and their baselines. eMplCOMSPS (red line) dominates over its baseline for all of phase 2. eMplCBT (purple line) and eMplCM (yellow line) also solve instances at a faster speed than their baseline for most time points. eGLCM (blue line) performs slightly worse than gLCM at the earlier time points, but beats the baseline for the remaining time points.

Comparison on Test Set 2 For SAT-2018, 17 SATCoin instances were submitted. For the experimental results reported in Table 3, we observe that compared to the baselines gLCM (solves 1) and MplCOMSPS (solves 2), their *expSAT* extensions, eGLCM (solves 5) and eMplCOMSPS (solves 6), show strong performance gains over these 17 instances[6]. We further evaluate the *expSAT* solvers on this benchmark by generating 52 hard instances (Test Set 2), which are different from the 17 instances submitted for SAT-2018.

Table 4 compares our *expSAT* extensions with their respective baselines for Test Set 2. The best performing *expSAT* extensions, eMplCM and eMplCOMSPS, solve 10 and 13 instances respectively, beating their baselines by each solving 9 additional instances. Compared to their baselines, eGLCM and eMplCBT solve 5 and 2 additional instances, respectively. Fig. 4 shows the solve time comparison for the 8 solvers for Test Set 2. Here, all of our *expSAT* solvers solve the problems at higher speed than their baselines at most of the time points.

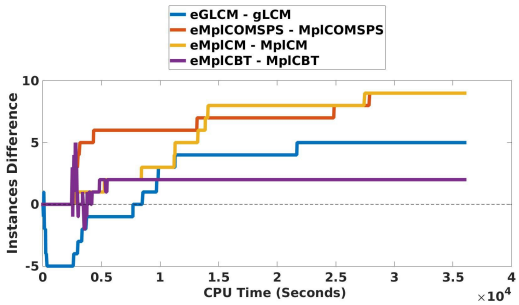
For this experiment, each of our extended solvers shows

[6]Compared to their baselines, eMplCM and eMplCBT solve equal number of instances over these 17 instances.

Table 4: Results for Test Set 2

System	SAT	UNSAT	Total
gLCM	3	4	7
eGLCM	**4 (+1)**	**8 (+4)**	**12 (+5)**
MplCOMSPS	2	2	4
eMplCOMSPS	**6 (+4)**	**7 (+5)**	**13 (+9)**
MplCM	0	1	1
eMplCM	**6 (+6)**	**4 (+3)**	**10 (+9)**
MplCBT	21	20	41
eMplCBT	**23 (+2)**	20	**43 (+2)**

Figure 4: Solve time comparison for Test Set 2



strong performance gains over its baseline. To put this experiment into perspective, we ran experiment with CryptoMiniSAT5[7], which is known to be a strong system for solving cryptographic benchmarks. This system solves 41 instances with average solve time 8907.47 secs, while our best performing *expSAT* solver eMplCBT solves 43, with average solve time of 2518.90 secs.

Analysis of the Experimental Results

For analysis of the results presented in the previous section, we use experimental data from gLCM and eGLCM, where `expVSIDS` is active for the whole time of a given run.

Given a set of instances, we define two subsets below.

- **exp**$^+$: Instances solved by eGLCM but not by gLCM, or gLCM takes longer time to solve than eGLCM.
- **exp**$^-$: Instances solved by gLCM but not by eGLCM, or eGLCM takes longer time to solve than gLCM.

GLR and Average LBD Score: In (Liang et al. 2017), it is shown that on average, a more efficient CDCL branching heuristic leads to higher GLR value and lower average LBD (aLBD) score of the learned clauses. We analyze GLR and aLBD scores. The top two rows of Table 5 show the average GLR values for **exp**$^-$ and **exp**$^+$ for Test Set 1 with gLCM and eGLCM.

- For **exp**$^-$, where the baseline gLCM is more efficient, gLCM has lower average GLR score. However, gLCM learns higher quality clauses (lower aLBD), on average.
- For **exp**$^+$, where eGLCM is more efficient, eGLCM generates conflicts at the same rate as the baseline. However, it learns higher quality clauses (lower aLBD), on average.

Our results on Test Set 1 are largely consistent with the observation from (Liang et al. 2017).

[7]https://www.msoos.org/cryptominisat5/

Table 5: Comparison of average GLR and average aLBD

Instance Set	System	exp$^-$			exp$^+$		
		#inst	avg. GLR	avg. aLBD	#	avg. GLR	avg. aLBD
Test Set 1 (2017+ 2018)	gLCM	156	0.47	**15.08**	245	0.49	15.88
	eGLCM		**0.49**	15.79		0.49	**14.78**
Test Set 2 (SATCoin)	gLCM	6	**0.61**	**25.46**	11	0.34	35.81
	eGLCM		0.30	33.55		**0.37**	**32.29**

Table 6: Comparison of average CD Phase length

Instance Set	System	exp$^-$		exp$^+$	
		#inst	avg. CDLen	#	avg. CDLen
Test Set 1 (2017+ 2018)	gLCM	156	30.27	245	8.60
	eGLCM		**29.97**		**8.26**
Test Set 2 (SATCoin)	gLCM	6	**4.96**	11	8.27
	eGLCM		9.38		**7.73**

The experimental data from Test Set 2 are strongly consistent with the observation of (Liang et al. 2017). The bottom two rows of Table 5 show that in each case, the better system has higher average GLR and lower average aLBD.

Reduction of Average CD Phase Length: Table 6 shows that for both test sets, exploration in eGLCM reduces the average CD phase length for both **exp**$^-$ and **exp**$^+$ in most of the cases. Thus exploration helps a solver to escape from the pathological state of conflict depression.

Exploration Statistics: What is the cost and benefit of performing exploration? On average, for the instances from Test Set 1, eGLCM incurs 92.53 seconds of overhead to perform exploration, which is about 3.94% of its average running time of 2346.12 seconds. With random exploration amid substantial CD phases, on average, eGLCM finds about 2 conflicts per 100 random steps.

Derived Clauses from Exploration: In *expSAT*, we do not learn the clauses that are derived from conflicts discovered during exploration. This is based on the following intuition. Whenever a CDCL search learns a clause c, c is immediately used by propagating the asserting literal (first UIP), which is hosted by c. However, in case of exploration, when a clause c is derived, such propagations do not immediately follow. Thus the utility of c is uncertain. We ran an experiment with eGLCM for Test Set 1, where we saved the learned clauses that are derived during exploration. The result is slightly worse than the result for eGLCM reported in Table 3.

Exploration Parameter Adaptation

A parameter setting that is effective for one instance may not be effective for another. Based on this intuition, we developed an algorithm named *paramAdapt* to dynamically control when to trigger exploration episodes, and how much exploration to perform in an exploration episode.

paramAdapt The three exploration parameters nW, lW, and p_{exp} are adapted between CDCL restarts based on the search behavior. A parameter setting is a triple $\Sigma = (nW, lW, p_{exp})$, 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 of Σ.

Table 7: Parameter values for the adaptive-*expSAT* Solvers

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

Table 8: Comparison between adaptive *expSAT* solvers and non-adaptive *expSAT* solvers with Test Set 1

System	2017			2018			Combined	
	SAT	UNSAT	Total	SAT	UNSAT	Total	Total	PAR-2 Score
eGLCM	84	**95(+1)**	179	103	**97(+1)**	200	379	4068
eGLCM$^{\mathbf{ad}}$	**85(+1)**	94	179	**105(+2)**	96	**201(+1)**	**380(+1)**	**4036**
eMplCOMSPS	**106(+7)**	**101(+2)**	**207(+9)**	**125(+2)**	96	**221(+2)**	**428(+11)**	**3442**
eMplCOMSPS$^{\mathbf{ad}}$	99	99	198	123	96	219	417	3665
eMplCM	104	**111(+3)**	215	128	100	228	443	3445
eMplCM$^{\mathbf{ad}}$	**107(+3)**	108	215	**130(+2)**	100	230	**445(+2)**	**3435**
eMplCBT	98	**113(+2)**	**211(+1)**	**138(+3)**	102	**240(+3)**	**451(+4)**	**3400**
eMplCBT$^{\mathbf{ad}}$	**99(+1)**	111	210	135	102	237	447	3410

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 Σ^{old} just to get a new setting Σ^{new}.

- If $\sigma^{new} < \sigma^{old}$, the performance of exploration is worse than before. First, Σ^{new} is set to the old Σ^{old}, then we perform an *increment*: Randomly select a parameter $p \in \Sigma$ 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 Σ^{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.

Experiments We repeat the same experiments for Test Sets 1 and 2 with four *expSAT* extensions with *paramAdapt* implemented on each of them. We denote by exp$\Psi^{\mathbf{ad}}$ the adaptive version of non-adaptive *expSAT* solver expΨ.

We set the default values of parameters Σ to $(5, 5, 0.02)$. The values of the other parameters are given in Table 7.

Table 8 shows the performance comparison between the non-adaptive and adaptive *expSAT* solvers. For Test Set 1, the overall performance of the non-adaptive versions is better: eMplCOMSPS and eMplCBT solve 11 and 4 more problems compared to their adaptive versions. eGLCM$^{\mathbf{ad}}$ solves 1 more instance than eGLCM. eMplCM$^{\mathbf{ad}}$ solves 2 more instances than its baseline.

For Test Set 2, the performance of most of the adaptive *expSAT* solvers are significantly better than their respective non-adaptive versions, as shown in Table 9. eMplCOMSPS$^{\mathbf{ad}}$ and eMplCM$^{\mathbf{ad}}$ solve 16 and 13 more problems than eMplCOMSPS and eMplCM, respectively. eGLCM$^{\mathbf{ad}}$ solves 9 more instances than eGLCM. Both eMplCBT$^{\mathbf{ad}}$ and eMplCBT solve an equal number of problems.

Table 9: Additional comparison with Test Set 2

System	SAT	UNSAT	Total
eGLCM	4	8	12
eGLCM$^{\mathbf{ad}}$	**13 (+9)**	8	**21 (+9)**
eMplCOMSPS	6	7	13
eMplCOMSPS$^{\mathbf{ad}}$	**14 (+8)**	**15 (+8)**	**29 (+16)**
eMplCM	6	4	10
eMplCM$^{\mathbf{ad}}$	**14 (+8)**	**9 (+5)**	**23 (+13)**
eMplCBT	**23 (+1)**	20	43
eMplCBT$^{\mathbf{ad}}$	22	**21 (+1)**	43

Table 10: A small scale performance analysis

1:System	2:Overhead	3:exp_GLR	4:exp_aLBD	5: GLR	6:aLBD	7:a. CDLen
eMplCOMSPS	**48.51 secs**	**0.0193**	**15.25**	0.51	**22.91**	**20.08**
eMplCOMSPS$^{\mathbf{ad}}$	198.21 secs	0.0179	15.78	0.51	23.39	20.77

Analysis Table 10 shows a small scale analysis with eMplCOMSPS and eMplCOMSPS$^{\mathbf{ad}}$, for which we observed the largest performance gap for Test Set 1. Compared to eMplCOMSPS$^{\mathbf{ad}}$, on average eMplCOMSPS has significantly lower overhead incurred in exploration (Col. 2), exploration finds conflicts at a faster rate (Col. 3), from which lower LBD (Col. 4) clauses are derived. During the search, both systems generate clauses at the same rate (Col. 5), however the average LBD of the learned clauses for eMplCOMSPS is lower (Col. 6), so is the average CD phase length for eMplCOMSPS (Col. 7). These data explain the better performance of eMplCOMSPS over eMplCOMSPS$^{\mathbf{ad}}$ for Test Set 1.

Related Work

Randomized exploration in SAT is used in local search methods such as GSAT (Selman, Levesque, and Mitchell 1992) and WalkSAT (Selman, Kautz, and Cohen 1993). The *Satz* algorithm (Li and Anbulagan 1997) 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 (Eén and Sörensson 2003). UCTSAT (Previti et al. 2011) employs Monte Carlo Tree Search (MCTS) to build a SAT search tree. Exploration can make a search process more robust by allowing an escape from *early mistakes* caused by inaccurate heuristics (Xie et al. 2014). Examples of recently popular exploration methods in search are MCTS (Browne et al. 2012) and the random walk techniques used in classical planning (Nakhost and Müller 2009). These techniques motivated our work on random exploration in CDCL SAT.

SATHYS (Audemard et al. 2010) employs both a CDCL SAT solver and a local search SAT solver. The latter helps the CDCL solver by identifying the most promising literal assignment to branch on, and the CDCL search process guides the local search process to flee from local minima. The Conflict History Based (CHB) (Liang et al. 2016a) and

Learning Rate Based (LRB) (Liang et al. 2016b) heuristics model variable selection as a Multi-Armed Bandit (MAB) problem, which is solved using the Exponential Recency Weighted Average (ERWA) algorithm. Both of these heuristics compute rewards from the conflict history of unassigned variables, in order to rank them. In contrast, we modify the VSIDS rank of variables based on the quality of conflicts generated by random exploration of the future states. Compared to the look-ahead based heuristic that maximize the GLR score (Liang et al. 2017), we perform nondeterministic exploration of the search space with a small subset of unassigned variables per random walk, and prioritize variables that generate high-quality conflicts. Since decision time is disregarded in their work, there is no direct basis for comparison.

Future Work

The ineffectiveness of VSIDS in conflict depressions can be addressed by performing exploration. Interesting research avenues to explore further include:
1. Integrate *expSAT* to LRB and CHB based systems.
2. Study exploration as in *expSAT* to guide polarity selection, e.g., by extending the *phase-saving* heuristic.
3. Develop machine learning methods to predict the onset of a long CD phase.
4. Better understand the relationship between properties of CD phases such as length and the performance of a solver.
5. Identify characteristics of SAT domains which influence the effectiveness of exploration.

Acknowledgements

We thank the anonymous reviewers for their valuable advice. This research is supported by Natural Sciences and Engineering Research Council of Canada PGS Doctoral (NSERC PGS-D) Award, President's Doctoral Prize of Distinction (PDPD), Alberta Innovates Graduate Student Scholarship (AIGSS), and NSERC discovery grant.

References

Audemard, G., and Simon, L. 2009. Predicting learnt clauses quality in modern SAT solvers. In *Proceedings of IJCAI 2009*, 399–404.

Audemard, G.; Lagniez, J.; Mazure, B.; and Sais, L. 2010. Boosting local search thanks to CDCL. In *Proceedings of LPAR-10*, 474–488.

Biere, A.; Heule, M.; van Maaren, H.; and Walsh, T. 2009. *Handbook of Satisfiability: Volume 185 Frontiers in Artificial Intelligence and Applications*. Amsterdam, The Netherlands, The Netherlands: IOS Press.

Browne, C.; Powley, E. J.; Whitehouse, D.; Lucas, S. M.; Cowling, P. I.; Rohlfshagen, P.; Tavener, S.; Liebana, D. P.; Samothrakis, S.; and Colton, S. 2012. A survey of Monte Carlo Tree Search methods. *IEEE Trans. Comput. Intellig. and AI in Games* 4(1):1–43.

Eén, N., and Sörensson, N. 2003. An extensible SAT solver. In *Proceedings of SAT 2003. Selected Revised Papers*, 502–518.

Gupta, A.; Ganai, M. K.; and Wang, C. 2006. SAT-based verification methods and applications in hardware verification. In *Proceedings of SFM 2006*, 108–143.

Li, C. M., and Anbulagan. 1997. Look-ahead versus look-back for satisfiability problems. In *Proceedings of CP 1997*, 341–355.

Liang, J. H.; Ganesh, V.; Poupart, P.; and Czarnecki, K. 2016a. Exponential recency weighted average branching heuristic for SAT solvers. In *Proceedings of AAAI 2016*, 3434–3440.

Liang, J. H.; Ganesh, V.; Poupart, P.; and Czarnecki, K. 2016b. Learning rate based branching heuristic for SAT solvers. In *Proceedings of SAT 2016*, 123–140.

Liang, J. H.; K., H. G. V.; Poupart, P.; Czarnecki, K.; and Ganesh, V. 2017. An empirical study of branching heuristics through the lens of global learning rate. In Gaspers, S., and Walsh, T., eds., *Proceedings of SAT 2017*, 119–135.

Luo, M.; Li, C.-M.; Xiao, F.; Manyá, F.; and Lu, Z. 2017. An effective learnt clause minimization approach for CDCL SAT solvers. In *Proceedings of IJCAI 2017*, 703–711.

Manthey, N., and Heusser, J. 2018. SATcoin - Bitcoin mining via SAT. In *Proceedings of SAT Competition 2018*, 67–68.

Massacci, F., and Marraro, L. 2000. Logical cryptanalysis as a SAT problem. *J. Autom. Reasoning* 24(1/2):165–203.

Moskewicz, M. W.; Madigan, C. F.; Zhao, Y.; Zhang, L.; and Malik, S. 2001. Chaff: Engineering an efficient SAT solver. In *Proceedings of DAC 2001*, 530–535.

Nakhost, H., and Müller, M. 2009. Monte-Carlo exploration for deterministic planning. In *Proceedings of IJCAI 2009*, 1766–1771.

Previti, A.; Ramanujan, R.; Schaerf, M.; and Selman, B. 2011. Monte-Carlo style UCT search for boolean satisfiability. In *Proceedings of AI*IA 2011*, 177–188.

Rintanen, J. 2012. Engineering efficient planners with SAT. In *Proceedings of ECAI 2012*, 684–689.

Selman, B.; Kautz, H. A.; and Cohen, B. 1993. Local search strategies for satisfiability testing. In *Cliques, Coloring, and Satisfiability, Proceedings of a DIMACS Workshop 1993*, 521–532.

Selman, B.; Levesque, H. J.; and Mitchell, D. G. 1992. A new method for solving hard satisfiability problems. In *Proceedings of the AAAI 1992*, 440–446.

Sutton, R. S., and Barto, A. G. 1998. *Introduction to Reinforcement Learning*. Cambridge, MA, USA: MIT Press, 1st edition.

Xiao, F.; Luo, M.; Li, C.-M.; Manyá, F.; and Lu, Z. 2017. MapleLRB_LCM, Maple_LCM, Maple_LCM_Dist, MapleLRB_LCMoccRestart and Glucose3.0+width in sat competition 2017. In *Proceedings of SAT Competition 2017*, 22–23.

Xie, F.; Müller, M.; Holte, R.; and Imai, T. 2014. Type-based exploration with multiple search queues for satisficing planning. In *Proceedings of AAAI 2014*, 2395–2402.