Analysis of comparative effectiveness of state-of-the-art heuristics for CDCL SAT solvers

Stepan Kochemazov
Matrosov Institute for System Dynamics and Control Theory SB RAS, Irkutsk, Russia
E-mail: veinamond@gmail.com

Abstract. The Conflict-Driven Clause Learning algorithms for solving the Boolean satisfiability problem comprise the major part of the methods used to solve various instances of the problems that arise in industry and science. In recent years there have been proposed several major heuristics for these algorithms which are assumed to be de facto good for the solvers' performance over diverse sets of benchmarks. The goal of this paper is to evaluate the contribution of each separate heuristic to the performance of a state-of-the-art solver, see the extent to which they are beneficial, and figure out if the heuristics have any particular features that need to be taken into account.

1. Introduction

The Boolean satisfiability problem (SAT) [1] is one of the most well-studied NP-complete problems in computer science. It is possible to effectively reduce a variety of problems from diverse areas to SAT and effectively solve them in that form. The success of SAT today is largely due to the development of the Conflict-Driven Clause Learning (CDCL) algorithm [2] that was accomplished at the end of the XX-th century.

Since SAT is NP-complete, modern implementations of CDCL employ a lot of heuristics. Some of them can be considered baseline, such as the Variable State Independent Decaying Sum (VSIDS) [3], restarts [4], or Literal Block Distance [5]. Others are either situational or were proposed only recently. The latter include Learnt Clause Minimization [6], Distance heuristic [7], Chronological Backtracking [8], Duplicate Learnts heuristic [9], and the use of Stochastic Local Search component [10]. The interesting observation is that the majority of applications, such as cryptography or bioinformatics, tend to use the older solvers developed prior to 2013, such as `MiniSAT` [11] or `glucose` [5]. The new CDCL heuristics proposed in recent years have yet to earn such acclaim.

In the present paper, the goal is to evaluate the contribution of the state-of-the-art CDCL heuristics proposed recently to the solvers' effectiveness and compare them with each other. The main studied question is whether the new heuristics do improve the solvers' effectiveness over a wide range of benchmarks or not. An additional goal is to understand the strengths and weaknesses of these heuristics.

A brief outline of the paper is as follows. The next section presents some background on CDCL SAT solvers and the considered heuristics. The third section contains the methodology

of experiments, including the description of several variants of the `Relaxed_LCMDCBDL_newTech` SAT solver that took 2nd place in the SAT Competition 2020. These variants differ only in the enabled heuristics. They are compared in the fourth section which is devoted to the computational experiments and their discussion. It is followed by the conclusion and acknowledgments.

2. Background

As was mentioned above, the modern CDCL SAT solvers are programmatic implementations of the algorithms for solving the Boolean satisfiability problem. Essentially, they are the tools to be used in practice for tackling hard problems, that can not be easily be solved using existing effective approaches. The progress in CDCL SAT solvers is mainly realized via annual SAT Competitions[1]. These competitions use benchmark sets comprised of instances from diverse application domains to test new heuristics and implementations of the CDCL algorithm. They are popular with the developers of SAT solvers and related technologies. The emergence of the `MiniSAT` solver in 2004 [11] marked an important stepping stone in the CDCL solvers development thanks to the high quality of its programmatic implementation. To this day, the majority of modern algorithms for solving SAT are either directly based on `MiniSAT` or employ many of the implementation techniques it popularized. It would not be far off to say, that `MiniSAT` can be viewed as one of the main progenitors of almost all modern SAT solvers. On the heuristics side, `MiniSAT` implemented VSIDS and the frequent restarts. Later, the state of the art in CDCL was extended by the concept of Literal Block Distance (LBD) [5]. However, since then the progress in the area somewhat stagnated. Despite many new heuristics being introduced, many SAT applications still employ `MiniSAT` and `glucose` solvers proposed more than ten years ago.

Nevertheless, the SAT competitions continue, and naturally, the heuristics which appear in the winners of competitions are viewed with great interest. In the context of this paper, several promising heuristics proposed in recent years will be considered. The brief outline and description of the analyzed heuristics follow.

- **LCM**: Learnt Clause Minimization – the heuristic proposed in [6], which is aimed at improving the quality of learnt clauses by applying to them a special procedure that makes it possible to remove the redundant literals from a clause.
- **DISTANCE**: the distance heuristic was first introduced in [7]. Its goal is to initialize the values of the branching heuristic by directing the search at the beginning. It is relatively expensive, so in most implementations the heuristic works for at most 100 000 conflicts.
- **CB**: Chronological Backtracking – this method described in [8] implements an attempt to make a partial return to the roots of SAT solving, in particular, to the Davis–Putnam–Logemann–Loveland (DPLL) algorithm. It allows the solver in certain conditions to ignore the nonchronological backtracking, which became one of the trademarks of CDCL solvers, in favor of a chronological one.
- **DL**: Duplicate learnts heuristic [9] attempts to use simple data mining methods to discover the learnt clauses that are repeatedly derived during the search and store them permanently.
- **SLS**: the Stochastic Local Search component augmented with *Rephasing* technique has been appearing in the growing number of solvers. While the idea of combining CDCL and SLS is far from novel, the implementation of the SLS component in `Relaxed_LCMDCBDL_newTech` [10] has several specific features that have not been used before.

The implementation of each heuristic usually contributes major changes to some of the core SAT solver components. Nevertheless, it is often possible to isolate the heuristic from the

[1] http://www.satcompetition.org/

rest of the algorithm. Since the goal of the present paper is to evaluate the heuristics and compare their effectiveness, it was decided to perform the computational experiments using the SAT solver which already implements all of the listed ones. In particular, there was used the `Relaxed_LCMDCBDL_newTech` solver that took the second place at SAT Competition 2020. Below it is referred to as `RLNT`. What makes it interesting compared to SAT Competition 2020 winner, the `kissat` solver, is that the lineage of `RLNT` can be easily followed back to `Minisat` [11]. Indeed, `RLNT` is based on `MapleLCMDistChronoBT-DL-v3` – the winner of SAT Race 2019, which is based on `MapleLCMDistChronoBT` – the winner of SAT Competition 2018, based on `MapleLCMDist` – the winner of SAT Competition 2017, based on `MapleCOMSPS` – the winner of SAT Competition 2016, based on `CominisatPS` [12], based on `Minisat` and `glucose`. Informally, it can be said that `MapleLCMDist` = `MapleCOMSPS` + **LCM** + **DIST**, `MapleLCMDistChronoBT` = `MapleLCMDist` + **CB**, `MapleLCMDistChronoBT-DL-v3`=`MapleLCMDistChronoBT`+ **DL**. Finally, `RLNT` = `MapleLCMDistChronoBT-DL-v3` + **SLS**. The reality is more complex in the latter two cases, since the difference between the corresponding solvers involves changes to some of the parameters of existing heuristics.

3. Methodology of experiments

To evaluate the contribution of heuristics in equal conditions, there was developed the implementation of `RLNT`, in which each of the heuristics listed in the previous section can be enabled or disabled independently via the use of `#ifdef` mechanism. It means that by manipulating the defined constants it is possible to change only the header of a single source file, recompile the program and obtain, e.g. `RLNT-noCB` solver, which has the Chronological Backtracking disabled. The general idea that is employed to compare the heuristics is to switch them off separately. The greater the decrease in the solver's performance from disabling a heuristic, the greater is its benefit for the solver.

3.1. Solvers

For the experiments there have been prepared the following solvers that are the variants of `RLNT`: `RLNT` itself, `RLNT-noSLS`, `RLNT-noDL`, `RLNT-noCB`, `RLNT-noDIST`, `RLNT-noLCM`, `RLNT-noDL-noDIST`. Here, `RLNT` followed by `no*` refers to the version of `RLNT` in which the heuristics corresponding to the solver name are disabled. The latter variant, `RLNT-noDL-noDIST` was evaluated because in the preliminary testing it showed good performance. In addition to that, in order to evaluate the progress in CDCL solvers from a slightly different perspective, the winners of SAT Competitions 2017-2019 were also included: `MapleLCMDist`, `MapleLCMDistChronoBT`, `MapleLCMDistChronoBT-DL-v3`. To offset them the versions of `RLNT` were prepared that coincide with the respective SAT competition winners in the enabled heuristics, e.g. `RLNT-noSLS-noDL` corresponds to `MapleLCMDistChronoBT`, `RLNT-noSLS-noDL-noCB` corresponds to `MapleLCMDist`. Finally, the version `RLNT-noSLS-noDL-noCB-noDIST-noLCM` is the variant in which all heuristics are disabled.

3.2. Benchmarks

To accurately assess the performance of CDCL SAT solvers it is necessary to use a diverse set of benchmarks that come from different applications. The benchmark sets used in SAT Competitions appear to satisfy this criterion, thus in this paper, they were used in the experiments. In particular, the test instances from the main tracks of SAT Competitions 2016[2] and 2020[3], and also SAT Race 2019[4] were used. They are freely available from the corresponding

[2] https://baldur.iti.kit.edu/sat-competition-2016/index.php?cat=downloads
[3] https://satcompetition.github.io/2020/downloads.html
[4] http://sat-race-2019.ciirc.cvut.cz/index.php?cat=downloads

websites. In total there were 1300 instances.

3.3. Evaluation criteria

Remind, that when a SAT solver is launched with a fixed time limit on some SAT instance, there are three possible outcomes. First, it can find a satisfying assignment, meaning that a SAT instance is *satisfiable*. Second, it can prove that there are no satisfying assignments and thus the input formula is *unsatisfiable*. If the solver terminates due to the time limit, then the status of the instance is unknown. Overall, taking into account the fact that a formula can have many satisfying assignments, it may seem that satisfiable instances are significantly easier to tackle for a SAT solver, compared to unsatisfiable instances, for which it is required to construct a specific proof. Empirical observations show that it is indeed true but to a limited extent. Both hard and simple cases of satisfiable and unsatisfiable instances are often encountered in practice.

In the experiments, each considered solver was launched once on each instance from the benchmark set. The outcome of the launch, together with the runtime of a solver was recorded. The statistics presented below include the count of solved instances, divided into counts for solved satisfiable and unsatisfiable ones. Similar to the SAT competitions, these numbers are accompanied by the PAR-2 score (Penalized Average Runtime with factor 2) which is computed as follows. First, a sum of terms is computed, where for each instance from the benchmark set the term is formed as a runtime of a solver if it solved this instance, or as 2 times the time limit if the solver did not. Then this sum is divided by the number of terms. Given the two solvers that solved the equal number of instances, the one with the lower value of PAR-2 is faster on average.

3.4. Computing platform

Following the standard SAT Competition procedure, all solvers were launched with a time limit of 5000 seconds. As a computing platform the PCs with 16-core AMD Ryzen 3950x CPUs and 32Gb RAM were used, operating under Ubuntu 20.10. The solvers were launched in 16 simultaneous threads without restrictions on memory usage.

4. Computational experiments

The results of the experiments are summarized in Table 1. The so-called cactus plots, that present the performance of the solvers on satisfiable instances and unsatisfiable instances, are showed in Figures 1 and 2. The plotline for each solver displays the runtimes of this solver over all benchmarks, ordered in ascending order. It means that on the cactus plot, the further the plotline is to the right – the larger number of instances were successfully tackled by the corresponding solver within the time limit, and the closer the line is to the bottom – the smaller is the average runtime of a solver over all benchmarks.

4.1. Discussion

The analysis of the results in Table 1, together with that presented in cactus plots at Figure 1 and 2 allows one to make several observations. First, it is quite clear that, at least on the employed benchmark set, the heuristics do not yield equal gain. The first three rows of the Table clearly show, that dropping some of the heuristics increases the solver's performance. In particular, it appears that disabling **DL** or **DIST** allow `RLNT` to solve several more satisfiabile and unsatisfiable instances. Disabling them together yields even greater benefit. Therefore, it is necessary to reevaluate the combination of heuristics for each particular solver if the goal is to achieve the best possible performance. It means that when incorporating new heuristics into a solver, it is best to study how it fits into the general picture. However, this makes it much harder to perform straightforward fair comparisons. Another interesting observation that can

Table 1. The results of computational evaluation of different solvers on the joint set of benchmarks from SAT Competitions 2016 and 2020 and SAT Race 2019 (1300 tests in total). SCR (Solution Count Ranking) refers to the total number of solved instances.SCR:SAT and SCR:UNSAT refer to the numbers of solved satisfiable and unsatisfiable benchmarks, respectively. PAR2 column refers to penalized average runtime with parameter 2.

Solver	SCR	SCR:SAT	SCR:UNSAT	PAR2
`RLNT-noDL-noDIST`	726	405	321	4841.34
`RLNT-noDIST`	721	400	321	4875.14
`RLNT-noDL`	721	403	318	4883.67
`RLNT`	715	399	316	4904.70
`RLNT-noCB`	697	378	319	5024.10
`RLNT-noSLS-noDL`	694	344	350	5179.37
`RLNT-noLCM`	684	387	297	5116.70
`MapleLCMDistChronoBT-DL-v3`	682	331	351	5252.24
`RLNT-noSLS-noDL-noCB`	682	337	345	5270.08
`RLNT-noSLS`	671	331	340	5300
`RLNT-noSLS-noDL-noCB-noDIST-noLCM`	670	340	330	5341.44
`MapleLCMDistChronoBT`	665	312	353	5413.83
`MapleLCMDist`	658	303	355	5483.10

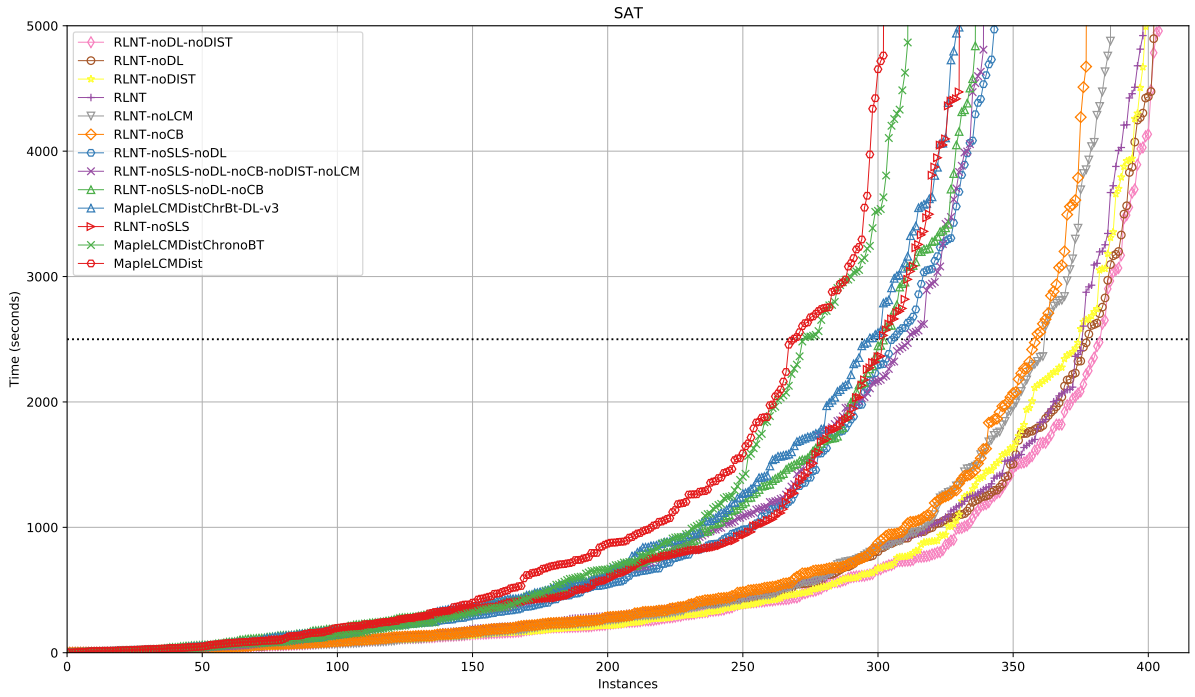

Figure 1. Cactus plot for considered solvers over satisfiable instances.

be drawn from Figure 1 is the following: it is clear that the **SLS** heuristic appears to give the solver the ability to cope with at least ≈ 50 more satisfiable test instances from the considered set of benchmarks than the competition, which is quite surprising. It is the **SLS** heuristic that is responsible for the large gap between two groups of plot lines at Figure 1. On the other

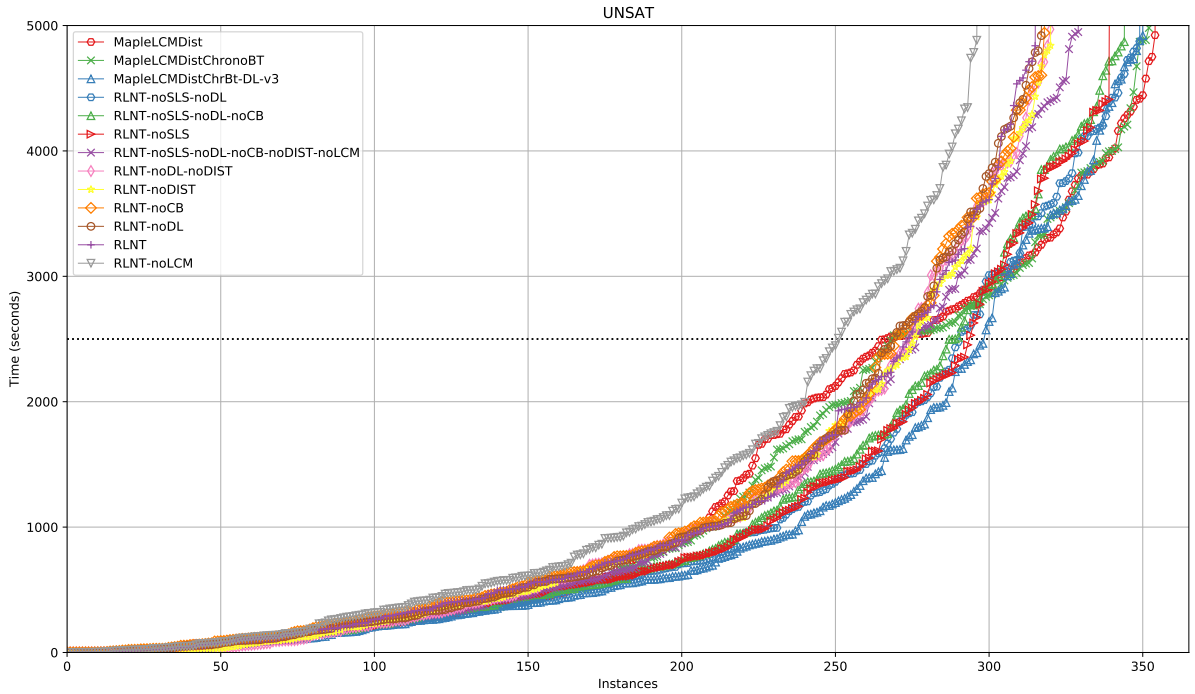

Figure 2. Cactus plot for considered solvers over unsatisfiable instances.

hand, from Table 1 it can be seen that dropping **SLS** results in a significant gain (about 30) in the number of unsatisfiable instances solved, which is, however, overshadowed by the decrease in the number of solved satisfiable tests. Apart from **SLS**, the next best heuristic appears to be **LCM** which mostly targets the unsatisfiable instances. The **CB** heuristic which helps the solver a lot with satisfiable instances takes third place. Another observation is that the overall architecture of SAT solvers indeed progressed slightly in recent years, since, for example, `RLNT-noSLS-noDL` shows better performance compared to `MapleLCMDistChronoBT` which uses the same set of heuristics.

Comparing the cactus plots at Figures 1 and 2 one can conclude that since 2017, the solvers started to switch focus to satisfiable instances, often at the cost of unsatisfiable ones. In particular, the `MapleLCMDist` solver from 2017 managed to solve the largest number of unsatisfiable instances and the smallest number of satisfiable ones out of all tested solvers. The switch in focus, however, is accompanied by overall progress that results in decreasing the average solver runtimes.

Now, let us look at the detailed statistics on satisfiable and unsatisfiable instances solved by each solver in three benchmark sets, corresponding to SAT Competition 2016 (SC2016), SAT Race 2019 (SR2019), and SAT Competition 2020 (SC2020). It is presented in Table 2. The horizontal line after the sixth row separates the solvers with **SLS** and the solvers without this heuristic. Observe, that for both SC2016 and SR2019 the enabling of **SLS** leads to the increase of the number of satisfiable instances solved by a moderate amount, from 3 to 12. On the other hand, for SC2020 the difference induced by **SLS** is at least 25 instances. It means that the benchmark set for SC2020 is (hopefully, inadvertently) biased towards the **SLS** heuristic. As it was mentioned above, enabling SLS in all cases results in the decrease of unsatisfiable instances solved, but by a relatively small amount. From the analysis of other results, it can be conjectured, that the benchmark set from SC2016 is biased towards **LCM**, but to a significantly

smaller degree. Observe that disabling **LCM** results in ≈ 10 less unsatisfiable instances solved in all cases. As for the SR2019 benchmark series, it appears to be more or less balanced and does not appear to be biased towards any specific heuristic. However, it should be taken into account that the proportion of satisfiable instances solvable under a time limit to that of unsatisfiable instances. From Table 2 it is clear that SC2016 favors heuristics targeting unsatisfiable instances, while SR2019 and SC2020 benchmark sets are better for solvers with heuristics that work well on satisfiable instances.

Table 2. Detailed statistics on satisfiable and unsatisfiable instances solved.

	SC2016		SR2019		SC2020	
	SAT	UNSAT	SAT	UNSAT	SAT	UNSAT
`RLNT-noDL-noDIST`	099	121	159	095	147	105
`RLNT-noDIST`	095	120	154	095	151	106
`RLNT-noDL`	090	119	161	094	152	105
`RLNT`	089	118	159	094	151	104
`RLNT-noCB`	085	122	156	094	137	103
`RLNT-noLCM`	087	107	159	089	141	101
`RLNT-noSLS-noDL`	083	139	150	099	111	112
`MapleLCMDistChronoBT-DL-v3`	077	142	145	098	109	111
`RLNT-noSLS-noDL-noCB`	080	139	147	099	110	107
`RLNT-noSLS`	081	135	142	097	108	108
`RLNT-noSLS-noDL-noCB-noDIST-noLCM`	081	128	147	094	112	108
`MapleLCMDistChronoBT`	076	145	141	099	095	109
`MapleLCMDist`	077	147	138	100	088	108

Taking into account all of the above, it is quite clear, that from the point of view of application domains, where the majority of SAT instances to solve are unsatisfiable, the modern CDCL heuristics are not very enticing. On the other hand, the domains which work mainly with satisfiable instances are likely to benefit from using the cutting-edge SAT competition winners. Overall, the more heuristics exist – the better, because since SAT is the NP-complete problem, it means that in advance, no one can predict whether some heuristic will work on the particular class of instances or not. That is why the choice makes the area richer with possibilities for improvement.

5. Conclusion

The results of the computational experiments indicate that in the CDCL SAT solvers that employ many separate heuristics it is important to sometimes spend time on reviewing their performance and tuning the existing configurations because sometimes disabling a heuristic can result in the performance gain comparable to that from adding a brand new one. The predispositions of some heuristics towards satisfiable or unsatisfiable instances should also be factored into the choice of whether to employ them in each particular practical case.

Acknowledgments

The research was supported by the RFBR grant No.19-07-00746.

References

[1] Biere A, Heule M, van Maaren H and Walsh T (eds) 2009 *Handbook of Satisfiability* (*Frontiers in Artificial Intelligence and Applications* vol 185) (IOS Press) ISBN 978-1-58603-929-5

[2] Marques-Silva J P, Lynce I and Malik S 2009 Conflict-driven clause learning SAT solvers in Biere *et al.* [1] pp 131–153

[3] Moskewicz M W, Madigan C F, Zhao Y, Zhang L and Malik S 2001 Chaff: Engineering an efficient SAT solver *Proceedings of the 38th Annual Design Automation Conference* DAC '01 pp 530–535
[4] Luby M, Sinclair A and Zuckerman D 1993 *Inf. Process. Lett.* **47** 173–180
[5] Audemard G and Simon L 2009 Predicting learnt clauses quality in modern SAT solvers *IJCAI* pp 399–404
[6] Luo M, Li C, Xiao F, Manyà F and Lü Z 2017 An effective learnt clause minimization approach for CDCL SAT solvers *IJCAI* pp 703–711
[7] Xiao F, Luo M, Li C M, Manya F and Lu Z 2017 MapleLRB_LCM, Maple_LCM, Maple_LCM_Dist, MapleLRB_LCMoccRestart and glucose-3.0+width in SAT competition 2017 *Proc. of SAT Competition 2017* vol B-2017-1 pp 22–23
[8] Nadel A and Ryvchin V 2018 Chronological backtracking *SAT* (*LNCS* vol 10929) pp 111–121
[9] Kochemazov S, Zaikin O, Semenov A A and Kondratiev V 2020 Speeding up CDCL inference with duplicate learnt clauses *ECAI 2020* (*FAIA* vol 325) pp 339–346
[10] Zhang X and Cai S 2020 Relaxed backtracking with rephasing *Proc. of SAT Competition 2020* vol B-2020-1 pp 15–16
[11] Eén N and Sörensson N 2004 An extensible sat-solver *SAT 2004* (*LNCS* vol 2919) pp 502–518
[12] Oh C 2015 Between SAT and UNSAT: The fundamental difference in CDCL SAT *SAT* (*LNCS* vol 9340) pp 307–323