Coverage-Based Clause Reduction Heuristics for CDCL Solvers

Hidetomo Nabeshima[1(✉)] and Katsumi Inoue[2]

[1] University of Yamanashi, Kofu, Japan
nabesima@yamanashi.ac.jp
[2] National Institute of Informatics, Tokyo, Japan
inoue@nii.ac.jp

Abstract. Many heuristics, such as decision, restart, and clause reduction heuristics, are incorporated in CDCL solvers in order to improve performance. In this paper, we focus on learnt clause reduction heuristics, which are used to suppress memory consumption and sustain propagation speed. The reduction heuristics consist of evaluation criteria, for measuring the usefulness of learnt clauses, and a reduction strategy in order to select clauses to be removed based on the criteria. LBD (literals blocks distance) is used as the evaluation criteria in many solvers. For the reduction strategy, we propose a new concise schema based on the coverage ratio of used LBDs. The experimental results show that the proposed strategy can achieve higher coverage than the conventional strategy and improve the performance for both SAT and UNSAT instances.

1 Introduction

Many heuristics, such as decision, phase selection, restart, and clause reduction heuristics, are used in CDCL solvers in order to improve performance. For example, Katebi et al., show that decision and restart heuristics have resulted in significant performance improvement in their paper evaluating the components of CDCL solvers [5]. In this paper, we focus on clause reduction heuristics, which remove useless learnt clauses in order to suppress memory consumption and sustain propagation speed. Clause reduction is practically required since CDCL solvers learn a large number of clauses while solving. The reduction heuristics consist of evaluation criteria to measure the usefulness of learnt clauses and the reduction strategy for selecting clauses to be removed based on the criteria.

As the former evaluation criteria, LBD (literals blocks distance) [1] is implemented in many solvers. LBD is an excellent measure to identify learnt clauses that are likely to be used frequently. In this paper, we present experimental evidence of the identification power of LBD in a wide range of instances. Moreover, we show that an appropriate threshold for LBD, which are used to decide if clauses should be maintained or not, is determined depending on a given instance. However, a certain fixed threshold of LBD is often used in latter reduction strategies. In this paper, we propose a new reduction strategy based on the coverage of used LBDs, which dynamically computes an appropriate LBD

S. Gaspers and T. Walsh (Eds.): SAT 2017, LNCS 10491, pp. 136–144, 2017.
DOI: 10.1007/978-3-319-66263-3_9

threshold in order to cover most propagations and conflicts. The experimental results show that our schema effectively maintains the used clauses and achieves performance improvement for both SAT and UNSAT instances.

The rest of this paper is organized as follows: Sect. 2 reviews clause reduction heuristics used in CDCL solvers. In Sects. 3 and 4, we provide experimental results, in order to clarify the property of LBD, and to point out some issues in the LBD-based reduction strategy, respectively. Our proposed reduction strategy is described in Sect. 5. Section 6 shows the experimental results and Sect. 7 concludes this paper.

2 Clause Reduction Heuristics

We briefly review the CDCL algorithm [3,6]. We assume that the reader is familiar with notions of propositional satisfiability (propositional variable, literal, clause, unit clause, unit propagation, and so on). The CDCL algorithm repeats the following two operations until a conflict occurs.

1. *Unit propagation*: the unassigned literal in each unit clause is assigned as true to satisfy the clause. This operation repeats until there is no unit clause.
2. *Decision*: when no unit clauses exist, an unassigned literal is selected and a truth value (true or false) is assigned to it.

For each assigned literal l, the *decision level of* l is defined as the number of decision literals on and before assigning l. By $dl(l)$, we denote the decision level of the literal l. When a conflict (falsified clause) occurs in the first step, the algorithm analyzes a cause of the conflict and *learns a clause* from the cause in order to prevent repeating the same conflict. The learnt clause is added to the clause database; then, the algorithm backjumps to the appropriate decision level computed from the clause.

CDCL solvers learn a large number of clauses during the search process of a given SAT instance. Hence, solvers should reduce the clause database periodically in order to suppress memory consumption and sustain propagation speed. In this section, we introduce reduction heuristics based on LBD, which was firstly introduced in Glucose solver. First, we present the evaluation criteria LBD in order to sort learnt clauses according to usefulness.

Definition 1 (Literals Blocks Distances (LBD) [1]). *The LBD of a clause C is defined as $|\{dl(l) \mid l \in C\} \cap \mathbb{N}|$, where $\mathbb{N}$ is the set of all natural numbers including 0; that is, the number of kinds of decision levels of literals in C.*

By $lbd(C)$, we denote the LBD of clause C. When a learnt clause is generated, the LBD of the clause is computed based on the current assignment. Additionally, the LBD is updated when the clause is used in unit propagations and the new LBD is smaller than the old one[1]. Literals with the same decision level have

[1] In Glucose 3.0 or later, the LBD update is executed only for clauses used in unit propagations on and after the first UIP in conflict analysis.

a possibility to be assigned at the same time in the future. Hence, small LBD clauses have a high possibility, which will be used in unit propagations and conflicts. We show the experimental evidence in the next section.

Next, we review the clause reduction strategy used in Glucose. The clause database reduction is executed every $l_{\text{first}} + l_{\text{inc}} \times x$ conflicts[2], where x is the number of reduction calls (initially $x = 0$). On each reduction, clauses are reduced according to the following policy that contains two exceptional conditions:

- Half of learnt clauses are removed in descending order of LBD except the following clauses.
  - keeps clauses whose LBDs ≤ 2 (these are called *glue clauses*).
  - keeps clauses used after the last reduction and updated LBD is less than a certain threshold.

In the following, we refer the above base policy and two exceptional conditions as **BP**, **E1** and **E2**, respectively. This Glucose reduction strategy and its derivatives are used in many solvers. For example, Lingeling dynamically selects either Glucose-based or classical activity-based strategies [4]. If the standard deviation of the LBDs of learnt clauses is too small or too large, then, the activity-based strategy is selected. MapleCOMSPS uses the reduction strategy combining both. This keeps clauses whose LBDs ≤ 6, while others are managed by the activity-based strategy. In addition, clauses with LBDs of 4 to 6, which have not been used for a while, are managed by the activity-based strategy [8]. In Sect. 4, we show some issues in the Glucose reduction strategy.

3 Experimental Evaluation of LBD

LBD is a well-known criterion for identifying learnt clauses that are likely to be used frequently. In this section, we present the experimental evidence of LBD usefulness in a wide variety of instances. From the results, we design our reduction strategy based on LBD. Throughout the paper, we use 755 instances, excluding duplicates, from the application instances used in competitions over the last 3 years,[3] as benchmark instances. All experiments were conducted on a Core i5 (1.4 GHz) with 4 GB memory. We set the timeout for solvers to 5,000 CPU seconds. We used our SAT solver GlueMiniSat 2.2.10. The main difference with Glucose is that GlueMiniSat uses the lightweight in-processing techniques [7]. Detailed results and the source code of GlueMiniSat can be found at http://www.kki.yamanashi.ac.jp/~nabesima/sat2017/.

Figure 1 shows the distributions of LBDs of learnt clauses (left) and used LBDs (right) for each instance. In this experiment, learnt clause reduction was disabled; that is, the solver held every learnt clause. The numbers in the legend represent LBDs. The red line in the left graph will be explained in Sect. 4. Each

[2] In Glucose 3.0 or later, l_{first} and l_{inc} are 2000 and 300 respectively [2].
[3] SAT 2014 competition, SAT-Race 2015 and SAT 2016 competition.

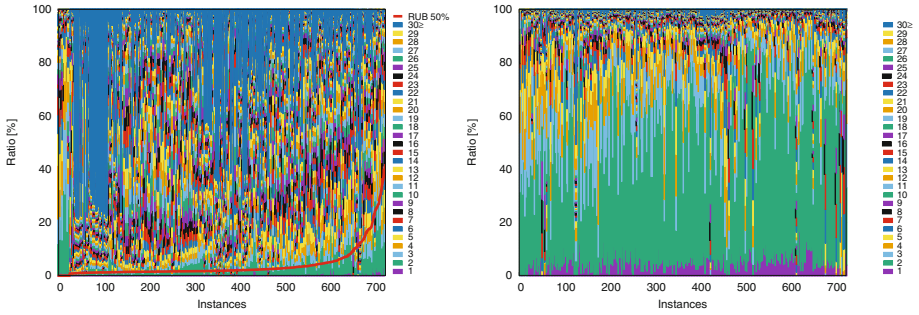

Fig. 1. Distribution of LBDs of learnt clauses (left) and used LBDs (right). Instances are sorted by ascending order of CPU time. (Color figure online)

stacked bar in the left graph represents the ratio distribution of LBDs, for all learnt clauses, after the solver stopped (e.g., if there are 10% learnt clauses whose LBDs are 2, the height of LBD 2 bar is 10%). In the right graph, each bar represents the ratio distribution of clause LBDs, which caused propagations or conflicts (e.g., if 50% of propagations or conflicts are caused by LBD 2 clauses, then the height of the LBD 2 bar is 50%).

The left graph shows that learnt clauses have various LBDs. In easy instances at the left end of the graph, small LBDs are somewhat dominant; however, the other instances have many different LBDs. On the other hand, from the right graph, it is clear that most propagations or conflicts are caused by small LBD clauses. This strongly supports the identification power of the LBD criterion.

4 Issues of Glucose Reduction Strategy

The Glucose reduction schema consists of **BP**, **E1** and **E2**, described in Sect. 2. In this section, we discuss the issues of the schema. We consider the base policy **BP**. Suppose that L_k is the number of learnt clauses after k-th reductions ($k \geq 1$) and r is the residual ratio ($0 \leq r < 1$, 0.5 in Glucose). L_k is defined as follows:

$$L_k = \begin{cases} r l_{\text{first}} & (k = 1) \\ r(L_{k-1} + l_{\text{first}} + (k-1)l_{\text{inc}}) & (k > 1) \end{cases} \tag{1}$$

We consider the difference $d_k = L_k - L_{k-1}$, which can be represented as $d_k = rd_{k-1} + rl_{\text{inc}}$. For this equation, when we add $-\frac{r}{1-r}l_{\text{inc}}$ to both sides, it represents a geometric progression with initial value $rl_{\text{first}} - \frac{r}{1-r}l_{\text{inc}}$ and common ratio r. Hence, we can get the following relationship:

$$L_k - L_{k-1} = (rl_{\text{first}} - \frac{r}{1-r}l_{\text{inc}}))r^{k-1} + \frac{r}{1-r}l_{\text{inc}} \tag{2}$$

The difference between L_k and L_{k-1} represents the number of clauses that can be newly held. The limit of $L_k - L_{k-1}$ as k approaches ∞ is a constant:

$$\lim_{k \to \infty} (L_k - L_{k-1}) = \frac{r}{1-r}l_{\text{inc}} \tag{3}$$

On the other hand, the interval between reductions increases exponentially. This means that the ratio of number of clauses that can be newly held, gradually approaches 0 as k increases. This is the first issue regarding **BP**.

The red line in the left graph of Fig. 1 represents the upper bound of the number of clauses held when following **BP**. In most instances, the number of glue clauses exceeds the bound. Glue clauses are never removed by **E1**. As a result, the solver can not hold new non-glue clauses at all, as long as it follows **BP** and **E1**. Even with a high residual ratio, the increment becomes a constant by (3); therefore, the issue essentially remains. Moreover, by keeping only glue clauses (**E1**) is sometimes insufficient to cover most propagations and conflicts. In the right graph of Fig. 1, the purple and green bars at the bottom represent the ratio of LBD 1 and 2. This indicates that the appropriate upper bound of LBD depends on a given instance. These are the second issue regarding **E1**.

In Glucose, clauses used after the last reduction and with LBD less than, or equal to 30, are not removed (**E2**). The right graph in Fig. 1 shows that this threshold can cover most propagations and conflicts; however, it may be overly retentive. This is the third issue related to **E2**.

In the next section, we propose a new concise reduction strategy to address the above mentioned concerns.

5 Coverage-Based Reduction Strategy

Most propagations and conflicts are caused by small LBD clauses. We propose a reduction strategy to dynamically compute the upper bound of LBD in order to cover most propagations and conflicts.

Let c be the specified coverage ($0 \leq c \leq 1$) and f_k be the number of times that LBD k clauses are used, where we call that a clause is *used* when it causes a unit propagation or a conflict, that is, when the clause becomes a unit or a falsified clause in a unit propagation process. We define the cumulative frequency up to k as $f_k^{\mathrm{cum}} = \sum_{i=1}^{k} f_i$ and the total frequency as $f^{\mathrm{tot}} = f_{|V|}^{\mathrm{cum}}$, where V is the set of variables at a given instance. The LBD threshold $lbd - thld(c)$ is defined as the minimum LBD l such that f_l^{cum} achieves the cover rate c of f^{tot} uses, that is,

$$lbd - thld(c) = l \quad \text{s.t.} \quad (f_{l-1}^{\mathrm{cum}} < cf^{\mathrm{tot}}) \ \wedge \ (cf^{\mathrm{tot}} \leq f_l^{\mathrm{cum}}). \tag{4}$$

This does not guarantee that the rate c of used clauses in the future will be covered by holding clauses with LBD $\leq lbd - thld(c)$, because a discarded clause may be required in order to propagate a clause with LBD $\leq lbd - thld(c)$. Nevertheless, we will present experimental results that our approach can achieve high coverage rate.

Next, we consider the trade-off between coverage and number of kept clauses. A high coverage requires the retention of a large number of clauses. Figure 2 exhibits the holding ratio, which is the number of maintained clauses at the termination of solver divided by the total number of learnt clauses, when we

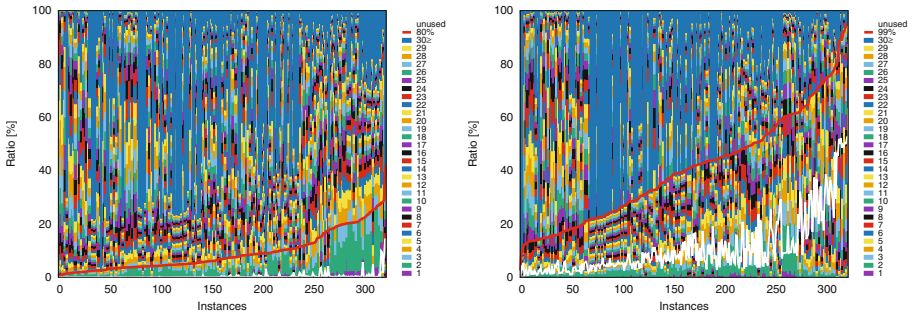

Fig. 2. Holding ratio of coverage of 80% (left) and 99% (right) for unsolved instances. Instances are sorted by ascending order of holding ratio. (Color figure online)

specify the coverage as 80% (left) and 99% (right). The red line represents the holding ratio and the white line denotes the ratio of unused and held clauses[4]. At the right end of the left graph, the red line is 30% and the white line is 10%. This means that a maximum of 30% of learnt clauses are required to cover at least 80% of uses of learnt clauses, and that a maximum of 10% of clauses are unused. The right graph at Fig. 2 shows that it is necessary to keep the majority of clauses in order to cover almost all uses, in the worst case, and that approximately half of them are not used.

In order to suppress the number of held clauses while covering propagations and conflicts as much as possible, we classify learnt clauses into three types: *core*, *support*, and *other* clauses. We make core clauses cover most uses (e.g. 80%), and support clauses cover the remains (e.g. 99%). We give support clauses a short lifetime since their number can be enormous; we give core clauses a longer lifetime, where the *lifetime* n of clause C means that C will be removed when it is unused while n reductions occur. We provide the formal definition of core, support, and other clauses. Let C be a clause, and c^{core} and c^{supp} be the specified coverage of core and support clauses ($c^{\mathrm{core}} \leq c^{\mathrm{supp}}$), respectively.

- C is a *core clause* if $lbd(C) \leq lbd - thld(c^{\mathrm{core}})$.
- C is a *support clause* if $lbd - thld(c^{\mathrm{core}}) < lbd(C) \leq lbd - thld(c^{\mathrm{supp}})$.
- Otherwise, C is an *other clause*.

The coverage-based clause reduction is executed every $l_{\mathrm{first}} + l_{\mathrm{inc}} \times x$ conflicts, same as in the Glucose schema, where x is the number of reduction calls (initially $x = 0$). For each reduction, we compute the core LBD threshold $lbd - thld(c^{\mathrm{core}})$ and the support LBD threshold $lbd - thld(c^{\mathrm{supp}})$ based on the frequency distribution of used LBDs. The lifetime of a core and support clause is the specified value l^{core} and l^{supp} ($l^{\mathrm{core}} \geq l^{\mathrm{supp}}$), respectively. Other clauses are removed at the next reduction (that is, the lifetime is 0). The computational cost of the coverage-based reduction strategy is $O(n)$, where n is the number of learnt

[4] A clause is unused if it does not produce any propagation or conflict, except for the UIP propagation immediately after being learn it.

clauses. Because the computation of the LBD threshold (4) is $O(m)$, where m is the maximum LBD, the removal of clauses exceeding the threshold is $O(n)$, and usually $m \ll n$. The Glucose reduction strategy requires $O(n \log n)$ since it needs to sort learnt clauses by their LBDs. Note that our reduction strategy does not impose the upper bound to the number of clauses (**BP**).

6 Experimental Results

We evaluated the coverage-based and Glucose reduction strategies. In the evaluation, we use the following parameters: $c^{\text{core}} = 0.8$, $c^{\text{supp}} = 0.99$, $l^{\text{core}} = 10$, $l^{\text{supp}} = 1$, $l_{\text{first}} = 2000$ and $l_{\text{inc}} = 300$. The first 4 parameters were determined by preliminary experiments. l_{first} and l_{inc} are the same as in Glucose. We also compared our approach with Glucose 4.0, MapleCOMSPS, and Lingeling. The latter two solvers use the Glucose-style schema as part of the reduction strategy, as described in Sect. 2. These solvers took the first and third place in the main track of the SAT 2016 competition, respectively[5].

Table 1. Solved instances, where "X (Y + Z)" denotes the number of solved instances (X), solved satisfiable instances (Y) and solved unsatisfiable instances (Z), respectively.

Solver	Solved instances
GlueMiniSat 2.2.10 (Glucose schema)	510 (255 + 255)
GlueMiniSat 2.2.10 (Coverage schema)	**524** (259 + 265)
Glucose 4.0	484 (244 + 240)
MapleCOMSPS	519 (**276** + 243)
Lingeling bbc	522 (249 + **273**)
Virtual best solver	597 (305 + 292)

Table 1 shows the number of instances solved by each solver and Fig. 3 is the cactus plot of these results. GlueMiniSat, with the Glucose schema, has better performance than Glucose. The coverage schema can further improve performance for both SAT and UNSAT instances. MapleCOMSPS and Lingeling show the superior results for SAT and UNSAT instances, respectively. GlueMiniSat, with the coverage schema, shows that the well-balanced result and total number of solved instances are comparable with these state of the art solvers.

Table 2 is the comparison of statistics between Glucose and coverage schema. Each value in the first 5 lines denotes the average for commonly solved 494 instances of both strategies. The first two lines in the table show that the Glucose schema reduces more clauses than the coverage schema; hence, the Glucose schema shows higher propagation speed. On the other hand, the coverage schema

[5] We exclude Riss 6, which ranked 2nd in the competition. Because it uses Linux-specific APIs, we could not compile it in our computing environment (Mac OS X).

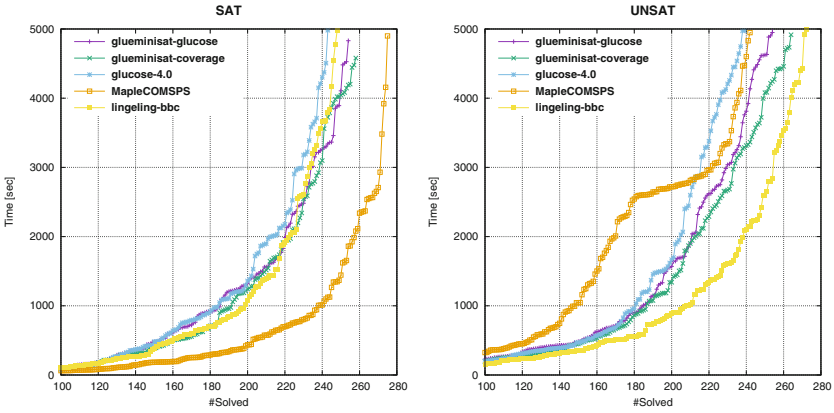

Fig. 3. Time to solve instances

Table 2. Comparison of two reduction strategies in GlueMiniSat.

Solver	Glucose schema	Coverage schema
Removed learnt [%]	76.4	72.9
Propagation speed [literals/sec]	1749748	1716494
CPU time [sec]	812.0	757.5
Conflicts	2519806	2325005
Reduction time [sec]	6.9	2.5
Coverage [%]	73.3	85.0
Precision [%]	67.0	74.1
Recall [%]	33.9	44.1

requires shorter CPU time and less conflicts in order to solve instances. It shows that the coverage schema can hold more useful clauses than the Glucose schema. In the coverage schema, the reduction of learnt clauses is slightly faster since the computational cost is $O(n)$ while the Glucose schema is $O(n \log n)$.

The last three lines in Table 2 are the results of different experiments, in which each solver does not actually remove learnt clauses to calculate coverage, precision, and recall. Each value indicates the average for commonly solved 406 instances. Coverage in Table 2 is the ratio of the number of used clauses that are caused only by maintained clauses to the total number of used clauses that are caused by all clauses. Precision is the ratio of used and held clauses to held clauses; recall is the ratio of used and held clauses to used clauses. In the coverage schema, these values have improved. This indicates that the coverage schema can better identify which clauses will be used.

7 Conclusion

We have shown that LBD can identify which clauses will be used, and proposed a concise and lightweight coverage-based reduction strategy, which provides an appropriate LBD threshold in order to cover most propagations and conflicts. The experimental results show that the coverage schema can effectively hold clauses to be used. Many solvers use LBD as an evaluation criterion for learnt clauses. Our approach can be applicable to such solvers.

References

1. Audemard, G., Simon, L.: Predicting learnt clauses quality in modern SAT solvers. In: Proceedings of IJCAI-2009, pp. 399–404 (2009)
2. Audemard, G., Simon, L.: Glucose 3.1 in the SAT 2014 competition (2014). http://satcompetition.org/edacc/sc14/solver-description-download/118. SAT Competition 2014 Solver Description
3. Bayardo Jr., R.J., Schrag, R.: Using CSP look-back techniques to solve real-world SAT instances. In: Proceedings of the 14th National Conference on Artificial Intelligence (AAAI 1997), pp. 203–208 (1997)
4. Biere, A.: Lingeling and Friends at the SAT Competition 2011 (2011). http://fmv.jku.at/papers/biere-fmv-tr-11-1.pdf. SAT Competition 2011 Solver Description
5. Katebi, H., Sakallah, K.A., Marques-Silva, J.P.: Empirical study of the anatomy of modern sat solvers. In: Sakallah, K.A., Simon, L. (eds.) SAT 2011. LNCS, vol. 6695, pp. 343–356. Springer, Heidelberg (2011). doi:10.1007/978-3-642-21581-0_27
6. Marques-Silva, J.P., Sakallah, K.A.: GRASP: a search algorithm for propositional satisfiability. IEEE Trans. Comput. **48**(5), 506–521 (1999)
7. Nabeshima, H., Iwanuma, K., Inoue, K.: On-the-fly lazy clause simplification based on binary resolvents. In: ICTAI, pp. 987–995. IEEE (2013)
8. Oh, C.: Between SAT and UNSAT: the fundamental difference in CDCL SAT. In: Heule, M., Weaver, S. (eds.) SAT 2015. LNCS, vol. 9340, pp. 307–323. Springer, Cham (2015). doi:10.1007/978-3-319-24318-4_23