The SAT Museum

Armin Biere[1,*], Mathias Fleury[1], Nils Froleyks[2] and Marijn J.H. Heule[3]

[1]University of Freiburg, Germany
[2]Johannes Kepler University Linz, Austria
[3]Carnegie Mellon University, Pittsburgh, PA, USA

Abstract
The virtual SAT Solver Museum is an effort towards preserving historical SAT solvers, by collecting and porting their source code to modern compilers and evaluating them on representative benchmark sets on the same hardware. This allows us to compare historic and modern solvers in the same environment. Our results clearly show a remarkable improvement of SAT solver performance in the last 30 years.

Keywords
Satisfiability, SAT Solvers, SAT Competition

1. Introduction

It has been stated that "No major performance breakthrough [happened in SAT solving] in close to two decades". Notable proponents of this claim are Karem Sakallah at the recent Simons Institute's seminar in 2023 and Joao Marques-Silva during his invited talk at POS 2019 [1, Slide 12 (or 53 of the total number)]. The SAT Museum exists to document the history of SAT solving and to show in contrast to these claims that indeed "SAT solvers are getting faster and faster".

The SAT Museum is curated by two authors of this paper; Armin Biere and Marijn Heule have put considerable effort into collecting and restoring the SAT solvers that have been published since the first SAT competitions more than two decades ago. Some results of this effort have been presented as a lightening talk at POS'20, as well as in the form of a (comparatively) high-impact tweet with preliminary plots for the SAT Competition 2020 benchmarks on Twitter.

Even though a first SAT competition was conducted more than 3 decades ago in 1992 [2], the current regular series of annual SAT competitions was started in 2002 [3] by Laurent Simon and Daniel Le Berre and in most years attracts dozens of SAT solver submissions. The SAT competition provides a fair environment where solvers compete on the same benchmarks and hardware. These competitions have been credited as a main driving force in advancing SAT-solving technology and are a well-recognized show-case with high visibility and impact far beyond the core SAT community.

Each year, the benchmark suite consists of a combination of old and new benchmarks. In recent years, at least 75% of the benchmarks were new, and no more than 14 out of 400 originated

14th International Workshop on Pragmatics of SAT (PoS 2023)
*Corresponding author.
0000-0001-7170-9242 (A. Biere); 0000-0002-1705-3083 (M. Fleury); 0000-0003-3925-3438 (N. Froleyks); 0000-0002-5587-8801 (M. J.H. Heule)

from the same research group to ensure diversity. For more information on the competition we refer to the yearly SAT competition proceedings, e.g., the SAT Competition 2022 proceedings [4], or to the last article describing the SAT Competition in 2020 [5].

To asses the progress in solver performance, we consider all winning solvers since the SAT Competition 2002, whose code we could find on the SAT Competition website or obtain through personal communication. Note, that there was no requirement to publish source code nor even binaries in earlier incarnations of the SAT Competition (Section 3). Besides providing data on competition winners we also include two historically important solvers: Boehm1, the winner of the first SAT competition in 1992 [2], as well as Grasp [6] from 1997.

We run on the same hardware (from 2016) all collected and patched solvers on six benchmark sets from SAT competitions spanning more than two decades, namely 2002, 2011, 2019, 2020, 2021, and 2022. We report results on each set separately, in order to address an argument brought forward by Laurent Simon at a recent POS workshop, that the benchmark selection method of more recent competitions might give a bias towards newer solvers and which arguably might not be observable on the SAT Competition 2011 benchmark set for example. Our data on the SAT Competition 2011 benchmark set refutes this argument as it clearly shows the same solver progress which we observed in other years.

While in general we see a big improvement in solver performance in these 30 years across all considered benchmark sets, the yearly improvement is mostly rather slow, except for performance jumps in some years, which arguably happen with a frequency of 3 to 5 years. Analyzing the reasons for this apparent progress, i.e., both with respect to algorithms, heuristics and implementation, and in particular distilling the core ideas leading to these performance jumps is considered an important follow-up work but out of the scope of this first study.

2. Preliminaries

For the sake of understanding this paper, no special knowledge of SAT is required and we refer to the *Handbook of Satisfiability* [7] for more details. In short, CDCL [8] and its predecessor DPLL [9] work on a partial assignment trying to satisfy a set of clauses. When a *conflict* (mismatch between the assignment and the constraints) arises, the partial model is adapted and CDCL learns new clauses to prevent the same conflict in the future.

On top of CDCL or DPLL, the set of clauses can be simplified by transforming the problem more significantly. In earlier solvers, these techniques were employed as *preprocessing* before running CDCL, whereas nowadays they are run interleaved with CDCL as *inprocessing*. We refer to the corresponding preprocessing chapter [10] in the SAT handbook for details.

In general, this work considers SAT solvers (some of them developed by the first author), which are run on problems from the SAT Competition (last two authors were frequently part of the committee running it and selecting benchmarks).

Therefore, a major thread to validity of this work, as noted by one reviewer, is that its authors are all stake holders in the SAT Competition, either as participants or as organizers. Showing newer solvers to be better clearly serves their interest to support the competition and how its artifacts are used in the scientific discourse on SAT. Nevertheless, we argue, that our carefully executed and extensive experiments are convincing and allow to reach the favorable conclusion,

that SAT solvers are getting faster and faster.

The second major thread to validity is related to the fact that new solvers during development are trained on at-that-time current set of benchmarks: To join the competition, developers check that new technique work on previous competition benchmarks. For example, Kissat-mab-hywalk-2022, the winner of 2022, is based on the 2021 winner (which in turn is based on the 2020 winner). And unsurprisingly, it performs better on the 2021 benchmarks than the 2021 winner. However, none of them has seen the 2022 benchmarks. It is also unlikely that it was trained to perform well on the 2002 benchmarks. To counteract this potential thread to validity, we have used a large benchmark, spanning more than two decades of competitions.

3. The Solvers

In this section, we list all tested solvers attempting to highlight some of their contributions. We selected most SAT Competition winners and some others for their historical significance.

Before Preprocessing. In context of the SAT solver Grasp [6] CDCL was proposed, even though the term CDCL was only later introduced [11]. The decision heuristic at that time attempted to satisfy as many clauses as possible and is considered to be very costly to compute in each search node. Improving such decisions heuristics was also the main topic for the DPLL and thus pre-CDCL SAT solvers participating in the first SAT competition in 1992, from which we include the winning solver Boehm1 [2]. The next historically most significant SAT solver is Chaff [12]. It introduced various techniques that are now commonly used in all SAT solvers, such as watched literals for efficient propagation and the VSIDS decision heuristic to quickly find good decisions. We also consider its 2004 variant [13].

In 2002, the solver Limmat [14] won the competition (by one instance in a tie-breaking round). It follows the ideas of zChaff (at a time when the source code was not available). In 2003 the Siege SAT solver [11] finished 3rd in the SAT Competition (but run hors concours). Its main features are *blocking literals* and the *variable-move-to-front* (VMTF) decision heuristic.

The SAT solver Berkmin [15] improved the Chaff bumping heuristics by more explicitly picking literals in recently learned clauses and also taking into account literals that appear during the conflict analysis and not only those appearing in the final conflict clause. Around this time, restarts were still mostly random. In 2003 the SAT solver MiniSat [16] appeared[1], introducing essential algorithmic and implementation optimizations, including learned clause minimization, exponential VSIDS, and lazy priority queue updates. It won for the first time in 2006. The solver is further considered an attempt at providing clean code by removing redundant features present in other SAT solvers.

CDCL and Preprocessing. In 2005 actually SatElite-GTI, a combination of the SatElite preprocessor [17] with MiniSat as back-end solver, won the competition by a big margin, i.e., contributing to one of those performance jumps we will see. In that year, MiniSat 2005 was also awarded, but it lost to the combination with SatElite. After this success many winning solvers followed this recipe and included SatElite as preprocessor, until in 2008 MiniSat's version 2.0

[1]This seminal paper received the first test-of-time award of the SAT conference in 2022.

won the competition. It is the first MiniSat version which combines CDCL with preprocessing in one code base and executable.

In 2006 MiniSat dominated the (first) SAT Race 2006 and in 2007 the idea of rapid restarts and phase saving helped the Rsat solver to win the SAT Competition 2007. This technique afterwards became standard in all solvers. Also on the CDCL side the fruits of using the glue (LBD) metric [18] of learned clauses to improve reduction of the learned clause data base as well as improved dynamic restart schemes let the Glucose solver [18] win in 2011 and 2012.

The Glucose solver accordingly formed the basis of the development of the MapleSAT solver series winning the competition three times in a row from 2016-2018. In 2016 it introduced the idea of interleaving different policies for SAT (fewer restarts / longer assignments) and UNSAT (more restart / short assignments) proposed by Chanseok Oh [19], contradicting earlier intuitions that restarts mostly help solvers to avoid heavy-tail phenomenon [20] in satisfiable formulas. The solver further included the new LRB decision heuristic and recursive reason side bumping [21] in 2016 [22].

In 2017 vivification [23] was incorporated into MapleSAT in the form of simplifying (aka "inprocessing" - see below) of learned clauses, while before vivification was only applied to original / irredundant clauses during preprocessing. In 2018 the next variant of MapleSAT won the competition, again extended by a different set of authors, by switching between the default CDCL version of non-chronological backtracking and chronological backtracking [24, 25]. In the SAT Race 2019 MapleSAT was again successful by filtering out redundant learned clauses through hashing [26] and enforcing deterministic switches between LRB and VSIDS [27].

Inprocessing Solvers. While the earlier listed solvers did not perform any global transformation on the formula or only do so at the beginning, a different line of work is to include techniques such as probing, subsumption, and blocked clauses during search.

In 2009, the winning SAT solver PrecoSAT [28] implemented this form of formula simplification during search as the first of its kind. This would later be called *inprocessing* [29]. While inprocessing can improve performance, when and for how long to schedule and preempt various inprocessing algorithms becomes both important and difficult to get right.

In 2010, CryptoMiniSat [30] won the competition. It is mainly known for its special handling of XOR clauses (parity or equivalence constraints) which are featured prominently but actually were never used as CryptoMiniSat 2010 could not recover XORs with more than 2 inputs from the CNF. Beyond that CryptoMiniSat features probing and hyper binary resolution in an inprocessing fashion. Initially based on MiniSat, development is still continuing today.

The solver Lingeling (winning 2013 [31] and 2014 [32]) utilized advanced inprocessing techniques, including equivalence reasoning and blocked clause elimination. These developments were enabled by the proper theoretical foundations for model reconstruction [29]. In 2015, the SAT solver abcdSAT [33] won the SAT Race 2014. It uses Lingeling as a preprocessor and Glucose as main solver and featured a new strategy to keep recently used clauses.

Finally, in 2020, the SAT solver Kissat [34] was introduced. Compared to its predecessor CaDiCaL [35], Kissat has fewer features, particularly inprocessors, which however are scheduled more aggressively. While the performance improvement in 2020 is usually attributed to the inclusion of a local search solver and target phases [36], it is worth noting that CaDiCaL was

actually the first solver to explore this. Successors of Kissat won the following two years; in 2021, a version with a more stable decision heuristic [37], and in 2022, an implementation featuring aggressive random walks [38].

4. Collecting and Porting Solvers

In our view the most important outcome of this study is to collect solvers which either in the competition or otherwise are important to the history of SAT solving. Furthermore we ported these legacy solvers to modern compilers. This finally also allowed us to run and compare them, also with more recent solvers, in a clean apple-to-apple comparison on the same hardware and on the same set of benchmarks to asses the progress in the last quarter of a century.

We started this endeavor in 2019, probably right on time, as collecting original solvers is becoming harder than we imagined. Most of the historic solvers still have webpages but links to actual code are dysfunctional. Some webpages disappeared completely. In these cases we reached out to the original authors, which dug through their old computers or found other ways to help us out to retrieve source code or binaries (see acknowledgments at the end).

On top of collecting original source code and binaries, we also provide patches to several legacy solvers, which allow us to compile them with modern compilers (we used gcc/g++ 9.4.0 for our experiments). Most of these issues were due to the g++ compiler becoming over the years more picky about what C++ constructs are accepted. Besides removing some warnings in theses patches, they also contain several fixes addressing bugs of some solvers, which led to incorrect results, but which after debugging were only due to hard coded limits in parsers or in the case of zChaff due to the code not being 64-bit clean. Patches will have to be updated for newer version of the gcc/g++, similar to the restoration process in an ordinary museum.

Besides porting solvers, we also fixed the parser of the solver Boehm1 to support DIMACS and to parse more than 1 000 variables. The implementation of the solver is actually recursive. During experiments we considered increasing the stack size to reduce the number of errors, but finally decided against this option and kept the default stack size (of 8 MB). In the end no solver run showed any discrepancy on the 6 competition benchmark sets we used in the experiments.

5. Performance Results

We ran all the benchmarks on 8-core Intel Xeon E5-2620 v4 CPUs running at 2.10 GHz (turbo-mode disabled) with a memory limit of 127 GB and a time limit of 5 000 seconds as in recent SAT competitions even though on slightly slower hardware. All data, including solvers, patches, log files and plots, is available at https://cca.informatik.uni-freiburg.de/satmuseum and [39].

Regarding results we want to stress again that SAT solver developers train on previous competitions: The winner of the 2022 competition is based on the winner from 2021, had access and most likely has been trained on the 2021 benchmarks to find better heuristics than its predecessor from 2021, which has not been trained on them nor on more recent problems sets.

We use CDFs (cumulative distribution function) and not cactus plots: the higher the solver the more problems solved and the more to the left the faster the solver. Our results are shown in Fig. 1 for the SAT Competition 2002, in Fig. 2 for the SAT Competition 2011, in Fig. 3 for the

SAT Competition 2019, in Fig. 4 for the SAT Competition 2020, in Fig. 5 for the SAT Competition 2021, and in Fig. 6 for the SAT Competition 2022. The conclusion is clear: consistently across all benchmarks of several years, recent solvers are better than old solvers with some minor variability. More recent solvers do solve more instances.

Nevertheless, there seems to be several major performance jumps in the reported results: one when preprocessing was introduced around 2006 and a second inconsistent one: 2019 on 2021/2022 benchmarks and 2016 on older benchmark sets with a minor improvement in 2020. The second jump can be attributed to a combination of local search (after 2019 in CaDiCaL and Kissat), rephasing (after 2016), and more aggressive bounded variable elimination.

The behavior of solvers varies across benchmarks. The SAT solver maple-compsps-drup is a striking example: after 2 500 seconds it changes the heuristics to switch to VSIDS (later variants would change on a more regular and deterministic interval). In 2002 the effect is quite strong and it solves many instances in a very short time (while still performing worse than Lingeling 2013). This effect is less pronounced in 2011 and barely visible later.

In Figure 7, we visualize the pairwise similarity between solvers. The highest similarity is observed between the two most recent solvers, Kissat (2021) and Kissat (2022), while the lowest similarity is observed between these two solvers and the oldest solvers, Boehm1 (1992) and Grasp (1997). The dendrogram above the heat-map depicts a hierarchical clustering automatically generated based on solver similarity. Remarkably, it very closely aligns with release years.

Generally, solvers developed after the introduction of a particular technique utilize that technique and gain an advantage on the same benchmarks. This is most evident with the three main clusters: the first groups solvers before introduction of preprocessing, and the other two clusters group solvers before and after 2016. In that year, a new heuristic was introduced and specialized phases targeting SAT and UNSAT problems became popular. The split between the early preprocessing solvers and the inprocessing solvers after 2009 is also easily discernible.

Related Work. In this work, we compare SAT solvers on SAT competition instances. In related work Dutertre compared various SAT solvers from 2019 (without restricting to competition winners) and MiniSat but on SMT bitvector benchmarks [40]. He observed improvements over MiniSat but the ranking did not reflect the results from the SAT Competition – the second best SAT solver finished 7th. He also tested various features in CaDiCaL and tested on problems hard for SAT solvers not requiring extensive theory reasoning. Recent work by Fazekas [41] on simplifying the interface between SAT and SMT solvers reaches similar conclusions.

In another related work by Kochemazov, Ignatiev, and Marques-Silva [42] the focus was on *incremental* SAT solving. They compared competition winners between 2016 and 2020 and MiniSat, but they did not observe an improvement over MiniSat on MaxSAT instances, except for two families. However, in recent work on incremental use of SAT solvers for backbone extraction [43] the more recent solver CaDiCaL surpasses MiniSat by a large margin.

The SAT heritage effort by Audemard, Paulevé, and Simon [44] also tries to preserve and enable to run historic SAT solvers. Their approach is based on system-level virtualization with docker containers and thus orthogonal to ours by compiling the original source code with historic compilers. They do not attempt to port and patch solvers, which means a comparison such as ours on 6 sets of competition benchmark sets will result in many discrepancies, particularly for

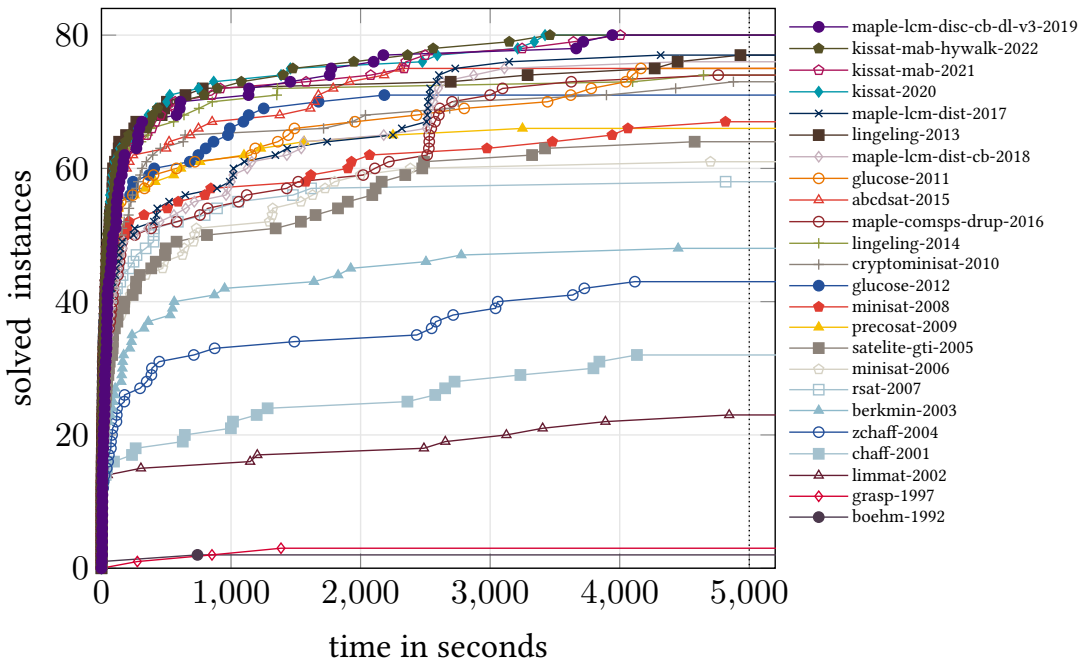

Figure 1: All time winners on the SAT Competition 2002 benchmarks (100 problems)

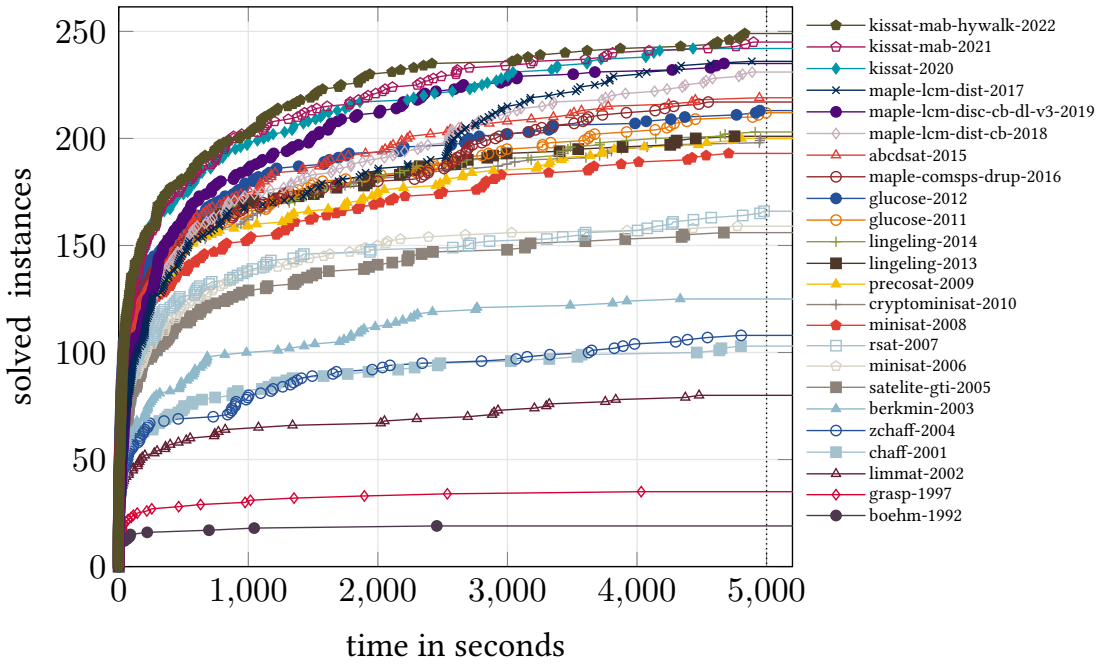

Figure 2: All time winners on the SAT Competition 2011 benchmarks (300 problems)

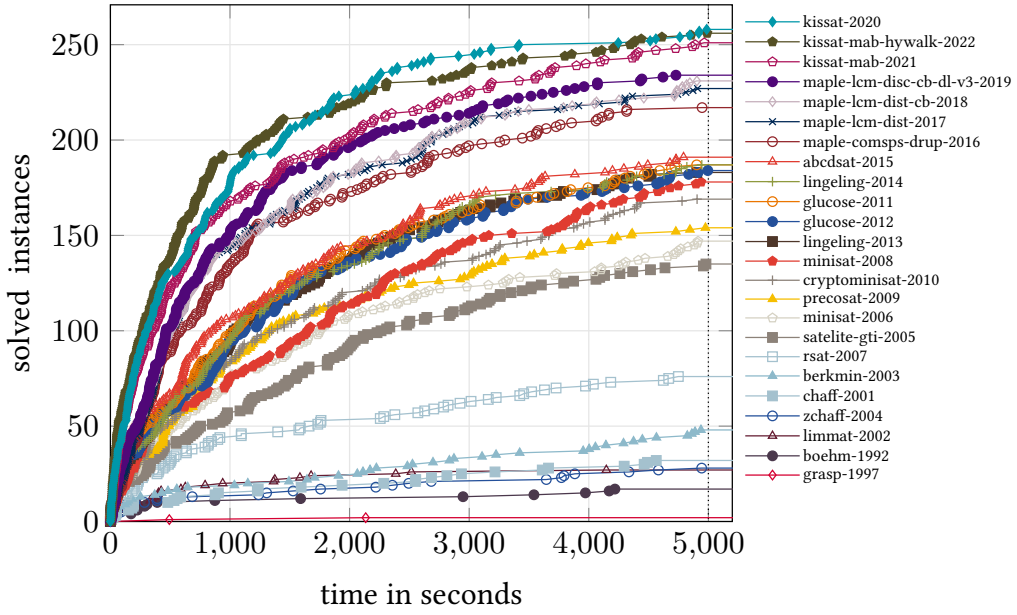

Figure 3: All time winners on the SAT Competition 2019 benchmarks (400 problems)

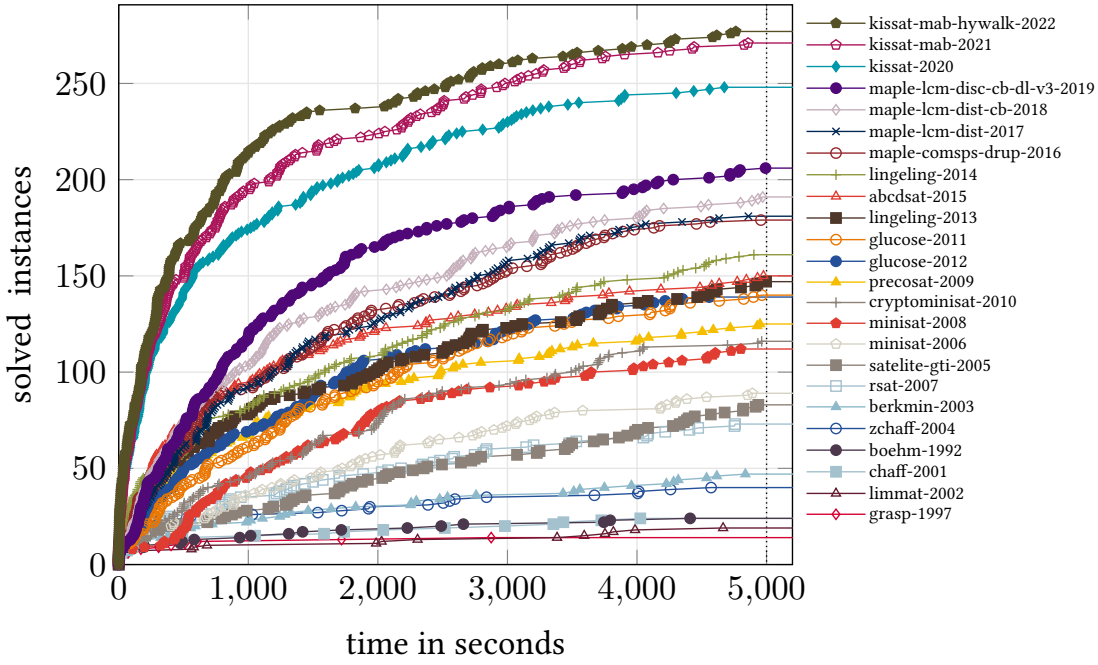

Figure 4: All time winners on the SAT Competition 2020 benchmarks (400 problems)

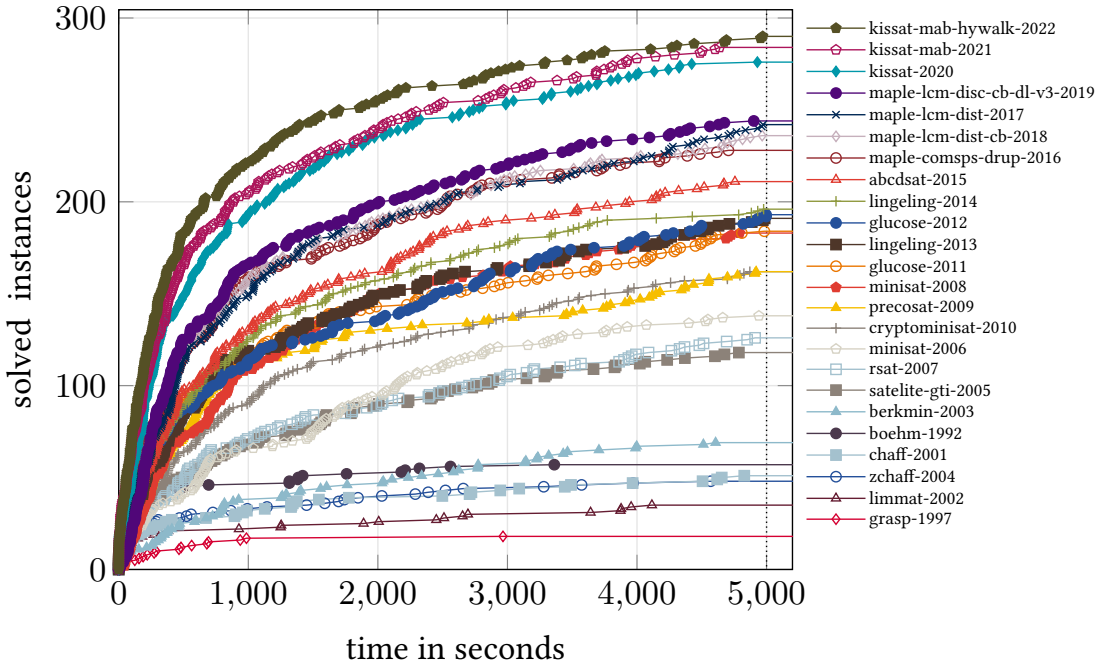

Figure 5: All time winners on the SAT Competition 2021 benchmarks (400 problems)

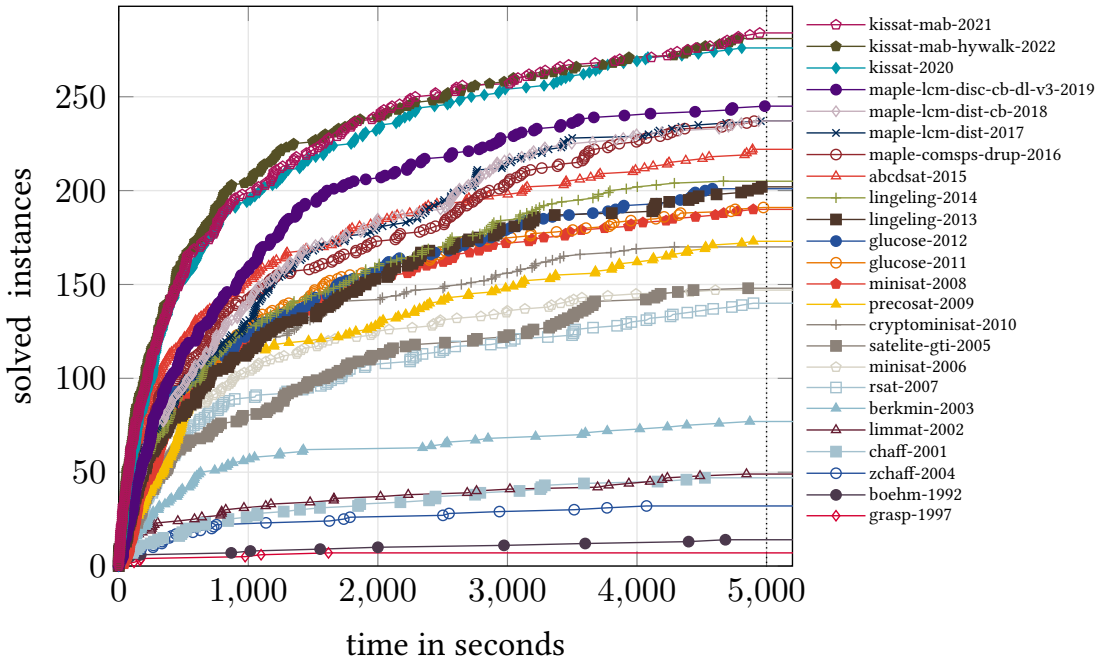

Figure 6: All time winners on the SAT Competition 2022 benchmarks (400 problems)

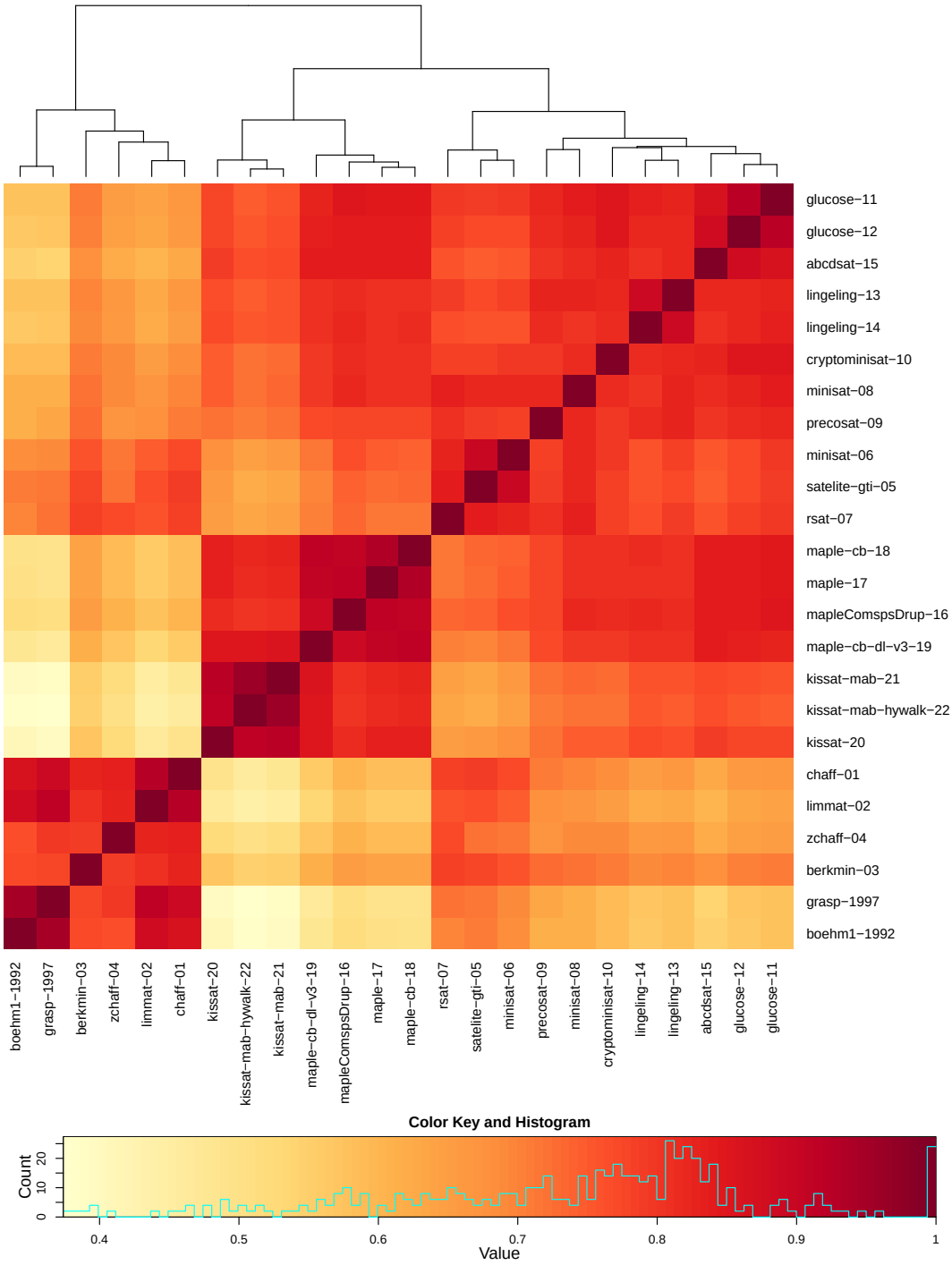

Figure 7: Heat-map and dendrogram (top) based on runtime similarity. The similarity between two solvers is defined by comparing the solving time they achieve on each of the 2000 benchmarks over the years. If a solver failed to solve an instance, we assign twice the timeout value (10 000). The absolute difference in solving time is then accumulated and normalized to the interval $[0, 1]$, where 1 indicates identical performance ($1 - \sum |t_i - t'_i| \ / \ 2000 \cdot 10\,000$). Darker regions indicate higher similarity between solvers. A more precise relation between color and similarity-value together with a histogram of the values that appear is given at the bottom. Above the heat-map we illustrates a hierarchical clustering, with solvers or clusters with high similarity join lower in the dendrogram, while clusters with significantly different performance are joined higher.

older solvers, and thus render it meaningless from the perspective of comparing performance. Our approach will likely need additional patches for newer compilers in the future though. However it is unclear whether container virtualization can survive decades without maintenance.

Finally, an on-first-sight related but in our view bogus experiment was conducted in 2020 by Fichte, Hecher, and Szeider [45]. It only focused on a small rather uncommon benchmark set [46] of 202 benchmarks as well as on a small set of solvers. The key feature of their set-up was to run unmodified legacy solver code on old legacy hardware (from 1999) as well as modern hardware (from 2019), with the goal to compare SAT solver progress due to algorithms / software (team SW) versus progress due to hardware improvements (team HW).

However, we argue that this goal was not reached, as the experiment ignored apparent discrepancies: If we take for example the problem `AProVE07-04.cnf` from the SAT Competition 2012, zChaff claims that this problem is SAT (without having made any decision) within 0.1 s, while all other solvers report UNSAT (as expected). We further observed 7 discrepancies for team SW, on the old Sparc architecture and 9 for team HW on the new architecture, not including problems solved only by zChaff. This actually changes the result substantially and makes the team SW look much better than what was reported in [45]. Amazingly, the solver Grasp also (incorrectly) claims that `AProVE07-04.cnf` is satisfiable. After pointing out these issues to the authors they promised to address these problems in an extended version of their paper.

6. Conclusion

In this work we have collected and fixed the source code of winning SAT solvers from the SAT competitions and compared their performance on many benchmarks. Overall more recent solvers solve more problems, rather consistently. Thus SAT solvers get faster and faster. We are looking forward to continue this preservation effort and performance evaluation on additional older and future solvers as well as benchmark sets.

Clearly, our presented results disprove the false view discussed in the introduction that there was no major progress in SAT solving in the last 20 years. Still, one nagging remaining issue with our work is that we do not provide a deeper understanding about the differences between solvers and whether all implemented techniques are useful. Are there some old techniques not part of modern SAT solvers which are still useful? And most important, can we produce even better SAT solvers by understanding this remarkable progress better?

Acknowledgments

This work is supported by the Austrian Science Fund (FWF), NFN S11408-N23 (RiSE), the LIT AI Lab funded by the State of Upper Austria, and the National Science Foundation under grant CCF-2229099. We further like to thank Daniel Le Berre for the original idea of conducting this study and also coming up with the name “SAT Museum” and are also grateful to the anonymous reviewers for their positive and very encouraging feedback.

We are also in debt to the colleagues who helped us to resurrect the source code or binaries of some of the discussed historic solvers, including Jingchao Chen (for the binary abcdSAT 2015 as the competition did not keep binaries), Eugene Goldberg and Yakov Novikov (for helping us

to retrieve berkmin 2003). Lintao Zhang, Matthew Moskewicz and Sharad Malik (for helping us to revive Chaff2), Hans Kleine Büning and Theodor Lettmann (for forwarding source code of the winning solver Boehm1 in 1992 by Max Böhm) and last but not least João Marques-Silva for providing a patched version of Grasp.

Finally we want to thank the authors of all the other solvers, the participants and organizers of past SAT competitions. Without their enthusiasm in supporting the competition and the community this study would not exist. Last but not least we want to thank Karem Sakallah for his infinite patience in related discussions on the progress in SAT solving and the participants of past editions of the POS workshop.

References

[1] J. P. Marques-Silva, SAT: Disruption, demise & resurgence, 2019. URL: http://www.pragmaticsofssat.org/2019/disruption.pdf, invited talk.

[2] M. Buro, H. Kleine Büning, Report on a SAT competition, Fachbereich Math.-Informatik, Univ. Gesamthochschule Paderborn, Germany, 1992.

[3] L. Simon, D. L. Berre, E. A. Hirsch, The SAT2002 competition, Ann. Math. Artif. Intell. 43 (2005) 307–342. doi:10.1007/s10472-005-0424-6.

[4] T. Balyo, M. J. H. Heule, M. Iser, M. Järvisalo, M. Suda (Eds.), Proc. of SAT Competition 2022 – Solver and Benchmark Descriptions, volume B-2022-1 of *Department of Computer Science Series of Publications B*, University of Helsinki, 2022.

[5] N. Froleyks, M. J. H. Heule, M. Iser, M. Järvisalo, M. Suda, Sat competition 2020, Artificial Intelligence 301 (2021) 103572.

[6] J. P. Marques-Silva, K. A. Sakallah, GRASP - a new search algorithm for satisfiability, in: R. A. Rutenbar, R. H. J. M. Otten (Eds.), Proceedings of the 1996 IEEE/ACM International Conference on Computer-Aided Design, ICCAD 1996, San Jose, CA, USA, November 10-14, 1996, IEEE Computer Society / ACM, 1996, pp. 220–227. doi:10.1109/ICCAD.1996.569607.

[7] A. Biere, M. J. H. Heule, H. van Maaren, T. Walsh (Eds.), Handbook of Satisfiability, volume 336 of *Frontiers in Artificial Intelligence and Applications*, 2nd ed., IOS Press, 2021. URL: https://doi.org/10.3233/FAIA336. doi:10.3233/FAIA336.

[8] J. P. Marques-Silva, I. Lynce, S. Malik, Conflict-driven clause learning SAT solvers, in: A. Biere, M. J. H. Heule, H. van Maaren, T. Walsh (Eds.), Handbook of Satisfiability, volume 336 of *Frontiers in Artificial Intelligence and Applications*, 2nd ed., IOS Press, 2021, pp. 133–182. doi:10.3233/FAIA200987.

[9] M. Davis, G. Logemann, D. W. Loveland, A machine program for theorem-proving, Commun. ACM 5 (1962) 394–397.

[10] A. Biere, M. Järvisalo, B. Kiesl, Preprocessing in SAT solving, in: A. Biere, M. J. H. Heule, H. van Maaren, T. Walsh (Eds.), Handbook of Satisfiability, volume 336 of *Frontiers in Artificial Intelligence and Applications*, 2nd ed., IOS Press, 2021, pp. 391 – 435.

[11] L. Ryan, Efficient algorithms for clause-learning SAT solvers, Master's thesis, Simon Fraser Univ., 2004. URL: https://www.cs.sfu.ca/~mitchell/papers/ryan-thesis.ps.

[12] M. W. Moskewicz, C. F. Madigan, Y. Zhao, L. Zhang, S. Malik, Chaff: Engineering an

efficient SAT solver, in: Proceedings of the 38th Design Automation Conference, DAC 2001, Las Vegas, NV, USA, June 18-22, 2001, ACM, 2001, pp. 530–535. doi:10.1145/378239.379017.

[13] Y. S. Mahajan, Z. Fu, S. Malik, Zchaff 2004: An efficient SAT solver, in: H. H. Hoos, D. G. Mitchell (Eds.), 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*, Springer, 2004, pp. 360–375. doi:10.1007/11527695_27.

[14] A. Biere, The Evolution from LIMMAT to NANOSAT, Report, ETH Zürich, 2004-04. doi:10.3929/ethz-a-006744011, technical Reports D-INFK.

[15] E. I. Goldberg, Y. Novikov, Berkmin: A fast and robust SAT-solver, in: 2002 Design, Automation and Test in Europe Conference and Exposition (DATE 2002), 4-8 March 2002, Paris, France, IEEE Computer Society, 2002, pp. 142–149. doi:10.1109/DATE.2002.998262.

[16] N. Eén, N. Sörensson, An extensible SAT-solver, in: E. Giunchiglia, A. Tacchella (Eds.), Theory and Applications of Satisfiability Testing, 6th International Conference, SAT 2003. Santa Margherita Ligure, Italy, May 5-8, 2003 Selected Revised Papers, volume 2919 of *Lecture Notes in Computer Science*, Springer, 2003, pp. 502–518. doi:10.1007/978-3-540-24605-3_37.

[17] N. Eén, A. Biere, Effective preprocessing in SAT through variable and clause elimination, in: F. Bacchus, T. Walsh (Eds.), Theory and Applications of Satisfiability Testing, 8th International Conference, SAT 2005, St. Andrews, UK, June 19-23, 2005, Proceedings, volume 3569 of *Lecture Notes in Computer Science*, Springer, 2005, pp. 61–75. doi:10.1007/11499107_5.

[18] G. Audemard, L. Simon, Predicting learnt clauses quality in modern SAT solvers, in: C. Boutilier (Ed.), IJCAI 2009, Proceedings of the 21st International Joint Conference on Artificial Intelligence, Pasadena, California, USA, July 11-17, 2009, 2009, pp. 399–404.

[19] C. Oh, Between SAT and UNSAT: the fundamental difference in CDCL SAT, in: Theory and Applications of Satisfiability Testing–SAT 2015–18th International Conference, Austin, TX, USA, September 24-27, 2015, Proceedings, 2015, pp. 307–323. doi:10.1007/978-3-319-24318-4_23.

[20] C. P. Gomes, B. Selman, N. Crato, H. A. Kautz, Heavy-tailed phenomena in satisfiability and constraint satisfaction problems, J. Autom. Reason. 24 (2000) 67–100. doi:10.1023/A:1006314320276.

[21] J. H. Liang, V. Ganesh, P. Poupart, K. Czarnecki, Learning rate based branching heuristic for SAT solvers, in: N. Creignou, D. L. Berre (Eds.), 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*, Springer, 2016, pp. 123–140. doi:10.1007/978-3-319-40970-2_9.

[22] C. Oh, COMiniSatPS the chandrasekhar limit and GHackCOMSPS, in: T. Balyo, N. Froleyks, M. J. H. Heule, M. Iser, M. Järvisalo, M. Suda (Eds.), Proc. of SAT Competition 2016 – Solver and Benchmark Descriptions, volume B-2016-1 of *Department of Computer Science Report Series B*, University of Helsinki, 2016, p. 29.

[23] M. Luo, C. Li, F. Xiao, F. Manyà, Z. Lü, An effective learnt clause minimization approach

for CDCL SAT solvers, in: C. Sierra (Ed.), Proceedings of the 26th International Joint Conference on Artificial Intelligence, IJCAI 2017, Melbourne, Australia, August 19-25, 2017, ijcai.org, 2017, pp. 703–711. doi:10.24963/ijcai.2017/98.

[24] A. Nadel, V. Ryvchin, Chronological backtracking, in: O. Beyersdorff, C. M. Wintersteiger (Eds.), 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*, Springer, 2018, pp. 111–121. doi:10.1007/978-3-319-94144-8_7.

[25] V. Ryvchin, A. Nadel, Maple_LCM_Dist_ChronoBT: Featuring chronological backtracking, in: T. Balyo, N. Froleyks, M. J. H. Heule, M. Iser, M. Järvisalo, M. Suda (Eds.), Proc. of SAT Competition 2018 – Solver and Benchmark Descriptions, volume B-2018-1 of *Department of Computer Science Report Series B*, University of Helsinki, 2018, p. 29.

[26] S. Kochemazov, O. Zaikin, A. A. Semenov, V. Kondratiev, Speeding up CDCL inference with duplicate learnt clauses, in: G. D. Giacomo, A. Catalá, B. Dilkina, M. Milano, S. Barro, A. Bugarín, J. Lang (Eds.), ECAI 2020 - 24th European Conference on Artificial Intelligence, 29 August-8 September 2020, Santiago de Compostela, Spain, August 29 - September 8, 2020 - Including 10th Conference on Prestigious Applications of Artificial Intelligence (PAIS 2020), volume 325 of *Frontiers in Artificial Intelligence and Applications*, IOS Press, 2020, pp. 339–346. doi:10.3233/FAIA200111.

[27] S. Kochemazov, O. Zaikin, V. Kondratiev, A. Semenov, MapleLCMDistChronoBT-DL, duplicate learnts heuristic-aided solvers at the SAT Race 2019, in: T. Balyo, N. Froleyks, M. J. H. Heule, M. Iser, M. Järvisalo, M. Suda (Eds.), Proc. of SAT Race 2019 – Solver and Benchmark Descriptions, volume B-2019-1 of *Department of Computer Science Report Series B*, University of Helsinki, 2019, p. 24.

[28] A. Biere, Lingeling, Plingeling, PicoSAT and PrecoSAT at SAT Race 2010, Technical Report 10/1, Johannes Kepler University Linz, FMV Reports Series, Institute for Formal Models and Verification, Johannes Kepler University, Altenbergerstr. 69, 4040 Linz, Austria, 2010. doi:10.350/fmvtr.2010-1.

[29] M. Järvisalo, M. J. H. Heule, A. Biere, Inprocessing rules, in: B. Gramlich, D. Miller, U. Sattler (Eds.), Automated Reasoning - 6th International Joint Conference, IJCAR 2012, Manchester, UK, June 26-29, 2012. Proceedings, volume 7364 of *Lecture Notes in Computer Science*, Springer, 2012, pp. 355–370. doi:10.1007/978-3-642-31365-3_28.

[30] M. Soos, K. Nohl, C. Castelluccia, Extending SAT solvers to cryptographic problems, in: O. Kullmann (Ed.), 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*, Springer, 2009, pp. 244–257. doi:10.1007/978-3-642-02777-2_24.

[31] A. Biere, Lingeling, Plingeling and Treengeling entering the SAT Competition 2013, in: A. Balint, A. Belov, M. J. H. Heule, M. Järvisalo (Eds.), Proc. of SAT Competition 2014 – Solver and Benchmark Descriptions, volume B-2013-1 of *Department of Computer Science Series of Publications B*, University of Helsinki, 2013, pp. 51–51.

[32] A. Biere, Yet another local search solver and Lingeling and friends entering the SAT Competition 2014, in: A. Balint, A. Belov, M. J. H. Heule, M. Järvisalo (Eds.), Proc. of SAT Competition 2014 – Solver and Benchmark Descriptions, volume B-2014-2 of *Department*

of Computer Science Series of Publications B, University of Helsinki, 2014, pp. 39–40.

[33] J. Chen, Minisat_bcd and abcdsat: Solvers based on blocked clause decom- position, in: A. Balint, A. Belov, M. J. H. Heule, M. Järvisalo (Eds.), Proc. of SAT Competition 2015 – Solver and Benchmark Descriptions, volume B-2015-1 of *Department of Computer Science Series of Publications B*, University of Helsinki, 2015, pp. 51–51.

[34] A. Biere, K. Fazekas, M. Fleury, M. Heisinger, CaDiCaL, Kissat, Paracooba, Plingeling and Treengeling entering the SAT Competition 2020, in: T. Balyo, N. Froleyks, M. J. H. Heule, M. Iser, M. Järvisalo, M. Suda (Eds.), Proc. of SAT Competition 2020 – Solver and Benchmark Descriptions, volume B-2020-1 of *Department of Computer Science Report Series B*, University of Helsinki, 2020, p. 51–53.

[35] A. Biere, CaDiCaL at the SAT Race 2019, in: M. J. H. Heule, M. Järvisalo, M. Suda (Eds.), Proc. of SAT Race 2019 – Solver and Benchmark Descriptions, volume B-2019-1 of *Department of Computer Science Series of Publications B*, University of Helsinki, 2019, pp. 8–9.

[36] S. Cai, X. Zhang, M. Fleury, A. Biere, Better decision heuristics in CDCL through local search and target phases, J. Artif. Intell. Res. 74 (2022) 1515–1563. doi:10.1613/jair.1.13666.

[37] M. S. Cherif, D. Habet, C. Terrioux, Kissat_MAB: Combining VSIDS and CHB through multi-armed bandit, in: T. Balyo, N. Froleyks, M. J. H. Heule, M. Iser, M. Järvisalo, M. Suda (Eds.), Proc. of SAT Competition 2021 – Solver and Benchmark Descriptions, volume B-2021-1 of *Department of Computer Science Report Series B*, University of Helsinki, 2021, p. 15–16.

[38] J. Zheng, K. He, Z. Chen, J. Zhou, C.-M. Li, Combining hybrid walking strategy with Kissat_MAB, CaDiCaL, and LStech-Mapl, in: T. Balyo, N. Froleyks, M. J. H. Heule, M. Iser, M. Järvisalo, M. Suda (Eds.), Proc. of SAT Competition 2022 – Solver and Benchmark Descriptions, volume B-2022-1 of *Department of Computer Science Report Series B*, University of Helsinki, 2022, p. 20–21.

[39] A. Biere, M. Fleury, N. Froleyks, M. Heule, The SAT Museum POS'23 artifact, 2023. doi:10.5281/zenodo.10037737.

[40] B. Dutertre, An empirical evaluation of SAT solvers on bit-vector problems, in: F. Bobot, T. Weber (Eds.), Proceedings of the 18th International Workshop on Satisfiability Modulo Theories co-located with the 10th International Joint Conference on Automated Reasoning (IJCAR 2020), Online (initially located in Paris, France), July 5-6, 2020, volume 2854 of *CEUR Workshop Proceedings*, CEUR-WS.org, 2020, pp. 15–25.

[41] K. Fazekas, A. Niemetz, M. Preiner, M. Kirchweger, S. Szeider, A. Biere, IPASIR-UP: user propagators for CDCL, in: M. Mahajan, F. Slivovsky (Eds.), 26th International Conference on Theory and Applications of Satisfiability Testing, SAT 2023, July 4-8, 2023, Alghero, Italy, volume 271 of *LIPIcs*, Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2023, pp. 8:1–8:13. URL: https://doi.org/10.4230/LIPIcs.SAT.2023.8. doi:10.4230/LIPIcs.SAT.2023.8.

[42] S. Kochemazov, A. Ignatiev, J. P. Marques-Silva, Assessing progress in SAT solvers through the lens of incremental SAT, in: C. Li, F. Manyà (Eds.), 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*, Springer, 2021, pp. 280–298. doi:10.1007/978-3-030-80223-3_20.

[43] A. Biere, N. Froleyks, W. Wang, Cadiback: Extracting backbones with cadical, in:

M. Mahajan, F. Slivovsky (Eds.), 26th International Conference on Theory and Applications of Satisfiability Testing, SAT 2023, July 4-8, 2023, Alghero, Italy, volume 271 of *LIPIcs*, Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2023, pp. 3:1–3:12. URL: https://doi.org/10.4230/LIPIcs.SAT.2023.3. doi:10.4230/LIPIcs.SAT.2023.3.

[44] G. Audemard, L. Paulevé, L. Simon, SAT heritage: A community-driven effort for archiving, building and running more than thousand SAT solvers, in: L. Pulina, M. Seidl (Eds.), 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*, Springer, 2020, pp. 107–113. doi:10.1007/978-3-030-51825-7_8.

[45] J. K. Fichte, M. Hecher, S. Szeider, A time leap challenge for SAT-solving, in: H. Simonis (Ed.), Principles and Practice of Constraint Programming - 26th International Conference, CP 2020, Louvain-la-Neuve, Belgium, September 7-11, 2020, Proceedings, volume 12333 of *Lecture Notes in Computer Science*, Springer, 2020, pp. 267–285. doi:10.1007/978-3-030-58475-7_16.

[46] H. H. Hoos, B. Kaufmann, T. Schaub, M. Schneider, Robust benchmark set selection for boolean constraint solvers, in: LION, volume 7997 of *Lecture Notes in Computer Science*, Springer, 2013, pp. 138–152.