Exploiting Glue Clauses to Design Effective CDCL Branching Heuristics

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. In conflict-directed clause learning (CDCL) SAT solving, a state-of-the-art criterion to measure the importance of a learned clause is called *literal block distance* (LBD), which is the number of distinct decision levels in the clause. The lower the LBD score of a learned clause, the better is its quality. The learned clauses with LBD score of 2, called *glue clauses*, are known to possess high pruning power. In this work, we relate glue clauses to decision variables. First, we show experimentally that branching decisions with variables appearing in glue clauses, called *glue variables*, are more conflict efficient than with nonglue variables. This observation motivated the development of a structure-aware CDCL variable bumping scheme, which increases the heuristic score of a glue variable based on its appearance count in the glue clauses that are learned so far by the search. Empirical evaluation shows the effectiveness of the new method on the main track instances from SAT Competitions 2017 and 2018 with four state-of-the-art CDCL SAT solvers. Finally, we show that the frequency of learned clauses that are glue clauses can be used as a reliable indicator of solving efficiency for some instances, for which the standard performance metrics fail to provide a consistent explanation.

Keywords: CDCL SAT · Branching Heuristics · Glue Clauses

1 Introduction

Given a formula $\mathcal{F}$ of boolean variables, the task of SAT solving is to determine a variable assignment that satisfies $\mathcal{F}$ or to report the unsatisfiability of $\mathcal{F}$ in case no such assignment exists. SAT is known to be NP-complete [5]. Despite the hardness, modern CDCL SAT solvers can solve large real-world problems from important domains, such as hardware design verification [8], software debugging [4], planning [21], and encryption [18, 23], sometimes with surprising efficiency. This is the result of a careful combination of its key components, such as preprocessing [6, 10] and inprocessing [11, 17], robust branching heuristics [14, 13, 19], efficient restart policies [2, 20], intelligent conflict analysis [22], and effective clause learning [19].

Clause learning prunes search space. As conflict discovery is the only way to learn clauses, the rate of discovery is critical for CDCL SAT solvers. As a large amount of learned clauses reduces the overall performance, the management of the learned clause database also becomes a key component of a modern CDCL SAT solver [19, 22].

In earlier CDCL SAT solvers, the size and recent activities of learned clauses were the dominant criteria for determining the relevance of learned clauses [7]. The CDCL SAT solver Glucose [1] was the first to apply a new measure called *literal block distance* (LBD), which indicates the number of distinct decision levels in a learned clause. The learned clauses with LBD score of 2, called *glue clauses*, are of particular interest [1, 20] because a glue clause connects a block of closely related variables, and thus a relatively small number of decisions are needed to make it a *unit clause* (i.e., a clause that has all but one literals assigned under the current partial assignment). A glue clause therefore may cause a faster generation of conflicts within fewer numbers of decisions, which leads to pruning of the search space. Simply put, glue clauses have higher potential to reduce search space more quickly than other learned clauses. For this reason, all modern CDCL SAT solvers permanently store glue clauses.

Inspired by the intuitive characteristics of glue clauses, we ask the following question: Can glue clauses be used to help re-rank decision variables to improve search efficiency? We call the decision variables that have appeared in at least one glue clause up to the current search state *glue variables*, and others *nonglue variables*.

The main contributions of this paper are:

– We conduct an experiment using the 750 instances from the main track of SAT Competition 2017 and 2018 (abbreviated as SAT-2017 and SAT-2018, respectively) with four state-of-the-art CDCL SAT solvers: glucose 4.1[1] (just called Glucose), MAPLECOMSPS_PURE_LRB[2] (abbreviated as MapleLRB), Maple_LCM_Dist[3] (abbreviated as MLD, winner of SAT-2017) and MapleLCMDistChronoBT[4] (abbreviated as MLD_CBT, winner of SAT-2018). Our experiment shows that decisions with glue variables are more conflict efficient than those with nonglue variables. Furthermore, glue variables are picked up by CDCL branching heuristics disproportionately more often.
– We design a structure-aware variable score bumping method called *Glue Bumping* (GB), which dynamically bumps activity score of a glue variable based on its current activity score and (normalized) *glue level*, which is a measure of the count of glue clauses in which the variable appears. The method is simple to implement.
– We implemented the GB method on top of the same four SAT solvers mentioned above. For the 750 instances from SAT-2017 and SAT-2018, all GB extensions solve more instances than the baselines and achieve lower PAR-2 scores[5]. One of our extended solver solves 9 additional instances over the instances from SAT-2017. According to [2], this level of performance gain closely resembles to the introduction of a critical feature, which is remarkable, given the simplicity of the new method.
– We provide evidence that the frequency of glue clauses in learned clauses may serve as a reliable indicator of solving efficiency. In [16], the authors reported cor-

[1] https://www.labri.fr/perso/lsimon/glucose/
[2] https://sites.google.com/a/gsd.uwaterloo.ca/maplesat/
[3] https://baldur.iti.kit.edu/sat-competition-2017/solvers/
[4] http://sat2018.forsyte.tuwien.ac.at/solvers/main_and_glucose_hack/
[5] A metric used in SAT competitions. Defined as the sum of all runtimes for solved instances + $2 * timeout$ for unsolved instances; lowest score wins.

relations between solving efficiency of branching heuristics and standard metrics based on the global learning rate (GLR) and average LBD (aLBD) scores - higher solving efficiency is indicated by higher average GLR and lower average aLBD. We show that these two measures do not provide a consistent explanation of solving efficiency for some subsets of SAT-2017 and SAT-2018, for which the correlations are highly expected to hold. However, using a new measure based on the frequency of learned clauses that are glue, we are able to provide a consistent explanation.

The next section provides preliminaries. Section 3 reports an experiment on the role of glue variables in CDCL SAT solving, which motivates the design of a bumping scheme in Section 4. Section 5 reports an experimental analysis. In Section 7 we explain why our standard bumping scheme does not work very well for Glucose and how to fix the issue. Section 8 reports some additional experimental results with the GB method. Section 9 is about related work and future directions can be found in Section 10.

2 Preliminaries

2.1 Inner Working of a CDCL Solver

A CDCL SAT solver works by extending an initially empty *partial assignment* using two operations in an interleaving fashion: a *branching decision* and *unit propagation* (UP). A branching decision selects an unassigned variable by using a branching heuristic and assigns a boolean value to it. Following a branching decision, UP simplifies $\mathcal{F}$ by deducing a new set of implied variable assignments. UP may lead to a conflict due to a falsified or conflicting clause. *Conflict analysis* determines the root cause of a conflict and generates a *learned clause* that is added to $\mathcal{F}$ to prevent the conflict from reappearing in the future, thereby pruning the search. Search continues from a *backjumping level* computed from the learned clause. We refer the reader to [3] for more details on CDCL SAT solving.

2.2 Terminologies

We review some terminologies used in this paper.

- **Activity Based Branching Heuristics** The standard CDCL branching heuristic, such as VSIDS [19], LRB [14] and CHB [13], maintains an activity score for each variable of a given formula. During the search, a variable's involvement in conflicts contribute to the increments of its activity score. At any given state of the search, the activity score of a variable measures its involvement in the recent conflicts.
- **Global Learning Rate** (GLR) This is defined as $\frac{n_c}{n_d}$, where n_c is the number of conflicts generated in n_d decisions [16], i.e., GLR measures the average number of conflicts that a solver generates per decision.
- **Literal Block Distance (LBD)** The LBD of a learned clause θ indicates the number of distinct decision levels in θ [1]. If LBD(θ) $= k$, then θ contains k propagation blocks, where each block has been propagated within the same branching decision. Intuitively, variables in a block are closely related. Learned clauses with lower LBD score tend to have higher quality.

- **Glue Clauses** These are the learned clauses with LBD score of 2 [1], which have the potential for fast propagations of truth values under a partial assignment.

Let $\mathcal{F}$ be a SAT formula. Suppose a CDCL solver Ψ is solving $\mathcal{F}$ and s is its current search state. At s, Ψ has taken $d > 0$ decisions and has learned a set of glue clauses. A *glue variable* is a variable that has appeared in at least one glue clause up to the search state s. Other variables that have not appeared in a glue clause are called *nonglue variables*. A *glue decision* is the branching decision that selects a glue variable and a *nonglue decision* is the branching decision that selects a nonglue variable. Suppose that until s, Ψ has taken gd glue decisions (resp. ngd nonglue decisions) which generated gc conflicts (resp. ngc conflicts).

- *Learning Rate* (LR): In contrast with GLR (global learning rate) where the rate of conflict generation is over all decisions, we are also interested in such rates over glue decisions only or over nonglue decisions only, up to a search state. *LR with glue decisions* is defined as $\frac{gc}{gd}$, while *LR with nonglue decisions* is defined as $\frac{ngc}{ngd}$.
- *Average LBD* (aLBD): This is the average LBD score per conflict generated solely by glue decisions or solely by nonglue decisions. Let $sumLBD_{gc}$ (resp. $sumLBD_{ngc}$) be the sum of LBD scores of the learned clauses derived from those gc (resp. ngc) conflicts. The *aLBD with glue decisions* (resp. *nonglue decisions)* is defined as $\frac{sumLBD_{gc}}{gc}$ (resp. $\frac{sumLBD_{ngc}}{ngc}$).

3 Conflict Efficiency of Glue Variables

In this section, we report an experiment that studies the role played by glue variables in CDCL SAT solving, which shows that glue decisions are more conflict efficient (i.e., achieve higher average LR and lower average aLBD, in general) than nonglue decisions and the branching heuristics of modern CDCL SAT solvers exhibit bias towards selection of glue variables over nonglue variables.

The solvers in this experiment are Glucose, MapleLRB, MLD, and MLD_CBT. The branching heuristics used in the first two solvers are, respectively, VSIDS [19] and LRB [14]. For the next two, the branching heuristics are based on a combination of three heuristics, VSIDS, LRB, and Dist [24].

We run all 750 instances used in the main track of SAT-2017 (350 instances) and 2018 (400 instances) with 5000 seconds timeout limit per instance. We instrumented the four solvers to collect the following statistics for each instance: (i) the numbers of glue and nonglue decisions, (ii) LR and aLBD for both glue and nonglue decisions, and (iii) the numbers of glue and nonglue variables. For each instance, all the measurements are taken at the final search state (i.e., either after satisfiability/unsatisfiability is determined or after timeout). All experiments are run on a Linux workstation with 64 Gigabytes RAM and processor clock speed of 2.40 GHZ.

3.1 Conflict Generation Power of Glue Variables

Table 1 shows a comparison of average LR and average aLBD for glue and nonglue decisions, grouped by satisfiable, unsatisfiable and unsolved instances. Comparing column D1 and D2, on average, all solvers achieve significantly higher LR with glue decisions. For all three categories of instances, MLD and MLD_CBT achieve significantly lower average LBD (compare columns E1 and E2) for glue decisions. For Glucose and MapleLRB, the numbers under E1 and E2 are largely comparable, without showing significant gaps.

(A) Systems	**(B) Type**	**(C) #Inst**	**(D) Average of Learning Rate (LR)**		**(E) Average of aLBD**	
			(D1) Glue Decisions	**(D2) Nonglue Decisions**	**(E1) Glue Decisions**	**(E2) Nonglue Decisions**
Glucose	SAT	180	**0.55**	0.41	18.44	**18.18**
	UNSAT	191	**0.56**	0.44	**11.2**	11.4
	Unsolved	379	**0.57**	0.48	**24.76**	25.48
MapleLRB	SAT	194	**0.47**	0.38	20.18	**19.25**
	UNSAT	190	**0.58**	0.46	**11.92**	12.39
	Unsolved	366	**0.48**	0.44	34.86	**33.39**
MLD	SAT	235	**0.47**	0.19	**31.76**	40.55
	UNSAT	207	**0.59**	0.27	**12.8**	30.1
	Unsolved	308	**0.52**	0.37	**24.23**	34.09
MLD_CBT	SAT	238	**0.51**	0.21	**32.1**	41.9
	UNSAT	215	**0.61**	0.27	**13.17**	24.74
	Unsolved	297	**0.53**	0.37	**25.25**	36.7

Table 1. Comparison of average LR (higher is better) and average aLBD (lower is better) for glue and nonglue decisions.

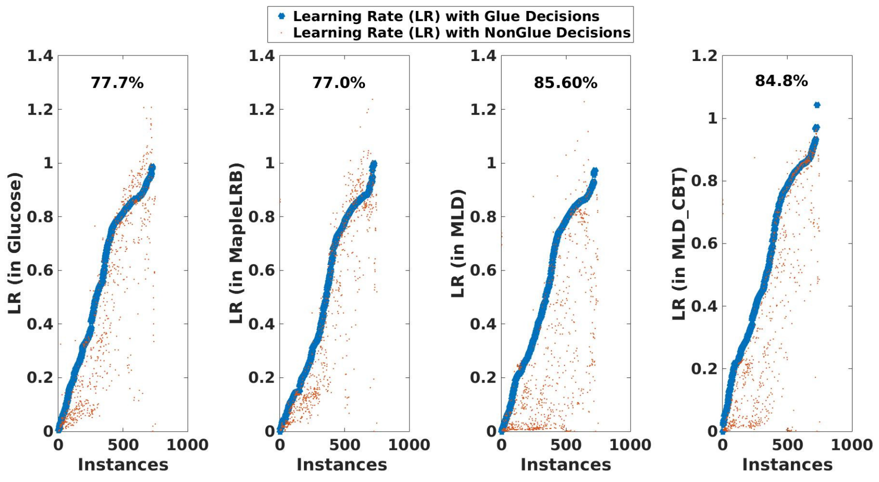

Fig. 1. Comparison of LR values for glue and nonglue decisions. Instances are sorted by the LR values of glue decisions. The number at the top of each plot represents the percentage of instances, for which LR of glue decisions are higher than LR of nonglue decisions.

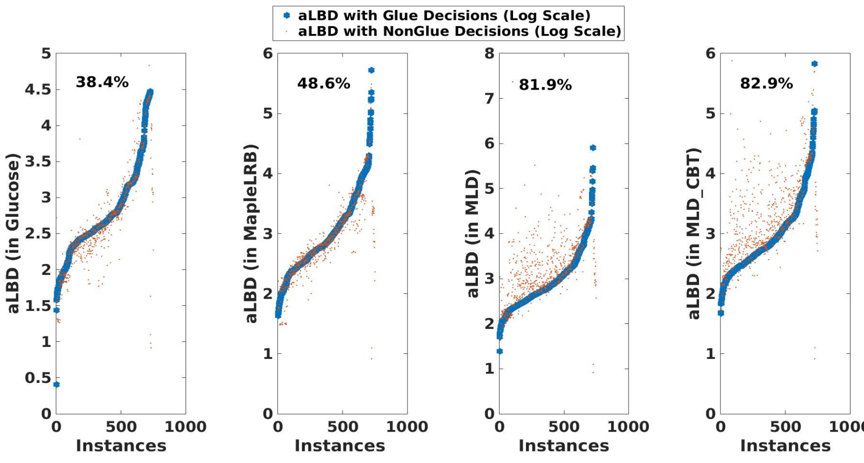

Fig. 2. Comparison aLBD scores (in Log Scale). Instances are sorted by the aLBD of glue decisions. The number at the top of each plot represents the percentage of instances, for which aLBD of glue decisions are lower than aLBD of nonglue decisions.

To confirm that the average values for these 2 measures reported in Table 1 reflect the actual distribution of these measures, we plot the LR and aLBD values for the 750 instances for the four solvers.

Figure 1 shows per instance LR values for both glue and nonglue decisions for the four solvers in four subplots. For all solvers and for large majority of the instances, glue decisions achieve higher LR than nonglue decisions.

Figure 2 shows per instance aLBD scores (in Log scale) for the 750 instances for glue and nonglue decisions.

– For Glucose and MapleLRB (first and second plots, Figure 2), for more than half of the instances, the aLBD score of the learned clauses by nonglue decisions is lower than the aLBD score of the learned clauses by glue decisions. The average values of aLBD under columns E1 and E2 in Table 1 for Glucose and MapleLRB reflect the ground data.
– We observe quite a different scenario in case of MLD and MLD_CBT (third and fourth plot, Figure 2). The aLBD scores of the learned clauses by glue decisions are lower for large majority of the instances. Again, the average values of aLBD under columns E1 and E2 in Table 1 for MLD and MLD_CBT reflect the ground data.

Overall, glue decisions are more conflict efficient than nonglue decisions for all the tested solvers. For average aLBD with glue decisions, the winners of the last two SAT competitions, MLD and MLD_CBT, generate substantially lower (better) values.

3.2 Selection Bias of Glue Variables

We are interested in the question: Do conflict guided CDCL branching heuristics exhibit any bias towards glue variables over nonglue variables?

Given a SAT formula $\mathcal{F}$ and a solver Ψ, we define *glue fraction* (GF) (resp. *nonglue fraction* (NF)) as the fraction of variables in $\mathcal{F}$ that are glue (resp. nonglue) variables, after Ψ ends its run with $\mathcal{F}$. GF (resp. NF) measures the pool size of glue (resp. nonglue) variables in $\mathcal{F}$ with respect to the total number of variables in $\mathcal{F}$.

Over the 750 instances, column B of Table 2 shows the average GF and average percentage of glue decisions and column C shows the average nonglue fraction and the average percentage of nonglue decisions. It shows that for all the four solvers, on average, the pool size of glue variables is significantly smaller than the pool size of nonglue variables (columns B1 and C1). For all the four solvers, on average, glue decisions relative to glue variables pool size are higher (column B2) than nonglue decisions (column C2) relative to the nonglue variables pool size.

(A) Systems	**(B) Average for Glue Variable**		**(C) Average for Nonglue Variables**	
	GF (B1)	**Glue Decisions % (B2)**	**NF (C1)**	**Noglue Decisions % (C2)**
Glucose	0.25	65.43%	0.75	34.57%
MapleLRB	0.21	63.14%	0.69	36.86%
MLD	0.22	47.60%	0.78	52.60%
MLD_CBT	0.22	48.76%	0.78	51.24%

Table 2. Biased Selection of Glue Variables

In summary, the four state-of-the-art CDCL SAT solvers make a much larger percentage of glue decisions against relatively smaller pools of glue variables. This shows the bias of these solvers towards selecting glue variables in branching decisions.

4 Activity Score Bumping for Glue Variables

From the above analysis, it is clear that decisions with glue variables are more conflict efficient than with nonglue variables. An interesting question is how we can exploit this empirical characteristic for more efficient SAT solving. Here, we present a score bumping method, called *Glue Bump* (GB), which bumps the activity score of glue variables. The amount of bumping for a glue variable depends on the appearance count of that variable in glue clauses and its current activity score.

Glue Level Let G be the set of learned glue clauses until search state s. The *glue level* of a glue variable v, denoted $gl(v)$, is defined to be the number of glue clauses in G in which v appears.[6] A higher glue level indicates higher potential to create conflicts.

4.1 The GB Method

By using the current activity scores and (normalized) glue levels of glue variables (we will comment on normalization shortly), the GB method bumps the activity scores of

[6] We omit the parameter s since the glue level of a variable is always computed w.r.t. a underlying search state by default, without confusion.

glue variables. This gives higher preference to recently active glue variables with high glue levels. The GB method is simple to implement and conveniently integrates with activity based standard CDCL heuristics.

The GB method modifies a CDCL SAT solver Ψ by adding the following two procedures, which are called at different states of the search. We denote by Ψ^{gb} the GB extension of the baseline solver Ψ.

```
Alg. 1: Increase Glue Level
Input: A newly learned glue clause $\theta$
1  For $i \leftarrow 1$ to $|\theta|$
2      $v \leftarrow varAt(\theta, i)$
3      $gl(v) \leftarrow gl(v) + 1$
4  End
```

```
Alg. 2: Bump Glue Variable
Input: A glue variable $v$

1  $bf_v \leftarrow activity(v) * \left(\frac{gl(v)}{|G|}\right)$
2  $activity(v) \leftarrow activity(v) + bf_v$
```

Increase Glue Level: Whenever Ψ^{gb} learns a new glue clause θ, it invokes Alg. 1. For each variable v in θ, the glue level of v is increased by 1 (line 3).

Bump Glue Variable: Alg. 2 bumps a glue variable v. It computes the *bumping factor* for v, denote bf_v, by combining both of the current activity score and normalized glue level of v (line 1). The bumping is performed by adding the bumping factor of v to the activity score of v, which becomes the new activity score for v (line 2).

Glue Level Normalization: The glue level of a glue variable can grow unboundedly with the discovery of more and more glue clauses. The activity score of a glue variable also grows, but at a different rate. Thus scaling the glue level is necessary.

We normalize $gl(v)$ to (0,1][7] by

$$\frac{gl(v)}{|G|}$$

where G is the set of glue clauses discovered by the search so far. The normalization scales the glue levels of glue variables by the *total number* of glue clauses discovered by the search so far.

Delayed Bumping of Glue Variables: Ψ^{gb} does not perform the bumping of v right after its hosting clause θ is discovered. It delays the bumping (i.e., the invocation of the *Bump Glue Variable* procedure) of v until it is unassigned by backtracking. This is a subtle point which we explain below.

– The glue clause θ is the latest learned clause and all the variables in θ including v are assigned at the current search state. At this stage, any score bumping that v receive would not be used until it gets unassigned.

[7] At a given state of the search, a given glue variable v appears in at least one glue clause. So, the glue level of v (which is the count of number of glue clauses in which v appears), $gl(v) > 0$. After dividing $gl(v)$ with $|G|$, the normalized glue level remains larger than 0. Hence, the normalization normalizes the glue level within the range (0,1].

– Let $T = d^e - d^s > 0$ be the decision window starting from the decision d^s that generates θ and ending at the decision d^e in which v gets unassigned. Within T, the search may generate more glue clauses in some of which v may appear. Furthermore, v may get involved in several conflicts during T and may have its activity score increased. It is clear that the bumping factor of v computed at d^e reflects a more recent measure than the one computed at d^s. By delaying the bumping of v until d^e when v has just got unassigned and become a candidate variable for branching, the GB method boosts the activity score of v by a more recent bumping factor.

5 Implementation and Experiments

5.1 Implementation

We implemented the GB method on top of the CDCL SAT solvers Glucose, MapleLRB, MLD, and MLD_CBT and call the extended solvers Glucose^{gb}, MapleLRB^{gb}, MLD^{gb}, and MLD_CBT^{gb}, respectively. The baseline solvers do not distinguish between glue and nonglue variables, except Glucose, which bumps activity scores of variables that are propagated from a glue clause.

In Glucose^{gb} and MapleLRB^{gb}, on the unassignment of a glue variable, the GB method updates the activity score of that glue variable by VSIDS and LRB, respectively, which are the heuristics used in their baselines. As remarked earlier, the baseline solvers MLD and MLD_CBT employ three heuristics, namely DIST, VSIDS and LRB, which are activated at different phases of the search. At any given phase, on the unassignment of a glue variable, MLD^{gb} and MLD_CBT^{gb} update the activity score of that glue variable for the currently active heuristic at that phase.

5.2 Experiments

We conduct our experiments with four extended solvers with the same set of 750 instances on the same machine with 5000 seconds timeout per instance. Here, we present comparisons between the extended solvers and their counterpart baselines in terms of solved instances, solved time and PAR-2 score.

Solved Instances Comparison Table 3 compares the four extended solvers with their baselines. Both MapleLRB^{gb} and MLD^{gb} solves 13 more instances (9 SAT, 4 UNSAT for the former and 11 SAT, 2 UNSAT for the latter). Glucose^{gb} solves 4 more instances (2 SAT, 2 UNSAT), and MLD_CBT^{gb} solves 2 additional instances (both SAT).

According to Audemard and Simon [2], solving 10 or more instances on a fixed set of instances from a competition by using a new technique, generally shows a critical feature. MapleLRB^{gb} solves 9 more instances over the instances from SAT-2017 and and MLD^{gb} solves 8 additional instances over the instances from SAT-2018. The gains with MapleLRB^{gb} and MLD^{gb} are significant and closely resemble to the introduction of a critical feature.

Systems	**SAT Comp-17**				**SAT Comp-18**				**SAT Comp-2017 and 2018**			
	SAT	**UNSAT**	**Total**	**PAR-2**	**SAT**	**UNSAT**	**Total**	**PAR-2**	**SAT**	**UNSAT**	**Total**	**PAR-2**
Glucose	83	96	179	1893	97	95	192	2274	180	191	371	4167
Glucose^{gb}	**86 (+3)**	96 (+0)	**182 (+3)**	**1868**	96 (-1)	**97 (+2)**	193 (+0)	**2273**	**182 (+2)**	**193 (+2)**	375 (+4)	**4141**
MapleLRB	80	95	175	1897	114	95	209	2069	194	190	384	3966
MapleLRB^{gb}	**87 (+7)**	**97 (+2)**	**184 (+9)**	**1824**	**117 (+3)**	**96 (+1)**	**213 (+4)**	**2027**	**204 (+10)**	**193 (+3)**	**397 (+13)**	**3851**
MLD	99	106	205	1635	136	101	237	1807	235	207	442	3442
MLD^{gb}	**103 (+4)**	**107 (+1)**	**210 (+5)**	**1593**	**143 (+7)**	**102 (+1)**	**245 (+8)**	**1725**	**246 (+11)**	**209 (+2)**	**455 (+13)**	**3318**
MLD_CBT	**103**	113	216	1565	135	**102**	237	1800	238	215	453	3365
MLD_CBT^{gb}	102 (-1)	**114 (+1)**	216 (+0)	**1539**	**138 (+3)**	101 (-1)	**239 (+2)**	**1756**	**240 (+2)**	215 (+0)	**455 (+2)**	**3295**

Table 3. Comparison of the four baseline solvers with their GB extensions for the instances from SAT-2017 and SAT-2018. The PAR-2 scores are scaled down by the factor of $\frac{1}{10{,}000}$.

Solve Time Comparison Figure 3 compares the performance of Glucose^{gb} (blue line), MapleLRB^{gb} (red line), MLD^{gb} (yellow line) and MLD_CBT^{gb} (purple line) against their baselines. This figure plots the difference in the number of instances solved as a function of time. At most points in time, each of MapleLRB^{gb}, MLD^{gb}, and MLD_CBT^{gb} solves more problems. This is particularly pronounced for MLD^{gb} (yellow line) at earlier time points, for MLD_CBT^{gb} (purple line) on mid range time points. The improvement for MapleLRB^{gb} (red line) remains steady, with a brief downward slope in the middle. Glucose^{gb} performs slightly worse than Glucose at most of the times.

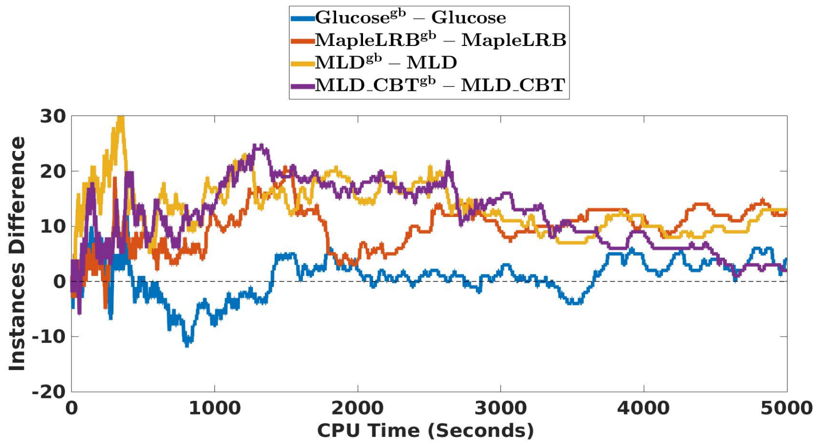

Fig. 3. Solve time comparisons. For any point above 0 in the vertical axis, our extensions solve more instances than their baselines at the time point in the horizontal axis.

PAR-2 Score Comparison In SAT competitions, solvers are ranked based on their PAR-2 scores. A PAR-2 score is computed as the sum of all runtimes for solved instances + $2 * timeout$ for unsolved instances; solvers of lower PAR-2 scores are better.

Table 3 shows that all our extended versions achieve a lower PAR-2 score than the baselines for all the problem sets. Overall, the percentage of PAR-2 score reductions (computed from the last column of Table 3) with MLD^{gb}, MapleLRB^{gb} and MLD_CBT^{gb} are 3.73%, 2.98% and 2.12%, respectively, which are considered significant with respect to SAT competition. For example, in SAT-2018 the winning solver has a PAR-2 score which is a reduction of only 0.81% over the runner-up.[8]

Glucose^{gb} also lowers the PAR-2 score but only by 0.60%. The improvement is less impressive than with other three GB extensions. In Section 7, we will discuss the reason and show that this performance gap is not an indication of ineffectiveness of the GB method.

Finally in this section, as many benchmarks in SAT-2017/SAT-2018 are of industrial strength, we provide the information about the benchmark families, for which our GB method is particularly efficient. Table 4 lists those benchmark families for which our GB extended solvers solve at least 2 more instances than their baselines.

GB Extensions	Benchmarks/SAT Comp	Solved by Baseline	Solved By GB extensions	% Improvements
Glucose^{gb}	Integer Prefix/2017	28	**32 (+4)**	**14.32%**
	Soos/2018	8	**11 (+3)**	**37.5%**
	Ofer/2018	9	**11 (+2)**	**18.18%**
MapleLRB^{gb}	T/2017	28	**31 (+3)**	**9.67%**
	Integer Prefix/2017	27	**30 (+3)**	**10.00%**
	Klieber/2017	17	**19 (+2)**	**10.52%**
	Chen/2018	2	**4 (+2)**	**50.00%**
	Ofer/2018	5	**7 (+2)**	**28.57%**
	Scheel/2018	18	**20 (+2)**	**10.00%**
MLD^{gb}	ak128/2017	11	**13 (+2)**	**15.38%**
	Heule/2018	16	**20 (+4)**	**20.00 %**
MLD_CBT^{gb}	Xiao/2018	7	**9 (+2)**	**22.22%**
	Collatz/2018	7	**10 (+3)**	**30.30%**

Table 4. Benchmark families for which the GB extended solvers solve at least two more instances than their baselines.

6 A New Measure of Solving Efficiency

In [16], the authors show that on average, better branching heuristics have higher GLR values and lower average LBD (aLBD) scores of the learned clauses. In Table 5, we compare our extended solvers and their baselines in terms of the average GLR values and average aLBD scores. All the solvers with GB extension generate conflicts at about the same rate as their corresponding baselines and achieve slightly smaller average aLBD scores. These results are largely consistent with [16].

Of course, one can pick up some subset of the benchmarks and show that the standard metrics that are based on average GLR and average aLBD may not be always applicable. On the other hand, for some subsets of benchmarks it may be highly expected that these metrics should be re-enforced. In this section, we select two subsets of this kind, but surprisingly the standard metrics do not provide a consistent explanation; they even lead to opposite conclusions. However, we show that a simple new measure,

[8] http://sat2018.forsyte.tuwien.ac.at/index.php?cat=rules

Systems	Glucose	Glucose^{gb}	MapleLRB	MapleLRB^{gb}	MLD	MLD^{gb}	MLD_CBT	MLD_CBT^{gb}
avg. GLR	0.49	0.49	0.48	0.48	0.40	0.40	0.40	**0.41**
avg. aLBD	20.09	**19.93**	24.88	**24.79**	27.73	**27.36**	27.59	**27.26**

Table 5. Comparison of average GLR and aLBD score for GB extension solvers and baselines over the 750 test instances.

based on the fraction of learned clauses that are glue clauses, provides a consistent explanation of solving efficiency.

6.1 Metrics for Solving Efficiency

We define a new performance metric called *Glue to Learned* (G2L). Then we present an analysis with three metrics, two standard ones and G2L on two different types of instances, where the baseline heuristics and their GB extensions show opposite strengths.

Glue to Learned (G2L). G2L represents the fraction of learned clauses that are glue clauses. More precisely, it is defined by $\frac{\#glue\_clauses}{\#learned\_clauses}$, where our solver Ψ has learned *#learned_clauses* clauses for a given run on a given formula, among which *#glue_clauses* are glue clauses.

Relating G2L to Solving Efficiency The performance of branching heuristics correlates well with average GLR and the average aLBD scores at large scale. However, these two metrics fail to explain the performance of the the baseline heuristics and their GB extensions for two specially designed subsets of instances from SAT-2017 and SAT-2018:

- $\mathbf{GB_{exclusive}}$: These instances are solved by Ψ^{gb}, but not by Ψ.
- $\mathbf{Baseline_{exclusive}}$: These instances are solved by Ψ, but not by Ψ^{gb}.

Table 6 compares the four baseline solvers and their GB extensions in terms of average GLR, average aLBD, and average G2L for $\mathbf{GB_{exclusive}}$ and $\mathbf{Baseline_{exclusive}}$ instances. For these two types of instances, it is expected that the solving efficiency will positively (resp. negatively) correlate with average GLR (resp. average aLBD).

We observe:

- Average GLR: For instances from $\mathbf{GB_{exclusive}}$ (Column C) and $\mathbf{Baseline_{exclusive}}$ (Column D), the better branching heuristics have lower average GLR values. This is surprising since the performance of branching heuristics is negatively correlated with average GLR values. This is highly inconsistent with the results reported in [16].
- Average aLBD: In both $\mathbf{GB_{exclusive}}$ and $\mathbf{Baseline_{exclusive}}$, the better heuristics have lower average aLBD in Glucose and MapleLRB based systems. This is consistent with the results from [16]. However, in MLD and MLD_CBT based systems, the better branching heuristics have higher average aLBD scores, which is inconsistent with the results of [16].

(A) Systems	**(B) Employed Heuristics**	**(C)** $\mathbf{GB_{exclusive}}$				**(D)** $\mathbf{Baseline_{exclusive}}$			
		#inst	**avg. GLR**	**avg. aLBD**	**avg. G2L**	**#inst**	**avg. GLR**	**avg. aLBD**	**avg. G2L**
Glucose	{VSIDS}	33	**0.56**	28.60	0.0005	29	0.59	**18.52**	**0.0015**
Glucose^{gb}	{VSIDS}^{gb}		0.53	**24.69**	**0.0016**		**0.62**	20.14	0.00078
MapleLRB	{LRB}	27	**0.50**	26.06	0.00073	14	0.47	**30.75**	**0.00046**
MapleLRB^{gb}	{LRB}^{gb}		0.46	**20.38**	**0.00126**		**0.48**	32.02	0.00037
MLD	{Dist/VSIDS/LRB}	28	**0.55**	**23.60**	0.00029	15	0.53	26.70	**0.0011**
MLD^{gb}	{Dist/VSIDS/LRB}^{gb}		0.51	26.04	**0.00032**		**0.58**	**23.21**	0.0009
MLD_CBT	{Dist,VSIDS,LRB}	26	**0.49**	**26.08**	0.0006	24	0.51	29.64	**0.00065**
MLD_CBT^{gb}	{Dist/VSIDS/LRB}^{gb}		0.43	36.24	**0.0011**		**0.55**	**25.42**	0.00037

Table 6. Comparison between baselines and their GB extensions for average GLR, average aLBD and average G2L for instance sets $\mathbf{GB_{exclusive}}$ and $\mathbf{Baseline_{exclusive}}$; Column B shows the heuristics employed for the systems in column A, where $\{x\}^{gb}$ in column B is the GB extension of baseline heuristic x. Column C (resp. Column D) shows three metrics: avg. GLR, avg. aLBD and avg. G2L for instance category $\mathbf{GB_{exclusive}}$ (resp. $\mathbf{Baseline_{exclusive}}$), where the subcolumn #inst shows the number of $\mathbf{GB_{exclusive}}$ (resp. $\mathbf{Baseline_{exclusive}}$) instances for which we are comparing the heuristics in Column B.

– Average G2L: For both $\mathbf{GB_{exclusive}}$ and $\mathbf{Baseline_{exclusive}}$, the better heuristics always achieve higher average G2L values. The biggest difference in G2L is 220% (0.0016-0.0005) for VSIDS and VSIDS^{gb} in Glucose and Glucose^{gb} for the $\mathbf{GB_{exclusive}}$. We observe a significantly larger average G2L values for all the other cases as well (compare the bold values in avg. G2L subcolumn with the values not in bold, for both columns C and D in Table 6).

To summarize, for instances for which one heuristic is better than the other, the correlation between the performance of branching heuristics and average GLR and average aLBD is not always consistent with the results of [16]. The average value of the new metric G2L positively correlates with the performance of the branching heuristics in each case.

7 Effect of Glue Level Normalization

Earlier, we noticed that Glucose^{gb} showed less improvement than the other GB extensions. Compared to its baseline, Glucose^{gb} solves 4 additional instances, lowers the PAR-2 score only by 0.60% (Table 3), and solves instances at a slower rate than its baseline at most time points (Figure 3).

Unlike the other 3 baseline solvers used in our experiments, the baseline solver Glucose already bumps variables that are propagated from glue clauses by using VSIDS [1]. These variables are a subset of what we call glue variables. Thus in Glucose^{gb}, these variables get bumped from two sources: from GB bumping and from VSIDS. We hypothesize that the relatively weak performance of Glucose^{gb} comes from this imbalance.

We tested this hypothesis by changing the glue level normalization method in GB to decrease the bumping factor in Alg. 2. For a given glue variable v, instead of dividing $gl(v)$ by $|G|$, we divide by a bigger factor: $\frac{gl(v)}{\sum_{\theta \in G} len(\theta)}$, where $len(\theta)$ is the number of variables in the glue clause θ. The sum is the total number of the glue variables discovered so far in the search. If the average length of the glue clauses in G is n, then in this version, $gl(v)$ is scaled-down n times more than before.

We repeated our experiment with this version. Over the 750 instances from SAT-2017 and 2018, Glucose^{gb} now solves 11 more instances than Glucose and and lowers the PAR-2 score by 2.86%. For the other three GB extensions, this reduction does not work well.

8 Additional Experimental Results

8.1 Results with Benchmarks from SAT-2016

We performed an additional experiment with all of our GB extended solvers for the bumping factor $\frac{gl(v)}{|G|}$ for the benchmark instances from SAT-2016 [9]. In the below, we summarize the results:

- Both Glucose and its GB extension solve equal number of problems (SAT 64, UNSAT 123, total 187).
- MapleLRB^{gb} solves (SAT 69, UNSAT 102, total 171) equal number of instances as its baseline *MapleLRB* (SAT 67, UNSAT 104, total 171).
- MLD^{gb} solves 2 more instances (SAT 73, UNSAT 135, total 208) than MLD (SAT 69, UNSAT 137, total 206).
- MLD_CBT^{gb} solves 2 less instances (SAT 66, UNSAT 137, total 203) than its baseline MLD_CBT (SAT 65, UNSAT 140, total 205).

For this benchmark set, the GB method does not work as well as it works for the benchmarks from SAT-2017 and 2018. Further tuning of the GB method is expected to improve the performance of the GB extended solvers on this benchmark set.

8.2 Experiment with Non-Delayed Bumping

We performed a smaller scale experiment with MLD over the 350 instances from SAT competition-2017, where we bump the score of the glue variables as soon as their hosting glue clause is learned (i.e., without delaying the bumping). MLD, with this version of glue variable bumping, solves 2 more UNSAT instances, but 2 less SAT instances than the baseline. As this non-delayed bumping did not appear to be promising with MLD, we did not perform any further experiment.

9 Related Work

As remarked earlier, Glucose [1] explicitly increases the activity scores of variables of the learned clause that were propagated by a glue clause. In their work, the bumping

[9] A total of 483 instances (283 applications, 200 crafted) after removing 17 duplicate instances between SAT-2016 and SAT-2017.

was based on VSIDS score bumping scheme. In contrast, we increase the activity scores of all variables that appear in glue clauses based on their normalized glue level.

In [12], the authors studied the behavior of Glucose with respect to *eigencentrality*, a precomputed static measure of ranking of the variables in industrial SAT instances. They show that the branched and propagated variables in Glucose have high eigencentrality and compared to the variables that appear in conflict clauses, the variables that appear in learned clauses are more eigencentral. In contrast, we dynamically characterize glue and nonglue variables within the course of a search and show that decisions with glue variables are more conflict efficient than decisions with nonglue variables.

The authors of [15] show that the VSIDS heuristic branches disproportionately more often on variables that are bridges between communities. Here, we have shown that CDCL heuristics branch disproportionately more often on glue variables with respect to their relatively smaller pool size.

In [9], the authors exploit the *betweeness centrality* measure of variables in industrial SAT formulas to design new heuristics. This measure is precomputed for a given instance. In contrast, we compute the normalized glue level of the variables dynamically during the search.

10 Summary and Future Work

In this work, we showed experimentally that decisions with variables appearing in glue clauses are more conflict efficient than decisions with other variables, and state-of-the-art CDCL SAT solvers tend to make glue decisions more often. Motivated by these observations, we developed a structure-aware CDCL variable bumping scheme, which increases the heuristic score of a glue variable based on the frequency of its appearance in glue clauses. Our empirical evaluation showed the effectiveness of the new method on the main track instances from SAT-2017 and SAT-2018 with four state-of-the-art CDCL SAT solvers. Lastly, we found that for some subsets of SAT-2017 and SAT-2018 benchmarks, our experimental data are surprisingly inconsistent with the standard performance metrics based on GLR and average LBD. We showed that for these subsets of benchmarks, the measure based on the fraction of learned clauses that are glue clauses provides a consistent explanation of our experimental data.

A number of questions deserve further considerations. The first is on the relationships between normalized glue level and other centrality measures, such as eigencentrality or betweenness centrality. The notion of glue level is central in our glue bumping scheme. Can we design clause deletion heuristics based on the notion of glue level? A similar question can be asked for the G2L metric: can we design more efficient branching heuristics based on this measure of solving efficiency?

Acknowledgements

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

References

1. Gilles Audemard and Laurent Simon. Predicting learnt clauses quality in modern SAT solvers. In *Proceedings of IJCAI 2009*, pages 399–404, 2009.
2. Gilles Audemard and Laurent Simon. Refining restarts strategies for SAT and UNSAT. In *Proceedings of CP 2012*, pages 118–126, 2012.
3. Armin Biere, Marijn Heule, Hans V. Maaren, and Toby Walsh. *Handbook of Satisfiability: Volume 185 Frontiers in Artificial Intelligence and Applications*. IOS Press, Amsterdam, The Netherlands, 2009.
4. Cristian Cadar, Vijay Ganesh, Peter M. Pawlowski, David L. Dill, and Dawson R. Engler. EXE: automatically generating inputs of death. In *Proceedings of CCS 2006*, pages 322–335, 2006.
5. Stephen A. Cook. The complexity of theorem-proving procedures. In *Proceedings of the 3rd Annual ACM Symposium on Theory of Computing 1971*, pages 151–158, 1971.
6. Niklas Eén and Armin Biere. Effective preprocessing in SAT through variable and clause elimination. In *Proceedings of SAT 2005*, pages 61–75, 2005.
7. Niklas Eén and Niklas Sörensson. An extensible sat-solver. In *Proceedings of SAT 2003. Selected Revised Papers*, pages 502–518, 2003.
8. 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, 2006.
9. Sima Jamali and David Mitchell. Centrality-based improvements to CDCL heuristics. In *Proceedings of SAT 2018*, pages 122–131, 2018.
10. Matti Järvisalo, Armin Biere, and Marijn Heule. Blocked clause elimination. In *Proceedings of TACAS 2010*, pages 129–144, 2010.
11. Matti Järvisalo, Marijn Heule, and Armin Biere. Inprocessing rules. In *Proceedings of IJCAR 2012*, pages 355–370, 2012.
12. George Katsirelos and Laurent Simon. Eigenvector centrality in industrial SAT instances. In *Proceedings of CP 2012*, pages 348–356, 2012.
13. Jia Hui Liang, Vijay Ganesh, Pascal Poupart, and Krzysztof Czarnecki. Exponential recency weighted average branching heuristic for SAT solvers. In *Proceedings of AAAI 2016*, pages 3434–3440, 2016.
14. Jia Hui Liang, Vijay Ganesh, Pascal Poupart, and Krzysztof Czarnecki. Learning rate based branching heuristic for SAT solvers. In *Proceedings of SAT 2016*, pages 123–140, 2016.
15. Jia Hui Liang, Vijay Ganesh, Ed Zulkoski, Atulan Zaman, and Krzysztof Czarnecki. Understanding VSIDS branching heuristics in conflict-driven clause-learning SAT solvers. In *Proceedings of Haifa Verification Conference, HVC 2015*, pages 225–241, 2015.
16. 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. In *Proceedings of SAT 2017*, pages 119–135, 2017.
17. 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 IJCAI 2017*, pages 703–711, 2017.
18. Fabio Massacci and Laura Marraro. Logical cryptanalysis as a SAT problem. *J. Autom. Reasoning*, 24(1/2):165–203, 2000.
19. Matthew W. Moskewicz, Conor F. Madigan, Ying Zhao, Lintao Zhang, and Sharad Malik. Chaff: Engineering an efficient SAT solver. In *Proceedings of Design Automation Conference, DAC 2001*, pages 530–535, 2001.
20. Chanseok Oh. Between SAT and UNSAT: the fundamental difference in CDCL SAT. In *Proceedings of SAT 2015*, pages 307–323, 2015.

21. Jussi Rintanen. Engineering efficient planners with SAT. In *Proceedings of ECAI 2012*, pages 684–689, 2012.
22. João P. Marques Silva and Karem A. Sakallah. GRASP: A search algorithm for propositional satisfiability. *IEEE Trans. Computers*, 48(5):506–521, 1999.
23. Mate Soos, Karsten Nohl, and Claude Castelluccia. Extending SAT solvers to cryptographic problems. In *Proceedings of SAT 2009*, pages 244–257, 2009.
24. Fan Xiao, Mao Luo, Chu-Min Li, Felip Manya‘, and Zhipeng Lu. MapleLRB_LCM, Maple_LCM, Maple_LCM_Dist, MapleLRB_LCMoccRestart and Glucose3.0+width in sat competition 2017. In *Proceedings of SAT Competition 2017*, pages 22–23, 2017.