EPiC Series in Computing

Volume 94, 2023, Pages 306–316

Proceedings of 24th International Conference on Logic for Programming, Artificial Intelligence and Reasoning

Analyzing Multiple Conflicts in SAT: An Experimental Evaluation *†

Albert Oliveras[1], Enric Rodríguez-Carbonell[1], and Rui Zhao[1]

Technical University of Catalonia, Barcelona
oliveras@cs.upc.edu, erodri@cs.upc.edu, rzhao@cs.upc.edu

Abstract

Unit propagation and conflict analysis are two essential ingredients of CDCL SAT Solving. The order in which unit propagation is computed does not matter when no conflict is found, because it is well known that there exists a unique unit-propagation fixpoint. However, when a conflict is found, current CDCL implementations stop and analyze that concrete conflict, even though other conflicts may exist in the unit-propagation closure. In this experimental evaluation, we report on our experience in modifying this concrete aspect in the CaDiCaL SAT Solver and try to answer the question of whether we can improve the performance of SAT Solvers by the analysis of multiple conflicts.

1 Introduction

Propositional satisfiability (SAT) is probably one of the most well-studied problems in Computer Science. From the theoretical point of view, it was the first problem to be proved NP-complete [12], establishing a landmark result. Proof complexity researchers have also found in SAT an excellent problem where studying the power and limitations of several proof systems [10]. But what is so particular about SAT is that, apart from the deep theoretical study of the problem, and probably thanks to it, algorithms that work very well in practice have also been developed. Because of that, it is now common to see SAT solvers being essential tools in areas such as Electronic Design Automation [28], Hardware Verification [25], Model Checking [9] and even Mathematics [18].

The most successful approach to SAT Solving is the Clause-Driven Conflict Learning (CDCL) procedure [21]. The seminal works of [27, 26] in the GRASP solver were later rationalized and extended further in the Chaff solver [23], starting what is sometimes known as "the SAT Revolution". The key ideas of CDCL have been extended with pre/inprocessing techniques [8], proof production [17], and new ideas for lemma removal [2], restarting mechanisms [7] and decision heuristics [5], among others.

*All authors are supported by grant PID2021-122830OB-C43, funded by MCIN/AEI/ 10.13039/501100011033 and by "ERDF: A way of making Europe".

†Source files can be found at https://github.com/dearzhaorui/multiple-conflicts-analyzing-in-sat

All this knowledge has allowed the implementation of extremely competitive and sophisticated solvers that incorporate a wide range of techniques, which are scheduled to be applied at different moments during the search process. Because of this, it is becoming increasingly difficult to find breakthrough ideas that lead to remarkable performance improvements. But finding these ideas is probably as important as performing critical analysis of new techniques and understand the reasons for their success, and also for their failure.

In this paper, we explore the simple idea of allowing unit propagation to collect a set of conflicting clauses, instead of only one, analyzing all of them and learning the best one(s) according to some criterion. This is not a new idea, as it has been presented in [20, 15, 11] and was implemented in the zChaff2004 system [19]. However, none of these papers provide a thorough experimental evaluation and, when some experimental results are presented, the conclusion is that this technique does not have a significant impact on runtime. In addition, SAT solvers at the time of some of these works have little to do with current ones, and hence, their findings might not be applicable anymore.

This paper is organized as follows. In Section 2 we give some basic preliminaries about SAT Solving. In Section 3 we study the potential of this technique without taking runtime into consideration. That is, we focus on the potential of this technique in terms of reducing the exploration of the search space. After that, in Section 4 we report on our experience trying to implement an efficient version of this technique that only learns one lemma per conflict. In Section 5, we make use of the chronological backtracking [24, 22] technique to learn *and propagate* multiple lemmas at a time. We conclude in Section 6.

2 Preliminaries

We will assume that the reader is familiar with the basic notions of propositional logic.

Unit Propagation. Given a formula F and an assignment α, *unit propagation* looks for a clause $C \in F$ where all literals are false in α except one, that is undefined, and adds this literal to α with reason C. This process is repeated until fixpoint or until a *conflicting* clause is found: one with all its literals being false in α. Efficient implementations of this procedure use the two-watched literal scheme [23]. In a nutshell, two non-false literals are watched in every clause. If these two non-false literals exist, the clause cannot unit propagate. Hence, when a literal l is added to the assignment, all clauses where $\neg l$ is watched are visited, since these are the only candidates for unit propagation. During this visit, we check whether the clause is conflicting, whether it unit propagates a literal, or else we can select another non-false literal to be watched.

CDCL. The Conflict-Driven Clause Learning (CDCL) algorithm [21] for determining whether a formula is satisfiable starts with an empty assignment α, which is extended and reduced by applying the following steps until the satisfiability of the formula is determined. First, unit propagation is applied. If no conflicting clause is found, either α is a total assignment (and hence satisfies the formula), or an unassigned literal (*decision literal*) is chosen and added to α. This choice is determined by dedicated decision heuristics [6]. If unit propagation finds a conflicting clause, this is analyzed by a *conflict analysis* procedure [30], which constructs a clause C (*lemma*) that can be safely conjuncted with the formula. It is guaranteed that, by removing enough literals from α (*backjumping*), this lemma will unit propagate a new literal.

Conflict Analysis. If we see an assignment as a sequence, any literal appearing in the sequence between the k-th decision literal and the $(k+1)$-one is said to belong to *decision level* k. In order to generate a proper lemma L, conflict analysis relies on this definition and on the

resolution rule, which, given clauses $l \vee C$ and $\neg l \vee D$ derives clause $C \vee D$. Initially, L is the conflicting clause and, while L contains more than one variable at the current decision level, the procedure takes the literal l appearing later in α (which will be of the current decision level) and sets $L := resolution(L, reason(l))$. When this process finishes, the only literal in L at the current decision level is called the *Unique Implication Point (UIP)*, and is the one that will be unit propagated after backjumping. Finally, a *lemma minimization* procedure [29, 13, 14] is executed, removing some literals from L and hence allowing the solver to learn a shorter lemma.

LBD. Adding too many lemmas will slow down unit propagation. This is why, periodically, CDCL solvers remove some learned lemmas. Currently, the most popular measure to determine the quality of a lemma is the *Literal Block Distance (LBD)*: the number of distinct decision levels of its literals. Low-LBD lemmas are preferred over those with large LBD. Although there exist lemma minimization procedures that allow one to reduce the LBD [1, 16], techniques implemented in state-of-the-art SAT solvers like CaDiCaL do not reduce this measure.

3 Potential of Analyzing Multiple Conflicts

In this section we will present some data that will allow us to assess to which extent we can improve the performance of SAT solvers by collecting several conflicting clauses during unit propagation, instead of stopping as soon as we find the first one. All experiments in this paper will be done by modifying the state-of-the-art SAT Solver CaDiCaL [4].

In this first experiment, we modified CaDiCaL as follows. During unit propagation, every time a conflicting clause is found, we store it in a set and continue propagating. Using a set is useful in order to avoid duplicates since the same conflicting clause might be visited, and found conflicting, several times. As usual, unit propagation finishes when all watch lists of all literals in the assignment have been visited. After that, we apply conflict analysis and lemma minimization to all conflicting clauses in order to derive a set of lemmas. Finally, we have to decide which lemma to learn. Since currently it is accepted that LBD is the best criterion to judge the quality of a lemma, we learn the one with lowest LBD. In case of a tie, we choose the one with the smallest size and, if the tie persists, the first lemma we have found is selected.

This corresponds to the ideal situation in which we can afford to apply unit propagation until fixpoint, and then analyze and minimize all conflicting lemmas. For the moment, we will forget about the overhead this causes to the SAT solver and present some interesting data we have extracted from the experiments. All experiments in this paper have been performed over the set of 400 benchmarks that were used in the Main Track of the 2022 SAT Competition [3]. We have run the systems on two sets of machines, with 10 and 8 units each, both 3.3Ghz 16GB Intel Xeon E-2124, but one set slightly faster than the other. For fairness, when doing comparisons about runtime, all systems are run on the same set of machines.

The first experiment we performed was to run the modified version of CaDiCaL explained above with a time limit of 900 seconds per benchmark. Despite this may seem a rather low time limit, it is enough to analyze the behavior of the system. The first question we wanted to answer is how often it happens that unit propagation finds more than one conflicting clause. The top-left histogram in Figure 1 summarizes the percentage of conflicts where multiple conflicting clauses were found. That is, how often it happens that unit propagation finds more than one conflicting clause. The bar over label 90 indicates that in 38 benchmarks, between 90 and 95% of the conflicts found multiple conflicting clauses. All histograms should be understood in the same way. It was surprising for us to see that in about 80% of the benchmarks more than 70% of the conflicts were multiple. Thus, very often several conflicting clauses are found. The average

number of different conflicting clauses can be seen in the histogram below. We can observe that there is a significant number of benchmarks for which, in case of a multiple conflict, the set of conflicting clauses to be analyzed is quite large. In about 80% of the benchmarks the average number is greater than 10, hence allowing the technique to analyze a large set of conflicts.

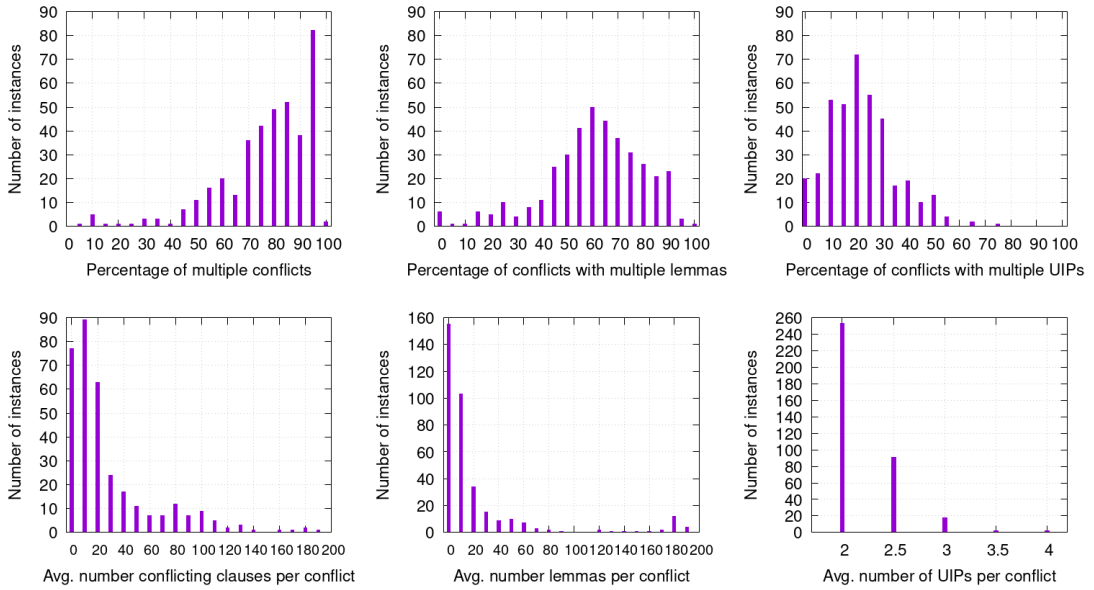

Figure 1: Multiple conflicts, multiple lemmas and multiple UIPs. All average histograms only consider the average over the conflicts with multiple lemmas, UIPs or conflicting clauses.

Second, we suspected that different conflicting clauses might lead to the same lemma. This would be harmful for our technique since it would mean that, although being syntactically different, they essentially correspond to the same conflicting situation. Concrete data can be seen again in Figure 1, where we plot the percentage of conflicts where more than one different lemma is generated. Results indicate that different clauses giving the same lemma is not uncommon, but still, in about 35% of the benchmarks more than 70% of conflicts gave different lemmas. Data about how many different lemmas are generated can be found in the histogram below, which shows, for example, that in about 60% of the benchmarks, when more than one lemma is found, we have on average more than 10 lemmas.

Yet another situation that could indicate that different lemmas are essentially trying to solve the same conflicting situation is when all lemmas have the same UIP. We analyzed how often this happens and we found that, in turn, the number of times when different UIPs are found is substantially lower than the number of times when different lemmas are found. Hence, finding multiple lemmas with the same UIP is not rare. The histograms on the right show that in almost no benchmark more than 70% of the lemmas contained different UIPs. In addition, when multiple UIPs are found, there are usually between 2 and 3 different UIPs on average. This result suggests that the diversity one can obtain by unit propagation up to fixpoint might not be as large as the initial experiments seemed to indicate.

However, one could still argue that obtaining a lemma that has the same UIP as the lemma found by standard CDCL, but a much lower LBD can have a very positive impact on performance. We computed how often the modified version of CaDiCaL found a lemma with LBD at

most 3, while the LBD of the lemma derived by analyzing the first found conflicting clause was larger than 10. These are somehow arbitrary values, but they will help in getting an idea of which type of LBD improvement we can obtain. The results can be seen in the histogram in Figure 2, where we clearly observe that this type of large LBD improvements are sporadic.

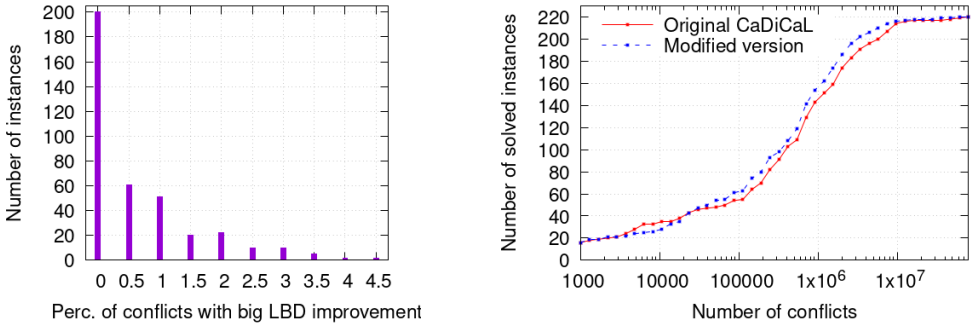

Figure 2: Big improvements in LBD and solved instances within a given number of conflicts.

Despite these somehow negative data, it turns out that the resulting system needs fewer conflicts to solve the problems, which indicates that the search behavior is better. We considered all benchmarks that CaDiCaL and the modified version could solve within 5000 seconds and compared the number of conflicts needed to solve them. Results are displayed on the cactus plot in Figure 2, where we can see that, given a certain number of conflicts, the modified version usually solves more instances. However, the benefit is quite limited and hence, even if all the extra work we are doing were free, the improvement would probably not be dramatic.

4 Implementation of a Limited but Efficient Version

The naive version we presented in the previous section is too costly with respect to unit propagation, conflict analysis and lemma minimization as we can see from the histograms in Figure 3. The histograms plot the percentage of the total runtime spent in these three procedures.

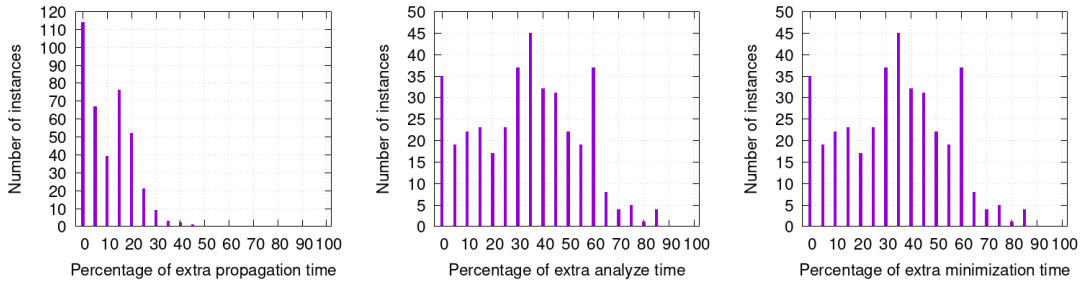

Figure 3: Percentages of extra propagation, conflict analysis and lemma minimization.

In order to improve the cost of these three operations we did the following improvements. With respect to unit propagation, once the first conflicting clause is found, we start a new procedure for unit propagation where neither the watches nor the model are modified. That is,

we only traverse the pending watch lists looking for conflicting clauses, without changing the assignment. That strongly reduces the amount of write operations of the procedure.

During the execution of conflict analysis, the number of lower-decision-levels already present in the lemma is a lower bound on the final lemma LBD. Hence, for each conflicting clause, we can stop its conflict analysis as soon as the LBD lower bound is larger than or equal to the lowest LBD found so far. Recall that lemma minimization will not change the LBD and hence, we will only minimize the lemma finally chosen to be learned.

However, as we learned from the data in Section 3, there are lots of conflicts to be analyzed and hence it is unlikely that only these improvements make it feasible to compute unit propagation up to fixpoint and analyze all conflicts within a reasonable amount of time. Fortunately, experimental results indicate us that there is no need to do so much work to get similar benefits. More concretely, in the experiments presented in the previous section, we also computed how often the best lemma was not the first one, and how often it was not within the first five conflicting clauses. Results in Figure 4 indicate that it is uncommon that the best lemma is not found within the first five conflicting clauses. As an example, in 75% of the benchmarks this happens in less than 15% of the conflicts on average. This empirically supports collecting only at most 5 conflicting clauses, because not too many good lemmas will be lost if we do so.

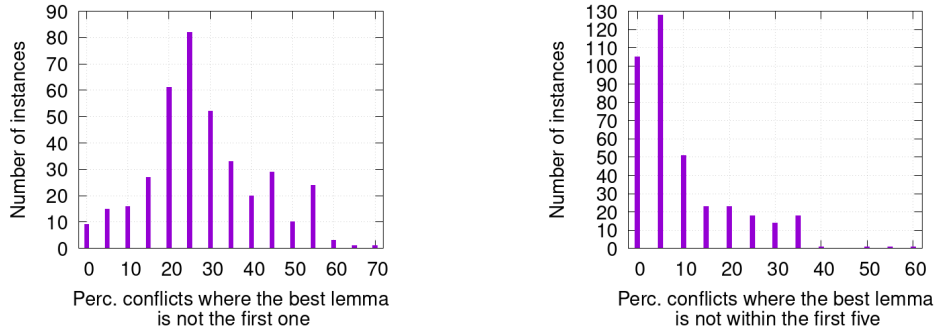

Figure 4: Position among all conflicting clauses of the best lemma.

Experiments revealed that limiting this was still not enough. The reason is that there are situations where we cannot find five conflicting clauses. In this case, unit propagation will be computed up to fixpoint. In order to avoid this, we count the number of clause dereferences performed after the first conflicting clause and stop propagation whenever this number exceeds a certain limit. We do this as an estimation of the amount of extra work being done. Finally, we added a third limit constraining the maximum number of watch lists with conflicting clauses that we visit. The rationale is that, since in CaDiCaL unit propagation is done in a breadth-first fashion, we do not want to consider conflicts found in the watch list of a literal that is very far away in the assignment. Most likely, such conflicts will be at a large distance from the decision literal in the implication graph, and hence the likelihood of a large-LBD lemma is high.

With these limitations on the amount of extra work done, the percentage of benchmarks where the extra propagation time is at least 5% has reduced from 70% to 25%. Regarding conflict analysis, this percentage has decreased from 90% to 10% of the benchmarks. Lemma minimization does not incur in any overhead because we are only minimizing one lemma, as in the standard setting. Even with this reduction on the extra work done, the percentage of cases with multiple conflicts, lemmas or UIPs is still large enough to provide some potential advantage.

For simplicity, we implemented these ideas on CaDiCaL with chronological backtracking disabled. We ran the original CaDiCaL and the modified one on the 400 benchmarks with 5 random seeds and a time limit of 5000 seconds per benchmark. The results in Section 3 seemed to indicate that there could be some gain but not a massive one and this is exactly what happens. The following table shows the number of solved instances per seed and system:

System	Seed 1	Seed 2	Seed 3	Seed 4	Seed 5	Median
Cadical-no-chronoBT	289	289	290	295	290	290
Multiple-conflicts-one-lemma	291	292	291	289	291	291

As we can see, the two systems are quite comparable, although the median of the modified version is slightly better. In Figure 5, we present two cactus plots with the number of solved instances with respect to time and number of conflicts. For each system, we only include the benchmarks for which it did not time out in any of the 5 seeds (276 for CaDiCaL and 278 for the version analyzing multiple conflicts). The time and conflicts displayed are the average over all seeds. Our modified version is a little bit more stable, in that it solves more benchmarks in all seeds, hence being less affected by randomness, but by a narrow margin (276 vs. 278).

The average number of conflicts is also a bit better, but it seems that even if all the extra work were free, the improvement in search would not pay off.

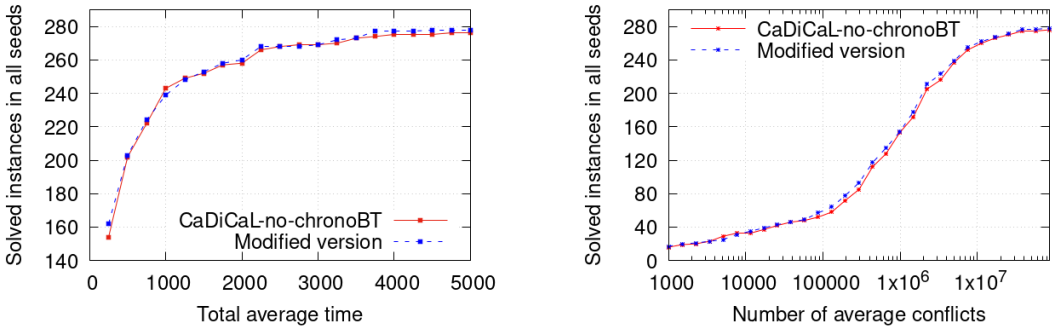

Figure 5: Solved instances within a given time or number of conflicts.

5 Learning Multiple Lemmas

Analyzing several conflicting clauses and only learning the best lemma according to some criterion wastes a lot of the work that has been done. Since we have computed several lemmas, a possibility is to use all of them to the maximum possible extent. On the one hand, lemmas are learned so that they will propagate in the future preventing similar conflicts, but they also have the property that, after backjump is performed, they immediately propagate.

Example. Let us assume that the current decision level is 7, and that we learn two lemmas, $L_1 := p\,(7) \vee \neg q\,(3) \vee r\,(2)$ and $L_2 := \neg s\,(7) \vee t\,(6) \vee \neg q\,(3)$, where the numbers between parentheses are the decision levels of the literals. If we learn L_1 we would backjump to level 3 and propagate p; for L_2 we would backjump to level 6 and propagate $\neg s$. However, we are in trouble if we want to learn and propagate both. If we learn L_1 and backjump to 3 lemma L_2 no longer propagates. Otherwise, if we learn L_2 and backjump to 6, lemma L_1 also unit

propagates but we cannot add p to the assignment because that assignment should happen at decision level 3 but we are at decision level 6. This would break several invariants of CDCL.

Fortunately, CaDiCaL implements chronological backtracking [24, 22] which, among other things, allows the assignment of the solver (sometimes known as the *trail*) to be not in order. That is, we might find a literal set at decision level 3 later in the assignment than a literal set at decision level 5. This flexibility allows us to fix the problem mentioned in the example above, and hence paves the way to use both lemmas to propagate right after conflict analysis.

The procedure we implemented is the following. We first collect a certain set of conflicting clauses. Then, since in chronological backtracking we might find clauses that were already conflicting at some previous decision level, we compute the lowest decision level at which some conflict occurred, select all conflicting clauses of that level, and choose among them the best lemma (i.e. lowest LBD) for each different UIP. Finally, we learn all such lemmas (as many as different UIPs), backjump and propagate all the different UIPs, which might be set at different decision levels. Experiments revealed that in more than 50% of the benchmarks more than 5% of the conflicts had more than one UIP, the most common case being two UIPs. Hence the number of times where this technique can be applied is not massive but not negligible either.

Again, we ran the 400 benchmarks with 5 random seeds and a time limit of 5000 seconds, and the comparison between the solved instances of the unmodified CaDiCaL with chronological backtracking and the version described in this section is the following:

System	Seed 1	Seed 2	Seed 3	Seed 4	Seed 5	Median
Cadical-chronoBT	296	287	293	289	291	291
Multiple-conflicts-multiple-lemmas	295	294	292	291	290	292

The comparison between the two systems is again very similar, being the modified version slightly better. Perhaps the most surprising fact can be seen in Figure 6, where cactus plots similar to the ones in the previous section are presented. In this case, CaDiCaL solves 275 benchmarks in all 5 seeds, whereas the version learning multiple lemmas solves 285. This is a remarkable difference and suggests that the modification results in a more stable system. One likely reason is that by analyzing multiple conflicts, we can select several UIPs and the best lemma for each one of them. On the other hand, in the original version this choice is determined by the order in which clauses appear in the watch lists, which is totally arbitrary.

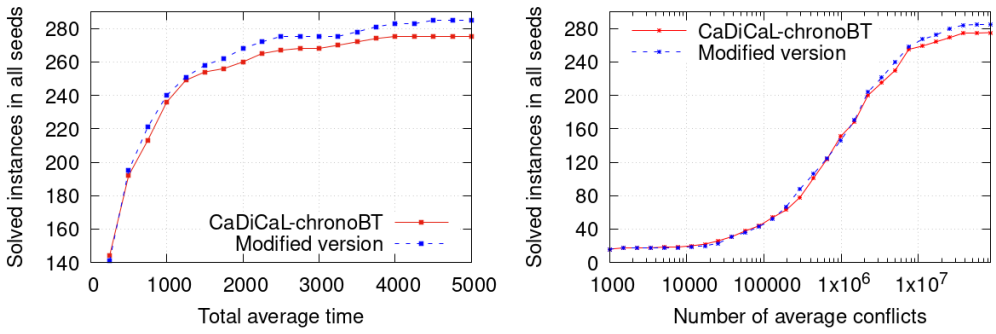

Figure 6: Solved instances within a given time or number of conflicts.

6 Conclusions and Future Work

We have presented a thorough analysis of the possible impact of analyzing multiple conflicts in SAT solvers. By performing a large number of controlled experiments, we have (i) learned that collecting and analyzing all conflicts only slightly improves the search space and (ii) understood the reasons for this limitation. Later we proved that these improvements can be translated to a more stable system by limiting the amount of extra work to be done and exploiting the benefits from chronological backtracking.

As future work, we might want to measure how often it happens that different lemmas are found, but they are equivalent modulo the binary implication graph, or subsumption relationships among them exist. This would shed more light on why the gain obtained by learning them is limited. Also, we might study other measures of quality to decide which lemmas are learned. In order to speed up performance, a parallel approach might be used to reduce the overhead of the additional computations. Also, trying to predict, using machine learning techniques, which conflicting clause will provide the best lemma would reduce the runtime. All these ideas should probably come together with mechanisms that allow one to detect on which instances the presented technique is useful, and enable or disable it accordingly.

References

[1] Gilles Audemard, Lucas Bordeaux, Youssef Hamadi, Saïd Jabbour, and Lakhdar Sais. A generalized framework for conflict analysis. In Hans Kleine Büning and Xishun Zhao, editors, *Theory and Applications of Satisfiability Testing - SAT 2008, 11th International Conference, SAT 2008, Guangzhou, China, May 12-15, 2008. Proceedings*, volume 4996 of *Lecture Notes in Computer Science*, pages 21–27. Springer, 2008.

[2] Gilles Audemard and Laurent Simon. Predicting learnt clauses quality in modern SAT solvers. In *Proceedings of the 21st international jont conference on Artifical intelligence*, IJCAI '09, pages 399–404, San Francisco, CA, USA, 2009. Morgan Kaufmann Publishers Inc.

[3] Tomas Balyo, Marijn J. H. Heule, Markus Iser, Matti Järvisalo, and Martin Suda. SAT Competition 2022. https://satcompetition.github.io/2022/.

[4] Armin Biere, Katalin Fazekas, Mathias Fleury, and Maximillian Heisinger. CaDiCaL, Kissat, Paracooba, Plingeling and Treengeling entering the SAT Competition 2020. In Tomas Balyo, Nils Froleyks, Marijn Heule, Markus Iser, Matti Järvisalo, and Martin Suda, editors, *Proc. of SAT Competition 2020 – Solver and Benchmark Descriptions*, volume B-2020-1 of *Department of Computer Science Report Series B*, pages 51–53. University of Helsinki, 2020.

[5] Armin Biere and Andreas Fröhlich. Evaluating CDCL variable scoring schemes. In Marijn Heule and Sean A. Weaver, editors, *Theory and Applications of Satisfiability Testing - SAT 2015 - 18th International Conference, Austin, TX, USA, September 24-27, 2015, Proceedings*, volume 9340 of *Lecture Notes in Computer Science*, pages 405–422. Springer, 2015.

[6] Armin Biere and Andreas Fröhlich. Evaluating CDCL variable scoring schemes. In Marijn Heule and Sean A. Weaver, editors, *Theory and Applications of Satisfiability Testing - SAT 2015 - 18th International Conference, Austin, TX, USA, September 24-27, 2015, Proceedings*, volume 9340 of *Lecture Notes in Computer Science*, pages 405–422. Springer, 2015.

[7] Armin Biere and Andreas Fröhlich. Evaluating CDCL restart schemes. In Daniel Le Berre and Matti Järvisalo, editors, *Proceedings of Pragmatics of SAT 2015, Austin, Texas, USA, September 23, 2015 / Pragmatics of SAT 2018, Oxford, UK, July 7, 2018*, volume 59 of *EPiC Series in Computing*, pages 1–17. EasyChair, 2018.

[8] Armin Biere, Matti Järvisalo, and Benjamin Kiesl. Preprocessing in SAT solving. In Armin Biere, Marijn Heule, Hans van Maaren, and Toby Walsh, editors, *Handbook of Satisfiability - Second*

Edition, volume 336 of *Frontiers in Artificial Intelligence and Applications*, pages 391–435. IOS Press, 2021.

[9] Armin Biere and Daniel Kröning. Sat-based model checking. In Edmund M. Clarke, Thomas A. Henzinger, Helmut Veith, and Roderick Bloem, editors, *Handbook of Model Checking*, pages 277–303. Springer, 2018.

[10] Sam Buss and Jakob Nordström. Proof complexity and SAT solving. In Armin Biere, Marijn Heule, Hans van Maaren, and Toby Walsh, editors, *Handbook of Satisfiability - Second Edition*, volume 336 of *Frontiers in Artificial Intelligence and Applications*, pages 233–350. IOS Press, 2021.

[11] Henrik Cao. Concurrent Multi-conflict Analysis in SAT Solvers, November 2022.

[12] Stephen A. Cook. The complexity of theorem-proving procedures. In *Proceedings of the third annual ACM symposium on Theory of computing*, STOC '71, pages 151–158, New York, NY, USA, 1971. ACM.

[13] Nick Feng and Fahiem Bacchus. Clause size reduction with all-uip learning. In Luca Pulina and Martina Seidl, editors, *Theory and Applications of Satisfiability Testing - SAT 2020 - 23rd International Conference, Alghero, Italy, July 3-10, 2020, Proceedings*, volume 12178 of *Lecture Notes in Computer Science*, pages 28–45. Springer, 2020.

[14] Mathias Fleury and Armin Biere. Efficient all-uip learned clause minimization. In Chu-Min Li and Felip Manyà, editors, *Theory and Applications of Satisfiability Testing - SAT 2021 - 24th International Conference, Barcelona, Spain, July 5-9, 2021, Proceedings*, volume 12831 of *Lecture Notes in Computer Science*, pages 171–187. Springer, 2021.

[15] Andrea Formisano and Flavio Vella. On multiple learning schemata in conflict driven solvers. In Stefano Bistarelli and Andrea Formisano, editors, *Proceedings of the 15th Italian Conference on Theoretical Computer Science, Perugia, Italy, September 17-19, 2014*, volume 1231 of *CEUR Workshop Proceedings*, pages 133–146. CEUR-WS.org, 2014.

[16] HyoJung Han and Fabio Somenzi. On-the-fly clause improvement. In Oliver Kullmann, editor, *Theory and Applications of Satisfiability Testing - SAT 2009, 12th International Conference, SAT 2009, Swansea, UK, June 30 - July 3, 2009. Proceedings*, volume 5584 of *Lecture Notes in Computer Science*, pages 209–222. Springer, 2009.

[17] Marijn J. H. Heule. Proofs of unsatisfiability. In Armin Biere, Marijn Heule, Hans van Maaren, and Toby Walsh, editors, *Handbook of Satisfiability - Second Edition*, volume 336 of *Frontiers in Artificial Intelligence and Applications*, pages 635–668. IOS Press, 2021.

[18] Marijn J. H. Heule, Oliver Kullmann, and Victor W. Marek. Solving and verifying the Boolean Pythagorean triples problem via cube-and-conquer. In Nadia Creignou and Daniel Le Berre, editors, *Theory and Applications of Satisfiability Testing - SAT 2016 - 19th International Conference, Bordeaux, France, July 5-8, 2016, Proceedings*, volume 9710 of *Lecture Notes in Computer Science*, pages 228–245. Springer, 2016.

[19] Yogesh S. Mahajan, Zhaohui Fu, and Sharad Malik. Zchaff2004: An efficient SAT solver. In Holger H. Hoos and David G. Mitchell, editors, *Theory and Applications of Satisfiability Testing, 7th International Conference, SAT 2004, Vancouver, BC, Canada, May 10-13, 2004, Revised Selected Papers*, volume 3542 of *Lecture Notes in Computer Science*, pages 360–375. Springer, 2004.

[20] Joao Marques-Silva. *Search algorithms for satisfiability problems in combinational switching circuits*. PhD thesis, University of Michigan, 1995.

[21] João Marques-Silva, Inês Lynce, and Sharad Malik. Conflict-driven clause learning SAT solvers. In Armin Biere, Marijn Heule, Hans van Maaren, and Toby Walsh, editors, *Handbook of Satisfiability - Second Edition*, volume 336 of *Frontiers in Artificial Intelligence and Applications*, pages 133–182. IOS Press, 2021.

[22] Sibylle Möhle and Armin Biere. Backing backtracking. In Mikolás Janota and Inês Lynce, editors, *Theory and Applications of Satisfiability Testing - SAT 2019 - 22nd International Conference, SAT 2019, Lisbon, Portugal, July 9-12, 2019, Proceedings*, volume 11628 of *Lecture Notes in Computer*

Science, pages 250–266. Springer, 2019.

[23] Matthew W. Moskewicz, Conor F. Madigan, Ying Zhao, Lintao Zhang, and Sharad Malik. Chaff: engineering an efficient SAT solver. In *Proceedings of the 38th annual Design Automation Conference*, DAC '01, pages 530–535, New York, NY, USA, 2001. ACM.

[24] Alexander Nadel and Vadim Ryvchin. Chronological backtracking. In Olaf Beyersdorff and Christoph M. Wintersteiger, editors, *Theory and Applications of Satisfiability Testing - SAT 2018 - 21st International Conference, SAT 2018, Held as Part of the Federated Logic Conference, FloC 2018, Oxford, UK, July 9-12, 2018, Proceedings*, volume 10929 of *Lecture Notes in Computer Science*, pages 111–121. Springer, 2018.

[25] Mukul R. Prasad, Armin Biere, and Aarti Gupta. A survey of recent advances in sat-based formal verification. *Int. J. Softw. Tools Technol. Transf.*, 7(2):156–173, 2005.

[26] João P. Marques Silva and Karem A. Sakallah. Conflict analysis in search algorithms for satisfiability. In *Eigth International Conference on Tools with Artificial Intelligence, ICTAI '96, Toulouse, France, November 16-19, 1996*, pages 467–469. IEEE Computer Society, 1996.

[27] João P. Marques Silva and Karem A. Sakallah. GRASP - a new search algorithm for satisfiability. In Rob A. Rutenbar and Ralph H. J. M. Otten, editors, *Proceedings of the 1996 IEEE/ACM International Conference on Computer-Aided Design, ICCAD 1996, San Jose, CA, USA, November 10-14, 1996*, pages 220–227. IEEE Computer Society / ACM, 1996.

[28] João P. Marques Silva and Karem A. Sakallah. Invited tutorial: Boolean satisfiability algorithms and applications in electronic design automation. In E. Allen Emerson and A. Prasad Sistla, editors, *Computer Aided Verification, 12th International Conference, CAV 2000, Chicago, IL, USA, July 15-19, 2000, Proceedings*, volume 1855 of *Lecture Notes in Computer Science*, page 3. Springer, 2000.

[29] Niklas Sörensson and Armin Biere. Minimizing learned clauses. In Oliver Kullmann, editor, *SAT*, volume 5584 of *Lecture Notes in Computer Science*, pages 237–243. Springer, 2009.

[30] L. Zhang, C. F. Madigan, M. W. Moskewicz, and S. Malik. Efficient Conflict Driven Learning in a Boolean Satisfiability Solver. In *International Conference on Computer-Aided Design, ICCAD '01*, pages 279–285, 2001.