Simplifying CDCL Clause Database Reduction

Sima Jamali(✉) and David Mitchell

Simon Fraser University, Burnaby, Canada
sima_jamali@sfu.ca, mitchell@cs.sfu.ca
http://www.cs.sfu.ca

Abstract. CDCL SAT solvers generate many "learned" clauses, so effective clause database reduction strategies are important to performance. Over time reduction strategies have become complex, increasing the difficulty of evaluating particular factors or introducing new refinements. At the same time, it has been unclear if the complexity is necessary. We introduce a simple online clause reduction scheme, which involves no sorting. We instantiate this scheme with simple mechanisms for taking into account clause activity and LBD within the winning solver from the 2018 SAT Solver Competition, obtaining performance comparable to the original. We also present empirical data on the effects of simple measures of clause age, activity and LBD on performance.

Keywords: Clause deletion · CDCL · Clause database reduction

1 Introduction

CDCL SAT solvers generate a very large number of new "learned" clauses, so clause management methods are central to solver performance [2,13]. In particular, most learned clauses must be deleted to keep the clause database of practical size, and the clause database reduction scheme is one of a small number of key heuristic mechanisms in a CDCL solver [3,14]. Typical clause maintenance strategies involve two stores of learned clauses, which we will call Core and Local. Clauses placed in Core are retained for the entire run. The size of Core is limited by being selective about which clauses are added. The large majority of learned clauses are placed in Local. The size of Local is limited by periodic deletion of "low quality" clauses, which are deemed unlikely to be of high future utility. The quality measure is typically a combination of size, age, literal block distance (LBD) and some measure of usage or activity [3,8–11,14,15].

Major changes to the general scheme are rare, but over time many refinements have combined to make the overall mechanism in the best recent solvers quite complex. Most details have intuitive explanations, and were chosen based on empirical performance. At the same time, the complexity seems perhaps a bit much relative to our understanding of "clause quality". This complexity makes

M. Janota and I. Lynce (Eds.): SAT 2019, LNCS 11628, pp. 183–192, 2019.
https://doi.org/10.1007/978-3-030-24258-9_12

it hard to evaluate the contributions of individual elements, and is an obstacle to adding new features or refined quality measures.

There are two main aspects to a clause deletion strategy. The first is a method to categorize clauses as likely to be useful (high quality), or not (low quality). The second is implementation of an algorithmic method to remove low quality clauses efficiently. In an idealized scheme, we might have a clause quality measure Q, and keep the clauses in a heap so that the lowest quality clause(s) can be removed when the clause database is deemed too large. Conventional wisdom is that using a heap would be too inefficient. It also seems unlikely that spending time to obtain the very worst clause is necessary. Thus, fast heuristics are desired. One scheme, which we call Delete-Half, is to periodically sort the clauses of Local and delete the half with lowest quality. This scheme has been very widely used for many years, but there are many other possible schemes. While some solvers use other schemes (e.g., [4,16]), we think much more investigation is justified. Regarding clause quality, we expect a very good clause quality measure to involve a combination of many factors. The dominant current quality measure uses VSIDS-like clause "activities". Unfortunately, the way activities are computed and maintained in practice makes it hard to combine activity with other measures of quality in a simple and meaningful way.

The goal of this work is to identify simple methods that might largely account for effectiveness of the best current schemes. We make the following contributions.

1. We introduce a new "online" clause deletion scheme which is simple to implement and maintains the size of Local at a desired value. It does not use sorting and in many natural instantiations takes constant time per conflict. The scheme is presented in Sect. 2.
2. We show that a simple instantiation of this scheme performs comparably to the state of the art. In particular, we implemented the scheme within MapleLCMDistChronoBT, the first-place solver from the 2018 SAT Competition [1,12]. This instantiation takes into account clause usage and LBD using very simple mechanisms. The resulting solver (Online-RU-T2Flag) and its performance are described in Sect. 4.
3. To aid in understanding the degree to which the particular methods play a role in solver performance, we present data from a number of experiments measuring performance or other properties. These appear throughout remaining sections.

Our performance evaluations are carried out using the 400 formulas from the main track of the 2018 SAT Solver Competition, with a 5000 second cut off. Our baseline solver for performance evaluation is MapleLCMDistChronoBT, winner of the competition and all other solvers are modified versions of it. The computations were performed on the Cedar compute cluster [6] on 32-core, 128 GB nodes with Intel "Broadwell" CPUs running at 2.1 Ghz.

1.1 MapleLCMDistChronoBT Clause Database Management

Many top-performing solvers in recent SAT Solver Competitions have been variants or derivatives of MapleSAT [11]. For simplicity, we focus on the first-place solver from the 2018 competition, MapleLCMDistChronoBT [12], which uses the deletion scheme introduced in COMiniSatPS [14,15].

This scheme has three clause databases, called Core, Tier2 and Local. The decision of where to store a newly learned clause in is based on its LBD: Core if LBD ≤ 3, Tier2 if $4 \leq$ LBD ≤ 6 and Local if $6 <$ LBD. If after 100,000 conflicts there are not enough clauses in Core, the core threshold is changed from 3 to 5. A clause may be moved from one DB to another based on LBD or usage. The LBD of each clause is recomputed whenever it is used in conflict analysis or the clause simplifying procedure [3]. If the LBD of a clause is sufficiently reduced, it is moved from Local to Tier2 or Core, or from Tier2 to Core. Every 10,000 conflicts, every clause in Tier2 that has not been used during the last 30,000 conflicts is moved to Local. Every 15,000 conflicts, all the clauses in Local are sorted by their activity and MapleLCMDistChronoBT deletes half of the clauses with lower activities. Clauses that are a reason for a current assignment and clauses with recent improvement in LBD are saved from deletion [3,14].

2 Online Clause Deletion

Our online clause deletion scheme is as follows. The clauses of Local are maintained in a circular list L with an index variable i that traverses the list in one direction. The index identifies the current "deletion candidate" L_i. We have a clause quality measure Q, and some threshold quality value q. When a new learned clause C needs to be stored in Local, we select a "low quality" clause in the list to be replaced with C by sequential search. As long as $Q(L_i) \geq q$, we increment i ("saving" clause L_i for one more "round"); The first time $Q(L_i) < q$, we replace L_i with C (deleting the "old" L_i). The clause quality measure threshold must be chosen so that there are always sufficiently many "low-quality" clauses in the list. There are algorithmic methods to ensure this (for example, using a feedback control mechanism) but it is not hard to obtain good practical performance without them.

Relating Delete-Half and Online Deletion. Consider a Delete-Half scheme with a sort-and-reduce phase every k conflicts. Roughly speaking (ignoring some details for simplicity) each clause is inspected every k conflicts, deleted if its quality is below the median of the current clauses in Local. If we instantiate our online scheme with $S = 2k$, and keep q sufficiently close to the median, we expect each clause to be inspected every k conflicts and deleted if its quality is below the median of the current clauses in Local. In this sense, the two schemes can be made quite close: we trade off sorting for dynamically estimating the median. In doing so, we get a clause database of uniform size, rather than one that significantly grows and shrinks.

Age-Based Deletion. A trivially implemented version of our scheme assumes $Q(C) < q$ for every clause C. This results in a pure age-based scheme: Each new learned clause replaces the oldest clause in Local. This very low-cost scheme works surprisingly well. Figure 1 shows a "cactus-plot" comparison of default MapleLCMDistChronoBT with 3 variants using online deletion. (The Local size limit is set to 80,000 clauses in all solvers using online deletion reported here.)

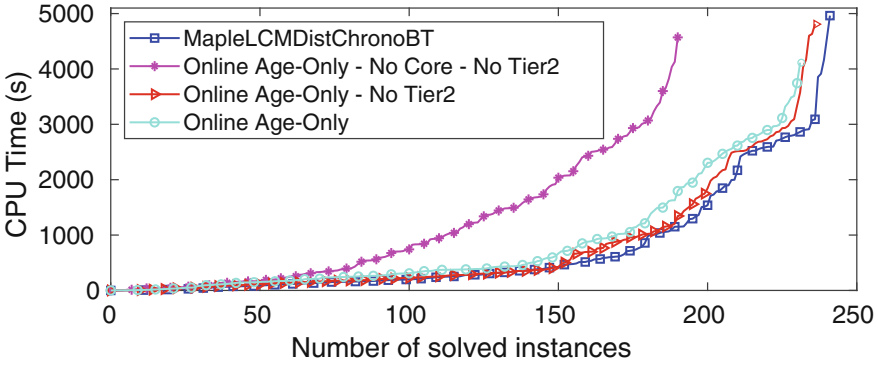

Fig. 1. Simple online deletion performance.

Online Age-Only - No Core, No Tier2. This has no permanent store at all, just pure age-based deletion of all learned clauses.

Online Age-Only - No Tier2. This version keeps clauses with LBD $\leq$3 or Size $\leq$4 permanently in Core, and uses pure age-based deletion from Local.

Online Age-Only. In this version we use Core and Tier2 just as in MapleLCMDistChronoBT, but use age-based online deletion from Local. If a clause is moved from Tier2 to Local, it replaces the oldest clause in Local.

Figure 1 shows that Core and Tier2 are important to the performance of MapleLCMDistChronoBT. It also shows that in the presence of Core and Tier2 a simple pure age-based deletion scheme for Local gives quite good performance.

We make two observations regarding this second point. First, in online deletion with Local of size S, if the probability of saving a clause is as most 0.5 (see Fig. 4), then every learned clause is kept for at least $S/2$ conflicts, giving it substantial time to be used. Delete-Half schemes generally do not ensure this. Second, age is highly correlated with usage rate, and can account for a large fraction of decisions that would be made based on clause activities. This is illustrated by Fig. 2, which shows the average usage rates of clauses that have been in Local for at least 10K conflicts, at different ages. The usage rate of most clauses drops very quickly.

3 Clause Usage

MiniSAT and many of its successors, including MapleLCMDistChronoBT, use clause "activity" scores in their clause deletion schemes [8,11,15]. If a clause is used in conflict analysis, its activity is "bumped", meaning it's activity score is increased by a reward value. The reward is initialized to 1 and divided by 0.999 (the decay factor) at each conflict, to similate decay of activities. To prevent activity overflow, when the activity of any clause reaches 1e20, all activity values and the reward value are divided by 1e−20 [5,8].

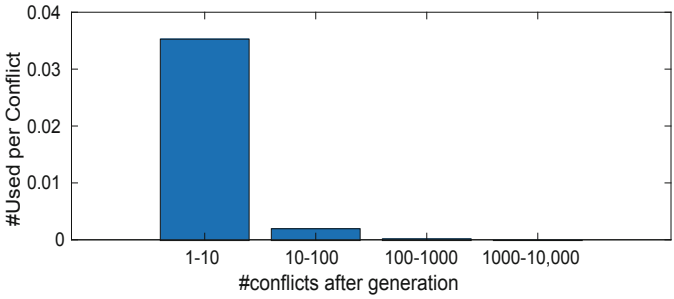

Fig. 2. Rate of use of clauses in Local at different ages.

This scheme, with many variations, has been widely used, but it also has inconvenient aspects as discussed above. We anticipated that, in the presence of Core, much simpler usage measures might be effective. Here we report two that we have considered. Both are extremely simple to implement. We follow their descriptions with reports of three experiments that may shed light on the performance of the RU measures.

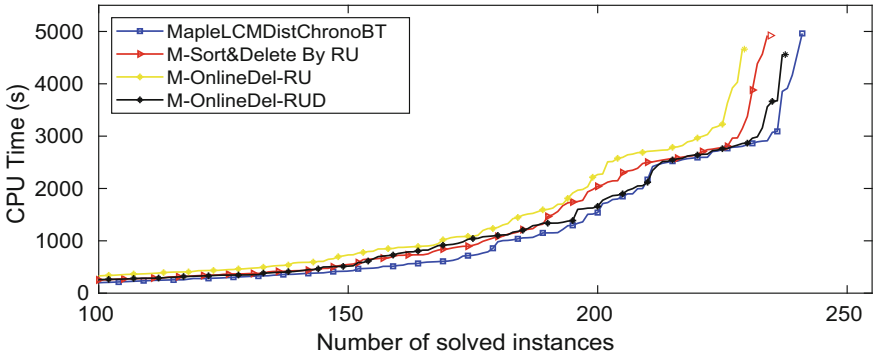

Fig. 3. Online deletion with recent usage

M-OnlineDel-RU. In this version, the measure Q of quality (or activity) is just the number of times the clause was used in conflict analysis during the last "round". That is, we count uses and reset the count to zero if the clause becomes the candidate for deletion but is saved. We denote this measure RU, for Recently-Used. If the threshold value is q (denoted RU $= q$), a clause will be saved if it was used q or more times in the last round.

M-OnlineDel-RUD. This is similar to M-OnlineDel-RU, but instead of resetting RU to 0 when a clause is saved, we decay it by dividing it by a constant. We call this measure RUD.

Figure 3 shows the performance of M-OnlineDel-RU with threshold RU $= 2$ and M-OnlineDel-RUD with RU $= 2$ and Decay constant 4. Both versions perform quite well, the decay version being almost as good as MapleLCMDistChronoBT. This suggests that online deletion using simple measures might compete effectively with Delete-Half using traditional activities.

To understand the effectiveness of RU versus traditional activities, we created a solver **M-Sort&Delete By RU** that is identical to MapleLCMDistChronoBT but does sorting and deletion from Local based on RU instead of activity. Figure 3 shows the performance is slightly inferior to MapleLCMDistChronoBT on our benchmark, lying between the performance of the two versions with online deletion. This suggests that we pay no penalty for using online deletion instead of the Delete-Half scheme, and confirms that in the presence of Core and Tier2 a simple usage measure can be almost as useful as traditional clause activities.

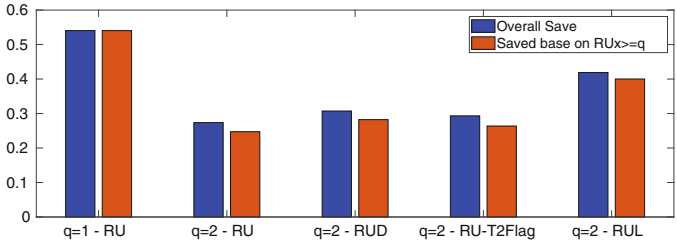

Fig. 4. Fraction of saved clauses in different online deletion schemes (Color figure online)

Fraction Saved by RU. Here we examine the fraction of clauses in Local that become candidates for deletion but are saved based on the RU measure. Figure 4 shows this value for several variations. In each pair of bars, the right bar (orange) shows the fraction of clauses with RU $\geq q$; the left bar (blue) shows the fraction of clauses saved based on either RU or because of being "locked" [8].

With $q = 1$, the probability of deletion is less than 1/2, and the performance of the solver is poor. In contrast, with $q = 2$, about three quarters of clauses are deleted, and the performance is quite good as shown in Fig. 3. In the remainder of the paper, all solvers using RU have q is set to 2.

Table 1. Commonality among high-activity clauses and recently-used clauses.

Formula	Local size	RU ≥ 1 (%)	RU ≥ 2 (%)	RUD ≥ 2 (%)	RUL ≥ 2 (%)
201	28470	12150 (100)	2162 (100)	2265 (97)	3591 (100)
CNP-5–200	28699	13177 (98)	4122 (100)	4923 (87)	2915 (99)
Karatsuba	25251	11091 (86)	1730 (90)	2111 (80)	5571 (89)
T62.2.0.cnf	7097	2474 (100)	521 (100)	618 (86)	1574 (100)
ae_rphp	30535	12586 (87)	6575 (94)	8004 (78)	7278 (94)
apn-sbox6	29422	15459 (87)	6423 (92)	7129 (85)	6282 (90)
cms-scheel	21828	8602 (100)	2454 (100)	2698 (92)	3752 (100)
courses	13869	3241 (100)	854 (100)	1092 (83)	1610 (100)
cz-alt-3–7	26577	11276 (99)	2346 (100)	2639 (93)	7247 (99)
dist9.c	26274	15150 (84)	4182 (92)	4614 (87)	6651 (90)
Average	**23802**	**10521 (93)**	**3137 (97)**	**3609 (87)**	**3395 (96)**

Clauses Saved by RU and Activity. We examined the clauses in Local just before the 10^{th} clause deletion in MapleLCMDistChronoBT, and measured their RU and activity values to see what fraction of clauses would be saved by our RU-based schemes. Table 1 shows the results for one formula from each of 10 families. The first column is the number of clauses in Local just before deletion. Other columns show the number of clauses that would be saved due to RU $\geq q$, and the fraction (in percent) of these clauses that have high enough activity to be saved by Delete-Half. On average this fraction is between 87 and 97%, suggesting that simple RU counters can account for a significant fraction of decisions based on activities.

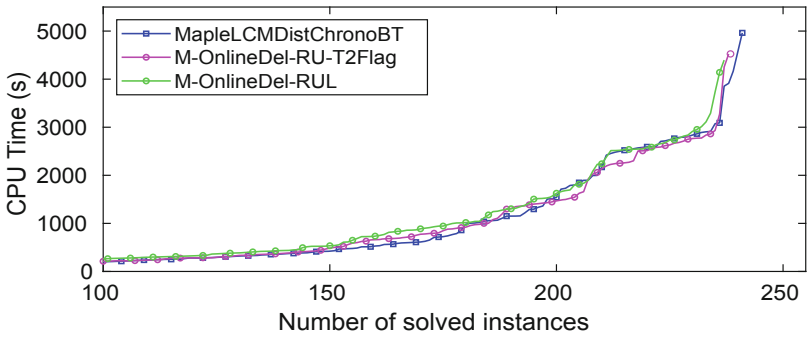

Fig. 5. Online deletion with usage and LBD

4 Clause LBD and Tier2

LBD is used in MapleLCMDistChronoBT for initial placement of a learned clause, and to move clauses between stores if the LBD changes. Here we report two simple methods to take into account LBD changes in a solver with online deletion and no Tier2. Figure 5 show the resulting performance.

M-OnlineDel-RU-T2Flag. Here we replace Tier2 with a rough simulation, by adding a "Tier 2 flag" to clauses in Local. We set the flag true if MapleLCMDistChronoBT would move it from Local to Tier2, and false for the reverse direction. Clauses with this flag true are always saved. This is not an accurate Tier2 simulation, because the size of the clause DB does not change appropriately. Nonetheless, the resulting performance is very close to the original solver.

M-OnlineDel-RUL. Here we take LBD into account by modifying the usage scoring. Instead of incrementing RU by 1 each time a clause is used, we increment by c/LBD, for a constant c. We call this RUL, for RU with LBD. The RUL values are re-set to zero when a clause is saved. The curve in Fig. 5 is the performance with $c = 20$.

Table 2. Performance on satisfiable *vs* unsatisfiable formulas.

Solver	# Solved	SAT	UNSAT
MapleLCMDistChronoBT	**241**	138	**103**
M-OnlineDel-RU	230	132	98
M-OnlineDel-RUL	237	140	97
M-OnlineDel-RUD	238	**141**	97
M-OnlineDel-RU-T2Flag	238	139	99

5 Discussion

We introduced a new, simple online clause deletion scheme, and reported the performance of instantiations of the scheme using clause age, LBD and very simple measures of usage. An implementation of the online scheme in MapleLCMDistChronoBT, the winning solver from the main track of the 2018 SAT Solver Competition, has performance almost as good as the original.

Online deletion requires less computation time than the Delete-Half scheme. However, the fraction of run time consumed by deletion in MapleLCMDistChronoBT is small, so this is not a major performance factor.

The online deletion schemes in this paper use age or age modified by a fixed quality threshold. A dynamic threshold may be more desirable, in which case we may use a feedback control scheme to ensure the threshold is such that the fraction of saved clauses is suitable (*cf* Fig. 4).

We continue to investigate more refined versions of our scheme, in particular with regard to clause quality measures and clause database size. Table 2 shows that our modified solvers are biased toward Satisfiable instances, and we will work on shifting this bias.

Acknowledgement. This research was supported and enabled in part by the Natural Sciences and Engineering Research Council of Canada (NSERC), WestGrid (www.westgrid.ca) and Compute Canada (www.computecanada.ca) [7].

References

1. The international SAT competitions web page. http://www.satcompetition.org
2. Ansótegui, C., Giráldez-Cru, J., Levy, J., Simon, L.: Using community structure to detect relevant learnt clauses. In: Heule, M., Weaver, S. (eds.) SAT 2015. LNCS, vol. 9340, pp. 238–254. Springer, Cham (2015). https://doi.org/10.1007/978-3-319-24318-4_18
3. Audemard, G., Simon, L.: Predicting learnt clauses quality in modern SAT solvers. In: Proceedings of the 21st International Jiont Conference on Artificial Intelligence, IJCAI 2009, pp. 399–404. Morgan Kaufmann Publishers Inc., San Francisco (2009)
4. Biere, A.: Pre, icosat@sc'09. solver description for SAT competition 2009. SAT Competitive Event Booklet (2009)
5. Biere, A., Fröhlich, A.: Evaluating CDCL Variable scoring schemes. In: Heule, M., Weaver, S. (eds.) SAT 2015. LNCS, vol. 9340, pp. 405–422. Springer, Cham (2015). https://doi.org/10.1007/978-3-319-24318-4_29
6. Cedar, A Compute Canada Cluster. https://docs.computecanada.ca/wiki/Cedar
7. Compute Canada: Advanced Research Computing (ARC) Systems. https://www.computecanada.ca/
8. Eén, N., Sörensson, N.: An extensible SAT-solver. In: Giunchiglia, E., Tacchella, A. (eds.) SAT 2003. LNCS, vol. 2919, pp. 502–518. Springer, Heidelberg (2004). https://doi.org/10.1007/978-3-540-24605-3_37
9. Goldberg, E., Novikov, Y.: BerkMin: A fast and robust SAT-solver. In: Lauwereins, R., Madsen, J. (eds.) Design, Automation, and Test in Europe, pp. 465–478. Springer, Dordrecht (2008). https://doi.org/10.1007/978-1-4020-6488-3_34
10. Jamali, S., Mitchell, D.: Centrality-based improvements to CDCL heuristics. In: Beyersdorff, O., Wintersteiger, C.M. (eds.) SAT 2018. LNCS, vol. 10929, pp. 122–131. Springer, Cham (2018). https://doi.org/10.1007/978-3-319-94144-8_8
11. Liang, J.H., Ganesh, V., Poupart, P., Czarnecki, K.: Learning rate based branching heuristic for SAT solvers. In: Creignou, N., Le Berre, D. (eds.) SAT 2016. LNCS, vol. 9710, pp. 123–140. Springer, Cham (2016). https://doi.org/10.1007/978-3-319-40970-2_9
12. Nadel, A., Ryvchin, V.: Chronological backtracking. In: Beyersdorff, O., Wintersteiger, C.M. (eds.) SAT 2018. LNCS, vol. 10929, pp. 111–121. Springer, Cham (2018). https://doi.org/10.1007/978-3-319-94144-8_7
13. Newsham, Z., Ganesh, V., Fischmeister, S., Audemard, G., Simon, L.: Impact of community structure on SAT solver performance. In: Sinz, C., Egly, U. (eds.) SAT 2014. LNCS, vol. 8561, pp. 252–268. Springer, Cham (2014). https://doi.org/10.1007/978-3-319-09284-3_20
14. 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). https://doi.org/10.1007/978-3-319-24318-4_23

15. Oh, C.: Improving SAT solvers by exploiting empirical characteristics of CDCL. Ph.D. thesis, New York University (2016)
16. Soos, M., Nohl, K., Castelluccia, C.: Extending SAT solvers to cryptographic problems. In: Kullmann, O. (ed.) SAT 2009. LNCS, vol. 5584, pp. 244–257. Springer, Heidelberg (2009). https://doi.org/10.1007/978-3-642-02777-2_24