SAT Competition 2020[☆]

Nils Froleyks[a], Marijn Heule[b], Markus Iser[c], Matti Järvisalo[d], Martin Suda[e]

[a] *Institute for Formal Models and Verification, Johannes Kepler University, Austria*
nils.froleyks@jku.at

[b] *Computer Science Department, Carnegie Mellon University, USA*
marijn@cmu.edu

[c] *Department of Informatics, Karlsruhe Institute of Technology, Germany*
markus.iser@kit.edu

[d] *HIIT, Department of Computer Science, University of Helsinki, Finland*
matti.jarvisalo@helsinki.fi

[e] *Czech Technical University in Prague, Czech Republic*
martin.suda@cvut.cz

Abstract

The SAT Competitions constitute a well-established series of yearly open international algorithm implementation competitions, focusing on the Boolean satisfiability (or propositional satisfiability, SAT) problem. In this article, we provide a detailed account on the 2020 instantiation of the SAT Competition, including the new competition tracks and benchmark selection procedures, overview of solving strategies implemented in top-performing solvers, and a detailed analysis of the empirical data obtained from running the competition.

Keywords: SAT, Boolean satisfiability, SAT Competition, SAT solvers, empirical evaluation, benchmarking

1. Introduction

From what was once mainly the archetypal intractable (in particular NP-complete) problem, propositional satisfiability (or Boolean satisfiability, SAT) has flourished into a success story of modern computer science [1]. This is due to advances in SAT solvers, i.e., implementations of decision procedures for SAT, which today form a central computational tool for solving real-world problem instances of various kinds of NP-hard search and optimization problems. With standardized input formats, readily-available APIs for incremental applications, and certified proof logging and checking capabilities, applications of SAT solver

[☆] satcompetition.github.io/2020

technology have branched from the first breakthrough applications in automated planning, test pattern generation and hardware verification to thousands of different application settings.

The success of SAT would not be possible without the persistent efforts of the SAT community to further improve the performance and robustness of SAT solvers. The SAT Competition series, with a history dating back to the early 90s, aims to support and provide further incentives for maintaining this progress. Organized yearly as an international open event, SAT Competitions (and their variants in the forms of SAT Races and a SAT Challenge) [2, 3, 4, 5, 6, 7, 8] have a consistent track record in receiving tens of solver submissions yearly, submitted by the community at large for obtaining a snapshot of the current state-of-the-art in practical SAT solving. Alongside participating solvers, the competition invites through open calls submissions of benchmark instances representing, in particular, new interesting applications scenarios of SAT solvers. Indeed, in addition to evaluating recently developed solvers, an important aspect of the SAT competition series is to collect on a yearly basis new benchmark sets, consisting of instances from various different application settings, which together with benchmark sets from previous years constitute a standard dataset for use in research papers and SAT solver development.

This article focuses on the 2020 instantiation of the SAT Competitions. To this end, we provide a detailed account of SAT Competition 2020 in terms of organizational details, competition tracks, participating solvers, benchmarks, and the empirical results from the competition. In terms of competition tracks, two new tracks, namely the cloud track and an application-specific track, were introduced in 2020, in addition to the already earlier established main, parallel, and incremental tracks; we provide motivation and the new organizational details for both of these new tracks. In terms of solvers, we provide an overview of solving strategies and other details implemented in the top-performing solvers from the competition, complementing the individual solver descriptions available in the 2020 competition proceedings [9]. As for benchmarks, we describe how the 2020 benchmark sets were constructed for each of the competition tracks, with an overview of the benchmarks contributed to the 2020 competition. In terms of empirical results we provide further analysis on the competition results, going beyond the standard rankings provided on the SAT competition web pages.[1] Finally, we also provide a discussion on lessons learned and ideas for future editions of SAT competitions.

This article is organized as follows. We start by providing an overview on the competition, including details on and motivations for the several competition tracks, the rules and other technical requirements of the competition, the ranking schemes used in evaluating the competing solvers, and the computing environments used for executing the competition (Section 2). We then provide an overview of the benchmark sets used in evaluating the solvers, including their origins and the selection process used for constructing the sets (Section 3).

[1] `https://satcompetition.github.io/2020/`

In Section 4 we provide an overview of the competition results followed by a survey on the solving strategies implemented distinctly in the top-ranking solvers in Section 5. Going considerably beyond the plain competition rankings, we provide in Section 6, an in-depth analysis of the competition data from different perspectives, including correlation analysis of runtime performance of solvers and marginal contributions of individual solvers to the "virtual best solver" and portfolios constructed from the competing solvers. The article is concluded with future prospects in Section 7.

2. Overview of SAT Competition 2020

In this section, we describe the individual 2020 SAT Competition tracks, explain the requirements for participation and the ranking criteria, as well as describe the computing infrastructure used for executing the competition.

2.1. Competition Tracks

SAT Competition 2020 consisted of seven tracks: Main track, No-Limits track, Planning track, "Glucose hack" track, Incremental Library track, Parallel track, and the Cloud Track for massively parallel SAT solvers.

2.1.1. The Main, No-Limits, Planning, and "Glucose hack" tracks

The focus of the traditional Main track is on sequential SAT solvers and their evaluation on structured, non-random benchmarks coming from various application areas.

To participate in the Main track, solvers needed to output certificates for both the satisfiable and the unsatisfiable instances. Moreover, the source code of the solver were required to be made publicly available. Solvers not complying with either of these two criteria were only evaluated in a so-called No-Limits track and were not eligible for the Main track awards. The No-Limits track thus enabled participation of closed-source solvers (not being able or willing to expose the source code for legal or other reasons) as well as portfolio solvers (combining two or more core SAT solvers developed by different groups of authors; c.f. Sect. 2.2). Without limit, submissions could be solvers that use a lookup table or similar to determine solutions. Thus, the No-Limits track was only evaluated with respect to *newly* submitted benchmark instances, i.e., on instances which were submitted to SAT Competition 2020.

However, solvers in No-Limits still competed against all other solvers submitted to the Main Track. Thus, to deserve a mention, a No-Limits solver would need to rank among the best-performing solvers among the Main Track participants. In 2020, the top ranked solvers in the No-Limits track were the same as in the Main track. This also indicates the stability of results under the exclusion of old benchmark instances.

Complementing the generality advocated by the standard SAT Competition tracks, in which solvers are evaluated on a set of benchmarks including instances from various types of different problem domains, for 2020 the organizers aimed

to experiment with the potential of a more application-specific track, each year highlighting a different problem domain where the SAT solving technology helps to advance the state of the art. In 2020, the Planning track represented the first trial instantiation of this idea. The focus of this track was specifically on efficiently solving instances arising from the domain of SAT-based automated planning [10]. Automated planning was chosen as the target problem domain of this first instantiation of the domain-specific tracks due to its centrality as one of the first breakthrough applications of SAT solvers. To this end, solvers participating in the Planning track were evaluated on 200 benchmark instances encoding planning problems. The same rules for participation as in the Main track applied to the Planning track. Solvers submitted to the Main track automatically participated in the Planning track.

The traditional Hack track (established 2009 as `Minisat` Hack track) was organized as a sub-track of the Main track for hacks of `Glucose 3` [11]. In the past, several advances in SAT solving required only small modifications of an established solver to achieve a considerable contribution. Hack tracks encourage participation of such small modifications. The limit for being considered a "hack" was—somewhat arbitrarily[2]—set to 1000 non-space character edit distance from the sources of `Glucose 3`. Unfortunately, in 2020 there were not enough participants in this sub-track and so we do not report on it in the results section.

We evaluated all 64 solver submissions (including different configurations of specific solvers) to the Main track. Out of the 64 solvers, eight were explicitly submitted to the No-Limits track. Four solvers were demoted to the No-Limits track due to outputting invalid unsatisfiability proof certificates. Six solvers were disqualified due to outputting truth assignments which did not satisfy the corresponding benchmark instance. This left us with 46 configurations of 22 solvers, including one Glucose hack.

2.1.2. Incremental Library Track

The Incremental Library track was first introduced in SAT Race 2015 [12] and also took place in SAT Competitions 2016 and 2017. In the Incremental Library track the underlying idea is to mimic scenarios where a SAT solver is used as a back-end solver in a more complex tool (typically solving a harder problem than SAT) and is called multiple times before the enclosing tool reaches its final state. "Incremental" here refers to the idea that the individual calls to the SAT solver are not independent, but may share a common subset of the input clauses or differ in the presence of additional unit clause assumptions [13, 14, 15]. Examples for applications of incremental SAT solving are counterexample-guided abstraction refinement (CEGAR) based approaches, e.g., for Bounded Model Checking [16], SAT-based planning [17], multi-agent path finding [18], and satisfiability modulo theories (SMT) solvers [19].

Instead of using or extending the DIMACS input format, in the the Incre-

[2]The specific threshold for edit distance is not central here; the idea is essentially to only allow relatively small changes to the Glucose code base, i.e., "quick hacks" to Glucose.

mental Library track a general incremental interface called IPASIR (Re-entrant Incremental Solver API) is employed [12]. The idea is that we actually run the enclosing tool on its own benchmark and communicate with the competing SAT solver through this interface. SAT solvers that are submitted for this track must hence implement the interface. Furthermore, it should be noted that the solutions output by a solver may, in general, influence the forthcoming invocations of the solver.

Six solver were submitted to the Incremental Library track. Two of the six solvers were disqualified due to outputting wrong answers.

2.1.3. Parallel Track

The Parallel track evaluates the runtime performance of SAT solvers making use of multiple processor cores in terms of wall-clock time. The benchmarks are the same as in the Main track. In contrast to the Main track, proof logging for unsatisfiable instances is not required in the Parallel track.[3]

A total of 14 solver configurations, based on 10 solvers, were submitted to the Parallel track. Three solver configurations were disqualified due to wrong answers.

2.1.4. Cloud Track

The Cloud track was a new development in the SAT Competitions for 2020. The track focuses on evaluating distributed solvers running on multiple machines in a network. Communication between the machines is possible using MPI and SSH. We received six solver submissions to the Cloud track.

2.2. Mandatory Participation Requirements

The following requirements were imposed for participating in SAT Competition 2020.

Source Code. The source code of submitted SAT solvers had to be made available (licensed for research purposes) except for the solvers participating only in the No-Limits track.

Description. A short system description was required for each solver submission, including a list of all authors involved in developing the solver, description of any non-standard algorithmic techniques and data structures implemented in the solver, as well as references to the relevant literature. These system descriptions have been collected and made available publicly in the non-refereed competition proceedings [9].

[3] Although this would, of course, be desirable, for the same reasons as in the Main track, currently there are no good solutions known for efficient proof logging of parallel solvers.

Benchmarks. The authors of solvers participating in the Main track were required to submit 20 "new" benchmark instances. The exact details of this rule are further explained in Section 3.1. In short, this rule guaranteed that the competition could be run on instances mostly unseen to the solver developers prior to the competition. Moreover, by making these benchmarks publicly available after the competition, the SAT community benefits by having an ever growing repository of diverse problems that next developments will target. The descriptions of the submitted benchmarks are also made available in the competition proceedings [9].

Input and Output Format. No-limits The benchmark instances were presented to the solvers in the de facto standard DIMACS input format for propositional formulas in conjunctive normal form (CNF). A simple extension of this format was to be adhered to when printing the satisfying assignment (see, e.g., [8], Section 2.4).

Where required, proofs of unsatisfiability were to be output in the DRAT format [20], either in its textual version—which is also very similar to the DIMACS input format—or in a more compact binary version (for more details, see [21], Unsat Certificates). Details on certification are further discussed in Section 2.4.

Number of Submissions. Due to the shear number of participants in the SAT Competitions, in order to make it feasible to run the whole competition, specific limits were set on the number of submitted solvers. In particular, each solver author was allowed to be an author of at most four different sequential solvers, two different parallel solvers, and one "Glucose hack" sub-track solver. Two solvers were considered different as soon as their sources differed or the compilation options were different, or different command line options were used (with the exception of an option enabling or disabling the proof output).

Portfolio Solvers. Apart from the No-Limits track, participants were not allowed to submit a portfolio of solvers, i.e., a combination of two or more core SAT solvers developed by different groups of authors.[4] This rule is mainly meant to encourage the SAT community to invest more effort into developing new solver code bases. Moreover, while we acknowledge that research on solver selection tools that typically orchestrate portfolio solvers is interesting, it is not at present the focus of the SAT competitions.

Organizers. The organizers of the competition were not allowed to participate.

[4] In other words, a submission of a combination of solvers was only possible if all the authors of all the parts were explicitly listed. This means that all the authors had to be notified if such participation was planned and had to consider it carefully, also taking into account the limited number of submissions per author as specified by the previous rule.

2.3. Solver Ranking and Disqualification

Solvers were ranked using a PAR-2 score based on a 5000-second timeout. A PAR-2 system assigns as many points as the amount of time (in seconds) it took the solver to solve a particular instance and twice the time limit, i.e. 10 000 points, if the instance was not solved. In particular, this means that the lower the score a solver obtained, the better the solver performs.

A solver was disqualified if it produced a wrong answer: specifically, if a solver reported "unsatisfiable" on an instance that was proven to be satisfiable by some other solver, or reported "satisfiable" but provided a wrong certificate. Solvers disqualified from the competition were not eligible for awards.

2.4. Certificates

In all tracks, solvers were required to output a solution (a satisfying truth assignment, i.e., a model on the instance in question) to certify recognizing a satisfiable instance. On the other hand, certificates for unsatisfiable instances (proofs) were required only in the Main track (besides the No-Limits track). In some cases, a solver output the correct result, but the respective certificate was wrong. Such solvers were demoted to the No-Limits track of the competition.

Each unsatisfiability proof produced by each solver was validated in a two-step fashion. First, the tool `DRAT-trim` [20] was used for initial checking and optimizing the proof, thereby obtaining a so-called LRAT proof file. An independent formally-verified checker `cake_lpr` [22] was then used for validating the LRAT proof as a correct proof of unsatisfiability.

In a few cases `DRAT-trim` ran into the verification timeout of 45,000 seconds. In the Main track, only those unsatisfiable benchmark instances for which the proof produced by a solver could be validated at least by `DRAT-trim` were considered solved by the solver. While there were several cases where `cake_lpr` ran out of resources, there was no case where `DRAT-trim` would accept a proof and `cake_lpr` would not.

2.5. Computing Environments

The Main, No-Limits, and Planning tracks were run on the StarExec cluster [23] with computing nodes equipped with Intel Xeon 2.4 GHz processors and 128 GB of memory. The time limit enforced on each solver for solving an instance was 5,000 seconds. In the Main track, proof validation was limited to 45,000 seconds per proof.) The solvers were allowed to use the full 128 GB of RAM.[5]

The Incremental Library Track was run on computers with 2x Intel Xeon E5430 2.66 GHz (4-Core) processors and 24 GB of RAM. The Parallel track was run on AWS m4.16xlarge machines with 64 virtual CPUs and 256 GB of memory, while the Cloud track was run on Amazon Web Services (AWS) m4.4xlarge

[5] Unfortunately, the memory limit of 24 GB, that was used in the previous years, was by mistake advertised on the competition web page prior to solver submission. This could have resulted in some solvers not "daring" to use the full 128 GB in the competition. We do not, however, have concrete evidence to support this possibility.

Algorithm 1: Benchmark Instance Selection

Data: I : Set of Instances, A : Set of Authors
Data: Functions $\alpha : I \to A$ and $\sigma : I \to \{\mathsf{sat}, \mathsf{unsat}, \mathsf{unknown}\}$
Result: S : Set of Selected Instances

$S \leftarrow \emptyset$
for $a \in A$ **do**
  $I_a^+ \leftarrow \mathsf{random}(7, \{e \in I \mid \alpha(e) = a \wedge \sigma(e) = \mathsf{sat}\})$
  $I_a^- \leftarrow \mathsf{random}(7, \{e \in I \mid \alpha(e) = a \wedge \sigma(e) = \mathsf{unsat}\})$
  if $|I_a^+| + |I_a^-| < 14$ **then**
    $l \leftarrow 14 - |I_a^+| - |I_a^-|$
    $I_a^? \leftarrow \mathsf{random}(l, \{e \in I \mid \alpha(e) = a \wedge \sigma(e) = \mathsf{unknown}\})$
  $S \leftarrow S \cup I_a^+ \cup I_a^- \cup I_a^?$
return S

machines with 16 virtual CPUs and 64 GB of memory. These tracks used wall-clock timeouts of 5,000 seconds and 1,000 seconds, respectively.

3. Benchmarks

For data-driven selection of benchmark instances, we used GBD Tools[6] which facilitates querying for instances with desired properties, e.g., by instance author, family, result or solver runtime [24]. We also use GBD Tools for distributing benchmark instances and their attributes to the general public.[7]

3.1. Selection of Instances

The "Bring Your Own Benchmarks" (BYOB) rule, first established in SAT Competition 2017 [25], was again followed in 2020. By this rule, solver authors are required to submit 20 benchmark instances to accompany a solver submission in order to participate in the competition. These benchmarks have to be "new" in the sense that instances included in benchmark sets from previous SAT competitions are not allowed. Furthermore, at least ten of the required 20 instances are required to be "interesting", interpreted in loose terms by the standard `Minisat` SAT solving needing at least one minute of runtime (on typical computing hardware) to solve an instance. It should be noted that new benchmarks could be submitted to the competition without needing to submit a solver. As a results, as detailed in Table 1, 27 authors contributed a set of 1,260 new benchmark instances from a wide range of different instance families.

We decided to include a total of 300 new benchmarks and a further 100 benchmarks from previous SAT Competitions to the main benchmark set of the 2020 competition. Key aims of benchmark selection is to ensure that (i)

[6] See `https://pypi.org/project/gbd-tools/`
[7] See `https://gbd.iti.kit.edu`

Family	Author	Submitted		Selected	
0/1 Integer Programming	Riveros	6		2	
Fermat		8		5	
Schur Coloring		4		2	
Sum Subset		5		2	
Anti-Bandwidth	Biere	187		14	
Baseball Lineup	Hickey	40		13	
Bit-Vector	Preiner	393		14	
Cellular Automata	Chowdhury	20		12	
CNF Miter	Manthey	38		7	
Coloring	Oostema	14		14	
Core-based Generator	Hartung	20		14	
Cover	Gacek	18		13	
Cryptography	Paxian	50	106	6	34
	Shaw	20		14	
	Soos	36		14	
Discrete Logarithm	Jingchao	20		7	
Edge Matching	Holten	58		7	
Flood-It Puzzle	Stiphout	40		0	
HGen	Guanfeng	20		13	
Hypertree Decomposition	Schidler	56		14	
Influence Maximization	Kochemazov	20		14	
Lam Discrete Geometry	Nejati	20		9	
Polynomial Multiplication	Maoluo	20		8	
Station Repacking	Newman	20		12	
Stedman Triples	Johnson	23		7	
Tensors	Savicky	20		14	
Termination Analysis	Yolcu	12		7	
Timetable	Djamegni	20		14	
Tournament	Heule	16		14	
Vlsat	Bouvier	36		14	
	Σ	1260		300	

Table 1: Families and amounts of newly submitted instances

the benchmark set includes enough many relatively hard-to-solve instances in order to differentiate the overall runtime performances of the competing solvers (without actually running the competing solvers during benchmark selection); (ii) the number of benchmarks included in the benchmark set from different problem domains is balanced across the problem domains, and that (iii) the benchmark set is also balanced in terms of the number of unsatisfiable and satisfiable instances included in the set.

To compile the set of 300 new instances, we first applied a hardness criterion by filtering out all instances solved by `Minisat` in less than ten minutes.[8] From

[8]Note that the limit of ten minutes is again somewhat arbitrary. This runtime hardness filter essentially aims to make sure that enough instances are included in the final benchmark set which allow for distinguishing in terms of relative performance between the best-performing

	SAT	UNSAT	UNKNOWN	Σ
New Instances	114	78	108	300
Old Instances	21	57	22	100
Σ	135	135	130	400

Table 2: Amount of old and new instances by result

the resulting 1,012 instances, in order to obtain a balanced benchmark set, we randomly selected k instances per author using the value k which ensured that the resulting set contains at least 300 instances. This *rule-based randomization* procedure is detailed as Algorithm 1. Specifically, we randomly selected seven satisfiable and seven unsatisfiable instances per author (Lines 3 and 4) and added instances of yet unknown result if this did not yield a total of 14 instances (Lines 5–7). Of the such obtained 308 instances, we randomly removed eight satisfiable instances, yielding a total of 114 satisfiable, 78 unsatisfiable and 108 instances of unknown satisfiability status.

We augmented the then obtained set of 300 new benchmarks with 100 instances from previous SAT competitions as follows. In order to further balance the number of satisfiable and unsatisfiable instances in the new benchmark set, we randomly selected 21 satisfiable, 57 unsatisfiable and 22 unknown instances. With additional constraints, we made sure not to select instances from benchmark families which are already represented in the set of 300 new instances (cf. Table 1). We also excluded random, agile and planning instances (due to the Planning track). The final main benchmark set contains 135 satisfiable, 135 unsatisfiable, and 130 instances of "unknown" status (cf. Table 2).

3.2. Planning Instances

Classical planning is the problem of finding a sequence of actions—a plan—that transforms the world from some initial state to a goal state. In 1992 Kautz and Selman [10] proposed to encode planning as satisfiability, constituting one of the hallmark early adoptions of SAT solving to solve real-world problems. In their encoding the problem of finding a plan of length i (*i.e.,* the *makespan*) is translated into a Boolean formula F_i that is satisfiable if a plan of length i or less exists. Their encoding is called sequential, whereas parallel encodings allow the execution of multiple actions in one step [26, 27, 28]. Finding the smallest makespan i for which F_i is satisfiable is important for SAT-based planning in general and the generation of this benchmark set in particular. The hardest formulas that a SAT-based planner has to solve are usually the last unsatisfiable F_i before the next higher makespan $i + 1$ is satisfiable [26].

competing solvers. Unfortunately this limit was much greater that the requirements imposed for "interesting" benchmark instances by the BYOB rule. It could be more sensible to impose the same ten minutes limit also for an instance being "interesting". However, this would require more efforts at least in terms of computation times from the solver authors in order to construct a set of interesting new benchmarks required by the BYOB rule.

Encoding		SAT	UNSAT
H	Tree-REX	15	11
P	Pasar	14	14
ME	Madagascar ∃-step	5	10
MS	Madagascar sequential	66	65
Σ		100	100

Table 3: Number of planning instances generated per encoding.

For the Planning track, the benchmark instances were generated using two SAT-based planners *Madagascar* [29] and *Pasar* [30]. We used Madagascar both in its default configuration to generate a parallel encoding based on ∃-step plans and to generate a sequential encoding. Pasar uses the *grounding routine* deployed by the well-known planner *Fast Downward* [31] to translate planning tasks into a different formalism and then encodes it to SAT using a parallel encoding. The classical planning benchmarks were selected from the *Satisfying* and *Optimal* tracks of the *International Planning Competitions* 2014 [9] and 2018 [10]. We only selected planning domains with *unit cost* and eliminated those that take more than 100 GB of memory to encode into SAT. We ran both Pasar and Madagascars ∃-configuration with a timeout of three hours on the remaining instances to find the minimal makespans. For each planning task where this process did not timeout, we generated a pair of satisfiable and unsatisfiable SAT instances. A significant number of instances from this set were not used as `Minisat` could solve them in under ten minutes. We augmented the remaining domains with the last unsatisfiable formulas generated for planning tasks where the minimal makespan could not be found. To generate the missing benchmarks, we use a *sequential* encoding together with bounds[11] on the optimal plan length.

In addition to the classical planning problems, we also included SAT instances generated by *Tree-REX* [32], a planner for *Hierarchical Task-networks* (HTN). In HTN planning, additional domain knowledge besides the problem description is provided. The HTN benchmarks were provided by the author of *Tree-REX*.

The instances of the Planning track are large in size compared to the Main track instances. Using the number of clauses as a metric, out of the 100 largest instances across both tracks, 86 belong to the Planning track benchmark set. The large size of Planning track instances can mainly be attributed to large numbers of binary clauses that SAT encodings of planning problems naturally produce. On average, more than 98% of the clauses are binary for planning instances. The average for the Main track instances is below 60%.

Table 3.2 shows the number of benchmarks generated by each encoding. For a complete list of the encoded planning tasks we refer to the generation

[9] `https://helios.hud.ac.uk/scommv/IPC-14/repository/benchmarksV1.1.zip`

[10] `https://bitbucket.org/ipc2018-classical/domains`

[11] Upper and lower bounds on plan length are available for some planning tasks from the *Optimal track* that have *unit cost* actions.

script.[12] The benchmarks of the Planning track adhere to the following naming convention: ⟨SAT/UNSAT⟩_⟨encoding⟩_⟨name⟩_⟨makespan⟩.cnf

3.3. Incremental Library Track Benchmarks

Benchmarks for the Incremental Library track consist of *benchmark applications* which implement and use the incremental SAT solver in their back-end as well as *benchmark instances* which serve as input to these applications. For evaluating solvers participating in the the Incremental Library track, we used six available IPASIR applications. For each of the six applications, we individually selected 50 application instances as follows.

Backbone Computation. Backbone variables [33, 34] are variables which take the same value in all models of a given SAT instance. The application genipabones incrementally determines backbone variables in a given satisfiable SAT instances using the so-called dual rail encoding [12]. We selected 50 of the smallest and easiest satisfiable instances from previous SAT competitions to evaluate solver performance with this application.

Essential Variables. Variables which have to be assigned in all partial models of a formula as *essential* (as opposed to *don't care*-values) [35]. The application genipaessentials incrementally determines essential variables in a given satisfiable formula [12]. For this application, we used the same 50 satisfiable instances as for backbone computation.

Longest Simple Paths (LSP). The application genipalsp determines longest simple paths in a graph [36]. We selected 50 LSP instances for our evaluation.[13]

Maximum Satisfiability (MaxSAT). The application genipamax solves partial MaxSAT problems by augmenting soft clauses with relaxation (or blocking) variables which are input to a cardinality constraint [37]. The MaxSAT problem is then solved by incrementally minimizing the bound of the cardinality constraint. For this application, we selected 50 instances from `MaxSAT Evaluation 2019`.[14]

Quantified Boolean Formulas. Ijtihad is a QBF solver which uses counterexample-guided expansion to incrementally solve a given QBF instance with a SAT solver [38]. Here we used 50 instances from `QBF Evaluation 2019`.[15]

Planning (SAS+). We selected 50 planning instances to evaluate incremental SAT solvers with Pasar, a planner which uses counterexample-guided abstraction refinement (CEGAR) [30].

[12] `https://satcompetition.github.io/2020/downloads/planning_generator.tar.xz`
[13] `http://algo2.iti.kit.edu/kalp/`
[14] `https://maxsat-evaluations.github.io/2019/`
[15] `http://www.qbflib.org/qbfeval19.php`

4. Competition Results

In this section, we provide a high-level overview of the results of SAT Competition 2020. Later on, we will provide an overview of some of the key and new solving techniques implemented in best-performing solvers (Section 5) as well as a more in-depth analysis of the competition results (Section 6). An overview of the top-10 solvers in each of the competition tracks discussed in the following is provided in Table 4.

4.1. Main Track

Starting with the Main track, Figure 1 shows the cumulative solved instances plot of the best-performing solver of the strongest ten teams (in short, the top-10 solvers) together with the Virtual Best Solver (VBS—see also 6.1). The best-performing solver overall on the combination of satisfiable and unsatisfiable instances is `Kissat-sat` and the runner-up is `Relaxed-newTech`. Notice that `Relaxed-newTech` solved more instances within the 2000-second limit. Third place, based on the PAR-2 score, went to `CMS-ccnr-lsids`. It solved two instances less than `CaDiCaL-alluip-trail`, but on the other hand solved various formulas more quickly. Similar observations have been made also in earlier recent SAT competitions where solvers were ranked based on the PAR-2 score. Furthermore, we observe more differences in overall runtime performance among the top solvers than what has been observed in the recent past competitions.

The four solvers `Kissat-sat`, `Relaxed-newTech`, `CaDiCaL-alluip-trail`, and `CMS-ccnr-lsids` performed significantly better that all other solvers submitted to the Main track (cf. Table 4). A very interesting observation is that these four top solvers all have a different code base. This has not been observed for many years; more typically many of the best-performing solvers have been based on same code bases.

The majority of the overall performance differences between the top-4 solvers and the other solvers is due to performance differences on satisfiable instances; see Figure 2. Indeed, in recent years, several techniques have been added to SAT solvers to improve their performance on satisfiable instance. Examples of such techniques are the integration of a local search solver and alternating between a SAT mode (infrequent restarts) and an UNSAT mode (frequent restarts and variable-move-to-front [39]). The best-performing solver in the Main SAT track on satisfiable instances is `Relaxed-newTech`, followed by `Kissat-sat` and `CMS-ccnr-lsids`.

Overall, solvers performed much more similarly on unsatisfiable instances than on satisfiable instances; see Figure 3 for the runtime performance on unsatisfiable instances. Only `Kissat-unsat`, the winner of the Main UNSAT track, performed significantly better than all other participating solvers. It is therefore not surprising that the VBS is reasonably close to `Kissat-unsat`. The solvers `CaDiCaL-trail` and `f2trc-s` placed, respectively, second and third in the Main UNSAT track.

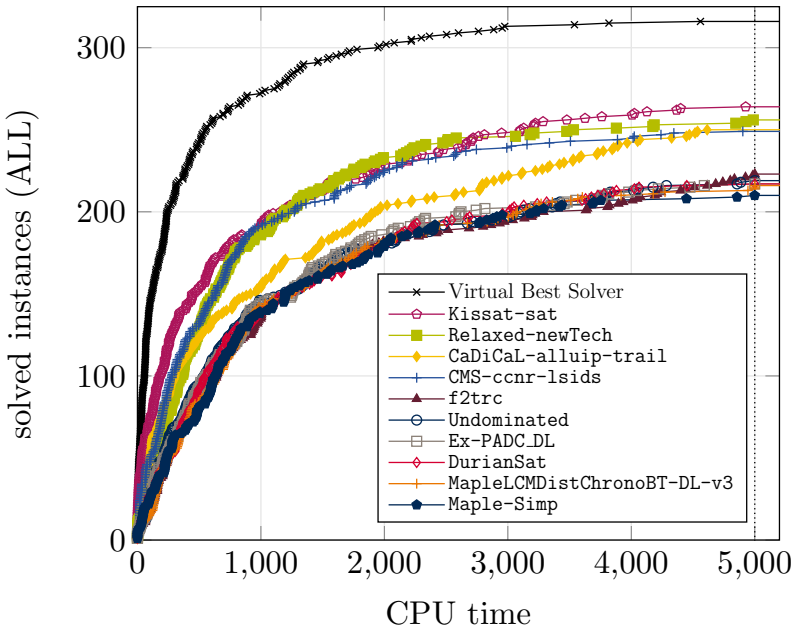

Figure 1: Performance of the top 10-solvers and the VBS on all Main track benchmarks

4.2. Planning Track

The competition in the Planning Track was more tight. The best solver `CaDiCaL-alliup-trail` solved only one instance more than the runner up `CMS-ccnr-lsids`. The PAR-2 scores of these two solvers were quite similar as well. The third ranked solver, `Kissat`, solved fewer instances, but its fast runtimes on several instances resulted in a strong PAR-2 score. Notice that these three solvers were also strong in the Main track. It should be noted that, somewhat disappointingly, none of the participating solvers were actually optimized for planning instances.

4.3. Parallel Track

Turning to the Parallel track, Figure 4 shows the performance of all participating parallel solvers. The best solver here is `Painless-MCOMSPS-STR32`. Interestingly, this solver used 32 threads on the 64 virtual cores that were available. In fact, it has been observed in already recent earlier SAT competitions that using fewer threads than the number of available virtual cores can be helpful; as threads compete for memory, using all virtual cores may be detrimental to overall performance. The runner up is `Plingeling`, while the third place goes to `ManyGlucose-32`. Interestingly, one can observe from Table 4 that only the `Painless-MCOMSPS-STR*` solvers and `Plingeling` had a lower PAR-2 score than the winner of the Main Track (`Kissat-sat`). It appears still to be challenging to beat the best-performing sequential solvers with with a parallel solver. (In

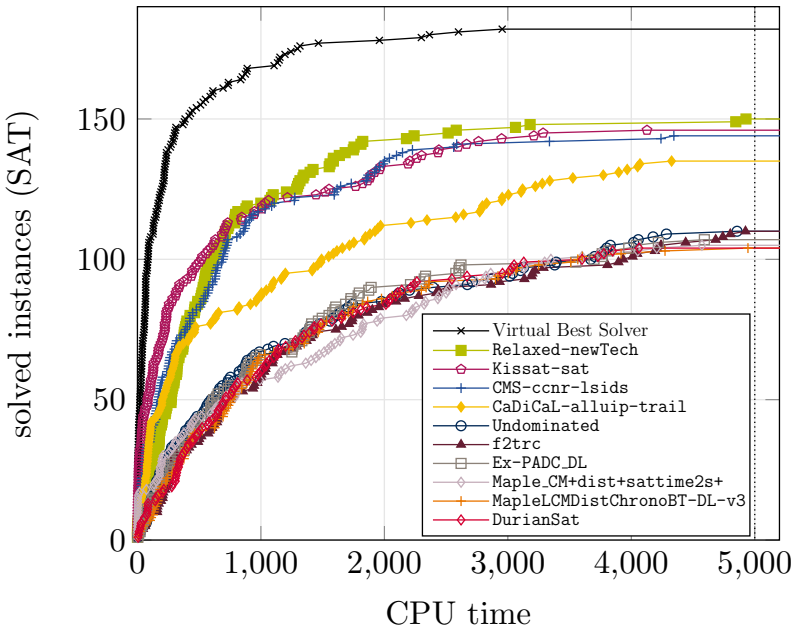

Figure 2: Performance of the top-10 solvers and the VBS on Main track satisfiable benchmarks

fact, as we will show later on, sequential portfolios including only a few Main track solvers show better performance than all of the parallel solvers.)

4.4. Cloud Track

The clear winner of the Cloud track is `Mallob-Mono` and the runner-up is `TopoSAT2` (see Figure 5). `Mallob-Mono` was able to solve more instances in 1000 seconds than the winner of the Parallel Track in 5000 seconds, which shows the potential of distributed SAT solving. The other four participants performed significantly worse. The massive parallelism in distributed SAT solving imposes additional challenges on scalable information sharing and search diversification. Since 2020 was the first year of this track, we expect a tighter competition in the future.

5. Winning Solvers

In this section we will provide an overview of the participating teams and solvers, and summarize new strategies implemented in the best-performing solvers of the 2020 competition. We start with a few remarks on the evolution of code-bases of well-known SAT solvers.

Main Track

Pl.	PAR-2	#	Solver
1	3926.2	264	Kissat-sat
(1)	4083.1	260	Kissat
2	4179.3	253	Relaxed-newTech
3	4266.7	248	CMS-ccnr-lsids
(3)	4278.0	250	CMS-ccnr
	4428.1	250	CaDiCaL-alluip-trail
	4429.6	250	CaDiCaL-alluip
	4436.5	245	Relaxed
	4501.2	243	CMS-walksat
	4554.0	243	CaDiCaL-trail

Parallel Track

Pl.	PAR-2	#	Solver
1	3316.6	283	Painless-MCOMSPS-str32
(1)	3714.7	271	Painless-MCOMSPS-str64
2	3743.4	269	Plingeling
3	3985.3	260	ManyGlucose-32
	4022.7	262	Painless-Maple-v1
	4036.3	258	ManyGlucose-64
	4103.3	260	Painless-Maple-v2
	4433.3	243	Syrup-Scavel
	4903.4	225	Treengeling
	5240.1	213	abcdsat-p20

Main Track, Satisfiable Instances

Pl.	PAR-2	#	Solver
1	2997.4	150	Relaxed-newTech
2	3127.6	146	Kissat-sat
3	3263.0	144	CMS-ccnr-lsids
(3)	3317.4	145	CMS-ccnr
	3355.5	143	Relaxed
	3721.2	139	CMS-walksat
	3830.5	134	Kissat
	3908.5	135	CaDiCaL-alluip-trail
	3909.6	135	CaDiCaL-alluip
	4265.6	126	CaDiCaL-trail

Parallel Track, Satisfiable Instances

Pl.	PAR-2	#	Solver
1	2853.7	153	Painless-MCOMSPS-str32
2	2913.6	154	Painless-Maple-v1
(2)	3082.7	151	Painless-Maple-v2
–	3196.9	148	Painless-MCOMSPS-str64
3	3805.9	133	Plingeling
	4048.3	130	ManyGlucose-64
	4076.2	130	ManyGlucose-32
	4675.4	119	Syrup-Scavel
	4907.1	114	Treengeling
	6337.5	85	abcdsat-p20

Main Track, Unsatisfiable Instances

Pl.	PAR-2	#	Solver
1	4315.1	124	Kissat-unsat
(1)	4335.6	126	Kissat
(1)	4724.8	118	Kissat-sat
2	4842.5	117	CaDiCaL-trail
–	4846.7	116	CaDiCaL-sc2020
(2)	4947.8	115	CaDiCaL-alluip-trail
(2)	4949.6	115	CaDiCaL-alluip
3	4991.4	110	f2trc-s
(3)	5051.4	109	f2trc
(3)	5054.3	110	f2trc-DL

Parallel Track, Unsatisfiable Instances

Pl.	PAR-2	#	Solver
1	3680.8	136	Plingeling
2	3779.5	130	Painless-MCOMSPS-str32
3	3894.3	130	ManyGlucose-32
(3)	4024.2	128	ManyGlucose-64
	4142.8	128	abcdsat-p20
	4191.1	124	Syrup-Scavel
	4232.5	123	Painless-MCOMSPS-str64
	4899.8	111	Treengeling
	5123.9	109	Painless-Maple-v2
	5131.9	108	Painless-Maple-v1

Planning Track

Pl.	PAR-2	#	Solver
1	6406.9	80	CaDiCaL-alluip-trail
(1)	6409.3	80	CaDiCaL-alluip
2	6466.9	79	CMS-ccnr-lsids
(2)	6471.9	79	CMS-ccnr
(2)	6472.9	79	CMS-walksat
3	6596.4	75	Kissat-unsat
	6650.1	79	CaDiCaL-trail
	6713.0	75	Maple-Mix
	6746.5	75	MapleCOMSPS-init
	6754.3	73	Maple-Simp

Cloud Track

Pl.	PAR-2	#	Solver
1	2603.8	299	Mallob-Mono
2	3146.8	278	TopoSAT 2
3	4797.6	213	Slime
	6800.8	132	Paracooba
	7299.0	110	CTsat
	8468.3	62	Paracooba-March

Table 4: Top 10 Solvers in Main, Planning, Parallel and Cloud Tracks: Place (Pl.), Score (PAR-2) and Number of Solved Instances (#) per Solver. Three awardees per track underlined.

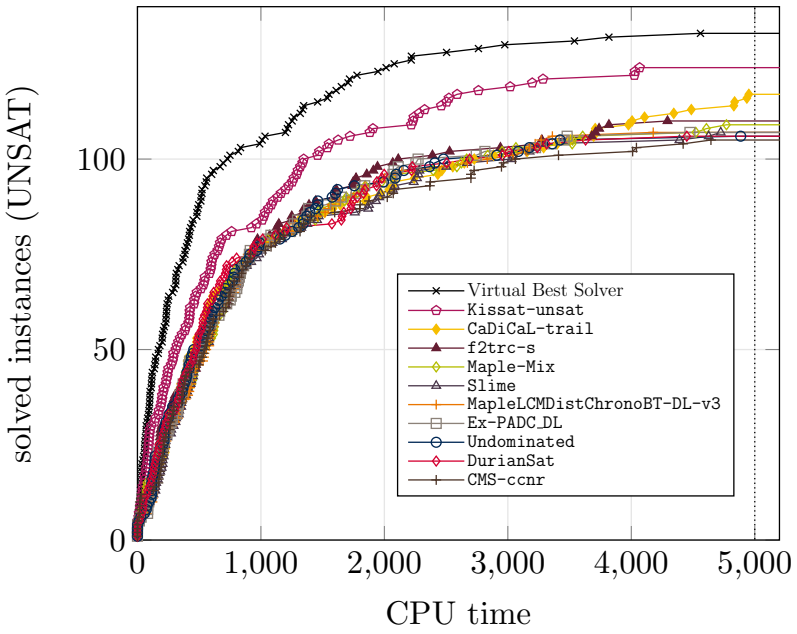

Figure 3: Performance of the top 10 solvers + VBS on Main track unsatisfiable benchmarks

5.1. Evolution of SAT Solver Code-Bases: "On the Shoulders of Giants"

Progress in SAT solvers is often based on successful modifications of existing and openly available solver code-bases. One well-known tree of code-base evaluation is rooted in the code-base of `Minisat` by Eén and Sörensson [40]. A well-known fork of `Minisat` is `Glucose` by Audemard and Simon [11]. In particular, `Glucose` introduced the influential literal block distance (LBD) heuristics for deciding which learned clauses to keep and which ones to forget during search [41]. The SAT solver `RISS` by Manthey is a further fork of `Glucose`, combining `Glucose` with the `Coprocessor` [42] preprocessor.

A further, more recent line of evolution in SAT solvers is rooted in the `CoMinisatPS` by Oh, which is itself a again a fork of `Minisat`, and which introduced three-tier clause-management [43]. Building on `CoMinisatPS`, the SAT solver `Maple` appeared as a series of forks presenting innovative branching heuristics at SAT Competition 2016 [44]. The at-the-time award-winning variant `MapleCOMSPS` by Liang et al. implements a hybrid branching heuristic of classic variable-state independent decaying sum (VSIDS) [45] and the newer learning rate based branching (LRB) [46].

For SAT Competition 2017, Luo et al. integrated learned clause minimization based on unit propagation (`LCM`) in their award-winning `Maple_LCM_Dist` [47] which also uses the new branching heuristic `Distance (Dist)` in an initial solving period [48]. In SAT Competition 2018, Ryvchin and Nadel successfully integrated conditional chronological backtracking (`ChronoBT`) [49] in their award

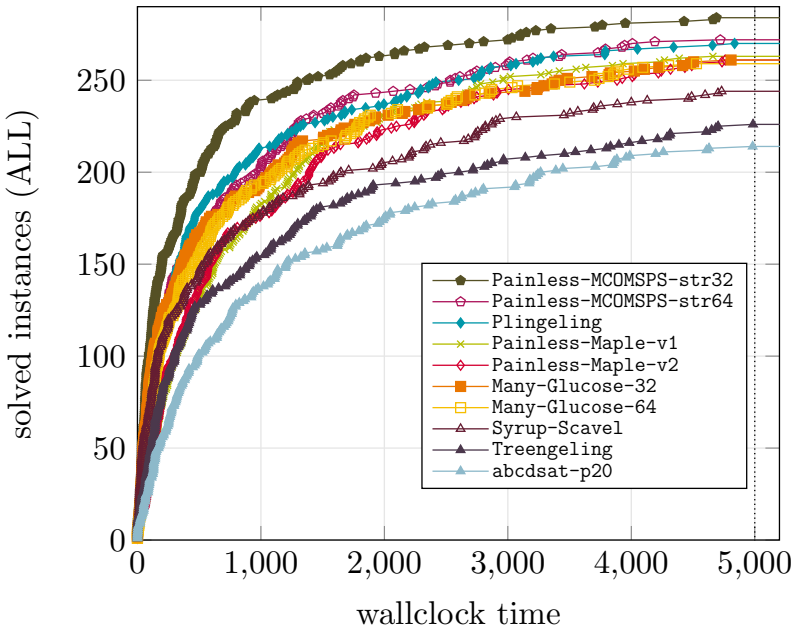

Figure 4: Performance of the solvers in the Parallel track.

award-winning solver `Maple_LCM_Dist_ChronoBT` [50].

Kochemazov et al. improved three-tier clause-management by persisting additional clauses through hash-based detection of repeatedly learned clauses and presented their award-winning `MapleLCMDistChronoBT-DL` in SAT Race 2019 [51]. As can be seen in Table 5, numerous submissions to SAT Competition 2020 are forks of some recent award-winning descendants of a `Maple`-based solver.

Also starting as a fork of `Minisat` with the integration of special treatment for XOR constraints [52], `CryptoMinisat` by Soos continues to be a state-of-the-art and feature-rich SAT solver. One highlight of `CryptoMinisat` are its advanced data-logging capabilities for statistical analysis of SAT solver behavior [53].

Many independent and award-winning code-bases can be found among the SAT solvers written by Biere. The sequential SAT solver `Lingeling` has been award-winning since SAT Competition 2011 and is still competitive in its parallel version `Plingeling` [54]. As of SAT Competition 2017, `CaDiCaL` by Biere is another independent representative of state-of-the-art SAT solvers and its improved re-implementation `Kissat` [55] was successful in the 2020 competition.

5.2. Sequential SAT Solvers

Sequential SAT solvers have been evaluated in the Main, Planning and Incremental Library track of SAT Competition 2020. 18 teams submitted a total of 48 solvers and configurations to the Main track and the Planning track of the competition, and four solvers participated the Incremental Library track.

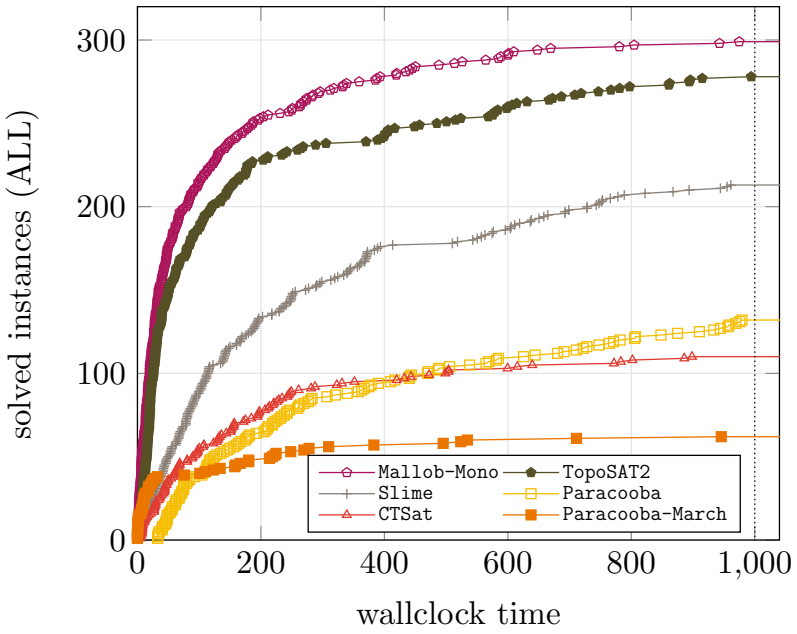

Figure 5: Performance of the solvers in the Cloud track.

Table 5 displays an overview of the participating teams, base solvers and their variants. In the following, we provide a short overview of the best-performing solvers of 2020, based mainly on the solver descriptions submitted to the 2020 competition proceedings by the authors of the individual solvers.

5.2.1. Kissat

Three configurations of `Kissat` were submitted to the 2020 competition, including one default configuration and two specialized configurations which are specifically tailored towards satisfiable and unsatisfiable instances, respectively. `Kissat` received *four awards*, achieving the first place in the Main track, the best score on unsatisfiable instances, the second-best score on satisfiable instances and the third place in the Planning track.

`Kissat` is a low-level re-implementation of `CaDiCaL` with new sophisticated lazy data-structures for clause state monitoring, e.g., through binary clause inlining, sentinel values and bit stuffing [56, 55]. Moreover, forward subsumption for learned clauses is mostly replaced by vivification algorithms [57]. Since conflict number has been observed to be too unstable for measuring the length of two alternating restart modes, `Kissat` uses the new unit "ticks" which approximates the number of cache-line accesses in unit-propagation [55]. `Kissat` also exploits autarkies to account for saved phases. In order to keep valuable information of saved phases, before each rephasing step `Kissat` computes the largest autarky for the assignment implied by the current saved phases [58]. As such an autarky

Team	Base Solver	Variant Name	M	U	S	P	I
Biere	Kissat	–	1	1	–	–	–
		sat	1	1	2	–	–
		unsat	–	1	–	3	–
Biere, Fleury	Cadical	SC2020	–	–	–	–	–
Zhang, Cai	MapleLCMDistCBT-DL	Relaxed	–	–	–	–	–
		Rel. newTech	2	–	1	–	–
Soos, Cai, Devriendt, Gocht, Shaw, Meel	CryptoMiniSat-CCAnr	–	3	–	3	2	1
		lsids	3	–	3	2	–
Soos, Selman, Kautz, Devriendt, Gocht	CryptoMiniSat-WalkSAT	–	–	–	–	–	–
Hickey, Feng, Bacchus	CaDiCaL	trail	–	2	–	–	–
		alluip	–	2	–	1	–
		alluip-trail	–	2	–	1	–
	MapleLCMDist	alluip-trail	–	–	–	–	–
Kochemazov	MapleLCMDistCBT	f2trc	–	3	–	–	–
		f2trc-s	–	3	–	–	–
	MapleLCMDistCBT-DL	f2trc	–	3	–	–	–
Kochemazov, Zaikin, Kondratiev, Semenov	MapleLCMDistCBT-DL-v3	–	–	–	–	–	–
Lonlac, Nguifo	MapleLCMDistCBT-DL-v3	Undominated	–	–	–	–	–
		Undom. Top16	–	–	–	–	–
		Undom. Top24	–	–	–	–	–
		Undom. Top36	–	–	–	–	–
Tchinda, Djamegni	ExMapleLCMDistCBT	padc_dl	–	–	–	–	–
		padc_dl_ovau_lin	–	–	–	–	–
		padc_dl_ovau_exp	–	–	–	–	–
		psids_dl	–	–	–	–	–
Shaw, Meel	MapleLCMDistCBT-DL-v3	DurianSat	–	–	–	–	–
Chen	MapleLCMDistCBT-DL	Maple_Mix	–	–	–	–	–
		Maple_Simp	–	–	–	–	–
Riveros	MapleLCMDistCBT	SLIME	–	–	–	–	–
Li, Wu, Xu, Chen	MapleLCMDistCBT-DL	Scavel	–	–	–	–	–
		Scavel01	–	–	–	–	–
		Scavel02	–	–	–	–	–
Liang, Oh, Nejati, Poupart, Ganesh	MapleCOMSPS_LRB_VSIDS_2	–	–	–	–	–	–
		init	–	–	–	–	–
Chowdhury, Müller, You	MapleLCMDistCBT-DL-v2.2	exp-V-LGB	–	–	–	–	–
		exp-V-L	–	–	–	–	–
		exp-L	–	–	–	–	–
		exp-V	–	–	–	–	–
Li, Luo, Xiao, Li, Manyà, Lü	MapleCM	+dist	–	–	–	–	–
		+dist+sat2s	–	–	–	–	–
		+dist+simp2	–	–	–	–	–
		used+dist	–	–	–	–	–
Kaiser, Hartung	MapleLCMDist	PauSat	–	–	–	–	–
Osama, Wijs	ParaFROST	–	–	–	–	–	–
		CBT	–	–	–	–	–

Table 5: *Teams* and *Solvers* in the Main, Planning and Incremental Library Track. Column *Base Solver* highlights solver genealogy, and *Variant Name* displays names of configurations and forks. Distinctive names are underlined. The rightmost columns indicate the awards given in the Main (M) track, for performance on unsatisfiable (U) and satisfiable (S) instances, and in the Planning (P) and Incremental Library (I) track.

might contain satisfying assignments which imply disconnected components, those variables are subject to subsequent variable elimination.

5.2.2. CryptoMiniSat

`CryptoMiniSat` received *four awards*, achieving the first place in the Incremental Library track, the second place in the Planning track, the third place in the Main track, and the third-best score on satisfiable instances. Two submitted variants, `default` and `LSIDS`, scored mostly adjacent ranks in the individual competition tracks.

The `LSIDS` variant of `CryptoMiniSat` comes with a new hybrid phase selection approach [59, 60]. `CryptoMiniSat` comes with an independent implementation of state-of-the-art hybrid branching heuristics which alternate between classic phase saving and target phase selection [56]. `CryptoMiniSat-CCAnr` regularly schedules short periods of local search and imports the best assignment for phase selection—a procedure which is known as "rephasing" from CaDiCaL [56]. In addition, `CryptoMiniSat-CCAnr` bumps the VSIDS scores of the first 100 variables in those clauses which the SLS solver weighs most hard to satisfy [60]. Inprocessing has been extended to include ternary resolution and more vivification [57]. `CryptoMinisat` alternates decay factors of its branching heuristics, thus avoiding the restriction to a "single best" configuration [60]. The submitted version of `CryptoMinisat` entails a new optimized implementation of Gauss-Jordan Elimination [61]. `CryptoMinisat` periodically executes the `BreakId` algorithm to calculate symmetry breaking clauses [62].

5.2.3. CaDiCaL AllUip

Based on `CaDiCaL`, its variants `Trail` and `AllUip` present implementations of a new *Trail Saving* approach [63] and the improved clause-learning heuristic *Stable AllUIP* [64]. Submitted were the three variants `Trail`, `AllUip` and `AllUip+Trail`. The variants including *Stable AllUIP* were the most successful in the 2020 competition, achieving in particular the first place in the Planning track.

Stable AllUip resolves additional clauses beyond the First Unit Implication Point (1-UIP) and keeps them whenever they are of smaller size and their LBD not greater than that of the 1-UIP clause. By monitoring the frequency of clauses which successfully pass that filter, the solver dynamically limits the amount of such extended learning attempts [45, 64]. The *Trail Saving* variant caches backtracked portions of the trail and uses them to restore decision levels during search if possible [63].

5.2.4. Relaxed newTech

The `Relaxed` fork of `MapleLCMDistCBT-DL` was first presented in SAT Race 2019 [65]. In SAT Competition 2020, its variant `newTech` showed a good performance especially on satisfiable instances. The solver received two awards, achieving the second place in the Main track and the best score on satisfiable instances.

`Relaxed` integrates short runs of the local search solver `CCAnr` through periodic export and import of assignments [65] and uses a probabilistic schedule for switching between ten phase selection modes. The `Relaxed newTech` variant uses occurrence counts of variables in unsatisfied clauses during stochastic local search runs to recalculate variable priorities for their modified branching heuristic [66].

5.2.5. Maple F2TRC

The `F2TRC` fork of `MapleLCMDistCBT` achieved the third best score on unsatisfiable instances. `F2TRC` comes with deterministic re-implementations of former winning strategies in `Maple`, e.g., by replacing time-based intervals through conflict-based intervals [67].

`F2TRC` introduces improved management of learned clauses in the tree tiers *core*, *tier2* and *local*, which are inherited from `CoMinisatPS` [43]. A dynamic size limit for the *core* tier triggers the reassignment of inactive clauses from *core* to *tier2*. To counter-act an observed starving of tier2, the conflict-based heuristic that controls demotion of clauses from *tier2* to *local* was replaced by a size-based heuristic [67].

5.3. Parallel SAT Solvers

Six teams submitted a total of ten solvers and configurations to the Parallel track. In the following, we outline the best-performing parallel solver implementations.

5.3.1. Painless MapleCOMSPS STR

`Painless-MCOMSPS-STR` integrates the solver `MapleCOMSPS` in the `Painless` parallelization framework [68, 69]. The authors submitted a `32` and a `64` threaded variant, which altogether won three awards, achieving the first place overall, the best score on satisfiable and the second-best score on unsatisfiable instances. Interestingly, the `32` threaded variant performed better than the `64` threaded variant.

`Painless` uses a generic interface to integrate a solver and abstracts away the implementation details of parallelism and concurrent data-structures. Due to this, implementations in `Painless` boil down to implementing parallelization and clause sharing strategies [68]. `Painless-MCOMSPS-STR` diversifies mainly via hard-coded configurations of the branching strategies LRB and VSIDS, and via sparse random initialization of variable polarities [70]. Two special solver instances perform concurrent clause strengthening [71] and Gaussian elimination, respectively. Regarding sharing, `Painless-MCOMSPS-STR` uses an all-to-all strategy with a fixed-size clause buffer and a dynamic LBD filter [9].

5.3.2. Plingeling

The parallel solver `Plingeling` achieved the best score on unsatisfiable instances, the second place in the overall evaluation, and the third-best score on satisfiable instances. `Plingeling` is built around the well-known `Lingeling`

and did not change since 2016. In a global master queue, `Plingeling` shares unit clauses, equivalences and short clauses with a size limit of 40 and an LBD limit of eight. `Plingeling` uses random seeds for diversification via variable polarities [55, 54].

5.3.3. ManyGlucose

`ManyGlucose` was submitted in `32` and `64` threaded variants. The `32` threaded variant won two awards in this competition, achieving the overall third place as well as the third-best score on unsatisfiable instances. `ManyGlucose` is a fork of `GlucoseSyrup` that uses strategies known from `ManySat` to achieve deterministic solver behavior [72, 73, 74].

5.3.4. Painless Maple

`Painless Maple` received the award for second-best performance on satisfiable instances. Interestingly, `Painless Maple` at the same time exhibits worst performance on unsatisfiable instances. `Painless Maple` integrates the solver `ExMapleLCMDistChronoBT` into the `Painless` parallelization framework [68]. It uses a sharing strategy in which the solvers are divided into those which only export clauses and others which import *and* export clauses and was submitted with two diversification variants `v1` and `v2`. `Painless Maple v1` diversifies via hand-crafted heuristic configurations and `Painless Maple v2` diversifies via randomized initialization of branching heuristics [9].

5.4. Massively Parallel SAT Solvers in the Cloud Track

Five teams submitted a total of six solvers and configurations in the Cloud track. In the following, we outline the best-performing massively parallel solver implementations.

5.4.1. Mallob Mono

`Mallob` is a fork of the massively parallel SAT solver `HordeSAT` [70]. `Mallob` performs dynamic load balancing through *malleable job scheduling* in case the input contains several SAT instances of varying priority. This functionality is disabled in the submitted variant `Mallob Mono`. `Mallob` uses `Lingeling-bcj` and as every 14th solver `Mallob` spawns the stochastic local search solver `YalSAT` [75]. Diversification is done via randomized sparse initialization of branching scores. `Mallob` shares clauses by organizing solvers in a binary tree in which clauses are asynchronously aggregated in a buffer which is passed along this tree from its leafs to the root. Each node performs a three-way merge of its local export buffer and the two incoming buffers. The aggregate that approaches the root of the binary tree is then broadcast to all solvers [76]. `Mallob` uses a global clause size limit and a dynamic size limit for the sharing buffer, which depends on its position in the binary tree and is larger the closer we get to the root. Clauses are sorted by their size during aggregation, such that smaller clauses are preferred over longer clauses. Duplicates are avoided by using a Bloom filter which is cleared periodically [77].

5.4.2. TopoSAT 2

`TopoSAT 2` [78] is a massively parallel SAT solver using `Glucose 3` [11]. The solver uses lock-free clause-exchange for solvers on the same machine and the message passing interface (MPI) to share clauses between machines [74]. `TopoSAT 2` strengthens clauses before export and delays clause import until the trail-size reaches a local minimum. `TopoSAT 2` diversifies via strategies used for branching, restarting, and clause forgetting [79].

5.4.3. Slime

`Slime` is built from `MapleLCMDistChronoBT` and was first submitted as a sequential solver to SAT Race 2019 with a new phase selection heuristic [80]. The new version of `Slime` submitted to the 2020 competition came with periodic randomization in geometrically increasing intervals [81]. Even thought its sequential version was unsuccessful in the Main track, the MPI-based cloud version of `Slime` achieved the third place in the Cloud track.

6. Differentiated Analysis of Main Track Results

In this section, we provide an additional analysis of the Main track results, going beyond the rankings. In particular, we focus on metrics complementing the PAR-2 score used for ranking the solvers in the actual competition.

6.1. Contributions to the Virtual Best Solver

The Virtual Best Solver (VBS) is a fictitious solver consisting of all solvers that actually participated in the competition (or a specific track) and an oracle which, when given an input instance, invokes the solver which performed the best on that instance. This way, the performance of the VBS highlights a certain upper bound on the performance achievable in principle by the participating solvers (cf. the figures in Section 4).

One can see that the VBS solves all instances that were solved by at least one solver and solves each instance in the best observed time. By quantifying how much each participating solver contributes to the performance of the VBS, we may attempt to establish which technology (as represented by the solvers) is the most important (and to what degree) in the observed state of the art in SAT solving. We consider here the following three related metrics conceptually derived from the notion of VBS.

VBS-1 "The fastest takes it all": For each solver, we count the number of times the solver was the fastest to solve an instance.

VBS-2 "Time aware, but proportional": A solver S solving an instance I in time T_I^S accrues the following fraction of a point for solving I: T_I^{VBS}/T_I^S, where T_I^{VBS} is the runtime of the best solver on I.

VBS-3 "Split the point for solving": We award each solver S solving an instance I the fraction $1/|\mathcal{S}_I|$ of points, where $\mathcal{S}_I$ is the set of solvers solving I.

VBS-1	%	Solver
46	11.5	Kissat-unsat
39	9.8	Kissat-sat
26	6.5	Kissat
18	4.5	CMS-ccnr-lsids
14	3.5	MapleCM+dist+sat2s
13	3.2	Relaxed
13	3.2	Maple-alluip-tr.
12	3.0	upGlucose3.0-PADC
10	2.5	CMS-ccnr
10	2.5	CaDiCaL-sc2020

VBS-2	%	Solver
131.5	32.9	Kissat-sat
122.6	30.6	Kissat
115.4	28.9	Kissat-unsat
92.3	23.1	CMS-ccnr-lsids
88.8	22.2	CMS-ccnr
84.1	21.0	CaDiCaL-alluip-tr.
84.0	21.0	CaDiCaL-alluip
83.0	20.7	CaDiCaL-sc2020
82.3	20.6	Relaxed-newTech
81.5	20.4	Relaxed

VBS-3	%	Solver
13.0	3.3	Kissat
11.9	3.0	Kissat-unsat
11.7	2.9	Relaxed-newTech
11.4	2.8	Kissat-sat
10.8	2.7	CaDiCaL-alluip-tr.
10.8	2.7	CaDiCaL-alluip
10.5	2.6	Relaxed
10.4	2.6	CaDiCaL-sc2020
10.3	2.6	CMS-walksat
9.6	2.4	CaDiCaL-trail

Table 6: VBS metrics for the results of the Main track, all instances. The total number of solved problems was 316. There were 48 (non-disqualified) solvers (and their configurations). Each table shows the first ten solvers sorted according to the respective metric.

We remark that the sum of VBS-1 points as well as the sum of VBS-3 points computed across all solvers is equal to the number of instances solved by at least one solver (later denoted *total*). This is obvious for VBS-1, as exactly one solver scores a point for solving an instance. In the case of VBS-3, where we discard the information about the solution times, we evenly split the one-point reward for solving an instance among those solvers which succeeded in solving the instance. In contrast, VBS-2 does not have this property as it in general distributes more than one point per instance. Similarly as VBS-1, it takes the solution time into account. Similarly as VBS-3, it does not award just the best solver on an instance. For example, a solver that uses twice as much runtime as the fastest solver on an instance receives a half a point. Furthermore, if all solvers solve an instance equally fast, each solver receives a whole point for the instance.

Table 6 provides the result of applying the just-described three metrics to the full results of the Main track. We can see that the respective leaderboards are generally dominated by `Kissat` in at least one of its configurations. VBS-1 tells us that `Kissat-unsat` was most often the fastest solver, in particular on 11.5 % of the solved instances.

The metric VBS-2 identifies `Kissat-sat` as the best solver. Its leading score of 32.9 % of the total is more difficult to interpret, though: a solver can score 32.9 % of VBS-2 total by solving 32.9 % of the solved instances in the best observed time and no others. However, we see from its VBS-1 score that

Iteration	Selected Solver	Solved	Contributes
1	Kissat-sat	264	264
2	CaDiCaL-alluip	250	22
3	f2trc-s	214	10
4	Relaxed-newTech	253	6
5	Kissat-unsat	238	4
6	Relaxed	245	3
7	CMS-walksat	243	3
8	CMS-ccnr-lsids	248	1
9	MapleCBT-DL-v3	211	1
10	DurianSat	210	1
11	exp-V-LGB-MLD-CBT-DL	194	1
	Total	–	316

Table 7: A greedy set cover of the solved instances by solvers of the Main track.

Kissat-sat solved 9.8 % of the solved instances in the best observed time (and some others). A solver can also score 32.9 % of VBS-2 total by solving all solved instances, but always being roughly three times slower than the VBS. The performance of Kissat-sat lies (clearly) somewhere between these two extremes.

Finally, according to VBS-3, the best solver is Kissat with 13.0 points, which amounts to 3.3 % of the distributed total score. The VBS-3 metric is generally the most evenly distributed one, at least among the first 10 solvers. (The last solver receives 0.9 points, which is 0.221 % of the total.) One can conclude from this that most of the benchmarks are solved by most of the well-performing solvers.

6.2. Greedy Set Cover

Another perspective on how much each solver contributes to the state of the art can be obtained by attempting to construct a sequential schedule of solvers (rather than relying on an oracle to pick one solver for each instance, as with VBS) and observing how big role each solver plays in such a schedule. Since constructing an optimal schedule tends to be computationally hard, we start here by presenting a computationally more efficient alternative—a greedy set cover approach.

With greedy set cover, we start with an empty schedule and iteratively consider each solver for the addition to the schedule obtained so far, picking the one with the highest "marginal contribution" in terms of the number of problems the new schedule will be able to solve. We demonstrate this on the actual data from the competition, again focusing in particular on the Main track results.

A greedy set cover of the solved instances by solvers of the Main track is presented in Table 7. In the first iteration, the solver which solved the highest number of instances is selected; in our case it was Kissat-sat with 264 instances as we know already from Table 4. With these 264 instances already covered, CaDiCaL-alluip is the best in further contributing to the set by additional 22

k	Time (s)	Solved	Best Schedule
1	5000	264	{ `Kissat-sat` }
2	2500	278	{ `Kissat-unsat`, `Relaxed-newTech` }
3	1666	272	{ `Kissat-unsat`, `Kissat-sat`, `CMS-ccnr-lsids` }
4	1250	262	{ `Kissat-unsat`, `Kissat-sat`, `CMS-ccnr-lsids`, `Maple-alluip-trail` }
5	1000	253	{ `Kissat-unsat`, `Kissat-sat`, `CMS-ccnr-lsids`, `Maple-alluip-trail`, `CaDiCaL-alluip` }

Table 8: Schedules that maximize the number of solved Main track instances for $k \in \{1, \ldots, 5\}$ solvers among which 5000 seconds are split uniformly.

instances in the second iteration. We can see that the further iterations tend to add very little, with the final four iterations adding one instance each. Note that each solver that managed to solve an instance uniquely (i.e., being the only solver that solved a particular instance) shows up in the greedy set cover. Indeed, the greedy set cover metric highlights solvers which are able to uniquely solve specific benchmark instances and thereby contribute to the current state of the art.

6.3. Time-Limited Schedules

The greedy set cover disregards the time it would take to execute the obtained "schedule" (of running the solvers that jointly cover all solved instances). However, we can also look at schedules that would fit in a prescribed time budget. A natural choice of the budget seems to be the original time limit of 5000 s.

To this end, by employing a brute-force approach, we first construct a sequence of schedules where the i-th schedule splits the available time of 5000 s uniformly among i solvers and solves the highest number of instances under these constraints. The results are presented in Table 8. We can see that the initial increase from 264 to 278 of "covered" instances when using two solvers instead of one (although allowing each to only use half of the time) does not continue further with additional solvers allowed, although it is still better to use three solvers in a fair time split (and cover 272 instances) than just one. Based on this observation, it is plausible that the really hard instances that were solved actually may require quite large runtime to get "cracked" by any solver and thus the advantage of adding more solvers to the schedule quickly diminishes.

We complement this "uniform time split" schedule by formulating the schedule construction problem as a MaxSMT formula and using the Z3 SMT solver [82] in its optimization mode [83] to solve it. For each solver (and its configuration) S we introduce an integer variable R^S denoting the number of seconds S runs in the new schedule. We then construct a formula with hard constraints $0 \leq R^S$ for every S and $\sum_{S \in \mathcal{S}} R^S \leq 5000$ and with one soft constraint for every instance I of the form

$$\bigvee_{S \in \mathcal{S}_I} T_I^S \leq R^S,$$

Solver	Time (s)	Solved	Contributes
Kissat-unsat	2523	219	219
Relaxed-newTech	2180	234	59
Kissat-sat	103	72	2
f2trc-DL	76	15	0
CaDiCaL-alluip	58	26	2
Maple-alluip-trail	34	12	1
MapleCM+dist+sat2s	18	18	1
CaDiCaL-sc2020	8	10	2
Total	5000	–	286

Table 9: An optimal 5000 s schedule for the Main track constructed using Z3.

k	Best k-Tuple	Score
1	{ Kissat-sat }	3926.2
2	{ Kissat-unsat, Relaxed-newTech }	3160.5
3	{ Kissat-unsat, Relaxed-newTech, CaDiCaL-2020 }	2986.4
4	{ Kissat, Relaxed-newTech, CaDiCaL-2020, Scavel01 }	2842.6
5	{ Kissat-unsat, Relaxed-newTech, CaDiCaL-2020, Scavel01, CMS-Walksat }	2757.3
6	{ Kissat-sat, Kissat-unsat, Relaxed, Relaxed-newTech, CaDiCaL-alluip-trail, f2trc-s }	2687.0
7	{ Kissat-sat, Kissat-unsat, Relaxed, Relaxed-newTech, CaDiCaL-alluip-trail, f2trc-s, CMS-Walksat}	2616.9
48	*Set of all Solvers*	2431.4

Table 10: Best performing k-tuples in terms of their VBS's PAR-2 score.

where $\mathcal{S}_I$ is the set of solvers which solved the instance I and T_I^S is the time it took solver S to solve I here rounded up to the nearest integer. (Note that while R^S are variables, i.e., unknowns, the T_I^S are known constants in the formula.)

Finding a solution which satisfies all hard constraints and as many soft constraints as possible, Z3 provided the schedule shown in Table 9 (in under two hours on a single core of a 2.30 GHz CPU). The table is sorted by R^S, the time the schedule allocates to individual solvers, with zero entries ignored. It is not clear to what degree is the obtained schedule unique and how much it relies on each solver being present and for how long. Nevertheless, it is interesting to observe the total number of problems covered, here 286, and compare it to the 278 achieved in Table 8 with the uniform split and two solvers.

As can be seen from the "contributes" column, the presence of `f2trc-DL` in the schedule is not necessary. The 15 instances this solver solves under 76 seconds were already covered by the preceding three solvers. This result is due to the fact that Z3 was not asked to produce a schedule with a minimal number of participating solvers. Indeed, allowing any other solver to run for the 76 "wasted" seconds would not increase the overall total.

6.4. Small Portfolios

The PAR-2 score of the VBS of all 48 submitted solvers in the Main track is 2431.4, which is close to 40% better than the PAR-2 score of 3926.2 of the single best solver `Kissat-sat`. Given the set of solvers S, the set of tuples of size k is defined as follows $P_k := \{T \mid T \in 2^S \wedge |T| = k\}$. We calculate the PAR-2 score for each VBS created from solver tuples in P_k. In Table 10, we report on the single best performing k-tuple $T_k \in P_k$ $(1 \leq k \leq 7)$.

Interestingly, each of the first five tuples $T_{k\leq 5}$ contains exactly one of the three `Kissat` variants. The set T_2 is composed of the two winners of the Main SAT and Main UNSAT tracks. For $i < 5$ the relation $T_i \subset T_{i+1}$ holds only under projection to base solvers due to the fluctuating variants of `Kissat`.

The composition changes more strongly in T_6. Interestingly, we now have both variants $\{$`Kissat-sat`, `Kissat-unsat`$\} \subset T_6$, and moreover it holds that $T_6 \subset T_7$. All solvers in $T_{k\leq 7}$ are among the top-performing solvers which received awards in the Main track, with the only exception of `Scavel01` $\in T_4 \cap T_5$ (cf. Table 4).

6.5. Score per Instance Family

Contributions to the VBS can be captured by clustering the instances by their family. We evaluate the runtimes of the three winning solvers of the Main track on those new families which are represented by at least 14 instances (cf. Table 1) and report their places and scores in Table 11. Interestingly, the overall best solver `Kissat-sat` is outperformed by the second and third ranked solvers `Relaxed-newTech` and `CMS-ccnr-lsids` on the *Anti-Bandwidth*, *Vlsat*, and *Influence Maximization* families by a large margin.

6.6. Similarity of Solvers

To investigate the similarity of solvers from the Main track, we define a similarity metric based on the measured runtimes. We start by removing 84 benchmarks that have not been solved by any solver. For the rest, a PAR-2 score is assigned to each instance for every solver, i.e., we set a score of 10,000

	`Kissat-sat`		`Relaxed-newTech`		`CMS-ccnr-lsids`	
Family	**Pl.**	**PAR-2**	**Pl.**	**PAR-2**	**Pl.**	**PAR-2**
Anti-Bandwidth	26	7435.6	2	5863.2	**1**	4958.7
Bit-Vector	**7**	8310.2	22	8772.4	42	9005.9
Coloring	**2**	5357.1	11	7589.3	13	7910.1
Core-Based Generator	**3**	1217.0	29	3170.7	4	1253.3
Cryptography	**4**	3451.4	6	3526.9	10	4124.2
Hypertree Decomposition	21	1173.3	**8**	1016.0	38	3188.0
Influence Maximization	22	2350.4	**3**	1838.9	11	2035.8
Tensors	**2**	1412.4	7	4411.2	11	7414.2
Timetable	41	7889.0	21	5103.8	**19**	5092.4
Tournament	4	10000.0	**2**	8715.0	4	10000.0
Vlsat	27	7149.7	5	5307.5	**1**	5026.4

Table 11: Place and PAR-2 of Winning Solvers in the Main Track per Instance Family

for unsolved instances. Each solver S is thus associated with the PAR-2 scores $S_1, \ldots, S_{316}$. The similarity of two solvers S and S', normalized to the interval $[0, 1]$ is defined as:

$$\mathsf{similarity}(S, S') = \quad 1 - \frac{\sum |S_i - S'_i|}{316 \cdot 10\,000}$$

We calculate the similarity of the 30 solvers with the best average PAR-2 score in the Main track. The results are shown in Figure 6 as a *heat map*, similar to the visualization in [12]. Additionally, the result of hierarchically clustering the solvers based on their similarity is illustrated as a *dendrogram*. The height at which two solvers or clusters are joined reflects how similar they are. For example, enabling *trail saving* in `CaDiCaL-alluip` has no impact on the runtime, resulting in a similarity above 0.999. Therefore, the two solvers are joined low in the dendrogram.

Interestingly, one identifiable large cluster consists of the `Maple`-descendants, all of which are modifications of the winners in the SAT Competition 2018 and the SAT Race 2019. The similarity within the cluster is high, except for `Scavel` and `exp_V_MLD_CBT_DL`. They form a subcluster with a lower similarity compared to the rest, but a high similarity within. In fact, the highest measured similarity, besides the aforementioned `CaDiCaL-alluip`, is observed between them. The two solvers are both based on `MapleLCMDistChronoBT-dl-v2.2`, but have different authors. This high similarity suggests that the changes they made either result in a very similar behavior or do not have a significant impact on the runtime performance.

The two configurations of `Relaxed` by *Zhang* and *Cai* use the same codebase as a lot of solvers in the `Maple`-cluster. However, the overall performance of the solver is better and closer to the `CMS`-cluster. The three `CMS` configurations differ in their implementation of stochastic local search (SLS) and have similar performance. The 2020 version of `CaDiCaL` exhibits weaker performance than the modifications based on the 2019 version and does not quite fit into any cluster.

The leftmost cluster in the heat map is comprised of other solvers originally written by *Biere*. What is interesting to note is that the `Kissat` configuration specialized for unsatisfiable instances joins the others configurations in the cluster high in the dendrogram. In fact, `Kissat-unsat` has the lowest average similarity to all other solvers in the top 30. This suggests its importance for an optimal portfolio.

6.7. Influence of Benchmark Selection on Solver Ranking

To evaluate the impact of benchmark selection on the solver ranking, we follow the experiments described in the tool suite `benchfeature` [84]. In particular, we first use random sampling to select subsets of the benchmarks used in the Main track. We start with 316 benchmarks that have been solved by at least one solver in the Main track and remove a number of benchmarks randomly. For each possible subset size (1–316) we generate 50 random samples. The solvers are assigned a new rank in ascending order of their PAR-2 score on each random

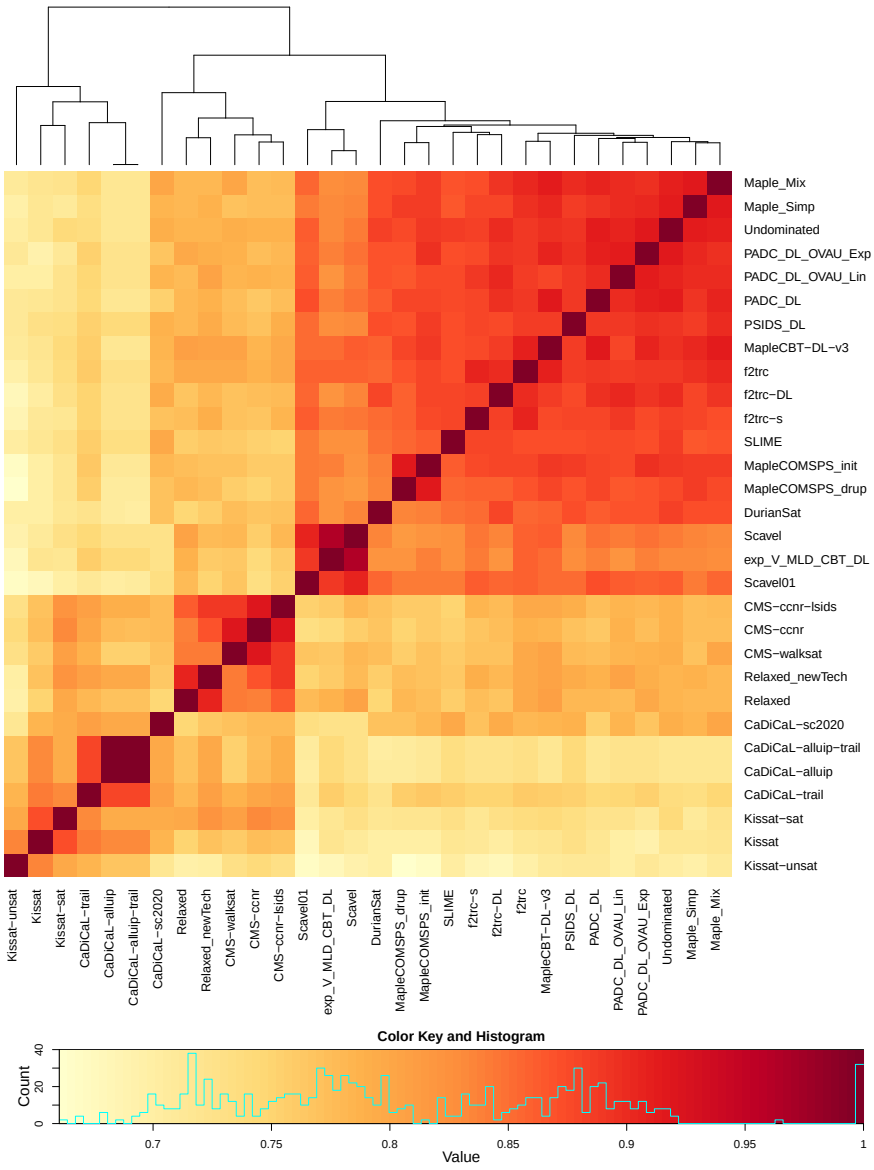

Figure 6: Heat-map and dendrogram (top) based on the runtime similarity of the solvers participating in the Main track. Darker regions mean that the solvers are more similar. A more precise relation between color and similarity-value together with a histogram of the values that appear is given at the bottom.

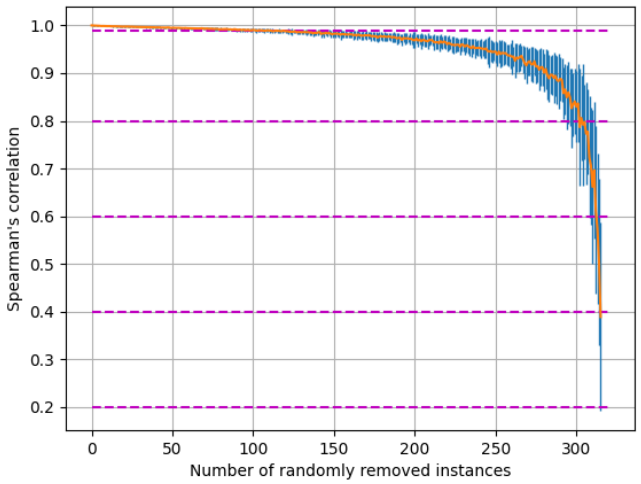

Figure 7: Mean and standard deviation of rank correlation under removal of random instances over 50 samples for each size

sample. Note that we never encounter a tie. This ranking can be seen as an estimate of the original ranking. If even relatively small random samples result in a good estimate, we draw a positive conclusion about the robustness of the ranking.

To determine how similar an estimate is to the original ranking, we calculate the *Spearman's rank correlation coefficient* of the two rankings. Spearman's rank correlation coefficient for two rankings r_1 and r_2 is defined by the following equation:

$$\rho = 1 - \frac{6 \cdot \sum_S (r_1(S) - r_2(S))^2}{n(n^2 - 1)},$$

where n is the number of solvers in the Main track (48), and the rankings $r_{\{1,2\}}(S)$ map a solver S to its rank, i.e., 1 for the best performing solver, whereas 48 is assigned to the solver with the highest average PAR-2 score.

The coefficient ρ is in the interval $[-1, 1]$, where a rank correlation of 1 means that the two rankings are equal and -1 means that one ranking is the reverse of the other. To give a better intuition for ρ, we list a few modifications to the original ranking together with the resulting rank correlation coefficient. The smallest change we can make is to switch the rank of two adjacent solvers, resulting in a high rank correlation $\rho = 0.9999$. Several small changes also result in a high rank correlation; repeating the same modification as before $n/2$ times to switch all pairs of adjacent solvers in rank still gives a value of $\rho = 0.9974$. On the other hand, switching the highest ranked solver (`Kissat-sat`) with the lowest results in a rank correlation of $\rho = 0.7602$. Moving `Kissat-sat` to the bottom of the ranking while moving every other solver up one rank gives a higher

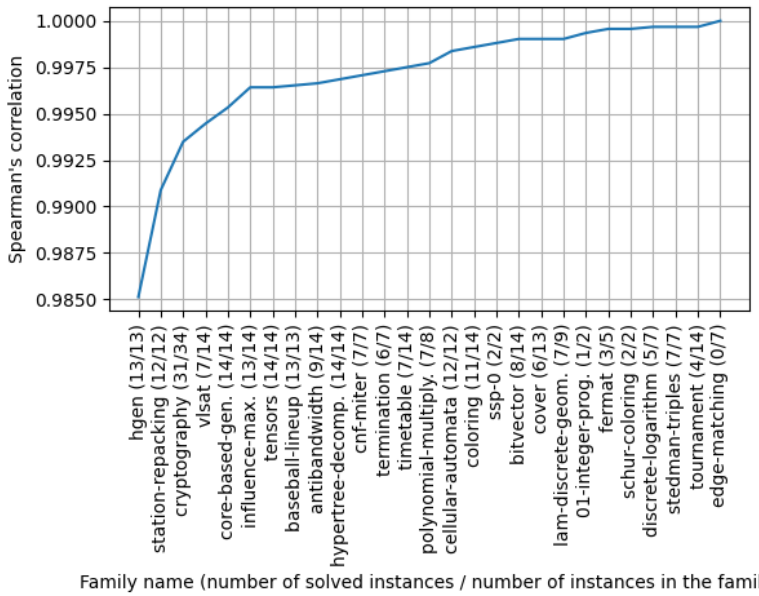

Figure 8: Change of rank correlation under removal of individual benchmark families

$\rho = 0.8776$. Doing the same to all three `Kissat` configurations (with ranks 1, 2 and 11, now 46, 47 and 48) results in $\rho = 0.6839$.

The mean and standard deviation of the computed correlation coefficients are depicted in Figure 7. The rank correlation is high even for relatively small samples. The average rank correlation drops below 0.99 only after randomly removing at least 95 benchmarks, which is 30% of the considered benchmark set. Accordingly, removing fewer benchmarks randomly has almost no effect on the ranking of the solvers. Furthermore, removing fewer than 200 (63%) benchmarks still results in an average rank correlation above 0.96. This suggests that the impact of the random selection in Algorithm 1 on the solver ranking is limited. The collected data cannot show whether all of the benchmarks originally submitted by the solver authors have a systematic bias. However, since each newly submitted benchmark family originates from a different domain and is often the result of current research, we can assume that the submitted families together are representative.

Figure 8 shows the rank correlation coefficient resulting from removing a complete benchmark family. As expected, removing a nonrandom subset can have a higher impact on the ranking even if it is small. Removing the 13 *hgen* benchmarks results in a (still high) rank correlation of 0.9895. Additionally, the ranking of the top five solvers stays the same. The individual removal of all other benchmark families results in a rank correlation above 0.99.

7. Conclusion and Prospects

The 2020 SAT Competition successfully continues the tradition of the SAT Competition series. In 2020, significant advances in SAT solvers compared to previous years were observed. Some of the more interesting observations on the winning solving strategies include the following. All winning solvers of the *Main* track periodically schedule runs of a stochastic local search (SLS) solver and import statistical information generated in unsuccessful SLS runs to reconfigure weights in their branching heuristics. As observed from the results of the *Parallel* track, it appears difficult to make proper use of more than 32 threads for SAT solving, as in some occasions the 32-threaded version of the same solver outperformed its 64-threaded counterpart. However, from the winner in the massively parallel *Cloud* track, we can learn that classical all-to-all clause-sharing can be outperformed by a more sophisticated clause-sharing architecture. It appears challenging to integrate and test sophisticated state-of-the-art methods in an incremental SAT solver and thus solvers usually disable parts of their features in the incremental use case. The winner of the *Incremental Library* track shows that it is worth integrating a full solver functionality in the incremental use case.

Prospects

In the instance selection, the author-wise balancing of satisfiable and unsatisfiable instances turned out often counterproductive as it did not lead to a more balanced overall selection of new instances. Moreover, this practice discriminated against authors who submitted solely satisfiable or unsatisfiable instances. The hardness criterion of 10 `Minisat` minutes was set higher than the hardness criterion of 1 `Minisat` minute of the "bring your own benchmarks" rule, which can be viewed as problematic. As lessons learned, in future competitions we will not aim to balance the benchmarks by satisfiability status on author level, will aim to be more consistent with the imposed *hardness* criteria for benchmark selection, and will also make sure to clearly communicate what it means for an instance to be counted as *unknown*.

The IPASIR interface facilitates the integration of SAT solvers into incremental applications. In contrast to benchmarking with instances given in the DIMACS CNF format, there are only a few benchmark applications available for the Incremental track. This calls for more community-level efforts for constructing a more diverse and well-organized repository of applications for incremental SAT solvers. Proper benchmarking and new tools for testing incremental SAT solvers may also help solver authors to deal with more complex use cases.[16]

This year, the Hack track was organized for hacks of `Glucose 3`. In the next competition, we plan on moving to the well-structured and documented state-of-the-art SAT solver `CaDiCaL`. The `CaDiCaL-alluip` solver—which is a modified `CaDiCaL`—has shown competitive performance in this competition.

[16] `https://github.com/fkutzner/IncrementalMonkey`

The first instantiation of an Application track was the Planning track organized as a one-time track in 2020. The results show that different solvers take the lead when we only evaluate for a single application, when compared to the overall Main track results. We intend to run further instantiations of the Application track. While none of the solvers that participated in the Planning track seemed particularly optimized for planning instances, we hope that in future iterations the community will pick up on the challenge of optimizing SAT solvers towards different focus applications, as a complementary challenge when compared to the generality of the Main track. In the next competition, the planned focus of Application track will be on SAT solver applications in cryptography.

The portfolio rule has been established to prohibit the participation of pure solver portfolios to stimulate the development of new codebases and to ensure fair competition among sequential solvers. The rule was challenged in this competition as it can be hurtful to cooperation in the community when solver authors use the work of other researchers as fully integrated subsystems in their own codebase. We aim to revisit and refine the portfolio rule in future instantiations of the competition to ensure that the rule does not unnecessarily hinder interesting algorithmic developments in SAT solving.

Acknowledgments

The authors thank StarExec for their support and resources to run the Main and the Planning tracks, and Amazon Web Services for their support and resources to run the Parallel and Cloud tracks.

Nils Froleyks is supported by the Austrian Science Fund (FWF) under projects W1255-N23, S11408-N23 and by the LIT AI Lab funded by the State of Upper Austria. Marijn Heule is supported by the National Science Foundation (NSF) under grant CCF-2010951. Matti Järvisalo was financially supported by Academy of Finland grants 322869 and 328718. Martin Suda was supported by the ERC Consolidator grant AI4REASON no. 649043 under the EU-H2020 programme, the Czech Science Foundation project 20-06390Y, and the project RICAIP, no. 857306 under the EU-H2020 programme.

References

[1] A. Biere, M. Heule, H. van Maaren, T. Walsh (Eds.), Handbook of Satisfiability, 2nd Edition, Vol. 336 of Frontiers in Artificial Intelligence and Applications, IOS Press, 2021.

[2] L. Simon, D. Le Berre, E. A. Hirsch, The SAT2002 Competition, Annals of Mathematics and Artificial Intelligence 43 (1) (2005) 307–342.

[3] D. Le Berre, L. Simon, The essentials of the SAT 2003 Competition, 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, Vol. 2919 of Lecture Notes in Computer Science, Springer, 2004, pp. 452–467.

[4] D. Le Berre, L. Simon, Fifty-five solvers in Vancouver: The SAT 2004 Competition, 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, Vol. 3542 of Lecture Notes in Computer Science, Springer, 2005, pp. 321–344.

[5] M. Järvisalo, D. Le Berre, O. Roussel, L. Simon, The international SAT solver competitions, AI Magazine 33 (1) (2012) 89–92.

[6] A. Balint, A. Belov, M. Järvisalo, C. Sinz, Overview and analysis of the SAT Challenge 2012 solver competition, Artificial Intelligence 223 (2015) 120–155.

[7] T. Balyo, M. J. H. Heule, M. Järvisalo, SAT Competition 2016: Recent developments, in: S. P. Singh, S. Markovitch (Eds.), Proceedings of the Thirty-First AAAI Conference on Artificial Intelligence, February 4-9, 2017, San Francisco, California, USA., AAAI Press, 2017, pp. 5061–5063.

[8] M. J. H. Heule, M. Järvisalo, M. Suda, SAT Competition 2018, Journal on Satisfiability, Boolean Modeling and Computation 11 (1) (2019) 133–154.

[9] T. Balyo, N. Froleyks, M. J. H. Heule, M. Iser, M. Järvisalo, M. Suda (Eds.), Proceedings of SAT Competition 2020; Solver and Benchmark Descriptions, Vol. B-2020-1 of Department of Computer Science Report Series B, Department of Computer Science, University of Helsinki, 2020.

[10] H. A. Kautz, B. Selman, Planning as satisfiability, in: B. Neumann (Ed.), 10th European Conference on Artificial Intelligence, ECAI 92, Vienna, Austria, August 3-7, 1992. Proceedings, John Wiley and Sons, 1992, pp. 359–363.

[11] G. Audemard, L. Simon, On the glucose SAT solver, International Journal on Artificial Intelligence Tools 27 (1) (2018) 1–25.

[12] T. Balyo, A. Biere, M. Iser, C. Sinz, SAT Race 2015, Artificial Intelligence 241 (2016) 45–65.

[13] N. Eén, N. Sörensson, Temporal induction by incremental SAT solving, Electronic Notes Theoretical Computer Science 89 (4) (2003) 543–560.

[14] A. Nadel, V. Ryvchin, O. Strichman, Ultimately incremental SAT, in: C. Sinz, U. Egly (Eds.), Theory and Applications of Satisfiability Testing - SAT 2014 - 17th International Conference, Held as Part of the Vienna Summer of Logic, VSL 2014, Vienna, Austria, July 14-17, 2014. Proceedings, Vol. 8561 of Lecture Notes in Computer Science, Springer, 2014, pp. 206–218.

[15] K. Fazekas, A. Biere, C. Scholl, Incremental inprocessing in SAT solving, in: M. Janota, I. Lynce (Eds.), Theory and Applications of Satisfiability Testing - SAT 2019 - 22nd International Conference, SAT 2019, Lisbon, Portugal, July 9-12, 2019, Proceedings, Vol. 11628 of Lecture Notes in Computer Science, Springer, 2019, pp. 136–154.

[16] A. Gupta, O. Strichman, Abstraction refinement for bounded model checking, in: K. Etessami, S. K. Rajamani (Eds.), Computer Aided Verification, 17th International Conference, CAV 2005, Edinburgh, Scotland, UK, July 6-10, 2005, Proceedings, Vol. 3576 of Lecture Notes in Computer Science, Springer, 2005, pp. 112–124.

[17] S. Gocht, T. Balyo, Accelerating SAT based planning with incremental SAT solving, in: L. Barbulescu, J. Frank, Mausam, S. F. Smith (Eds.), Proceedings of the Twenty-Seventh International Conference on Automated Planning and Scheduling, ICAPS 2017, Pittsburgh, Pennsylvania, USA, June 18-23, 2017, AAAI Press, 2017, pp. 135–139.

[18] P. Surynek, Unifying search-based and compilation-based approaches to multi-agent path finding through satisfiability modulo theories, in: S. Kraus (Ed.), Proceedings of the Twenty-Eighth International Joint Conference on Artificial Intelligence, IJCAI 2019, Macao, China, August 10-16, 2019, ijcai.org, 2019, pp. 1177–1183.

[19] R. Brummayer, A. Biere, Boolector: An efficient SMT solver for bit-vectors and arrays, in: Tools and Algorithms for the Construction and Analysis of Systems, 15th International Conference, TACAS 2009, Held as Part of the Joint European Conferences on Theory and Practice of Software, ETAPS 2009, York, UK, March 22-29, 2009. Proceedings, Vol. 5505 of Lecture Notes in Computer Science, Springer, 2009, pp. 174–177.

[20] N. Wetzler, M. Heule, W. A. J. Hunt, DRAT-trim: Efficient checking and trimming using expressive clausal proofs, in: C. Sinz, U. Egly (Eds.), Theory and Applications of Satisfiability Testing - SAT 2014 - 17th International Conference, Held as Part of the Vienna Summer of Logic, VSL 2014, Vienna, Austria, July 14-17, 2014. Proceedings, Vol. 8561 of Lecture Notes in Computer Science, Springer, 2014, pp. 422–429.

[21] T. Balyo, N. Froleyks, M. J. H. Heule, M. Iser, M. Järvisalo, M. Suda, SAT Competition 2020, `https://satcompetition.github.io/2020/` (2020).

[22] Y. K. Tan, Cake_lpr on Github, `https://github.com/tanyongkiam/cake_lpr` (2020).

[23] A. Stump, G. Sutcliffe, C. Tinelli, StarExec, a Cross Community Logic Solving Service, `https://www.starexec.org` (2012).

[24] M. Iser, C. Sinz, A problem meta-data library for research in SAT, in: D. Le Berre, M. Järvisalo (Eds.), Proceedings of Pragmatics of SAT 2015, Austin, Texas, USA, September 23, 2015 / Pragmatics of SAT 2018, Oxford, UK, July 7, 2018, Vol. 59 of EPiC Series in Computing, EasyChair, 2019, pp. 144–152.

[25] T. Balyo, M. J. Heule, M. Järvisalo (Eds.), Proceedings of SAT Competition 2017: Solver and Benchmark Descriptions, Vol. B-2017-1 of Department

of Computer Science Report Series B, Department of Computer Science, University of Helsinki, 2017.

[26] J. Rintanen, K. Heljanko, I. Niemelä, Planning as satisfiability: Parallel plans and algorithms for plan search, Artificial Intelligence 170 (12-13) (2006) 1031–1080.

[27] M. Wehrle, J. Rintanen, Planning as satisfiability with relaxed exist-step plans, in: M. A. Orgun, J. Thornton (Eds.), AI 2007: Advances in Artificial Intelligence, 20th Australian Joint Conference on Artificial Intelligence, Gold Coast, Australia, December 2-6, 2007, Proceedings, Vol. 4830 of Lecture Notes in Computer Science, Springer, 2007, pp. 244–253.

[28] T. Balyo, Relaxing the relaxed exist-step parallel planning semantics, in: 2013 IEEE 25th International Conference on Tools with Artificial Intelligence, IEEE Computer Society, 2013, pp. 865–871.

[29] J. Rintanen, Madagascar: Scalable planning with SAT, in: Proceedings of the 8th International Planning Competition (IPC-2014), 2014, pp. 1–5.

[30] N. C. Froleyks, T. Balyo, D. Schreiber, PASAR–planning as satisfiability with abstraction refinement, in: P. Surynek, W. Yeoh (Eds.), Proceedings of the Twelfth International Symposium on Combinatorial Search, SOCS 2019, Napa, California, 16-17 July 2019, AAAI Press, 2019, pp. 70–78.

[31] M. Helmert, The Fast Downward planning system, Journal of Artificial Intelligence Research 26 (2006) 191–246.

[32] D. Schreiber, D. Pellier, H. Fiorino, et al., Tree-REX: SAT-based tree exploration for efficient and high-quality HTN planning, in: J. Benton, N. Lipovetzky, E. Onaindia, D. E. Smith, S. Srivastava (Eds.), Proceedings of the Twenty-Ninth International Conference on Automated Planning and Scheduling, ICAPS 2018, Berkeley, CA, USA, July 11-15, 2019, AAAI Press, 2019, pp. 382–390.

[33] R. Monasson, R. Zecchina, S. Kirkpatrick, B. Selman, L. Troyansky, Determining computational complexity from characteristic 'phase transitions', Nature 400 (1999) 133–137.

[34] M. Janota, I. Lynce, J. Marques-Silva, Algorithms for computing backbones of propositional formulae, AI Communications 28 (2) (2015) 161–177.

[35] R. E. Bryant, Boolean analysis of MOS circuits, IEEE Transactions on Computer Aided Desisgn of Integrated Circuits and Systems 6 (4) (1987) 634–649.

[36] K. Fieger, T. Balyo, C. Schulz, D. Schreiber, Finding optimal longest paths by dynamic programming in parallel, in: P. Surynek, W. Yeoh (Eds.), Proceedings of the Twelfth International Symposium on Combinatorial Search, SOCS 2019, Napa, California, 16-17 July 2019, AAAI Press, 2019, pp. 61–69.

[37] T. Philipp, P. Steinke, PBLib – A library for encoding pseudo-boolean constraints into CNF, in: M. Heule, S. A. Weaver (Eds.), Theory and Applications of Satisfiability Testing - SAT 2015 - 18th International Conference, Austin, TX, USA, September 24-27, 2015, Proceedings, Vol. 9340 of Lecture Notes in Computer Science, Springer, 2015, pp. 9–16.

[38] R. Bloem, N. Braud-Santoni, V. Hadzic, U. Egly, F. Lonsing, M. Seidl, Expansion-based QBF solving without recursion, in: N. Bjørner, A. Gurfinkel (Eds.), 2018 Formal Methods in Computer Aided Design, FMCAD 2018, Austin, TX, USA, October 30 - November 2, 2018, IEEE, 2018, pp. 1–10.

[39] L. Ryan, Efficient algorithms for clause-learning SAT solvers, Master's thesis, Simon Fraser University (2004).

[40] 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, Vol. 2919 of Lecture Notes in Computer Science, Springer, 2004, pp. 502–518.

[41] 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.

[42] N. Manthey, Coprocessor 2.0 - A flexible CNF simplifier - (tool presentation), in: A. Cimatti, R. Sebastiani (Eds.), Theory and Applications of Satisfiability Testing - SAT 2012 - 15th International Conference, Trento, Italy, June 17-20, 2012. Proceedings, Vol. 7317 of Lecture Notes in Computer Science, Springer, 2012, pp. 436–441.

[43] C. Oh, Between SAT and UNSAT: the fundamental difference in CDCL SAT, in: M. Heule, S. A. Weaver (Eds.), Theory and Applications of Satisfiability Testing - SAT 2015 - 18th International Conference, Austin, TX, USA, September 24-27, 2015, Proceedings, Vol. 9340 of Lecture Notes in Computer Science, Springer, 2015, pp. 307–323.

[44] J. H. Liang, V. Ganesh, P. Poupart, K. Czarnecki, Learning rate based branching heuristic for SAT solvers, in: N. Creignou, D. Le Berre (Eds.), Theory and Applications of Satisfiability Testing - SAT 2016 - 19th International Conference, Bordeaux, France, July 5-8, 2016, Proceedings, Vol. 9710 of Lecture Notes in Computer Science, Springer, 2016, pp. 123–140.

[45] L. Zhang, C. F. Madigan, M. W. Moskewicz, S. Malik, Efficient conflict driven learning in boolean satisfiability solver, in: R. Ernst (Ed.), Proceedings of the 2001 IEEE/ACM International Conference on Computer-Aided Design, ICCAD 2001, San Jose, CA, USA, November 4-8, 2001, IEEE Computer Society, 2001, pp. 279–285.

[46] J. H. Liang, O. C., G. V., C. K., P. P., MapleCOMSPS, MapleCOMSPS LRB, MapleCOMSPS CHB, in: T. Balyo, M. J. Heule (Eds.), Proceedings of SAT Competition 2016: Solver and Benchmark Descriptions, Vol. B-2016-1 of Department of Computer Science Report Series B, Department of Computer Science, University of Helsinki, 2016, pp. 52–53.

[47] 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 Twenty-Sixth International Joint Conference on Artificial Intelligence, IJCAI 2017, Melbourne, Australia, August 19-25, 2017, ijcai.org, 2017, pp. 703–711.

[48] F. Xiao, M. Luo, C.-M. Li, F. Manyà, Z. Lü, MapleLRB LCM, Maple LCM, Maple LCM Dist, MapleLRB LCMoccRestart and Glucose-3.0+width in SAT Competition 2017, in: Balyo et al. [25], pp. 20–21.

[49] 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, Vol. 10929 of Lecture Notes in Computer Science, Springer, 2018, pp. 111–121.

[50] V. Ryvchin, A. Nadel, Maple_LCM_Dist_ChronoBT: Featuring chronological backtracking, in: Heule et al. [85], pp. 29–29.

[51] S. Kochemazov, O. Zaikin, V. Kondratiev, A. Semenov, MapleLCMDistChronoBT-DL, duplicate learnts heuristic-aided solvers at the SAT Race 2019, in: Heule et al. [86], pp. 24–24.

[52] 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, Vol. 5584 of Lecture Notes in Computer Science, Springer, 2009, pp. 244–257.

[53] M. Soos, R. Kulkarni, K. S. Meel, Crystalball: Gazing in the black box of SAT solving, in: M. Janota, I. Lynce (Eds.), Theory and Applications of Satisfiability Testing - SAT 2019 - 22nd International Conference, SAT 2019, Lisbon, Portugal, July 9-12, 2019, Proceedings, Vol. 11628 of Lecture Notes in Computer Science, 2019, pp. 371–387.

[54] A. Biere, LINGELING and friends entering the SAT Challenge 2012, in: A. Balint, A. Belov, D. Diepold, S. Gerber, M. Järvisalo, C. Sinz (Eds.), Proceedings of SAT Challenge 2012: Solver and Benchmark Descriptions, Vol. B-2012-1 of Department of Computer Science Report Series B, 2012, pp. 33–33.

[55] A. Biere, K. Fazekas, M. Fleury, M. Heisinger, CaDiCaL, Kissat, Paracooba, Plingeling and Treengeling entering the SAT Competition 2020, in: Balyo et al. [9], pp. 50–53.

[56] A. Biere, CaDiCaL at the SAT Race 2019, in: Heule et al. [86], pp. 8–9.

[57] C. Li, F. Xiao, M. Luo, F. Manyà, Z. Lü, Y. Li, Clause vivification by unit propagation in CDCL SAT solvers, Artificial Intelligence 279 (2020) Article 103197.

[58] B. Kiesl, M. J. H. Heule, A. Biere, Truth assignments as conditional autarkies, in: Y. Chen, C. Cheng, J. Esparza (Eds.), Automated Technology for Verification and Analysis - 17th International Symposium, ATVA 2019, Taipei, Taiwan, October 28-31, 2019, Proceedings, Vol. 11781 of Lecture Notes in Computer Science, Springer, 2019, pp. 48–64.

[59] A. Shaw, K. S. Meel, Designing new phase selection heuristics, in: L. Pulina, M. Seidl (Eds.), Theory and Applications of Satisfiability Testing - SAT 2020 - 23rd International Conference, Alghero, Italy, July 3-10, 2020, Proceedings, Vol. 12178 of Lecture Notes in Computer Science, Springer, 2020, pp. 72–88.

[60] M. Soos, S. Cai, J. Deviendt, S. Gocht, A. Shaw, K. S. Meel, CryptoMiniSat with CCAnr at the SAT Competition 2020, in: Balyo et al. [9], pp. 27–28.

[61] M. Soos, S. Gocht, K. S. Meel, Tinted, detached, and lazy CNF-XOR solving and its applications to counting and sampling, in: S. K. Lahiri, C. Wang (Eds.), Computer Aided Verification - 32nd International Conference, CAV 2020, Los Angeles, CA, USA, July 21-24, 2020, Proceedings, Part I, Vol. 12224 of Lecture Notes in Computer Science, Springer, 2020, pp. 463–484.

[62] J. Devriendt, B. Bogaerts, M. Bruynooghe, M. Denecker, Improved static symmetry breaking for SAT, in: N. Creignou, D. Le Berre (Eds.), Theory and Applications of Satisfiability Testing - SAT 2016 - 19th International Conference, Bordeaux, France, July 5-8, 2016, Proceedings, Vol. 9710 of Lecture Notes in Computer Science, Springer, 2016, pp. 104–122.

[63] R. Hickey, F. Bacchus, Trail saving on backtrack, in: L. Pulina, M. Seidl (Eds.), Theory and Applications of Satisfiability Testing - SAT 2020 - 23rd International Conference, Alghero, Italy, July 3-10, 2020, Proceedings, Vol. 12178 of Lecture Notes in Computer Science, Springer, 2020, pp. 46–61.

[64] R. Hickey, N. Feng, F. Bacchus, Cadical-trail, Cadical-alluip, Cadical-alluip-trail, and Maple-LCM-Dist-alluip-trail at the SAT Competition 2020, in: Balyo et al. [9], p. 10.

[65] X. Zhang, S. Cai, Four relaxed CDCL solvers, in: Heule et al. [86], pp. 35–36.

[66] X. Zhang, S. Cai, Relaxed backtracking with rephasing, in: Balyo et al. [9], pp. 15–16.

[67] S. Kochemazov, F2TRC: deterministic modifications of SC2018-SR2019 winners, in: Balyo et al. [9], pp. 21–22.

[68] L. L. Frioux, S. Baarir, J. Sopena, F. Kordon, PaInleSS: A framework for parallel SAT solving, in: S. Gaspers, T. Walsh (Eds.), Theory and Applications of Satisfiability Testing - SAT 2017 - 20th International Conference, Melbourne, VIC, Australia, August 28 - September 1, 2017, Proceedings, Vol. 10491 of Lecture Notes in Computer Science, Springer, 2017, pp. 233–250.

[69] J. H. Liang, O. C., G. V., C. K., P. P., MapleCOMSPS LRB VSIDS and MapleCOMSPS CHB VSIDS, in: Balyo et al. [25], pp. 20–21.

[70] T. Balyo, P. Sanders, C. Sinz, HordeSat: A massively parallel portfolio SAT solver, in: M. Heule, S. A. Weaver (Eds.), Theory and Applications of Satisfiability Testing - SAT 2015 - 18th International Conference, Austin, TX, USA, September 24-27, 2015, Proceedings, Vol. 9340 of Lecture Notes in Computer Science, Springer, 2015, pp. 156–172.

[71] S. Wieringa, K. Heljanko, Concurrent clause strengthening, in: M. Järvisalo, A. V. Gelder (Eds.), Theory and Applications of Satisfiability Testing - SAT 2013 - 16th International Conference, Helsinki, Finland, July 8-12, 2013. Proceedings, Vol. 7962 of Lecture Notes in Computer Science, Springer, 2013, pp. 116–132.

[72] G. Audemard, L. Simon, Glucose and Syrup: Nine years in the SAT Competitions, in: Heule et al. [85], pp. 24–25.

[73] G. Audemard, L. Simon, Lazy clause exchange policy for parallel SAT solvers, in: C. Sinz, U. Egly (Eds.), Theory and Applications of Satisfiability Testing - SAT 2014 - 17th International Conference, Held as Part of the Vienna Summer of Logic, VSL 2014, Vienna, Austria, July 14-17, 2014. Proceedings, Vol. 8561 of Lecture Notes in Computer Science, 2014, pp. 197–205.

[74] Y. Hamadi, S. Jabbour, L. Sais, ManySAT: a parallel SAT solver, Journal on Satisfiability Boolean Modeling Computation 6 (4) (2009) 245–262.

[75] A. Biere, CADICAL, LINGELING, PLINGELING, TREENGELING and YALSAT entering the SAT Competition 2018, in: Heule et al. [85], pp. 13–14.

[76] D. Schreiber, Engineering HordeSat towards malleability: mallob-mono in the SAT 2020 cloud track, in: Balyo et al. [9], pp. 45–46.

[77] D. Schreiber, P. Sanders, Scalable SAT solving in the cloud, in: Theory and Applications of Satisfiability Testing - SAT 2021 - 24th International Conference, Barcelona, Spain, July 4-9, 2021, Proceedings, 2021, in press.

[78] T. Ehlers, D. Nowotka, Tuning parallel SAT solvers, in: D. Le Berre, M. Järvisalo (Eds.), Proceedings of Pragmatics of SAT 2015, Austin, Texas, USA, September 23, 2015 / Pragmatics of SAT 2018, Oxford, UK, July 7, 2018, Vol. 59 of EPiC Series in Computing, EasyChair, 2019, pp. 127–143.

[79] T. Ehlers, M. Kulczynski, D. Nowotka, P. Sieweck, TopoSAT 2, in: Balyo et al. [9], pp. 60–60.

[80] O. Riveros, SLIME: A minimal heuristic to boost SAT solving, in: Heule et al. [86], pp. 38–38.

[81] O. Riveros, SLIME SAT solver, in: Balyo et al. [9], pp. 59–59.

[82] L. M. de Moura, N. Bjørner, Z3: an efficient SMT solver, in: C. R. Ramakrishnan, J. Rehof (Eds.), Tools and Algorithms for the Construction and Analysis of Systems, 14th International Conference, TACAS 2008, Held as Part of the Joint European Conferences on Theory and Practice of Software, ETAPS 2008, Budapest, Hungary, March 29-April 6, 2008. Proceedings, Vol. 4963 of Lecture Notes in Computer Science, Springer, 2008, pp. 337–340.

[83] N. Bjørner, A. Phan, νz - maximal satisfaction with Z3, in: T. Kutsia, A. Voronkov (Eds.), 6th International Symposium on Symbolic Computation in Software Science, SCSS 2014, Gammarth, La Marsa, Tunisia, December 7-8, 2014, Vol. 30 of EPiC Series in Computing, EasyChair, 2014, pp. 1–9.

[84] K. Fazekas, D. Kaufmann, A. Biere, Ranking robustness under sub-sampling for the SAT Competition 2018, presented at Pragmatics of SAT 2019 (2019). URL `http://fmv.jku.at/benchfeature/`

[85] M. J. H. Heule, M. Järvisalo, M. Suda (Eds.), Proceedings of SAT Competition 2018: Solver and Benchmark Descriptions, Vol. B-2018-1 of Department of Computer Science Report Series B, Department of Computer Science, University of Helsinki, 2018.

[86] M. J. H. Heule, M. Järvisalo, M. Suda (Eds.), Proceedings of SAT Competition 2019: Solver and Benchmark Descriptions, Vol. B-2019-1 of Department of Computer Science Report Series B, Department of Computer Science, University of Helsinki, 2019.