Improving Implementation of SAT Competitions 2017–2019 Winners

Stepan Kochemazov^(✉)

Matrosov Institute for System Dynamics and Control Theory SB RAS,
Irkutsk, Russia
veinamond@gmail.com

Abstract. The results of annual SAT competitions are often viewed as the milestones showcasing the progress in SAT solvers. However, their competitive nature leads to the situation when the majority of this year's solvers are based on previous year's winner. And since the main focus is always on novelty, it means that there are times when some implementation details have a potential for improvement, but they are just inherited from solver to solver for several years in a row. In this study we propose small modifications of implementations of existing heuristics in several related SAT solvers. These modifications mostly consist in employing a deterministic strategy for switching between branching heuristics and in augmentations of the treatment of *Tier2* and *Core* clauses. In our experiments we show that the proposed changes have a positive effect on solvers' performance both individually and in combination with each other.

Keywords: SAT · SAT solvers · Heuristics

1 Introduction

The Conflict-Driven Clause Learning (CDCL) solvers [13] form the core of the algorithms for solving the Boolean satisfiability problem (SAT) [4]. Every year the community proposes new heuristics aimed at improving their performance. To test them in close to real-world conditions, the SAT competitions are held. They evaluate the prospective solvers' performance in the same computational environment over the sets of test instances gathered from various areas where the SAT solvers are applied.

To show that a new heuristic for SAT solving contributes to state of the art, it is usually implemented on top of one of the well-known SAT-solvers such as MiniSAT [5], glucose [1], CryptoMiniSat [20], Cadical [2] or, more often, the most recent SAT competition(s) winner (e.g. [8, 12, 17]). The problem with the

The research was funded by Russian Science Foundation (project No. 16-11-10046) with an additional support from Council for Grants of the President of the Russian Federation (stipend no. SP-2017.2019.5).

© Springer Nature Switzerland AG 2020

L. Pulina and M. Seidl (Eds.): SAT 2020, LNCS 12178, pp. 139–148, 2020.

https://doi.org/10.1007/978-3-030-51825-7_11

latter is that the solver with a new heuristic must be compared with the original solver without modifications to show that the increase in performance is not due to fixing some aspects of original solver’s implementation. This is not a bad thing as it is, but it can lead to solvers accumulating undesired traits. One of the examples of the latter is the non-deterministic switching between branching heuristics, first introduced in **MapleCOMSPS** [9] which won in the main track of SAT Competition 2016. This behavioral trait was inherited by the winners of SAT Competitions 2017 and 2018, and many participants of the SAT Race 2019.

In the present paper we propose several changes to the common implementation aspects of the solvers that won at the main tracks of SAT Competitions 2017 to 2019. The main contributions are as follows:

1. We analyze possible variants of deterministic switching between branching heuristics and show that some of them yield consistently better results compared to the non-deterministic switching at 2500 s.
2. We adjust the treatment of the so-called *Tier2* clauses and show that accumulating them in a slightly different manner results in better overall performance.
3. We show that contrary to the intuition provided in [18], it is better to sometimes purge some learnts from the *Core* tier in order to increase the propagation speed and the effectiveness of a solver on hard instances.

We use the **MapleLCMDistChronoBT** solver as the main object for analysis and experimentation. To evaluate improvements and modifications we use the set of SAT instances from the SAT Race 2019. After having figured out the modifications that serve our goals best, we implement them in the winners of SAT Competition 2017 and SAT Race 2019. Then we test the resulting six SAT solvers on a wide range of benchmarks.

2 Background

The presentation of the details on implementation of heuristics in SAT solvers implies that the reader is familiar with the architecture of CDCL SAT-solvers in general and with key advancements in the area during the recent years.

The SAT competitions are annual competitive events aimed at the development of SAT solving algorithms. In the course of the main track of the competition, the solvers are launched on each test instance with the time limit of 5000 s. The performance is evaluated using two criteria: Solution-Count Ranking (SCR) – the number of test instances successfully solved by an algorithm within the time limit, and Penalized Average Runtime (PAR-2) computed as the sum of solver runtimes on solved instances plus $2 \times$ the time limit for unsolved ones, divided by the total number of tests.

In the paper we study the solvers **MapleLCMDist**, **MapleLCMDistChronoBT** and **MapleLCMDistChronoBT-DL-v3**, which won at SAT Competitions 2017 to 2019. All three solvers are based on the **MapleCOMSPS** solver that won at SAT

Competition 2016 [8]. The latter uses the foundation of COMiniSatPS [18], which combined the better traits of the well-known MiniSAT [5] and Glucose [1] solvers.

One of the main novelties of COMiniSatPS was the special treatment of learnt clauses depending on their *literal block distance* value (lbd). The notion of lbd was first proposed in [1] and is equal to the number of distinct decision levels of the literals in a learnt clause. COMiniSatPS splits all learnt clauses into three tiers: *Core*, *Tier2* and *Local* and handles them differently. The second major novel feature of COMiniSatPS was to use two sets of activity values for the branching heuristic. In particular, the solver employed two sets of scores maintained via Variable State Independent Decaying Sum (VSIDS) [16]. They used different VSIDS decay values, but what’s more important, their use was tied to periodic switching between restart strategies. For years after the introduction of Minisat [5], the SAT solvers mainly relied on *luby* restarts [11], but after the triumphal appearance of Glucose [1], the CDCL solvers mostly switched to the much faster *glucose*-restarts. COMiniSatPS combined both, and used one set of VSIDS scores with luby restarts and another set with glucose restarts.

In 2016 the *Learning Rate Branching* (LRB) heuristic [8] was proposed as an alternative to VSIDS. It was implemented in MapleCOMSPS as well as in several other solvers [9]. MapleCOMSPS is based on COMiniSatPS and uses LRB branching together with luby restarts and VSIDS-branching with glucose restarts. Unfortunately, the solver did not inherit the deterministic switching strategy employed by COMiniSatPS to combine different phases of solving. Instead, it relies on LRB + luby restarts for the first 2500s of the search and VSIDS + glucose restarts for its remainder, which results in a non-deterministic behaviour. Having realized their oversight, the MapleCOMSPS’s authors submitted to SAT competitions 2017–2019 only the deterministic variants of their solvers.

The winner of the main track of the SAT Competition 2017, MapleLCMDist introduced Learnt Clause Minimization [12] into MapleCOMSPS. It is an expensive inprocessing method [6] periodically applied to learnt clauses from *Tier2* and *Core* with the goal to remove irrelevant literals from them. The main track winner of SAT Competition 2018 called MapleLCMDistChronoBT had augmented MapleLCMDist with chronological backtracking [17], which was later studied in [15]. Finally, the winner of SAT Race 2019 was MapleLCMDistChronoBT-DL-v3, which used the duplicate learnts (DL) heuristic [7], that tracks duplicate learnts and sometimes adds them into *Tier2* or *Core*.

3 Improving Implementation of MapleLCMDistChronoBT

In this section we describe the three-phase experiment aimed at improving the implementation of MapleLCMDistChronoBT. The main question we strive to answer is if it is possible to retain its good overall performance but make the solver deterministic. In the first phase of experiments we implement several deterministic strategies for switching between branching heuristics and evaluate their performance. In the second phase we use the best-behaved switching strategy from the first phase to check whether it is possible to forego one of the

branching heuristics and let the solver work with a single one. In the third phase we propose and implement a small change to handling *Tier2* and *Core* learnt clauses and see how it affects the general performance of a solver.

In all experiments we used a single node of the “Academician V.M. Matrosov” computing cluster of Irkutsk Supercomputer Center [14]. It is equipped with two 18-core Intel Xeon E5-2695 CPUs and 128 GB DDR4 RAM. The solvers were launched in 36 simultaneous threads.

3.1 On Switching Between Branching Heuristics

First we combed through the available deterministic implementations in the solvers from recent SAT competitions and found several variants for switching between branching heuristics that are summed up in Table 1. All of them assume that VSIDS is used with glucose restarts and LRB with luby restarts, thus we refer to them as to VSIDS and LRB phases, respectively. The `fcm` column corresponds to the implementation in `COMiniSatPS`. The variants `f1` and `f2` denote the implementations first appeared in `MapleCOMSPS_LRB` [9] and `MapleCOMSPS_LRB_VSIDS_2` [10], respectively. The `v3` variant employs the scheme used in the `MapleLCMDistChronoBT-DL-v3` solver [7], which uses propagations to measure the phases’ sizes.

Table 1. The considered schemes for switching between branching heuristics

	<code>fcm</code>	<code>f1</code>	<code>f2</code>	<code>v3</code>
Initial <code>phase_allotment</code>	100 conflicts	100 conflicts	10000 conflicts	30 M props
VSIDS multiplier	2	1	1	1
<code>phase_allotment</code> multiplier	1.1/1	1.1/1	2/1	1.1/1.1

All variants in Table 1 use `phase_allotment` variable to store the number of conflicts (or propagations) allocated for the next phase. They all first launch the LRB phase and then the VSIDS phase. Some of them allocate more resources to the VSIDS phase, in that case the value in the VSIDS multiplier row of Table 1 is different from 1. The value of `phase_allotment` is multiplied by a constant after each LRB/VSIDS phase and the cycle repeats anew. Actually, all strategies but `v3` only increase the value of `phase_allotment` at the end of the VSIDS phase.

We implemented the strategies from Table 1 in `MapleLCMDistChronoBT`. The performance of the resulting 4 solvers on the benchmarks from SAT Race 2019 is presented in form of a cactus plot in Fig. 1 with the corresponding detailed statistics in Table 1. They are accompanied by the original solver and 5 other solver variants, which will be described in detail below. Two of them (`-LRB` and `-VSIDS`) employ only a single branching heuristic and the other three use different combinations of augmented handling of *Tier2* clauses (`-t`) and the procedures for reducing *Core* database (`-rc`).

Fig. 1. Cactus plots depicting the performance of `MapleLCMDistChronoBT` with altered implementations of switching strategies between branching heuristics and of handling learnt clauses over the SAT Race 2019 benchmarks.

Table 2. The detailed statistics on `MapleLCMDistChronoBT` with different variants of switching between branching heuristics and heuristics for handling *Tier2* and *Core* learnts on SAT instances from SAT Race 2019

Solver	SCR	SAT	UNSAT	PAR-2
<code>MapleLCMDistChronoBT</code>	236	138	98	4799
<code>MapleLCMDistChronoBT-f1</code>	231	135	96	4882
<code>MapleLCMDistChronoBT-f2</code>	235	140	95	4716
<code>MapleLCMDistChronoBT-v3</code>	232	137	95	4787
<code>MapleLCMDistChronoBT-fcm</code>	228	133	95	4985
<code>MapleLCMDistChronoBT-LRB-f2</code>	223	134	89	4996
<code>MapleLCMDistChronoBT-VSIDS-f2</code>	203	115	88	5589
<code>MapleLCMDistChronoBT-f2-t</code>	236	139	97	4720
<code>MapleLCMDistChronoBT-f2-rc</code>	240	143	97	4631
<code>MapleLCMDistChronoBT-f2-t-rc</code>	242	143	99	4556

As it can be seen from both Table 2 and Fig. 1, the best deterministic variant in the considered conditions is `f2`. Its distinctive feature is the `phase_allotment` multiplier equal to 2. Thus, the solver alternates between long phases of rapidly increasing size in which it relies on a single heuristic. The similar behaviour is achieved by the `v3` scheme, where a large number of propagations is allocated to phases. Compared to `f2`, `v3` starts from longer phases, that grow in size significantly slower.

3.2 On the Importance of Separate Branching Strategies

It was first noted in [18] and later reaffirmed in [3] that it is beneficial to maintain different variable activity values for use with different restart strategies. Also, in [18] it was pointed out that a strong correlation exists between the effectiveness of glucose restarts for proving unsatisfiability and that of Luby restarts for finding satisfying assignments. Based on the success of the solvers that employ Luby restarts with LRB and glucose restarts with VSIDS it is easy to make a conclusion that LRB shows better performance on SAT and VSIDS on UNSAT instances. To evaluate at a glance the highlighted issues we implemented two versions of the `MapleLCMDistChronoBT-f2`: one that uses only a single set of LRB activity values, and one that relies only on VSIDS. They both still switch between luby restarts and glucose restarts deterministically after the `f2` fashion. We denote them as `MapleLCMDistChronoBT-LRB-f2` and `MapleLCMDistChronoBT-VSIDS-f2`, respectively.

The results of these solvers are included in Fig. 1 and Table 2. They point at the following two conclusions. First, that indeed two separate sets of activities for branching during different restart strategies work better than a single set. Second, that the relation between LRB, luby restarts and satisfiable instances, and VSIDS, glucose restarts and unsatisfiable instances is more complex than it might seem and warrants further investigation that goes out of the scope of the present paper.

3.3 Improving the Handling of *Core* and *Tier2* Learnts

Recall, that in [18, 19], it was proposed to split learnt clauses into three tiers. *Core* tier accumulates and indefinitely holds the clauses with `lbd` not exceeding the value of the `core_lbd_cut` parameter, which is equal to 3 or 5 depending on a SAT instance. *Tier2* stores for a prolonged time period the clauses with slightly higher `lbd` than necessary to be included into *Core* (with `core_lbd_cut` $< \text{lbd} \leq 6$). The remaining clauses are put into the frequently purged *Local* tier.

We investigated how many learnts were pushed into these tiers by the considered solver over a wide range of benchmarks. It turned out that for about 80% instances from SAT Race 2019, `MapleLCMDistChronoBT-f2` accumulates more than 50 000 core learnts. The average sizes of *Tier2* also vary quite significantly since the distribution of `lbd` of conflict clauses is specific to each particular SAT instance. E.g. the capacity of *Tier2* can be as low as 200 learnts on average and as high as 12 000 learnts. In [19] it was specifically noted that the number of learnt clauses in *Tier2* may vary depending on the restart strategy and other factors and that it is perfectly normal. However, this observation did not take into account the Learnt Clause Minimization (LCM) [12] which was proposed in 2017 and is applied to *Core* and *Tier2* in `MapleLCMDistChronoBT`. Thus the instances with higher average `lbd` of learnt clauses do not benefit as much from LCM compared to the others due to the fact that such clauses often do not survive in *Tier2* long enough to be minimized.

These observations lead us to two small modifications in handling the *Tier2* and *Core* learnt clauses. The first one consists in periodically reducing the *Core* database. For this purpose as soon as the *Core* size exceeds a pre-specified limit (in our implementation it is equal to 50000 and is multiplied by 1.1 each time the procedure is invoked), we sort *Core* learnts in the ascending order based on their lbd and the size for equal lbd. Then we move all the clauses from the second half (with larger lbd and size), that did not participate in any of the most recent 100000 conflicts into *Tier2*.

The second improvement is to reorganize how the *Tier2* is purged. In the original implementation of **MapleLCMDistChronoBT** the *Tier2* is reduced every 10000 conflicts and as a result all clauses that have not participated in the most recent 30000 conflicts are moved to *Local*. We propose first to accumulate *Tier2* learnts until a pre-specified size limit (7000), and second during the purge to preserve only half of the clauses that participated in the most recent conflicts.

We implemented proposed modifications in **MapleLCMDistChronoBT-f2**. The inclusion of the procedure for reducing *Core* is labelled as `-rc` and the modification to handling of *Tier2* learnts is labelled as `-t`. The results of the corresponding variants are shown at Fig. 1 and in the lower part of Table 2. It is clear that the combination of proposed heuristics improves the solver’s performance.

4 Experimental Evaluation

In the previous section we figured out the prospective deterministic configuration of the **MapleLCMDistChronoBT** solver. Since the proposed implementation changes do not depend on anything specific to chronological backtracking, we implemented the same changes into the winners of the SAT Competition 2017 and SAT Race 2019. We mark the improved variants of the solvers with `-f2-t-rc`.

Fig. 2. Cactus plots showing the performance of improved implementations of SC2017-SR2019 winners on the combined benchmark set from SC2017, SC2018 and SR2019.

To evaluate the performance of constructed solvers we launched 3 original ones and 3 modifications on all benchmarks from SAT Competitions 2017 and 2018, and that from SAT Race 2019. The cactus plot of the resulting experiment is presented in Fig. 2. The detailed statistics is shown in Table 3. Here, MLCMD stands for `MapleLCMDist`, MLCMDCh for `MapleLCMDistChronoBT` and MLCMDCh-DL for `MapleLCMDistChronoBT-DL-v3`. An improved implementation of each solver is marked with a star.

Table 3. Detailed statistics of improved implementations of the winners of SC2017-SC2018-SR2019 on the instances from SC2017, SC2018 and SR2019. Each cell contains: SCR: SAT: UNSAT: PAR-2 (in seconds).

	SC2017	SC2018	SR2019	Total
MLCMD	206:100:106:4706	229:129:100:4718	225:128:97:4999	660:357:303:4812
MLCMD*	204: 99:105:4693	237:135:102:4541	242:144:98:4597	683:378:305:4607
MLCMDCh	208: 97:111:4659	235:133:102:4573	236:138:98:4799	679:368:311:4678
MLCMDCh*	217:103:114:4373	239:136:103:4490	242:143:99:4556	698:382:316:4477
MLCMDCh-DL	208: 99:109:4586	243:142:101:4402	237:141:96:4675	688:382:306:4553
MLCMDCh-DL*	204: 92:112:4707	242:139:103:4423	243:146:97:4547	689:377:312:4553

Observe that the new implementation improved the performance of all three solvers. The least affected solver is the winner of SR2019: it already employs a viable deterministic strategy for switching between branching heuristics. Also the solver was tuned over benchmarks from SC2017 and SC2018. What is interesting is that the modified solver tackled 6 more unsatisfiable problems than the original, however at a cost of 5 satisfiable ones. This is the largest increase over three solvers on unsatisfiable instances. The largest overall increase in performance was achieved by `MapleLCMDist`. The improved implementation made it possible to solve 23 more instances (21 SAT and 2 UNSAT) and decrease the PAR-2 by about 4%. The overall best performing solver is `MapleLCMDistChronoBT`. It surpassed the closest competitor by 9 solved instances and solved the largest number of UNSAT instances.

Informally, from all the plots and experiments it follows that a proper deterministic strategy for switching between branching heuristics smooths a solver's performance. Observe the evident bump on the lines plotting the performance of non-deterministic solvers near 2500 s mark in Fig. 2. Also, the improved implementations show a drastic performance difference for shorter time limits that smooths out closer to 5000 s. The second modification, which mainly consists in periodical purges of the *Core* database, helps the solver to increase its performance for harder instances, when the amount of learnt clauses in *Core* starts to hinder the rate at which all CDCL procedures operate. The improved implementation shows the increase in both UNSAT and SAT instances solved. There are more SAT instances solved because the considered benchmarks contain more

SAT tests (solvable by at least one participating solver) than UNSAT. All implementations and logs of the experiments are available online¹.

Acknowledgements. The author thanks anonymous reviewers for their helpful comments and Oleg Zaikin and Alexander Semenov for many fruitful discussions.

References

1. Audemard, G., Simon, L.: Predicting learnt clauses quality in modern SAT solvers. In: IJCAI, pp. 399–404 (2009)
2. Biere, A.: CaDiCaL, Lingeling, Plingeling, Treengeling, YalSAT entering the SAT competition 2017. In: Proceedings of SAT Competition 2017, vol. B-2017-1, pp. 14–15 (2017)
3. Biere, A.: CaDiCaL at the SAT Race 2019. In: Proceedings of SAT Race 2019, vol. B-2019-1, pp. 8–9 (2019)
4. Biere, A., Heule, M., van Maaren, H., Walsh, T. (eds.): Handbook of Satisfiability, Frontiers in Artificial Intelligence and Applications, vol. 185. IOS Press (2009)
5. 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
6. Järvisalo, M., Heule, M.J.H., Biere, A.: Inprocessing rules. In: Gramlich, B., Miller, D., Sattler, U. (eds.) IJCAR 2012. LNCS (LNAI), vol. 7364, pp. 355–370. Springer, Heidelberg (2012). https://doi.org/10.1007/978-3-642-31365-3_28
7. Kochemazov, S., Zaikin, O., Kondratiev, V., Semenov, A.: MapleLCMD istChronoBT-DL, duplicate learns heuristic solvers at the SAT Race 2019. In: Proceedings of SAT Race 2019, vol. B-2019-1, p. 24 (2019)
8. 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
9. Liang, J.H., Oh, C., Ganesh, V., Czarnecki, K., Poupart, P.: MapleCOMSPS, MapleCOMSPS_LRB, MapleCOMSPS_CHB. In: Proceedings of SAT Competition 2016, vol. B-2016-1, pp. 52–53 (2016)
10. Liang, J.H., Oh, C., Ganesh, V., Czarnecki, K., Poupart, P.: MapleCOMSPS_LRB_VSIDS and MapleCOMSPS_CHB_VSIDS. In: Proceedings of SAT Competition 2017, vol. B-2017-1, pp. 20–21 (2017)
11. Luby, M., Sinclair, A., Zuckerman, D.: Optimal speedup of Las Vegas algorithms. Inf. Process. Lett. **47**(4), 173–180 (1993)
12. Luo, M., Li, C., Xiao, F., Manyà, F., Lü, Z.: An effective learnt clause minimization approach for CDCL SAT solvers. In: IJCAI, pp. 703–711 (2017)
13. Marques-Silva, J.P., Lynce, I., Malik, S.: Conflict-driven clause learning SAT solvers. In: Biere et al. [4], pp. 131–153
14. Irkutsk Supercomputer Center of SB RAS. <http://hpc.icc.ru>
15. Möhle, S., Biere, A.: Backing backtracking. In: Janota, M., Lynce, I. (eds.) SAT 2019. LNCS, vol. 11628, pp. 250–266. Springer, Cham (2019). https://doi.org/10.1007/978-3-030-24258-9_18

¹ <https://github.com/veinamond/SAT2020>.

16. Moskewicz, M.W., Madigan, C.F., Zhao, Y., Zhang, L., Malik, S.: Chaff: engineering an efficient SAT solver. In: Proceedings of the 38th Annual Design Automation Conference, DAC 2001, pp. 530–535 (2001)
17. 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
18. 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
19. Oh, C.: Improving SAT solvers by exploiting empirical characteristics of CDCL. Ph.D. thesis, New York University (2016)
20. 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