Overview and Analysis of the SAT Challenge 2012 Solver Competition

Adrian Balint[a,1], Anton Belov[b,2], Matti Järvisalo[c,3,*], Carsten Sinz[d,4]

[a] *Institute of Theoretical Computer Science, Ulm University, Germany. Email:* `adrian.balint@uni-ulm.de`
[b] *Complex and Adaptive Systems Laboratory University College Dublin, Ireland. Email:* `anton.belov@ucd.ie`
[c] *HIIT & Department of Computer Science, University of Helsinki, Finland. Email:* `matti.jarvisalo@cs.helsinki.fi`
[d] *Institute of Theoretical Informatics, Karlsruhe Institute of Technology (KIT), Germany. Email:* `carsten.sinz@kit.edu`

Abstract

Programs for the Boolean satisfiability problem (SAT), i.e., SAT solvers, are nowadays used as core decision procedures for a wide range of combinatorial problems. Advances in SAT solving during the last 10–15 years have been spurred by yearly solver competitions. In this article, we report on the main SAT solver competition held in 2012, SAT Challenge 2012. Besides providing an overview of how SAT Challenge 2012 was organized, we present an in-depth analysis of key aspects of the results obtained during the competition.

Keywords: Boolean satisfiability, SAT solvers, competitions, solver ranking, empirical analysis

1. Introduction

The problem of Boolean satisfiability (or propositional satisfiability, SAT) is that of determining whether a given propositional logic formula has a solution, or in other words, is satisfiable [1]. SAT is the canonical NP-complete problem [2]—and among the most fundamental ones in computer science. In addition to its theoretical importance, SAT has become a central declarative approach to formulating and solving combinatorial problems, due to major advances in robust implementations of SAT solvers. Modern SAT solvers are now routinely used in a vast number of different AI and industrial applications, of which hardware and software verification [3, 4, 5] and planning [6, 7] are classical examples. Besides using SAT solvers "directly" to solve a given problem, they are also—often iteratively—employed as core NP-solvers within procedures for more complex decision or optimization problems such as Satisfiability Modulo Theories (SMT) [8, 9, 10], Quantified Boolean Formulas (QBF) [11, 12], Answer Set Programming (ASP) [13, 14, 15, 16], Maximum Satisfiability (MaxSAT) [17, 18, 19, 20], and Minimal Unsatisfiable Subformula (MUS) extraction [21, 22, 23, 24], as well as various SAT-based counterexample-guided abstraction refinement (CEGAR) approaches [25, 26, 11, 12, 27, 28, 29, 30, 31, 32] to solving problems within and beyond NP.

The *SAT solver competitions* (see [33] for an overview, and [34, 35, 36, 37, 38] for individual competition reports) organized during the last 10–15 years have progressed SAT solver technology by providing incentives for pushing the efficiency of SAT solvers further. SAT Challenge 2012 (SC 2012, in short), the main SAT solver competition held in 2012, was organized as a satellite event to the 15th International Conference on Theory and Applications of Satisfiability Testing (SAT 2012, Trento, Italy) and stands in the tradition of the SAT Competitions[5] [33] held yearly from 2002 to 2005 and biannually starting from 2007, and the SAT-Races held in 2006, 2008, and 2010[6]. This article

[*] Corresponding author. Phone: +358 50 3199 248, Fax: +358 9 1915 1120.

[1] Author supported by the Deutsche Forschungsgemeinschaft under grant SCHO 302/9-1

[2] Author supported by Science Foundation of Ireland, PI grant BEACON (09/IN.1/I2618).

[3] Author supported by Academy of Finland under grants 132812, 251170 COIN Centre of Excellence in Computational Inference Research, 276412, and 284591.

[4] Author supported in part by the "Concept for the Future" of Karlsruhe Institute of Technology within the framework of the German Excellence Initiative.

[5] See `http://www.satcompetition.org`.

[6] See `http://fmv.jku.at/sat-race-2006` (SAT-Race 2006), `http://baldur.iti.uka.de/sat-race-2008` (SAT-Race 2008), and `http://baldur.iti.uka.de/sat-race-2010` (SAT-Race 2010), respectively.

provides an overview of SC 2012. It summarizes the rules and organization, and gives a detailed analysis of the results. This article does not give algorithmic or implementation details of the partipating solvers. Readers interested in these details are referred to [39], which includes short descriptions written by the solver developers as part of their SC 2012 submission, as well as descriptions of benchmark instances submitted to SC 2012. General information about SC 2012 is available through the competition website[7].

The rest of this article is organized as follows. We start with an overview of organizational issues of SC 2012, including descriptions of the competition rules, tracks, ranking scheme, and the computing environment used for running the competition (Section 2). We then turn to describing in detail the competition benchmark selection and generation process used for the different benchmark categories (Section 3). We also provide a review of the benchmark selection methods used for various related (constraint solving) competitions. This is followed by statistics on the participating solvers and their authors (Section 4), and a brief overview of the results of SC 2012 in terms of solver rankings (Section 5). An in-depth analysis of the results of the competition is then presented (Section 6). Before we conclude, we briefly outline some lessons learned and suggestions for improvements of future SAT competitions (Section 7),

1.1. The Boolean Satisfiability Problem in Short

For each Boolean variable x, there are two literals, x and $\neg x$. A *clause* is a disjunction of literals; a formula in *conjunctive normal form* (CNF) is a conjunction of clauses. A truth assignment α is a function from Boolean variables to $\{0, 1\}$. A clause C is satisfied by α if $\alpha(x) = 1$ for some literal $x \in C$, or $\alpha(x) = 0$ for some literal $\neg x \in C$. A CNF formula F is satisfiable if there is an assignment that satisfies all clauses in F, and unsatisfiable otherwise. The NP-complete Boolean satisfiability (SAT) problem asks whether a given CNF formula F is satisfiable.

CNF provides the standard input language for most off-the-shelf SAT solvers available today. The DIMACS input format [35], specified in 1996 as a textual representation for formulas in CNF, is now widely used and was adopted by the SAT solver competitions from the beginning. Naturally, any propositional formula can be represented in CNF using a standard linear-size encoding [40] or one of the more intricate CNF encodings developed later (see, e.g., [41]).

2. Overview of SAT Challenge 2012

2.1. Organization

The main organizers of SC 2012 were Adrian Balint (Ulm University, Germany), Anton Belov (University College Dublin, Ireland), Matti Järvisalo (University of Helsinki, Finland), and Carsten Sinz (Karlsruhe Institute of Technology, Germany). Important technical assistance related to the execution of the competition was provided by the SC 2012 Technical Assistants, Daniel Diepold (Ulm University, Germany) and Simon Gerber (Ulm University, Germany).

Open calls for participation (for both solver and benchmark submissions) were issued and advertised on various mailing lists. Researchers from both academia and industry were openly invited to submit their solvers—in either source code or binary format—to SAT Challenge 2012. We did not make submission of source code mandatory, as we also wanted to attract solvers from industrial participants, for whom disclosing the source code was not feasible. This followed the tradition of previous SAT-Races, but was different from the previous SAT Competitions that required open source solver submissions.

2.2. Participation and Evaluation

An entrant to the SAT Challenge 2012 was a SAT solver submitted in either source code or as a binary. In order to obtain reproducible results, the submitters were asked to refrain from using non-deterministic program constructs to the extent possible. Solvers making stochastic decisions during execution were required to provide a command-line option for random seed initialization.

The input and output format requirements were the same as those used for the SAT Competitions and SAT-Races in previous years, specified, e.g., in the 2011 SAT Competition rules, Sections 4.1 and 5.1–5.2 [8]. Solvers were required

[7]http://baldur.iti.kit.edu/SAT-Challenge-2012
[8]http://www.satcompetition.org/2011/rules.pdf

to provide a satisfying truth assignment for satisfiable instances. Any solver that either claimed that an unsatisfiable instance is satisfiable, or produced a truth assignment that does not satisfy a satisfiable instance, was deemed incorrect and was hence disqualified.

Solvers were assessed based on the number of instances solved within the runtime limit. If several solvers solved the same number of instances, as a secondary criterion, the cumulated runtime (CPU time for sequential solvers, wall-clock time for parallel solvers) of all solved instances was used to rank the solvers.

The organization committee reserved the right to restrict participation of a solver to certain tracks, to allow only a limited number of solvers submitted by the same person, and to submit their own systems or other systems of interest to the competition. Systems submitted by one of the organizers were not considered in the official ranking and were not eligible to win awards.

2.3. Benchmark Categories and Competition Tracks

All competition entrants had to solve a set of benchmark instances in DIMACS CNF format drawn from a larger pool of instances. This pool included benchmarks from previous SAT Competitions and SAT-Races, as well as additional instances, both submitted in response to the call for benchmarks and benchmarks generated by the organizers. The exact benchmark set was not disclosed in advance. The instances from the benchmark pool used in SC 2012 were manually categorized beforehand into three different categories: *application*, *hard combinatorial*, and *random*; Sections 3 and 7.4 provide more details on the benchmark selection and categorization.

The following competition tracks, characterized by the type of benchmarks used in the tracks, were organized.

Three *main tracks for sequential solvers:*

- *Application SAT+UNSAT*: problem encodings (both satisfiable and unsatisfiable) from real-world applications.
- *Hard Combinatorial SAT+UNSAT*: hard combinatorial problems (both satisfiable and unsatisfiable) to challenge current SAT solving algorithms (similar to the previous SAT Competitions' category "crafted").
- *Random SAT*: satisfiable k-SAT instances generated uniformly at random for different clause lengths k.

Two *special tracks*:

- One track for *Parallel Solvers*: In this track, the same problem instances as in the Application Main Track were used. However, solvers could use up to eight computing cores.
- One track for *Sequential Portfolio Solvers*: A portfolio solver is a solver that combines different (sequential) SAT algorithms. It may have, e.g., run multiple solvers in a time-slicing manner on a given SAT instance, or selected one solver out of a set of given ones (e.g., determined by a machine-learning approach based on some metric) to tackle the problem. In this track, one third of the benchmarks was selected from the application, one third from the hard combinatorial, and one third from the random category. Within each category (except Random SAT), a mixture of satisfiable and unsatisfiable instances was used.

This collection of tracks was the end-result of an attempt to find a balance between the very large number of tracks (pure SAT, pure UNSAT, SAT+UNSAT for each of the categories, Application, Crafted, and Random; and instantiation-specific special tracks) organized in the SAT Competitions, and the strict application orientation of the SAT-Races (only "industrial" Application SAT+UNSAT). Unsatisfiable instances were ruled out from the SC 2012 Random Track based on the observation that there has been little progress as well as very few solvers; the dominating solver on Random UNSAT in the SAT Competitions has repeatedly been the lookahead solver March [42]. The 2009 SAT competition and the 2010 SAT-Race had benchmark category specific special tracks for parallel solvers, while the 2011 SAT Competition included a wall-clock based timeout (in addition to CPU-time based), intuitively favouring parallel solvers. While the SC 2012 special track for parallel solvers similarly employed a wall-clock based timeout, the benchmarks in the track were evenly selected from the three main tracks. The SC 2012 special track for sequential portfolio solvers was the first of its kind.

2.4. *Computing environment*

Evaluation of solvers was performed on identical nodes of the bwGrid cluster [43] of State of Baden-Württemberg, Germany. Each cluster node had the following specification:

- Hardware: Two 4-core Intel Xeon E5440 processors (2.83 GHz with 12 MB L2 cache per CPU), 16-GB RAM.
- Software: Scientific Linux OS, kernel 2.6.18, glibc 2.5, GCC 4.1.2, javac 1.6.0, 32-bit and 64-bit executables supported.

In the three Main Tracks and the Sequential Portfolio Solvers Track, a solver could use one core of one CPU and 6 GB of main memory. Two solvers were executed in parallel on each computing node (i.e., one solver per physical CPU). A runtime limit of 900 seconds CPU time was enforced per solver and benchmark instance, with the help of the **runsolver** tool [44] also used in previous competitions. In the Parallel Solvers Track, all eight cores and 12 GB of main memory were available. Only one solver was executed on each cluster node. A runtime limit of 900 seconds wall-clock time was enforced. A total of 2.2 CPU years were used to run SC 2012—not counting the testing of solvers and the filtering of instances beforehand, which also used around the same amount of computing resources.

2.5. *EDACC: Experiment Design and Administration for Computer Clusters*

The EDACC system [45, 46] was utilized for organizing SC 2012.[9] EDACC is a distributed computing system similar to the BOINC project [47]. It was inspired by the SatEx system [48] used in earlier SAT Competitions. EDACC consists of a central database (DB), a graphical user interface, a computation client, and a web front-end. All data, including solvers and their parameters, instances, and solver output, was stored in the DB. The computation client is responsible for the execution of experiments (running the solvers on the instances). The graphical user interface was used by the organizers to create the tracks and monitor the experiments. The web front-end was used for providing an automated submission and testing platform for the submitters. A submitted solver was automatically tested on a small representative set of instances and the results were automatically reported to the submitter. The participants could then analyze the results of their solver and submit a bug-fixed version of their solver when necessary. After running the competition, the participants could analyze their results within the web front-end that provides a wide range of statistical and graphical analysis possibilities, including:

- generation of box plots and cactus plots[10] (for comparing the results of multiple solvers), scatter plots (for pair-wise comparisons of solvers), and runtime matrix plots (for analyzing the variance of solver performances);
- comparison of distributions (Kolmogorow-Smirnow test, Wilcoxon rank sum test);
- distribution and kernel density estimation;
- probabilistic domination comparison of two solvers;
- computation of rankings using different ranking schemes; and
- SOTA (*state-of-the-art-contributor*) and VBS (*virtual best solver*) analysis (for definitions of SOTA and VBS, see Sections 3.1.1 and 3.2, respectively).

EDACC offers all major functionalities to organize algorithmic competitions and is freely available online[11] under an MIT license.

[9] All results of SC 2012 can be accessed at `http://www.satcompetition.org/edacc/SATChallenge2012/experiments`.

[10] Cactus plots have been traditionally used for presenting results of solver runtime comparisons in SAT and related solver competitions as well as often in research papers focusing on SAT solving techniques. A cactus plot gives the number of solved instances (y-axis) as a function of a per-instance timeout, and are closely related to *runtime CDFs* that give the *percentage* of instances solved as a function of a per-instance timeout. Hence cactus plots directly communicate the absolute number of instances solved within different runtime timeout values, while runtime CDF give the number of instances solved relative to the size of the benchmark instance set used.

[11] `https://github.com/edacc`

3. Benchmark Selection and Generation

In this section we first briefly survey and analyze the benchmark selection methods used in solver competitions related to SC 2012. We then outline the selection (for Application and Hard Combinatorial tracks) and generation (for Random tracks) processes for benchmarks used in SC 2012, and describe the benchmark set selected for SC 2012.

3.1. *Review of Benchmark Selection Methods*

In the following, we will review bechmark selection methods applied in four related constraint solver competition series: the CADE ATP System Competitions (CASC) [49][12], the SAT Competitions [33][13], the SMT Competitions [50][14], and the ASP Competitions [51, 52, 53][15]. A common theme among the selection processes is a (sometimes implicit) two-stage selection: in the first stage the benchmarks are *ranked* according to their perceived difficulty; in the second stage the benchmarks are selected based on some combination of their rank and other properties, such as whether or not the benchmark is *new* (i.e., not used in previous competitions).

3.1.1. *CADE ATP System Competitions (CASC)*

The Automated Theorem Prover (ATP) Competitions are perhaps the longest continuously running series of system competitions in our field. The first competition close to the current form was held at the CADE-13 conference in 1996. The design of the competition is presented in [54]. The paper also contains the original methodology for the ranking of the benchmarks (and the solvers). The methodology has been slightly modified with the introduction of the concepts of *system ranking by subsumption* and the *state-of-the-art (SOTA) system* [49]. The benchmark selection methodology used in the most recent competition, CASC-J6, follows [49], and overviewed here next.

The benchmark problems for the competition are taken from the TPTP Problem Library[16], which is an online repository of problem instances used for evaluation of theorem provers in the ATP community, and which is split into thematic *categories*. The library is “frozen” prior to the start of the competition. The ATP systems submitted for the competition itself, are used to rank the benchmarks. The difficulty of benchmarks is determined by the ability of so-called *SOTA contributors* to solve them. Let $B = \{b_1, \ldots, b_n\}$ be the set (pool) of available benchmarks, and let $S = \{s_1, \ldots, s_k\}$ be the set of solvers submitted to the competition. For a solver $s_i \in S$, let $B_i \subseteq B$ denote the set of benchmarks solved by s_i within a timeout. Solver s_i is said to *subsume* solver s_j if $B_i \supset B_j$. Furthermore, s_i is a *SOTA contributor* if no other solver subsumes it (i.e., there is no j with $B_i \subset B_j$). In other words, given that the sets B_i, $1 \leq i \leq k$, form a partially ordered set (*poset*, ordered by set inclusion), SOTA contributors are the maximal elements in the poset. The *SOTA problem rating* r_i for a benchmark b_i is then

$$r_i = \frac{\text{number of SOTA contributors that failed on } b_i}{\text{number of SOTA contributors}} \quad .$$

The benchmarks are rated within their corresponding categories. In case the number of SOTA contributors is less than a certain threshold (3), the non-SOTA contributors that solve the most problems are used. The benchmarks with SOTA rating of 0 are referred to as *easy*, those with a rating strictly between 0 and 1 are *difficult*, and those with rating 1 are *unsolved*. For CASC-J6, the problems with a rating in the interval $[0.21, 0.99]$ were selected [55]. Note that this implies that the *unsolved* benchmarks are not used in the competition.

3.1.2. *SAT Competitions*

The benchmark selection process used in SAT Competitions is presented in detail on the SAT Competition 2009 website[17]. Similar to SMT-COMP (discussed later), the benchmarks for the application and the crafted categories are rated based on the performance of the top three solvers from the previous competition. In the 2011 competition, the

[12]CASC-23 (2011) is at `http://www.cs.miami.edu/~tptp/CASC/23/`
[13]SAT Competition 2011 is at `http://www.satcompetition.org/2011/`
[14]SMT-COMP 2011 is at `http://www.smtcomp.org/2011/`
[15]ASP Competition 2011 is at `https://www.mat.unical.it/aspcomp2011/FrontPage`
[16]`http://www.cs.miami.edu/~tptp/`
[17]`http://www.satcompetition.org/2009/BenchmarksSelection.html`

benchmarks were rated using "SAT 2009 reference solvers" [56]. A benchmark is rated as (i) *easy*, if it is solved within 30 seconds by all solvers; (ii) *hard*, if its not solved by any solver within the timeout value of the first phase of the competition (1200 sec); (iii) *medium*, in all other cases. The competition benchmark sets are then selected given the following constraints. *Rating distribution:* 10% easy, 40% medium, 50% hard; *new vs existing (i.e., used in previous competitions):* 45% existing, 55% new; *source distribution:* not more than 10% from the same source.

The instances in the random category of SAT Competition 2011 were taken from (uniform random) k-CNF distributions for $k = 3, 5, 7$, i.e., for each clause, k variables were drawn uniformly at random from the set of all variables, and each variable drawn was negated with probability $1/2$. The *medium* instances were taken very close to the *clause-variable phase transition ratio* [57, 58] to ensure approximately 50 % of UNSAT instances; the *large* instances were taken slightly below the phase transition. The *medium* instances where classified into SAT and UNKNOWN (probably UNSAT) using the SLS solver **gnovelty+** [59] – the instances that are solved within the timeout are classified as SAT. Note, however, that the organizers indicate that in most cases the instances were solved "within seconds". The proportion of SAT/UNKNOWN instances among the *medium* instances of the final benchmark set is 50/50. The satisfiability status of the *large* instances was presumed to be SAT, due to their clause density below the threshold (cf. Section 3.3.3). The final set of benchmarks consisted of approximately $2/3$ of *medium* and $1/3$ of *large* benchmarks.

3.1.3. SMT Competitions

The benchmark rating system used in the recent SMT Competition (SMT-COMP 2012) is described in [60]. The rating system differs from the previously discussed systems in two aspects: (i) the solvers that "finished in good standing" in the previous year's competition (SMT-COMP 2011) were used rather than the solvers submitted to the 2012 competition[18], and (ii) the solving *time* is taken into account. The problem rating r is given by $r = \frac{5\ln(1+A^2)}{\ln(1+30^2)}$, where A is the average time over all solvers, in minutes, to solve the problem. 30 is the timeout value used in the 2011 competition. If a solver fails to solve the problem within the timeout, its solving time is taken to be 30 minutes. Thus, according to [60], the rating system recognizes the fact that problems that require more time by more solvers are more difficult. The logarithm is used to mark a larger change in difficulty at smaller time values than at larger ones, and the square is used to "flatten the curve slightly at the end". Given the problem rating, the benchmarks for the competition are then selected by choosing the same number of problems uniformly at random from each of the five intervals $[0, 1]$, $(1, 2]$, $(2, 3]$, $(3, 4]$, $(4, 5]$. For each of the subdivisions of benchmarks (i.e., for the different theories), 5% of benchmarks are chosen from the random category, 10% from crafted, and the rest from the industrial applications category.

3.1.4. ASP Solver Competitions

All ASP competitions to date appear to be using the benchmark selection process proposed for the first competition, held in 2007. The process is outlined in Section 4 of [51]. After fixing a set of five solvers for evaluating benchmark hardness (details for the set of solvers used in SC 2012 are provided in Section 3.3, a benchmark is considered *suitable* if at least one solver is capable of solving it within the timeout (600 seconds), and at most three solvers can solve it within 1 second. The set of benchmarks used for ranking the solvers in the competition is constructed by choosing random benchmarks from the pool of available benchmarks, until the desired number (100 overall) of *suitable* benchmarks is obtained. Similarly to CASC, the benchmarks are ranked using the solvers submitted to the competition.

3.2. Analysis of Benchmark Ranking and Selection Methods

It is well-known that the hardness of satisfiable random k-SAT instances close to the phase transition point increases as the number of variables is increased. However, for the heterogenous sets of application and hard combinatorial instances, instance size does not correlate well with the hardness of the instances. Hence empirical testing is required in order to rate the practical hardness of such instances. In the ASP and CASC competitions, the solvers submitted to the competition are used to rate the benchmarks, so rating/selection is done *a posteriori*. For SAT competitions, including SC 2012, such an *a posteriori* rating is not computationally feasible due to the large number of

[18]The definition of "good standing" is not given, but it seems that in some cases there's a large number of such solvers (above: five).

participating solvers. So both SAT and SMT competitions revert to the evaluation of hardness using some, typically few, best-performers from previous years. As we demonstrate later, a problem that might arise in this setting is that the selected benchmark set can be (strongly) biased towards a particular solver. So the selection must be done carefully, taking this potential bias into account. However, we do want to point out that, as further discussed in Section 7, eliminating such bias is not entirely unproblematic.

In the SOTA problem rating system used in CASC, the difficulty of any particular problem is proportional to the number of SOTA contributors that fail to solve it. This allows to reduce the influence of weak systems, since the fact that many weak (i.e., non-SOTA) systems fail to solve the problem does not necessarily mean that the problem is difficult. The SMT-COMP rating system also takes into account the time used by the solvers. However, given that all problems with solving times in the range $(0.5 \cdot timeout, timeout]$ (i.e., including the unsolved problems) get the rating of $(4, 5]$, no more than 20% of difficult benchmarks (with very few unsolved ones) get into the selected problem set. As a result, in SMT-COMP 2011[19], for example, the top solver in many cases managed to solve all, or close to all, of the selected benchmarks. The rating system used in previous SAT Competitions also takes into account the solving time, though in a less refined manner than SMT-COMP. However, the difficulty of benchmarks is judged using three solvers only, chosen from the top performing solvers in the competition of the previous year. Additionally, it appears that the number of *hard* benchmarks in SAT Competitions is too large, especially for the crafted category. Table 1 shows the percentages of the benchmarks solved by the *virtual best solver* (**VBS**) and the top-3 solvers in each of the categories in the 2009 and 2011 SAT Competitions. For each benchmark instance, the running time of the virtual best solver (VBS) is defined as the running time of the fastest solver out of all solvers participating in a competition. For example, the fact that the VBS solved 77% of benchmarks in the 2011 SAT Competition crafted category implies that almost ¼ of the benchmark set was not solved by any participating solver. Given the fact that the solver ranking system used in the competition is based on the number of instances solved by at least one solver (i.e., *solution-count ranking*), ¼ of the experiments in this category were a posteriori redundant in terms of determining the final result.

Another important factor influencing benchmark rating in competitions are resource limitations, such as CPU time and memory. For competitions that can afford rating of the benchmarks using the competing systems (e.g., CASC and ASP Competition) the resource limits used in the competition itself is an obvious choice. However, when the systems used to rate benchmarks are chosen from the top performers of the previous competition (as in the SAT and SMT Competitions), the choice becomes less clear: how does one account for the possible, and expected, progress of the systems since the previous competition? Applying the resource limits of the competition itself might result in a benchmark set that is too "easy".

Clearly, one of the objectives of the benchmark selection process is to create a heterogeneous set of benchmarks. While for the Random track this objective can be achieved by varying the parameters of the instance generator, for the Application and the Hard Combinatorial tracks this issue is quite challenging. A typical approach, taken for example in CASC and SAT Competitions, is to limit the proportion of benchmarks that come from a single submitter. Previous SAT competitions enforced a limit of 10% on the fraction of benchmarks contributed by one submitter. CASC competitions use an undocumented algorithm to determine a "fair" proportion, thus making the somewhat arbitrary 10% limit more refined [61]. However, a benchmark submitter can contribute multiple benchmark sets that often differ significantly in structure and in the application context — this makes the author-based grouping of benchmarks somewhat limited. A possible way to address this problem is to group the benchmarks into manually-defined "buckets" that cluster benchmarks according to a specific application domain (see Table 2 for an example of such clustering). In [62], the authors propose to cluster the benchmarks according to their *feature-vectors*, such as those used by portfolio-based solvers (cf. [63]) to determine which solver to run on a particular benchmark. This approach, however, also has drawbacks: for one, it presumes that the feature vectors capture the structure correctly. In addition, it to some extent complicates certain analysis tasks, such as finding a solver that performs best in a specific application domain. A possible solution to this is to combine the "bucket"-based method with feature vectors—this is a topic for further research.

[19]`http://www.smtexec.org/exec/?jobs=856`

Table 1: Percentage of instances solved by the top-3 solvers and the *virtual best solver* (VBS) of the 2009 and 2011 SAT Competitions.

	SAT Competition 2011				SAT Competition 2009			
Category	VBS	1st	2nd	3rd	VBS	1st	2nd	3rd
Application	86%	72%	70%	69%	78%	70%	70%	67%
Crafted	77%	54%	52%	51%	67%	56%	55%	53%
Random	82%	68%	64%	63%	89%	71%	62%	58%

3.3. SAT Challenge 2012 Benchmark Selection

The benchmark selection process is noticeably influenced by the solution-count ranking scheme used. Under this scheme, a central requirement is that the selected set of benchmarks should contain as few as possible benchmarks that would not be solved by any submitted solver. At the same time, the set should contain as few as possible benchmarks that would be solved by all—including the weakest—submitted solvers. In order to level out the playing field for competitors who do not have the resources to tune their solvers on all benchmark sets used in the previous competitions, an additional requirement is that the selected set should contain as many benchmarks as possible that were not used in previous SAT Competitions—we refer to these benchmarks as *unused* from now on. Finally, the selected set should not contain a dominating number of benchmarks from one source (domain, benchmark submitter).

The benchmarks for the Application and the Hard Combinatorial tracks were drawn from a pool containing benchmarks that were either (i) used in the past five competitive SAT events (SAT Competitions 2007, 2009, 2011 and SAT Races 2008, 2010); (ii) submitted to these five events but not used (*unused* benchmarks); or (iii) new benchmarks submitted to SC 2012 (the descriptions for these benchmarks can be found in [39]). As elaborated in Section 7.4, the categorization of benchmarks into the Application vs. the Hard Combinatorial category is far from straightforward, and might need to be revised in the future competitions. For SAT Challenge 2012 we used a traditional categorization, following the previous SAT competitions. As with the previous SAT competitions, the benchmarks for the Random track were generated from scratch. We used a new generation and filtering procedure described in Section 3.3.3.

The empirical hardness of the benchmarks (used to rate the benchmarks for the Application and Hard Combinatorial tracks, and to filter the generated benchmarks in the Random track) was evaluated using a selection of well-performing SAT solvers from the 2011 SAT Competition. Our first attempt to select the state-of-the-art (SOTA) contributors [49], as in the CASC and ASP competitions, from the second phase of the 2011 SAT Competition failed due to the fact that *all* solvers from the second phase turned out to be SOTA contributors. Driven by the restrictions on computational resources, we ultimately selected five SAT solvers among the best performing solvers from the Application, the Crafted and the Random tracks of the 2011 SAT competition. Among the best performing solvers, preference was given to solvers that solved the highest number of benchmarks uniquely. We also tried to diversify the original code-bases of the solvers (so that, for example, not all solvers were based on Minisat). Clearly, this is not an ideal solution. However, we did not arrive at a better one within the resourcesavailable at the time The selected solvers for each track are listed in the subsequent sections.

The hardness of the benchmarks was evaluated using the same cluster on which the actual solver evaluation was run. The *rating* of a benchmark within the Application and Hard Combinatorial categories was defined as follows:

`easy` — benchmarks that were solved by all 5 solvers in under 90 seconds. These benchmarks are extremely unlikely to contribute to the (solution-count) ranking of SAT solvers in SC 2012, as all reasonably efficient solvers are expected to solve these instances within the 900-second timeout.

`medium` — benchmarks not in `easy` that were solved by all 5 solvers in under 900 seconds. Though these benchmarks are expected to be solved by the best-performing solvers, they can help to rank the weaker solvers.

`too-hard` — benchmarks that were not solved by any of the 5 solvers within 2700 seconds (3 times the timeout). Most of these benchmarks are expected to be unsolved by all competing solvers. Inclusion of (many of) such benchmarks was infeasible due to limited computational resources.

`hard` — the remaining benchmarks, i.e., the benchmarks not in `easy` or `medium` that were solved by at least one solver within 2700 seconds. These are expected to be the most useful for ranking the best-performing solvers.

Table 2: Statistics on the 600 Application benchmarks selected for SC 2012.

Rating	Count	Satisfiability Status	Count	Used/Unused	Count
easy	57	SAT	264	used	289
medium	246	UNSAT	333	unused	311
hard	291	UNKNOWN	3		
too-hard	6				

Source	Count	Contributor (new benchmarks)
2D strip packing	10	
Bioinformatics	28	
Diagnosis	59	
FPGA routing	2	
Hardware verification: BMC	11	
Hardware verification: BMC, IBM benchmarks	60	
Hardware verification: CEC	20	
Hardware verification: pipelined machines (P. Manolios)	60	
Hardware verification: pipelined machines (M. Velev)	54	
Planning	46	
Scheduling[21]	9	Peter Grossmann
Software verification: bit verification	60	
Software verification: BMC	14	
Termination	33	
Crypto: AES[21]	11	Matthew Gwynne
Crypto: DES	10	
Crypto: MD5	14	
Crypto: SHA	10	
Crypto: VMPC	13	
Miscellaneous/unknown	76	

This rating of the benchmarks is similar to the one used in the 2009 and 2011 SAT Competitions[20], except that by singling out and disregarding the benchmarks that would almost certainly not be solved by any submitted solver (these are the `too-hard` benchmarks), we aimed at increasing the effectiveness of the selected sets for ranking the solvers.

Once the hardness of the benchmarks in the pool was established, 600 benchmarks were selected from the pool. During the selection we attempted to keep the 50-50 ratio between the `medium` and `hard` benchmarks, and, at the same time, to make sure that no benchmarks from the same source were over-represented ($> 10\%$ of the selected set). Benchmarks from the sources that were over-represented in the pool were selected by uniform random sampling from each over-represented source, taking into account the benchmark hardness. Due to the shortage of available benchmarks, this latter requirement forced us to select about 10% `easy` as well as a number of `too-hard` benchmarks. The details for each selected set differ, and are provided in the following sections. Section 3.3.3 provides additional details for the generation and filtering of Random benchmarks.

3.3.1. Application Benchmarks

The five SAT solvers used to evaluate the hardness of the application instances were **CryptoMiniSat** (ver. StrangeNight2-st), **Lingeling** (ver. 587f), **glucose** (ver. 2), **QuteRSat** (ver. 2011-05-12), **RestartSAT** (ver. B95). All solvers were obtained from the SAT Competition 2011 website.[22] The set of application benchmarks was drawn from a pool of 5472 instances. Some statistics on the set of the 600 selected instances are presented in Table 2.

[20]`http://www.satcompetition.org/2009/BenchmarksSelection.html`
[21]Includes new benchmarks submitted to SC 2012. Detailed descriptions of the benchmarks are provided in [39].
[22]`http://www.satcompetition.org/2011`

Table 3: Statistics on the 600 Hard Combinatorial benchmarks selected for SC 2012.

Rating	Count	Satisfiability Status	Count	Used/Unused	Count
easy	52	SAT	368	used	284
medium	39	UNSAT	226	unused	316
hard	503	UNKNOWN	6		
too-hard	6				

Source	Count	Contributor (new benchmarks)
Automata synchronization	8	
Edge matching	32	
Ensemble computation[22]	12	Janne H. Korhonen
Factoring	43	
Fixed-shape forced satisfiable[22]	29	Anton Belov
Games: Battleship	28	
Games: Hidoku[22]	3	Norbert Manthey
Parity games	26	
Pebbling games	13	
Horn backdoor detection via vertex cover[22]	59	Marco Gario
MOD circuits	35	
Parity (MDP)	7	
Quasigroup	40	
Ramsey cube	8	
rbsat	53	
sgen[22]	47	Ivor Spence
Social golfer problem	2	
Sub-graph isomorphism	46	
Van der Waerden numbers	41	
XOR chains	2	
Miscellaneous	66	

Overall, we achieved a fairly balanced mix between `medium` and `hard` benchmarks, SAT and UNSAT benchmarks, and a reasonable distribution among the various sources. The proportion of previously used benchmarks was quite high. While undesirable, as explained in the beginning of Section 3.3, this was unavoidable due to the small number of new benchmark submissions.

3.3.2. Hard Combinatorial Benchmarks

The five SAT solvers used to evaluate the hardness of the application instances were **clasp_2.0** (ver. R4092-crafted), **SArTagnan** (ver. 2011-05-15), **MPhaseSAT** (ver. 2011-02-15), **sattime** (ver. 2011-03-02), **Sparrow UBC** (ver. SATComp11). Note that we added the SLS-based solver Sparrow UBC to the set — this is due to the fact that some of the benchmarks in the Hard Combinatorial category are "random-like". However, since this solver is incomplete, it was not considered for determining the hardness of UNSAT instances. All solvers were obtained from the SAT Competition 2011 website. The set of hard combinatorial benchmarks was drawn from a pool of 1743 instances. Table 3 presents some statistics on the set of the 600 selected instances.

Note that while the selected benchmarks are balanced well among various sources, the proportion of *hard* benchmarks is very high. This is due to the fact that, among the 1743 benchmarks in the pool, there are only 39 instances of *medium* difficulty. Approximately ⅓ of the pool consists of *easy* instances, ⅓ of *hard*, and ⅓ of *too-hard*. Thus, the selected set is more difficult for the solvers in SC 2012 than the set of Application instances. The imbalance between SAT and UNSAT instances is explained by the fact that a large proportion of the *hard* instances were satisfiable, and we were forced to take almost all *hard* benchmarks from the pool.

3.3.3. Random SAT Benchmarks

The benchmark set for the Random SAT track contains 600 instances, generated according to the uniform random k-SAT model. The instances were divided into five major classes: k-SAT for $k = 3, 4, 5, 6, 7$. Each class contains ten subclasses with varying clauses-to-variables ratios and numbers of variables. Each subclass contains 12 instances. In the following we assume that n denotes the number of variables in a k-SAT formula, m is the number of clauses, and $\alpha = \frac{m}{n}$ is the clause density. The satisfiability status of a random k-SAT instance is not known *a priori*, although for each k there is a threshold value α_k for the clause density such that all instances generated with an $\alpha < \alpha_k$ are with high probability satisfiable, and all instances generated with an $\alpha > \alpha_k$ are with high probability unsatisfiable (as m, n tend to infinity). Instances generated at the threshold ratios or near them are the most challenging for complete and local search methods [57, 58]. For large n, the best approximations of the threshold ratios are given in [64] and listed in Table 4.

Table 4: Threshold values α_k for different k

k	3	4	5	6	7
α_k	4.267	9.931	21.117	43.37	87.79

Previous SAT Competitions also used the uniform random generation model (with a small exception: the $2 + p$ instances [65] used in 2007). Note that only k-SAT instances for $k = 3, 5, 7$ were used in these competitions, and for each k only two different ratios were considered (one also containing unsatisfiable instances). For further background, we refer to [66] for details on the random instances used in the 2005 SAT Competition.

For SC 2012, we generated k-SAT instances for $k = 3, 4, 5, 6, 7$. Starting from these values, we applied the following generation model: For each k, two extreme points (α_k, n_k) and (α_{nt}, n_{nt}), with $\alpha_{nt} < \alpha_k$ and $n_{nt} > n_k$, were fixed:

- n_k: the largest number of variables a formula generated at the threshold α_k was allowed to have (such that the top three best solvers for the random category in SAT Competition 2011 were still able to solve these problems in 2700 seconds).

- α_{nt}: the largest clauses-to-variables ratio for the number of variables n_{nt} (again based on our estimate of the behavior of best known solvers).

The values of the extreme points used in SC 2012 are presented in Table 5. For each k, ten combinations of (α, n) were chosen on the line between (α_k, n_k) and (α_{nt}, n_{nt}), giving a total of 50 combinations (for the full listing, see Appendix A). The intuition behind this generation scheme is twofold. First, we wanted to allow an analysis of the influence of the clause-to-variable ratio on the performance of different solvers. On the other hand, we also wanted to keep the difficulty of the instances at a certain level, which means that by lowering the clause-to-variable ratio we have to increase the number of variables. In the previous competitions instances generated on the threshold ratio where solved by all SLS solvers and had no influence on the ranking scheme.

For each chosen combination (α, n), we generated 100 instances, resulting in a total of 1000 instances per k-value, and thus a total of 5000 instances.

We have opted to use a new generator because existent generators used in previous competitions did not meet our quality criteria; especially (i) clauses produced should be unique, and (ii) the used random number generators should pass several currently known randomness tests. Consequently, our new generator (freely available online[23]) uses the SHA1PRNG generator (part of the Sun Java implementation) that has passed all randomness tests considered by L'Ecuyer and Simard in [67, p. 22].

To filter out the unsatisfiable instances within the generated set of 5000 instances, we used the best performing solvers from the SAT Competition 2011 random track: **Sparrow2011**, **sattime2011**, **EagleUP** and **adaptG2WSAT2011**. Additionally, we used **survey propagation v 1.4** [68], **adaptiveWalkSAT** [69], **adaptive probSAT** [69] and **adapt-novelty+** from UBCSAT [70]. Each solver was run only once on each instance using a cutoff of 2700 seconds (3

[23]`http://sourceforge.net/projects/ksatgenerator/`

Table 5: The values of the extreme points (α_k, n_k) and (α_{nt}, n_{nt}) used to generate the random benchmarks.

k	α_k	n_k	α_{nt}	n_{nt}
3	4.267	2000	4.2	40000
4	9.931	800	9.0	10000
5	21.117	300	20	1600
6	43.37	200	40	400
7	87.79	100	85	200

times the SC 2012 timeout). If a instance was solved by at least one solver, it was declared satisfiable. Otherwise, the satisfiability status of the instance was marked as UNKNOWN. From each of the 50 sets of instances generated for each (α, n) combination, we randomly chose 12 instances that were determined satisfiable in the filtering phase. The resulting set of a total of 600 instances constitutes the benchmark set used in the Random SAT Track.

3.4. Solver Bias in Benchmark Selection

We now discuss potential pitfalls of the SC 2012 benchmark selection process. Recall that the ranking of the benchmarks in the benchmark pool was done using a small set of SAT solvers that were known to perform well in the previous competitions. Once the benchmarks were ranked, a subset of benchmarks was selected, based on a set of requirements, such as the distribution of hardness, satisfiability status, etc. Since the best-performing SAT solvers were used for rating, the solvers might have been expected to perform somewhat similarly across the benchmark pool. The number of instances in the pool solved by any two solvers used for ranking should be close. However, this might not be the case across the set of *selected* benchmarks. As an extreme, consider the case where only two solvers **A** and **B** are used for ranking of the benchmarks in the pool S, and assume that the set of benchmarks $S_A \subset S$ solved by **A** and the set $S_B \subset S$ solved by **B** are disjoint, and that $|S_A| = |S_B|$. If the set $S' \subset S$ of benchmarks selected for the competition is drawn uniformly from S, then we should expect that the number of instances in S' solved by **A** and **B** is similar. However, since S' might be constructed under various additional constraints (such as satisfiability status, old vs new, etc), it might be the case that $S' \subset S_A$, and so the selected set is strongly biased towards solver **A**.

To our knowledge, such *solver bias* in the selected benchmarks has not been brought to light in the existing literature and has not been investigated in prior competitions. However, this issue has the potential to significantly affect the competition's results. In SAT Challenge 2012, the problem surfaced during the analysis of the results of the Application SAT+UNSAT track (see Table 6 for the preview), where the 2011 reference solver **lingeling (SAT**

Table 6: Preview of the original results of the Application SAT+UNSAT track on the full benchmark set (detailed results are in Sec. 5) compared with the results on subset of benchmarks corrected for solver selection bias. The subset consists of 500 benchmarks. Rank changes with respect to the original results are boldfaced. Reference solvers are missing the rank numbers and are typeset in italics.

Rank	Solver	#solved	%solved	Adj. rank	Adj. #solved	Adj. %solved
1	SATzilla2012 APP	531	88.5	1	434	86.8
2	SATzilla2012 ALL	515	85.8	2	421	84.2
3	Industrial SAT Solver	499	83.2	3	417	83.4
-	*lingeling (SAT Comp. 2011 Bronze)*	488	81.3	-	389	77.8
4	interactSAT	480	80.0	**5**	401	82.0
5	glucose	475	79.2	**4**	405	81.0
6	SINN	472	78.7	6	395	79.0
7	ZENN	468	78.0	7	393	78.6
8	Lingeling	467	77.8	8	392	78.4
9	linge_dyphase	458	76.3	**15**	377	75.4
10	simpsat	453	75.5	10	387	77.4
11	glue_dyphase	452	75.3	**9**	391	78.2
-	*glueminisat (SAT Comp. 2011 Silver)*	452	75.3	-	378	75.6
-	*glucose (SAT Comp. 2011 Gold)*	451	75.2	-	392	78.4

Comp. 2011 Bronze) showed an unexpectedly high performance on the benchmark set selected for the competition. The fact that this solver was one of the five solvers used for the ranking of the benchmarks suggested a possible bias towards the reference solvers. Further analysis of the evaluation data confirmed our hypothesis. To evaluate the impact of the bias on the competition results, we corrected the bias by removing 100 benchmarks from the selected set, so that the performance of the five solvers used for ranking of the benchmarks was relatively even. The rankings were then re-calculated using this corrected benchmark set — the results are presented in Table 6. Since the (original) results of the competition have already been presented to the community, and since the selection bias did not affect the rankings of the winners, we have opted to keep the results on the original set. However, as demonstrated in Table 6, the effect of the bias was close to being dramatic: the 4th and the 5th ranked solvers switched their positions (although, since one of these two solvers (glucose) is single-engine, and the other one (interactSAT) a multi-engine, this would not have changed the final standings). The solver **linge_dyphase** dropped from the 9th place to 15th, with the solver **glue_dyphase**, previously ranked 11th, taking its place. Also, notice that on the corrected set the comparative performance of the reference solver **lingeling (SAT Comp. 2011 Bronze)** is just above the 10th ranked competition solver, as opposed to being 4th.

4. Entrants

In total, 65 solvers were submitted to SC 2012 by 55 authors from 27 research groups with 12 different countries of affiliation. Eight solvers had to be disqualified due to erroneous results they produced; thus 57 solvers participated in SC 2012.[24] The number of authors per country of origin is provided in Table 7. Apart from five solver submissions with co-authors from IBM Research, all solver authors had academic affiliations.

Table 7: Number of authors per country of affiliation.

Country	**Number of authors**
France	12
Germany	10
USA	9
Japan	6
Canada	5
Australia	3
China	3
Taiwan	2
The Netherlands	2
Austria	1
Finland	1
Israel	1

The number of solver submissions to each competition track is provided in Table 8. A clear majority of the solvers were submitted as pre-compiled binaries; only three solvers were submitted in source-code. Notice, however, that this number does not tell the true number of open-source solvers participating in SC 2012. Most of the winning solvers are publicly available after the competition.

After the submission deadline, we categorized the solver submissions into three different types of approaches based solely on the solver descriptions provided by the authors (i.e., without taking additional input from the solver submitters into account):

- *Single-engine solvers*: An implementation of a white-box[25] solver consisting of a single main algorithmic ap-

[24]The disqualified solvers are not considered further in this paper, i.e., they have been removed from all figures, tables, etc. related to the results of SC 2012.

[25]Here "white-box" refers to the fact that the inner components of logic should be available for inspection by the person submitting the solver. Submitting the binary of a solver implemented by another individual, for example, would not fit into this category.

Table 8: Number of solvers submitted to each competition track.

Track	Number of entrants
Application SAT+UNSAT	33
Hard Combinatorial SAT+UNSAT	37
Random SAT	11
Parallel	23
Sequential Portfolio	4

proach (e.g., conflict-driven clause learning [71, 72, 73, 74, 75], lookahead [76], stochastic local search [77]). We note that preprocessors are not considered individual algorithms, and are allowed in this category as well.

- *Interacting multi-engine approaches*: An implementation that combines multiple different solver implementations / different types of algorithms in an interleaving fashion, possibly (but not necessarily) with information exchange between the different executed solvers.
- *Portfolio approaches*: Based on a predefined set of solver implementations. For each input, execute one or more solvers in a black-box fashion. Solver selection may be based e.g. on pre-trained machine learning models.

We acknowledge that this categorization is somewhat rough, and only one possible way of categorizing the solver submissions; this issue is discussed further in Section 7.5.

5. Overview of Results

This section provides the solver rankings of SAT Challenge 2012, highlighting the best-performing solvers that were awarded, as well as a discussion of the detailed results of each track. Full rankings are provided in Appendix B. The complete results can be found online[26].

5.1. Awarded Solvers

For each competition track and solver category, the best-performing solvers, with a listing of their developers and the main algorithmic approach applied by the solvers, were:

Main Track: Application SAT+UNSAT

Best Single-Engine Solver in the Application Track: **glucose** [78]
Authors: Gilles Audemard and Laurent Simon.
Type of algorithm: Conflict-driven clause learning (CDCL), based on Minisat [73].

Best Interacting Multi-Engine Approach in the Application Track: **Industrial Satisfiability Solver (ISS)**
Authors: Yuri Malitsky, Ashish Sabharwal, Horst Samulowitz, and Meinolf Sellmann.
Type of algorithm: Hybridization of systematic (mostly CDCL-based) and local search solvers, using meta-restarts and interleaved preprocessing.

Best Portfolio Approach in the Application Track: **SATzilla 2012 APP**
Authors: Lin Xu, Frank Hutter, Jonathan Shen, Holger H. Hoos, and Kevin Leyton-Brown.
Type of algorithm: Portfolio including a large range of both systematic and local search solvers. Solver selection based on pre-trained cost-sensitive classification models [63].

Main Track: Hard Combinatorial SAT+UNSAT

Best Single-Engine Solver in the Hard Combinatorial Track: **clasp-crafted** [15]
Authors: Benjamin Kaufmann, Torsten Schaub, and Marius Schneider.
Type of algorithm: CDCL, native implementation.

[26] `http://www.satcompetition.org/edacc/SATChallenge2012/`

Best Interacting Multi-Engine Approach in the Hard Combinatorial Track: **interactSAT_c**
Author: Jingchao Chen.
Type of algorithm: Hybridization of systematic (CDCL solver clasp, lookahead solver **March**) and local search (**sparrow2011**) solvers.

Best Portfolio Approach in the Hard Combinatorial Track: **SATzilla 2012 COMB**
Authors: Lin Xu, Frank Hutter, Jonathan Shen, Holger H. Hoos, and Kevin Leyton-Brown.
Type of algorithm: Portfolio including a large range of both systematic and local search solvers. Solver selection based on pre-trained cost-sensitive classification models.

Main Track: Random SAT

Best Solver in the Sequential Random SAT Track: **CCASat**
Authors: Shaowei Cai, Chuan Luo, and Kaile Su.
Type of algorithm: Local search, employing "configuration checking with aspiration" (CCA) [79].

Parallel Track: Application SAT+UNSAT

Best Solver in the Parallel Track: **pfolioUZK**
Authors: Andreas Wotzlaw, Alexander van der Grinten, Ewald Speckenmeyer, and Stefan Porschen.
Type of algorithm: Portfolio including a range of both systematic and local search solvers, inspired by the ppfolio portfolio. Simple solver selection, based on a combination of "uniformity-based selection" (depending on whether all clauses of a formula have exactly the same length) and a static solver configuration setup, allocating one solver to each available CPU core.

Sequential Portfolio Track

Best Solver in the Sequential Portfolio Track: **SATzilla 2012 ALL**
Authors: Lin Xu, Frank Hutter, Jonathan Shen, Holger H. Hoos, and Kevin Leyton-Brown.
Type of algorithm: Portfolio including a large range of both systematic and local search solvers. Solver selection based on pre-trained cost-sensitive classification models.

For more details on the algorithmic and implementation-level details of the individual solvers, we refer the reader to the solver descriptions collection released as a technical report [39].

5.2. Detailed Results

Cactus plots[27]—representing for each solver the number of instances that can be solved (x-axis) within a given timeout (y-axis)—for each competition track, including for clarity only the Top-10 best performing solvers and the reference solvers, are provided in Figures 1–5. Similarly, numerical data for the Top-10 solvers and the reference solvers is provided in Tables 9–13, and the full tables are provided in Appendix B.

In the result tables, solvers are ordered by the number of solved instances, and ties are broken taking the cumulative runtime into account. We also provide a second ranking (**T-Rank**), where only solvers of the same type (portfolio, single-engine, etc.) are taken into consideration. Besides the number of solved instances (**#solved**), we also give the percentage of solved instances (**%solved**), the cumulative runtime over all solved instances (**time (cum.)**), as well as the median runtime over all solved instances (**time (med.)**).

5.2.1. Application SAT+UNSAT Track

The Application SAT+UNSAT Track was dominated by portfolio and multi-engine solvers. They took the first four places (not taking the reference solver **lingeling** from SAT Competition 2011 into account). The winner, **SATzilla2012**

[27]We use cactus plots instead of the almost equivalent empirical cumulative distributions functions for better visualization. This is because the prime information of interest, the number of solved instances, should be plotted on the axis that has most space (in our plots the x-axis). As the difference in the number of solved instances is very small, we use a linear scale instead of a logarithmic scale. Note that EDACC offers the possibility to also plot the cumulative distributions and also supports logarithmic scales.

APP, even comes close to the *virtual best solver*. This suggests that high variability and adaptability in solver heuristics is extremely advantageous.

glucose was the best single-engine solver, exhibiting clearly improved performance over its 2011 version, now solving 79.2% of the instances compared to only 75.2% in 2011. It is also interesting to observe that the median runtime of the 2011 version of **glucose** is much lower than that of any other solver. The exact reason for this behavior is unclear, and may be caused by a different adjustment of solver heuristics compared to previous versions (see [39, p. 21]).

As reference solvers we have selected the best solvers from the 2011 SAT Competition, as well as additional solvers of general interest (such as the well-known and popular solver **minisat**).

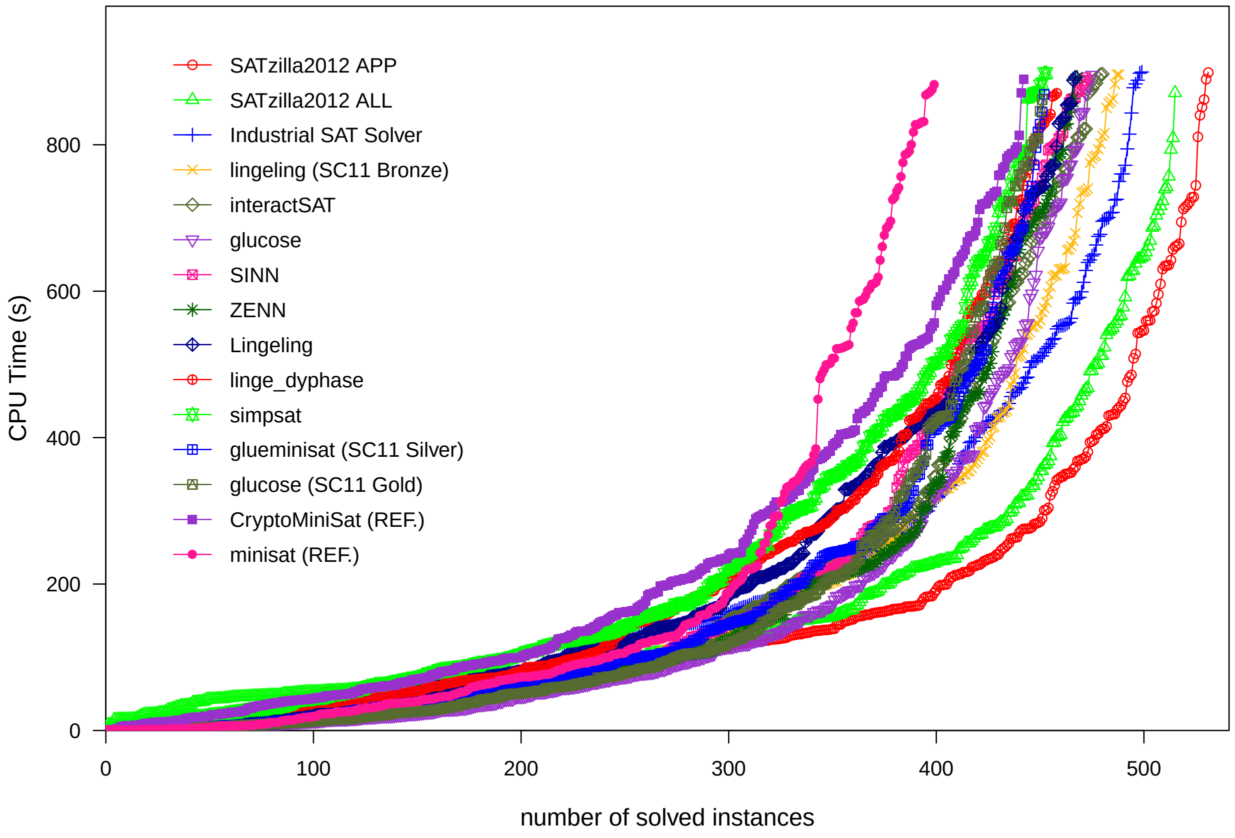


Figure 1: Cactus plot for the Application track. The total number of benchmarks in the track was 600.

5.2.2. *Hard Combinatorial SAT+UNSAT Track*

Similarly to the Application SAT+UNSAT Track, this track was dominated by portfolio and multi-engine solvers (see Figure 2 on Page 17 and Table 10 on Page 18). The best single-engine solver, **clasp-crafted**, comes in only on place seven, solving approximately 18% less instances than the best solver, **SATzilla2012 COMB**.

There is a quite considerable gap between the second and third best solver (over 12% in number of solved instances), as well as between the first five and the following solvers (over 8% in number of solved instances). Perhaps even more severe is the difference in median runtime between the first five and the sixth best solver, with a factor of almost 3.5.

Table 9: Results for Application SAT+UNSAT main track: Top-10 and reference solvers

Solver type	Rank	T-Rank	Solver	#solved	%solved	time (cum.)	time (med.)
vbs	-	-	*Virtual Best Solver (VBS)*	568	94.7	56528	30.3
portfolio	1	1	SATzilla2012 APP	531	88.5	85194	114.0
portfolio	2	2	SATzilla2012 ALL	515	85.8	86638	122.2
multi-engine	3	1	Industrial SAT Solver	499	83.2	93705	160.2
reference	-	-	*lingeling (SAT Comp. 2011 Bronze)*	488	81.3	84715	135.3
multi-engine	4	2	interactSAT	480	80.0	87676	152.5
single-engine	5	1	glucose	475	79.2	71501	114.4
single-engine	6	2	SINN	472	78.7	86302	146.4
single-engine	7	3	ZENN	468	78.0	74019	124.7
single-engine	8	4	Lingeling	467	77.8	91973	185.5
single-engine	9	5	linge_dyphase	458	76.3	90192	204.4
single-engine	10	6	simpsat	453	75.5	95737	222.0
reference	-	-	*glueminisat (SAT Comp. 2011 Silver)*	452	75.3	68818	145.7
reference	-	-	*glucose (SAT Comp. 2011 Gold)*	451	75.2	62424	77.8
reference	-	-	*CryptoMiniSat*	442	73.7	95035	240.6
reference	-	-	*minisat*	399	66.5	65633	189.5

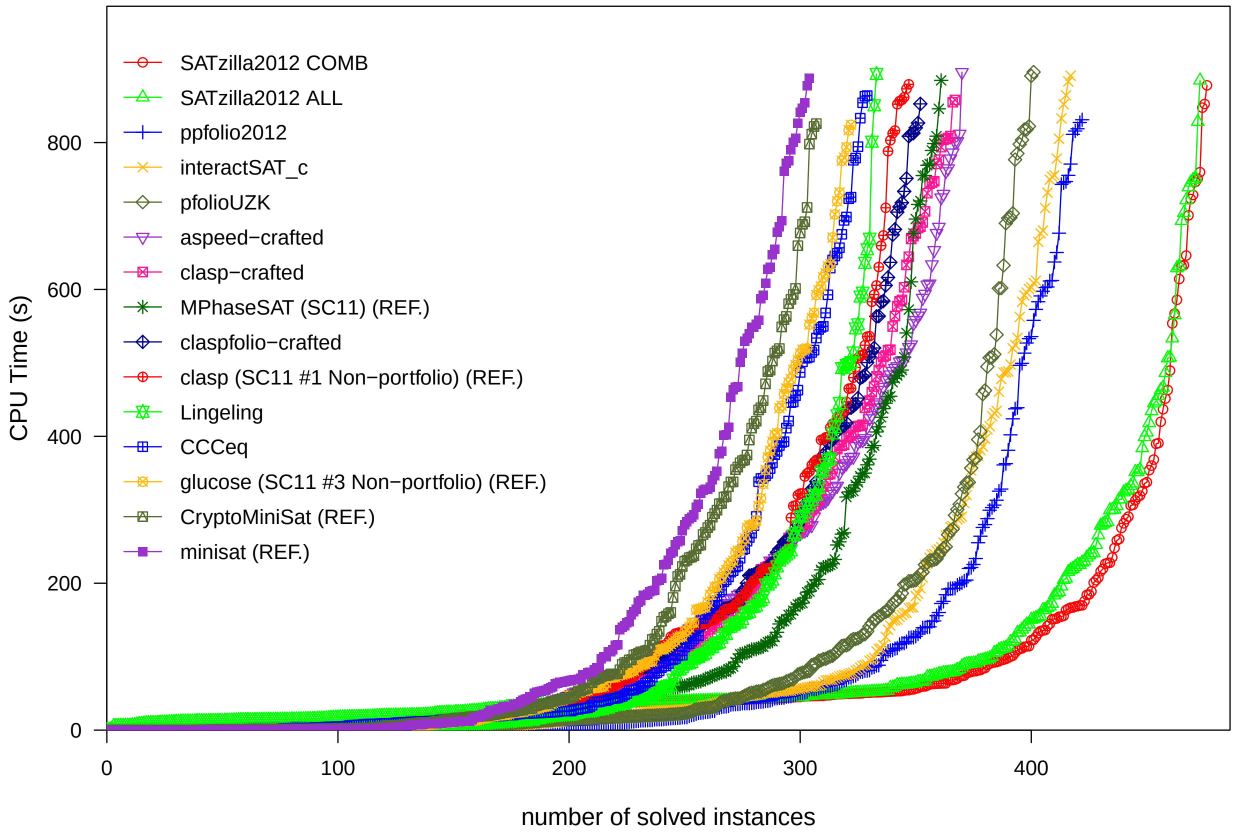


Figure 2: Cactus plot for the Hard Combinatorial track. The total number of benchmarks in the track was 600.

Table 10: Results for Hard Combinatorial SAT+UNSAT main track: Top-10 and reference solvers

Solver type	**Rank**	**T-Rank**	**Solver**	**#solved**	**%solved**	**time (cum.)**	**time (med.)**
vbs	-	-	*Virtual Best Solver (VBS)*	529	88.2	24848	1.3
portfolio	1	1	SATzilla2012 COMB	476	79.3	38108	45.4
portfolio	2	2	SATzilla2012 ALL	473	78.8	41765	45.2
portfolio	3	3	ppfolio2012	422	70.3	35784	50.5
multi-engine	4	1	interactSAT_c	417	69.5	40313	56.6
portfolio	5	4	pfolioUZK	401	66.8	34187	77.7
portfolio	6	5	aspeed-crafted	370	61.7	49239	269.3
single-engine	7	1	clasp-crafted	367	61.2	49317	277.0
reference	-	-	*MPhaseSAT (SAT Comp. 2011)*	361	60.2	35006	172.6
portfolio	8	6	claspfolio-crafted	352	58.7	42522	296.7
reference	-	-	*clasp (SAT Comp. 2011 #1 Non-portfolio)*	347	57.8	41038	322.2
single-engine	9	2	Lingeling	333	55.5	27313	291.0
multi-engine	10	2	CCCneq	329	54.8	36311	454.6
reference	-	-	*glucose (SAT Comp. 2011 #3 Non-portfolio)*	322	53.7	34546	515.4
reference	-	-	*CryptoMiniSat*	307	51.2	32414	682.9
reference	-	-	*lingeling*	305	50.8	29095	801.4
reference	-	-	*minisat*	304	50.7	39055	843.9
reference	-	-	*Sparrow2011*	217	36.2	19972	900.0
reference	-	-	*EagleUP (SAT Comp. 2011)*	34	5.7	997	900.0

5.2.3. *Random SAT Track*

In the Random SAT Track, portfolio solvers also fared quite well, but were beaten by a new single-engine solver, **CCASat**. The local-search solver **CCASat** employs *configuration checking* that originates from local search algorithms for the Minimum Vertex Cover problem, and combines it with the aspiration mechanism from tabu search. This new algorithm solves over 30% more instances in the competition than the second best solver. Compared to the best solver of 2011, it solved almost 40% more instances. The median runtime of **CCASat** is also much lower than that of the other solvers. The success of **CCASat** also impressively shows that improving core algorithms is of prime importance, being even more successful than competing portfolios.

Table 11: Results for Random SAT main track

Solver type	**Rank**	**T-Rank**	**Solver**	**#solved**	**%solved**	**time (cum.)**	**time (med.)**
vbs	-	-	*Virtual Best Solver (VBS)*	558	93.0	72841	39.2
single-engine	1	1	CCASat	423	70.5	76206	218.8
portfolio	2	1	SATzilla2012 RAND	321	53.5	80796	714.4
portfolio	3	2	SATzilla2012 ALL	306	51.0	83273	845.6
reference	-	-	*Sparrow2011 (SAT Comp. 2011 Gold)*	303	50.5	76396	876.1
reference	-	-	*EagleUP (SAT Comp. 2011 Bronze)*	283	47.2	83787	900.0
single-engine	4	2	sattime2012	269	44.8	80345	900.0
portfolio	5	3	ppfolio2012	253	42.2	70903	900.0
reference	-	-	*sattime2011 (SAT Comp. 2011 Silver)*	236	39.3	67237	900.0
portfolio	6	4	pfolioUZK	230	38.3	55584	900.0
single-engine	7	3	ssa	150	25.0	35316	900.0
single-engine	8	4	gNovelty+PCL	123	20.5	40240	900.0
single-engine	9	5	BossLS	103	17.2	18934	900.0
single-engine	10	6	sparrow2011-PCL	81	13.5	22788	900.0

5.2.4. *Special Tracks*

The SC 2012 special tracks were the Parallel Track Application SAT+UNSAT and the Sequential Portfolio Track.

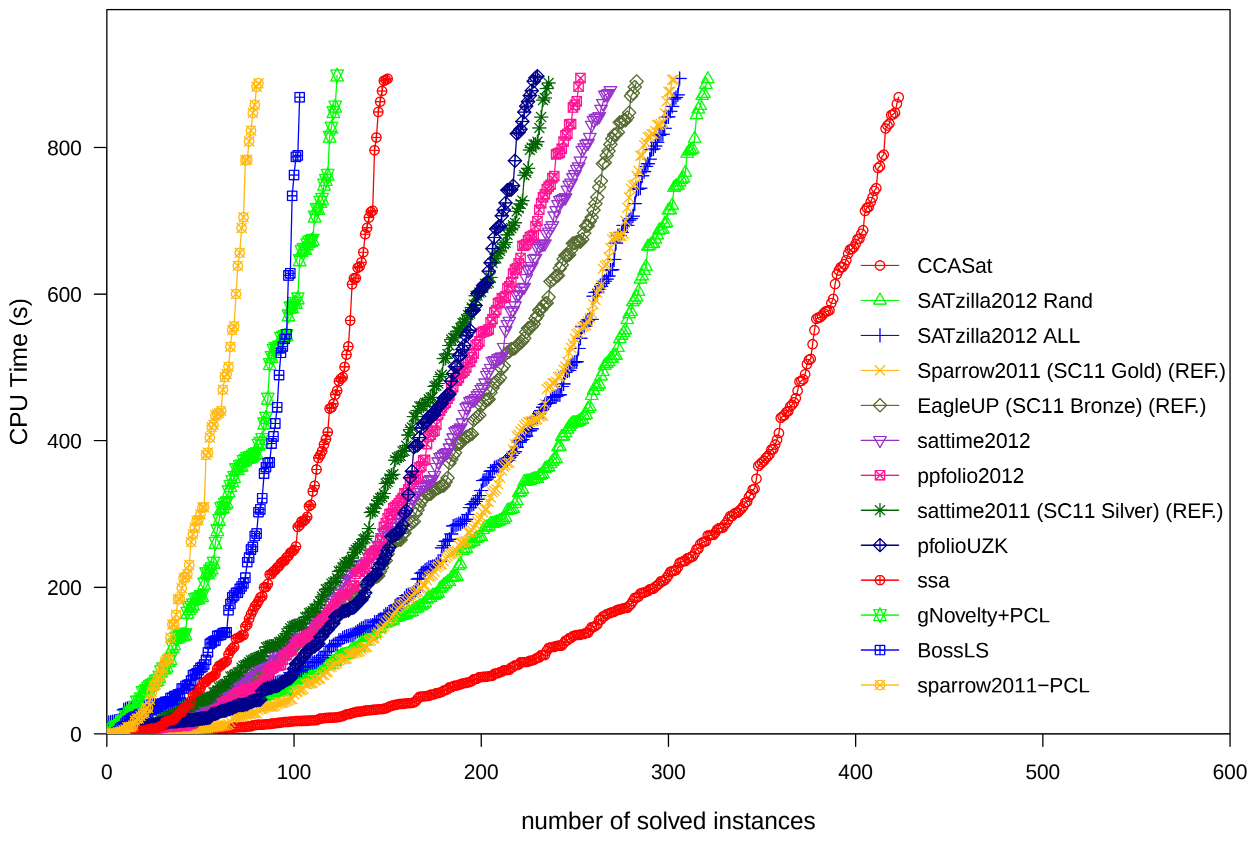


Figure 3: Cactus plot for the Random SAT track. The total number of benchmarks in the track was 600.

In the Parallel Track, concurrent and parallel solving algorithms could make use of all eight cores of a cluster node. Here, the runtimes are given as wall-clock times, which means that each solver had approximately eight times the compute resources available compared to the (sequential) Application SAT+UNSAT Track.

One would expect that—by having more compute power available—the solvers are much stronger now and solve more instances. However, this turned out not to be the case. The best performing solver in the Parallel Track, **pfolioUZK**, solved exactly the same number of instances (531) as the best solver in the sequential Application SAT+UNSAT Track, **SATzilla2012 APP**—although, the median wall-clock runtime of **pfolioUZK** is 65% lower than that of **SATzilla2012 APP**.

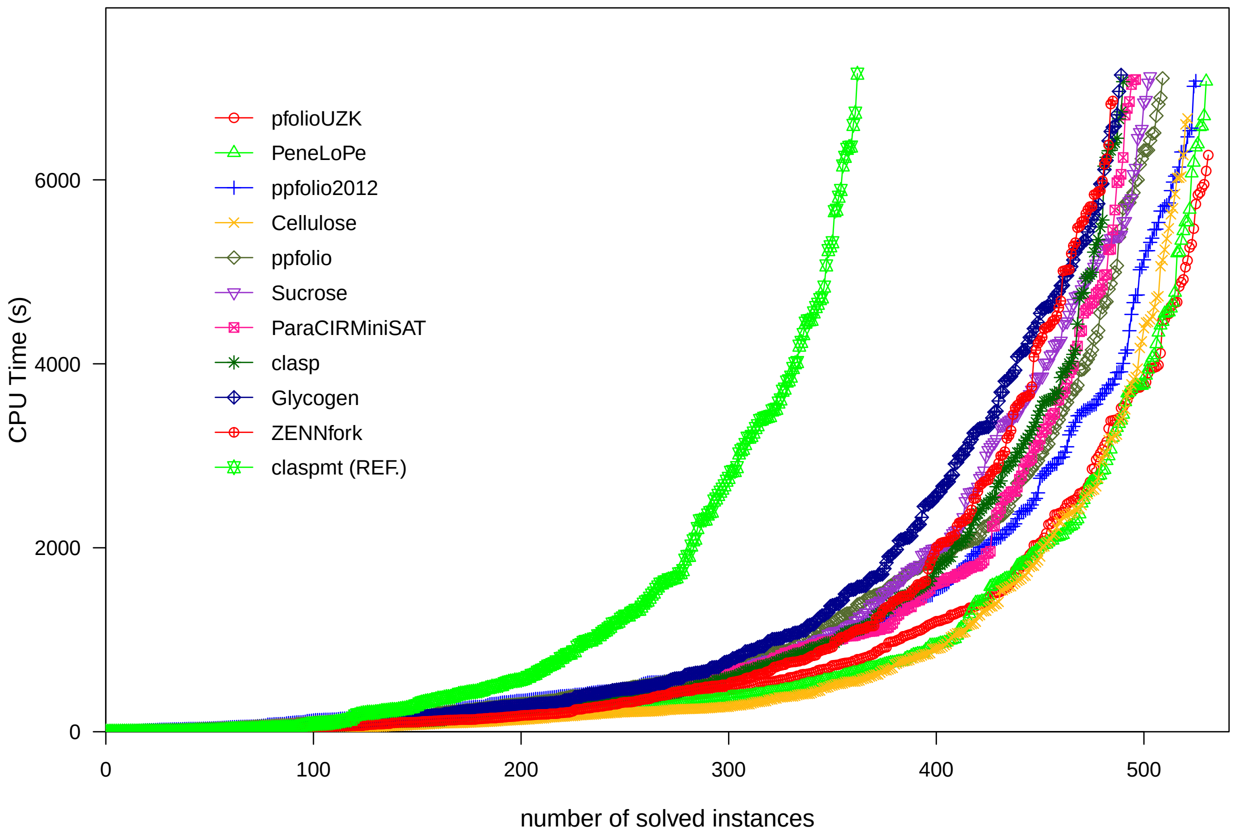


Figure 4: Cactus plot for the Parallel Application track. The total number of benchmarks in the track was 600.

Comparing the sequential and parallel versions of **pfolioUZK**, the improvement by the additional CPU power becomes more obvious. Whereas the sequential version of **pfolioUZK** ranked 16th in the sequential Application SAT+UNSAT Track, solving 404 instances, the parallel version fared much better, solving 531 instances. The median solving time also improved by a factor of five. This can be assumed to be close to the expected gain of 700% on the instances solved by both versions of the solver.

Unfortunately, among the Top-10 solvers from the sequential Application SAT+UNSAT Track, only one (**ZENN**) participated (in a slightly different version, **ZENNfork**) in the parallel track, which makes an assessment of the state-of-the-art in parallel SAT solver technology harder. It is also quite surprising that the parallel version, **ZENNfork**, solved only 3.5% more instances than the sequential **ZENN** solver.

The solver **claspmt** ranked 5th in the 2011 SAT Competition Application Track. **clasp** is the follow-up version of **claspmt** of 2012, in which multi-threading support has been built in. **clasp** solved 35% more instances, which, for this solver at least, shows the considerable progress made over one year.

Table 12: Results for Parallel Application track: Top-10 and reference solvers. The total number of benchmarks in the track was 600.

Solver type	**Rank**	**Solver**	**#solved**	**%solved**	**time (cum.)**	**time (median)**
vbs	-	*Virtual Best Solver (VBS)*	576	96.0	39670	19.6
parallel	1	pfolioUZK	531	88.5	72390	69.1
parallel	2	PeneLoPe	530	88.3	62967	54.4
parallel	3	ppfolio2012	525	87.5	78833	91.4
parallel	4	Cellulose	521	86.8	53705	42.0
parallel	5	ppfolio	509	84.8	75400	91.3
parallel	6	Sucrose	503	83.8	76120	80.7
parallel	7	ParaCIRMiniSAT	496	82.7	63497	86.7
parallel	8	clasp	490	81.7	62424	77.8
parallel	9	Glycogen	489	81.5	76241	97.1
parallel	10	ZENNfork	485	80.8	73808	89.1
reference	-	*claspmt*	362	60.3	56435	352.3

Table 13: Results for Sequential Portfolio Track

Solver type	**Rank**	**Solver**	**#solved**	**%solved**	**time (cum.)**	**time (median)**
vbs	-	*Virtual Best Solver (VBS)*	484	80.7	64805	65.5
portfolio	1	SATzilla2012 ALL	433	72.2	68033	139.9
portfolio	2	ppfolio2012	370	61.7	65598	376.0
portfolio	3	pfolioUZK	362	60.3	69485	391.6

In the Sequential Portfolio Track, one third of the competition benchmark instances came from each of the three categories Application, Hard Combinatorial, and Random. Unfortunately, only three solvers participated in this "mixed" track, even though we believe the track would have been rather interesting for showcasing the potential of portfolio solvers as generic SAT solvers. **SATzilla2012 ALL**, which is optimized for such a mixed set of problem instances, could outperform the other two solvers, solving 17% more instances than the second best solver **ppfolio2012**. It is noteworthy that the generic mixed heuristics of **SATzilla2012 ALL** performed quite well also in the tracks specialized on only one category of benchmarks, where it consistently ranked just one place behind the more specialized versions of **SATzilla2012**, namely **SATzilla2012 APP**, **SATzilla2012 COMB**, and **SATzilla2012 RAND**.

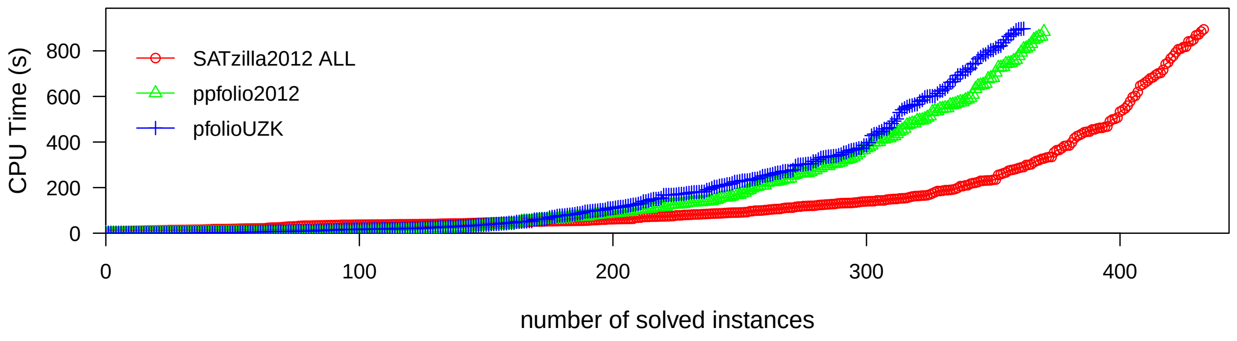


Figure 5: Cactus plot for the Sequential Portfolio track. The total number of benchmarks in the track was 600.

6. Analysis

In this section we provide a more detailed analysis of the results of SC 2012.

6.1. Impact of Ranking Schemes

The solution-count ranking scheme (SCR) used in SC 2012 ranks solvers according to the number of solved instances. Ties are broken by the cumulative CPU time. The SCR scheme has been used in the SAT Competitions and SAT-Race since 2009, replacing the purse-based ranking (PBR) [80] after a questionnaire about the preferred ranking scheme was done by the organizers of the 2009 SAT Competition. In addition to being easy to understand, SCR defines a transitive relation between solvers, i.e. the relative ranking of two solvers cannot be influenced by a third solver, which is not true for PBR.

However, SCR clearly also has disadvantages. One is that the produced rankings can be highly dependent of the enforced timeout. Another is that the runtime of the solvers plays only a marginal role within the ranking and is taken into account only in case of ties in the number of instances solver: a solver S that solves n instances will lose against a solver S' that solves $n+1$ instances independent of the average runtime of S and S'. Due to this, SCR has been criticized in multiple papers [80, 81]. One proposed alternative is the careful-ranking scheme (CR) [80] which is based on pairwise comparisons of solvers inspired by statistical tests. However, the CR scheme does not define a transitive relation. Another possibility is to rank solvers by their *penalized average runtime (PAR)* [82]. This idea gives a family of PARx ranking schemes over different penalization factors x. Denoting the imposed timeout limit by t, the PARx measure computes the average runtime by counting timeouts as having value $x \cdot t$. The parameter x controls the balance between the average runtime (over the successful runs) and the number of solved instances. Notice that SCR is equivalent to PARx in the limit $x \to \infty$. Therefore, it is to be expected that the correlation between PARx and SCR approaches 1 as x grows.

Here we study correlations between the rankings produced by the ranking schemes SCR, CR, and PARx for $x = 1, 2, 10$ using the SC 2012 data. Two possible correlation coefficients of interest are Kendall's τ and Spearman's ρ coefficients. Whereas the former measures the relative number of the number of disagreements and agreements between two rankings, the latter takes also into account the absolute differences in the disagreements. Intuitively, if two ranking methods rank a solver very differently, Spearman's ρ coefficient will be lower than Kendall's τ. One should note here that small differences between two rankings are not relevant as long as the big picture remains unchanged, i.e, a solver ranked among the best-performing solvers by one scheme should not be ranked as one of the worse/worst-performing solvers). Keeping this in mind, we use Spearman's ρ correlation coefficient for our analysis.

The results are shown in Tables 14, 15, and 16 for the Application, Hard Combinatorial, and Random SAT main track data, respectively. The SCR scheme correlates well with the PARx rankings; this correlation increases with x, as the higher the value of x, the more the unsuccessful runs are penalized. Even when runs are penalized with a factor of 10 (PAR10), some solvers could compensate this penalization with overall short runtimes, ranking higher under PAR10 than under SCR. The CR scheme (using noise of 5 seconds) correlates better with PAR10 and SCR than with PAR1 and PAR2, which is to some extent surprising, as we would have expected that CR emphasizes more the average runtime than the number of successful runs. We note to the interested reader that the EDACC SC 2012 web front-end provides (under "Ranking") all the aforementioned ranking methods (except PBR), allowing anyone to analyze further the rankings of one's interest.

Table 14: Spearman's rank correlation coefficient ρ between the different ranking schemes on the Application Track data. The higher the value the more correlated two ranking methods are.

ρ	SCR	CR	PAR1	PAR2	PAR10
SCR	1.000	0.917	0.967	0.991	0.999
CR		1.000	0.900	0.911	0.918
PAR1			1.000	0.953	0.968
PAR2				1.000	0.987
PAR10					1.000

Table 15: Spearman's rank correlation coefficient ρ between the different ranking schemes on the Hard Combinatorial Track data.

ρ	SCR	CR	PAR1	PAR2	PAR10
SCR	1.000	0.943	0.992	0.999	1.000
CR		1.000	0.936	0.936	0.940
PAR1			1.000	0.990	0.991
PAR2				1.000	0.999
PAR10					1.000

Table 16: Spearman's rank correlation coefficient ρ between the different ranking schemes on the Random SAT Track data.

ρ	SCR	CR	PAR1	PAR2	PAR10
SCR	1.000	0.968	0.990	0.993	1.000
CR		1.000	0.946	0.949	0.968
PAR1			1.000	0.998	0.990
PAR2				1.000	0.993
PAR10					1.000

Ranking the solvers from the different tracks according to the before mentioned ranking schemes would have changed the ranking of the top three solvers only when using CR and in this case only slightly (i.e. a change of the second or third solver would have occurred).

6.2. Stability of Rankings with Respect to the Timeout

The ranking of the solvers within each track also depends on the selected timeout, which in SC 2012 was 900 seconds. The EDACC web front end allows to simulate the ranking with lower timeouts. To measure the stability of the ranking with respect to different timeouts, we have computed the ranking for timeouts of 450, 180 and 90 seconds, which corresponds to $1/2$, $1/5$ and $1/10$ of the original timeout. For each ranking we then computed the Spearman correlation coefficient between the original ranking and the simulated ones with lower timeout; the results are shown in Table 17.

The rankings for the sequential tracks remain quite stable if we reduce the timeout to 450 seconds, implying that we would have obtained almost the same results (rankings) by using only half of the resources in these tracks. Only the sequential Application track would have produced a different 3rd ranking. However, for the parallel track, the disagreement is already on the first ranked solver, suggesting that the results in this track are very sensitive to the timeout value (see also Section 6.3). Using 180 seconds as the timeout resulted in considerably larger changes in the rankings, especially for the Application tracks. These changes are even more pronounced when using only 90 seconds. Interestingly, the ranking variations are very low for the Hard Combinatorial and Random tracks even when using only $1/10$ of the original timeout.

Table 17: Spearman's rank correlation coefficient ρ between the original ranking (established upon a timeout of 900 seconds) and the simulated ranking when the timeout is 450, 180 and 90 seconds. The value in brackets represents the rank position of the first disagreement.

Track	450sec.	180sec.	90sec.
Application	0.978 (3)	0.871 (3)	0.695 (1)
Hard Combinatorial	0.990 (7)	0.980 (7)	0.948 (7)
Random	0.990 (7)	0.939 (4)	0.919 (3)
Application parallel	0.962 (1)	0.899 (1)	0.828 (1)
Portfolio	1.000 (-)	1.000 (-)	0.700 (2)

Table 18: Descriptive statistics of the distribution of the sub-sample rankings of the top 15 solvers in each of the competition tracks, taken over 1000 sets of 300 (uniform, out of 600) randomly drawn benchmarks in each track. The left-most column shows the original ranking over the 600 benchmarks used in each of the tracks of the competition. Each of the subsequent groups of 3 columns corresponds to each of the competition tracks. In each group, the mean, the standard deviation, and the median of the sub-sample ranks are shown. Columns for tracks with less than 15 solvers contain empty entries. Entries where the sum-sample ranking differ drastically from the original ranking are boldfaced.

Orig	**Application**			**Hard Comb.**			**Random**			**App. Parallel**			**Seq. Portfolio**		
Rank	**Avg.**	**Std.**	**Med.**	**Avg.**	**Std.**	**Med.**	**Avg.**	**Std.**	**Med.**	**Avg.**	**Std.**	**Med.**	**Avg.**	**Std.**	**Med.**
1	1.00	0.03	1.0	1.15	0.36	1.0	1.00	0.00	1.0	1.74	0.86	**2.0**	1.00	0.00	1.0
2	2.00	0.08	2.0	1.85	0.36	2.0	2.08	0.28	2.0	1.89	0.92	**2.0**	2.14	0.34	2.0
3	3.01	0.18	3.0	3.29	0.46	3.0	2.92	0.28	3.0	3.06	0.93	3.0	2.86	0.34	3.0
4	4.76	1.15	4.0	3.79	0.57	4.0	4.32	0.58	4.0	3.48	1.02	4.0			
5	5.32	1.26	5.0	4.92	0.28	5.0	5.14	0.71	5.0	5.88	1.12	**6.0**			
6	6.25	1.27	6.0	6.26	0.45	6.0	5.56	0.64	6.0	6.03	1.19	**6.0**			
7	7.12	1.36	7.0	6.76	0.48	7.0	6.98	0.14	7.0	7.12	1.27	7.0			
8	7.38	1.63	8.0	8.00	0.28	8.0	8.02	0.13	8.0	7.48	1.43	8.0			
9	9.71	1.76	**10.0**	9.91	1.24	9.0	9.07	0.32	9.0	9.13	1.30	9.0			
10	**11.20**	1.91	**11.0**	11.05	1.61	**11.0**	9.93	0.32	10.0	10.52	1.29	10.0			
11	**11.11**	1.85	**11.0**	11.24	1.62	**11.0**	10.98	0.13	11.0	10.99	1.36	11.0			
12	11.41	1.48	**12.0**	**12.28**	2.40	**12.0**				11.64	1.45	12.0			
13	11.60	1.71	**12.0**	**12.28**	2.07	**12.0**				12.58	1.66	13.0			
14	13.12	1.15	14.0	15.72	2.24	**16.0**				15.24	1.31	**15.0**			
15	15.01	0.19	15.0	16.15	2.74	**16.0**				15.36	1.44	**15.0**			

6.3. Stability of Rankings with Respect to the Benchmarks Set

To evaluate the impact of the selected benchmark set on the solver rankings, we performed the following experiment. For each track, we sub-sampled 300 out of 600 benchmarks uniformly at random, and ranked the participating solvers on the resulting sample; we refer to the resulting ranking as *sub-sample ranking*. This procedure was repeated 1000 times. The descriptive statistics of the resulting distribution of the solver's sub-sample rankings are presented in Table 18. With the exception of the Application Parallel track, the rankings of the top performing solvers are quite stable: the median sub-sample rankings match the original rankings. However, in the Application Parallel track, the sub-sample rankings of the top two solvers vary to a very large degree. Hence the solvers would likely have switched ranks on a smaller set of benchmarks. A similar observation can be made for solvers ranked 9-14 in the Application track, 10-15 in the Hard Combinatorial track, and even 5-6 in the Parallel track. Many of the solvers ranked below 15 (not shown in the table) are in a similar situation as well. The only track that is very stable with respect to the selected set of benchmarks is the Random track. This is likely due to the fact that the random benchmark set is very homogeneous in terms of structural properties of the instances.

Regarding categorization of benchmarks, we note that some benchmark instances have both Application and Hard Combinatorial benchmarks characteristics. One notable example is the class of SAT encoded cryptographic problems, such as attacks against the block ciphers AES and DES. The computational hardness assumptions underlying the construction of these ciphers might suggest categorizing the resulting benchmarks as Hard Combinatorial, while the domain and the typically weak performance of the solvers in the Hard Combinatorial track on these instances suggest their classification as Application benchmarks. To evaluate the potential impact of re-classification of these benchmarks, we re-computed the rankings of the solvers in the Application track on the subset of the application benchmarks that excludes the cryptographic instances. The results are presented in Table 19. While the rankings of the winners have not been affected, already the solvers on the 4th and 5th position swapped their places. The changes in the rankings of lower-ranked solvers are even more dramatic with some solvers gaining or loosing up to three positions in the rankings. The results demonstrate that the issue of clear benchmark categorization has to be addressed in future competitions.

6.4. How Similar are the Submitted Solvers

As a measure of similarity between solvers, we computed the Spearman rank correlation between the results of all pairs of solvers. Rank correlation is better suited for analyzing the performance similarities of solvers than, e.g.,

Table 19: Comparison of the (original) rankings of the top 15 solvers in the Application SAT+UNSAT track on the full set of benchmarks with the rankings of the solvers on the set without the 58 cryptographic instances. Rank changes with respect to the original results are highlighted with bold typeface. Reference solvers are not listed.

	Full set (600 instances)			No crypto (542 instances)		
Solver	**Rank**	**#solved**	**%solved**	**Rank**	**#solved**	**%solved**
SATzilla2012 APP	1	531	88.0	1	475	87.0
SATzilla2012 ALL	2	515	85.0	2	464	85.0
Industrial SAT Solver	3	499	83.0	3	460	84.0
interactSAT	4	480	80.0	**5**	436	80.0
glucose	5	475	79.0	**4**	437	80.0
SINN	6	472	78.0	**7**	430	79.0
ZENN	7	468	78.0	**8**	427	78.0
Lingeling	8	467	77.0	**6**	435	80.0
linge_dyphase	9	458	76.0	**11**	420	77.0
simpsat	10	453	75.0	**13**	409	75.0
glue_dyphase	11	452	75.0	**9**	424	78.0
CCCneq	12	452	75.0	12	411	75.0
TENN	13	451	75.0	**10**	421	77.0
CCCeq	14	446	74.0	14	406	74.0
ppfolio2012	15	423	70.0	15	385	71.0

Pearson correlation that would only reveal possible linear correlations. The resulting correlation matrix is clustered hierarchically (under average distance) and plotted as a heat map.[28] This kind of plot was also used in [83] to analyze the contribution of solvers within a portfolio solver.

Figures 6–9 show the obtained hierarchical clusterings as *dendrograms* (top) and the correlation matrices (bottom). In the correlation matrices, rows and columns correspond to solvers, and are ordered in such a way that "similar" solvers are adjacent. Each entry in the correlation matrices gives the degree of correlation between two solvers. Instead of numeric entries, a color code is used for displaying correlations: darker colors correspond to high correlations, whereas lighter colors indicate a low degree of correlation. Below the matrix, the translation from color codes to numerical values is shown, together with a histogram indicating the frequency with which each correlation value occurs.

Application Track.. Figure 6 shows the correlation and clustering of the solvers from the Application track (excluding disqualified solvers). The dendrogram as well as the correlation matrix show clearly that **March** behaves quite differently from all the other solvers.[29] As **March** implements a lookahead DPLL algorithm, whereas all other solvers are based on the CDCL approach, this is not very surprising. In contrast to **March**, the performance of CDCL solvers seems to be very similar. In particular, there are several pairs of solvers with almost identical behavior:

- **simpsat** and **CryptoMiniSat** (the former has been implemented based on the latter);
- **satUZK** and **satUZKs** (the latter is a version of the former with added preprocessing); and
- **glucose** and **glue_dyphase** (the latter is a variant of the former with a slightly modified phase selection strategy).

The dendrogram also reveals three larger clusters of related solvers, where the first ranges from **SINN** to **riss**, the second from **caglue** to **satUZKs**, and the third from **pfolioUZK** to **simpsat**. It can be assumed that solvers (excluding the reference solvers) in the first cluster are either based on or incorporate very similar techniques to **minisat**, in the second cluster similar to **glucose**, and in the third to the solvers **lingeling** or **CryptoMiniSat**.

Hard Combinatorial Track.. Figure 7 shows the results for the Hard Combinatorial track (excluding disqualified solvers). The two classes of SLS and CDCL/DPLL solvers can be easily detected in the clustering (solvers **CCASat**

[28]We used the $heatmap.2$ package from the R statistical computing language.

[29]A similar observation was made in [83] in the context of evaluating solver contributions in the SATzilla 2011 portfolio solver.

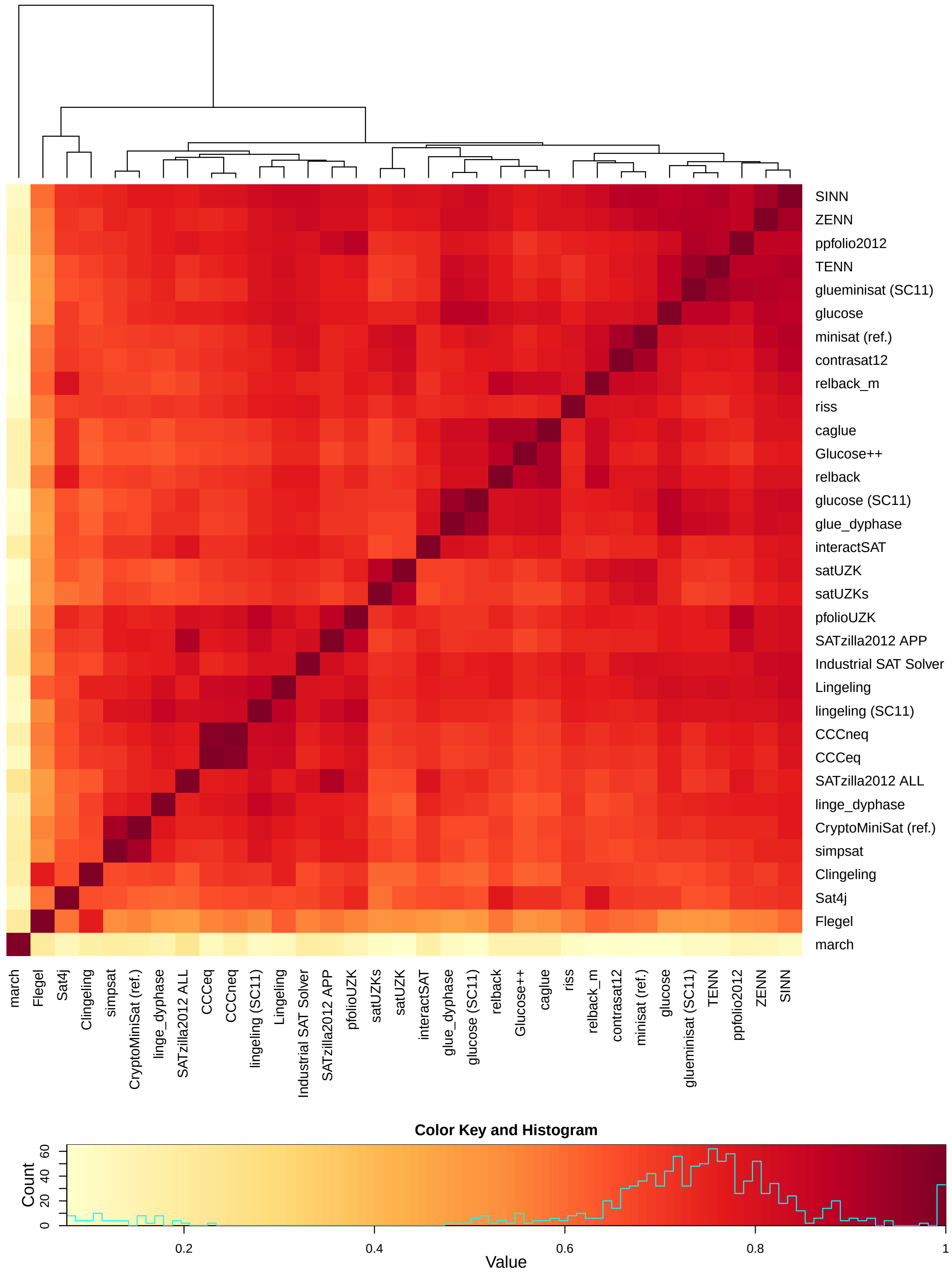
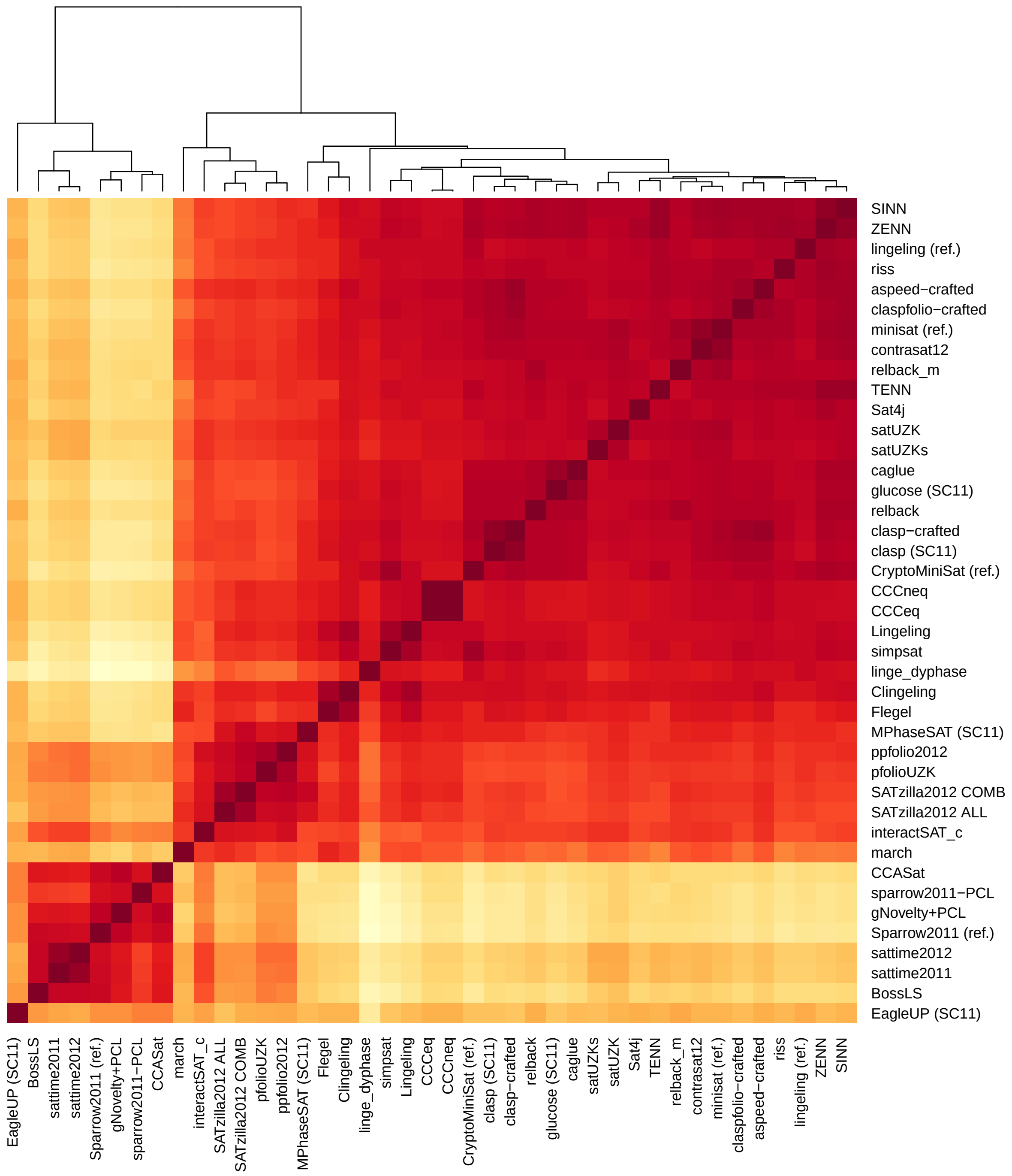


Figure 6: Clustered correlation matrix of the results of the solvers from the **Application Track**. Dark areas correspond to high correlation, whereas light areas correspond to low correlation between the solvers.



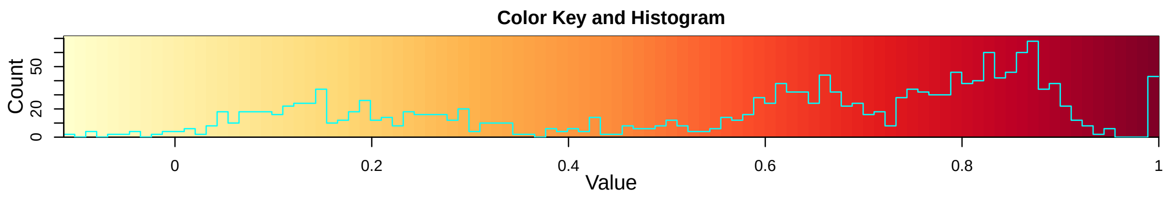


Figure 7: Clustered correlation matrix of the results of the solvers from the **Hard Combinatorial Track**. Dark areas correspond to high correlation, whereas light areas correspond to low correlation between the solvers.

to **EagleUp**, and **SINN** to **March**, respectively). CDCL solvers are in the top-right part, and SLS solvers show up on bottom-left. Between the SLS and the CDCL class we can also recognize the set of portfolio approaches, **ppfolio2012**, **pfolioUZK**, **SATzilla2012 COMB**, and **SATzilla ALL**. Two other solvers, **interactSAT_c** and **March**, are strongly correlated (**interactSAT_c** uses the lookahead solver **March** as a sub-solver). Also worth noting is that all portfolio solvers are heavily using the **MPhaseSAT** solver from the 2011 SAT Competition added as a reference solver. This is not surprising as **MPhaseSAT** had the largest unique solver contribution (the number of instances solved only by **MPhaseSAT**) in the 2011 SAT Competition crafted category; this is also the case for the SC 2012 Hard Combinatorial track when disregarding the portfolio solvers. The **MPhaseSAT** solver [84] uses a phase heuristic inspired by lookahead solvers which is quite expensive to compute, but appears to provide a key to solving some of the harder instances. The solver **EagleUP** behaves quite differently from the other SLS solvers, which may be due to the incorporation of unit propagation, a feature that is missing from the other SLS solvers.

It is also interesting that the performance of the portfolio solvers (**ppfolio2012** to **interactSAT_c**) is quite similar to that of CDCL solvers in this track, although these portfolio solvers integrate both SLS and CDCL components. This might indicate that constituent CDCL solvers dominate the behavior of such portfolios on hard combinatorial problems.

SLS solvers have an important contribution in the Hard Combinatorial track, being able to solve some satisfiable instances that the competing CDCL solver cannot solve. Analyzing the set of CDCL solvers together with only one single SLS solver, **sattime2012**, reveals that **sattime2012** has a unique solver contribution of 39 instances. The VBS is using SLS solvers on 192 out of 516 instances.

Random. Figure 8 shows the correlation and clustering of the solvers from the Random SAT track. There is a relatively large cluster around **Sparrow2011**, the winner of the Random SAT track of the 2011 SAT Competition. All portfolio solvers (**SATzilla2012 ALL** to **ppfolio2012**) and also the non-portfolio solver **CCASat** are relatively highly correlated with **Sparrow2011**, suggesting that the portfolio solvers often run **Sparrow2011** or a solver exhibiting similar performance as **Sparrow2011**. **CCASat** tries to mimic the behavior of **Sparrow2011** using a technique called *configuration checking with aspiration* (CCA). The portfolio solvers **pfolioUZK** and **ppfolio** are highly correlated, which might be due to the fact that the former is based on the latter. The SLS solver **BossLS** behaves quite differently from all other solvers. A reason for this might be the extensive preprocessing performed by this solver, including unit propagation, failed literal detection, and asymmetric blocked clause elimination. In general, the degree of correlation in the Random SAT track is much lower than in the Application and Hard Combinatorial track; the diversity of solving approaches submitted was much higher for this track.

Parallel Application. Figure 9 shows the correlation and clustering of the solvers from the Application Parallel track. Four major clusters can be detected here:

- the set of parallel portfolio solvers (**pfolioUZK** to **ppfolio2012**);
- the solver families **CCC[n]eq** (hybrid lookahead plus CDCL); and **Plingeling/Treengeling** (CDCL);
- the parallelized versions of **glucose** and **minisat** (**Sucrose** to **Minifork**); and
- the solvers **claspmt**, **splitter**, and **CryptoMiniSat**.

The approaches used by these solvers are quite different. The portfolio solvers use different base solvers running in parallel with different strategies, with no or only minimal clause exchange. The solvers in the second class (**CCC**-variants and descendants of **lingeling**) are based on search space splitting, use forms of learned clause exchange, and, in the case of the cube-and-conquer solvers of the **CCC** family, combine a CDCL algorithm with a lookahead-approach for determining how to split the search space. Parallel derivations of **glucose** use competition parallelism (i.e., differently configured versions of a CDCL base solver running in parallel on the whole SAT instance) with forms of clause exchange. The solvers **ZENNfork** and **Minifork**, based on **minisat**, perform search space splitting, but no clause exchange. Solvers in the last group, consisting of **claspmt**, **splitter** and **CryptoMiniSat**, implement specialized algorithms, e.g., iterative partitioning in case of **splitter**.

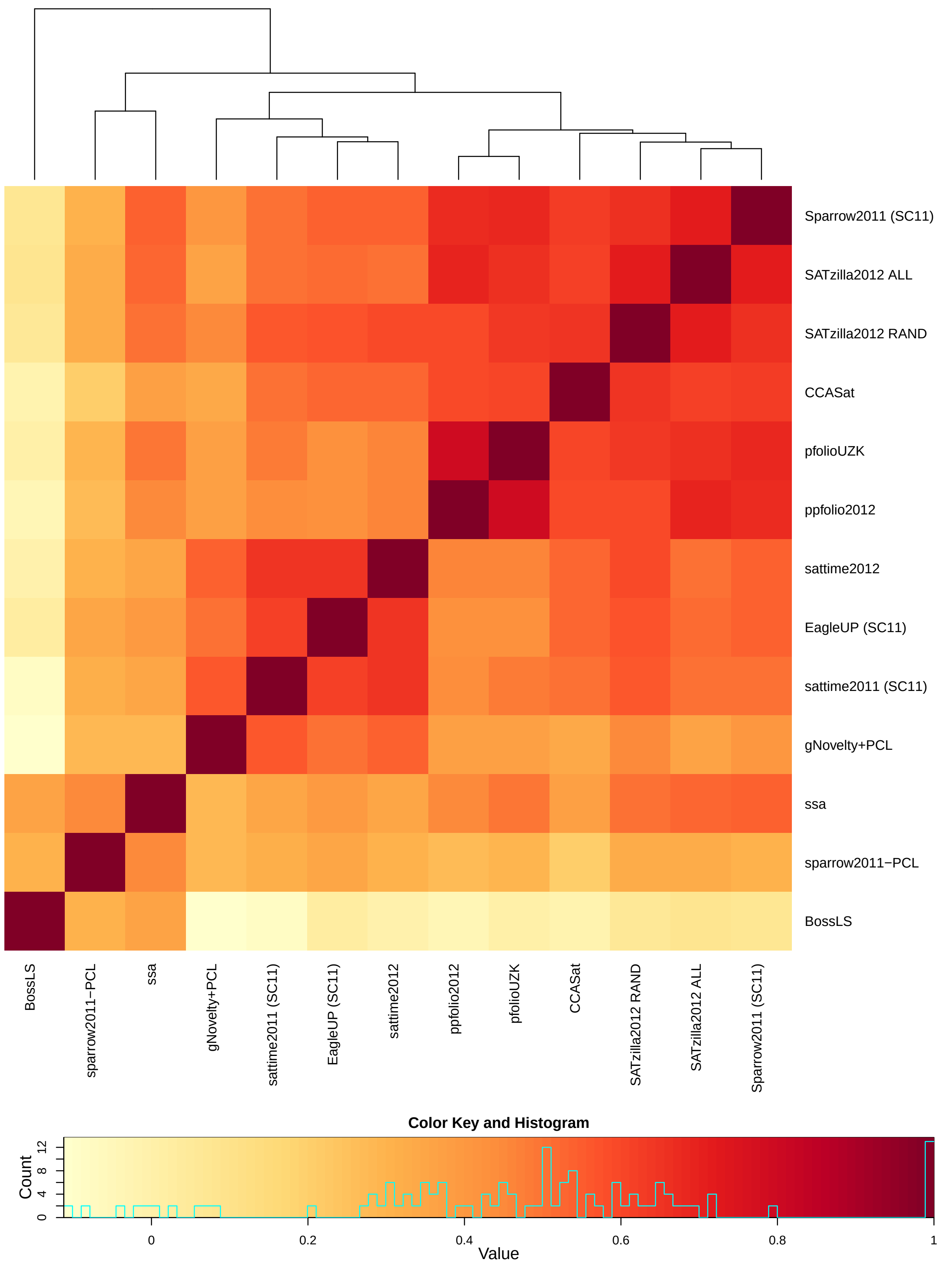
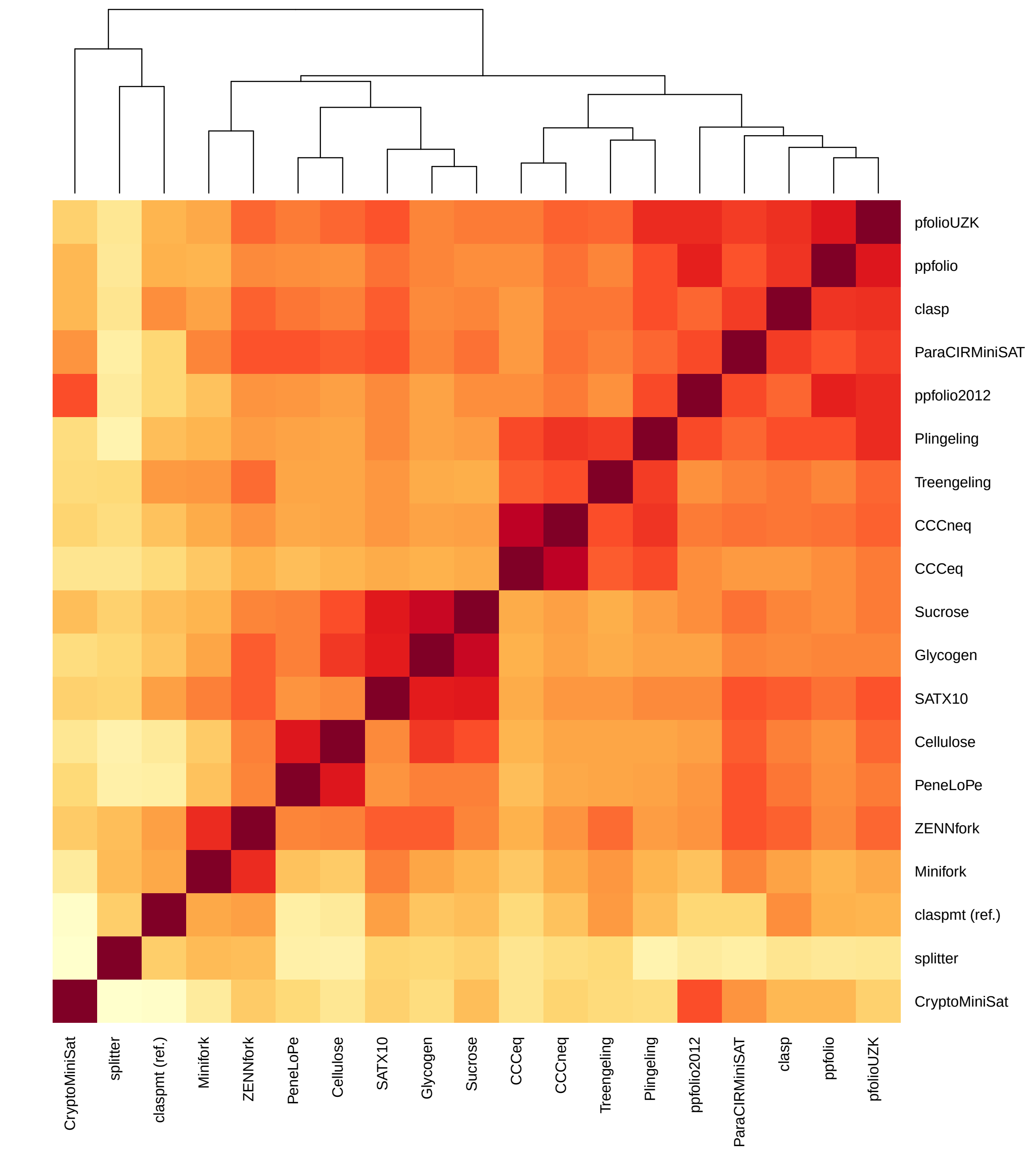


Figure 8: Clustered correlation matrix of the results of the solvers from the **Random Track**. Dark areas correspond to high correlation, whereas light areas correspond to low correlation between the solvers.



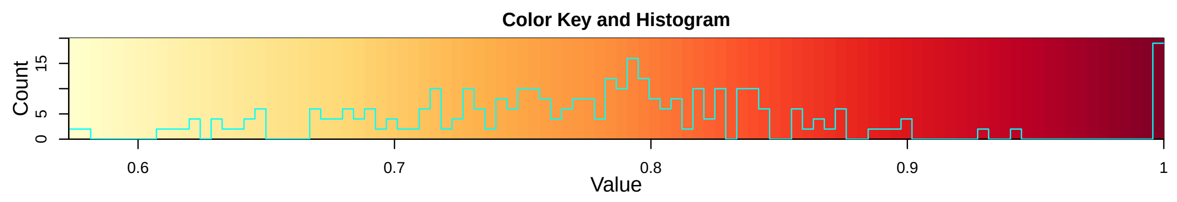


Figure 9: Clustered correlation matrix of the results of the solvers from the **Parallel Application Track**. Dark areas correspond to high correlation, whereas light areas correspond to low correlation between the solvers.

As with the Random SAT track, the degree of correlation in the Parallel Application track is much lower than in the Application and Hard Combinatorial track. It is also interesting to observe that the top seven solvers cover only two parallelization approaches, namely portfolio (**pfolioUZK**, **ppfolio2012**, **ppfolio** and **ParaCIRMiniSAT**) and competition parallelism with clause exchange, using **glucose** as the base solver (**PeneLoPe**, **Cellulose** and **Sucrose**).

6.5. Minimum Solver Set(s)

Here we address the question *What is a minimum set of solvers that would solve all instances solved by any solver within a track?* More formally, consider the set of solvers and the instances they have solved within the given timeout as a collection of subsets $S = \{S_1, S_2, S_3, \ldots, S_n\}$, where S_i is the subset of instances that was solved by solver S_i. Let $I = \bigcup_{i=1}^{n} S_i$ be the set of solved instances. Now we can formulate the question as a minimum set cover problem, where the task is to find the minimum number of subsets S_i to cover the set I. Note that all solvers that have a unique contribution will be part of the minimum solver set. Within the EDACC web front-end this problem is solved by encoding the problem as Max-SAT and using a branch-and-bound Max-SAT solver (**akmaxsat** [85]) to find all optimal solution. The unique solver contribution is also a measure of interest but is highly dependent on the set of solvers used to compute it. When analyzing the results, one should keep in mind that the inclusion of derivatives of a given solver in the set will reduce the unique solver contribution of this solver dramatically. In the following we provide the minimum solver sets for some of the tracks, excluding reference, offtrack, and disqualified solvers.

Application track. In the Application track there are six minimal sets of solvers, each of size 7, that do not differ significantly in terms of overall runtime (less than 5%). These sets consist of the solvers **glucose**, **linge_dyphase**, **interactSAT**, **Industrial SAT Solver**, **SATzilla2012 APP**, together with one solver from the set {**Glucose++**, **glue_dyphase**}, and one from {**CCCeq**, **CCCneq**, **Lingeling**}. Observe that the minimal sets include a large number of multi-engine solvers, as well as one portfolio solver. As these types of solvers include multiple solvers in their code-base, we also computed the minimal set over the sequential single-engine solvers only. The result was a single set of size 9, consisting of the solvers **glucose**, **linge_dyphase**, **Glucose++**, **Lingeling**, **simpsat**, **TENN**, **contrasat12**, **riss**, **ZENN**, suggesting, perhaps, a good portfolio composition. Notice, however, that the number of instances solved by the single-engine only solvers is 553 (vs. 562 for all). Interestingly, the solver **linge_dyphase**, ranked 9th overall in the Application track, is included in all minimal sets.

Hard Combinatorial track. In this track, there is a single minimal set of solvers of size 7. This set consists of the solvers **gNovelty+PCL**, **linge_dyphase**, **claspfolio-crafted**, **interactSAT_c**, **caglue**, **SATzilla2012 COMB**, and **Flegel**. Three of the solvers are either portfolio or multi-engine, and are clearly important contributors to the set, since the minimal set computed over sequential solvers only contains 11 solvers **gNovelty+PCL**, **linge_dyphase**, **sattime2011**, **BossLS**, **simpsat**, **sparrow2011-PCL**, **caglue**, **clasp-crafted**, **March**, **Lingeling**, **Sparrow2011**, and solves only 511 instances, instead of 527 solved by the minimum set over all types of solvers. Observe that over a half of the minimum set sequential solvers are SLS solvers, i.e., these solvers show strong performance on some of the satisfiable benchmark instance in the track.

Random track. All solvers had a unique solver contribution, thus being part of the minimum solver set. This is to some extent surprising, especially in light of the relatively homogenous class of benchmarks. However, we were not able to pinpoint a simple explanation for this behavior, which we in fact believe to be due to a combination of (i) the intrinsically randomized search heuristics applied within the solvers (local search), (ii) the benchmark set being to some extent more heterogeneous than (e.g., in the recent SAT Competitions before SC 2012, the random tracks did not include instances for clause lengths $k = 4, 6$), and (iii) in combination with point (ii), some of the solvers appeared to have been tuned to perform mainly on random instances generated using very similar parameter value combinations as the ones used in the recent SAT Competitions, witnessed e.g. by not being able to perform well on instances with even clause lengths. The fact that all solvers had a unique solver contribution explains also why portfolio approaches can be so successful on this type of instances. An optimal portfolio (of only core solvers) which is equivalent to the virtual best solver would solve 515 instances, which is almost 100 instances more than the best solver (423).

Summary. The minimum solver sets for the different main tracks differ in both the solver techniques used in the solvers, as well as the size of the minimum solver sets. For the Application track, the minimum solver sets are dominated by CDCL-based solvers, which is to be expected. We believe that the relatively small size of these minimum

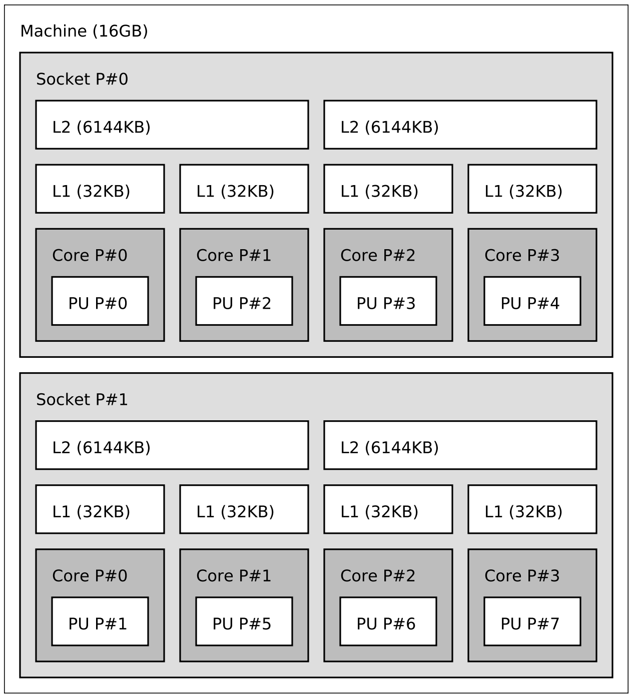


Figure 10: The CPU and cache architecture of the computing nodes of the bwGRID cluster used in SAT Challenge 2012.

solver sets may be due to the fact that the CDCL-based solvers are relatively similar to each other. Interestingly, local search solvers do not contribute[30] to the minimum solvers sets, although approximately half of the benchmarks in the track were satisfiable. This is in contrast with the Hard Combinatorial track, in which around half of the solvers within the minimum solver sets implement local search, evidently heavily contributing to solving the satisfiable benchmarks. This difference between the Application and Hard Combinatorial tracks may partly be explained by the fact that there were less local search solver submissions to the Application track than to the Hard Combinatorial track (which may suggest that there is currently less research effort put into improving local search techniques for application-type instances than for hard combinatorial type of instances). However, we believe that one should not underestimate the fact that local search solver offer a competitive approach to solving various types of satisfiable instances within the Hard Combinatorial track benchmarks. Unsurprisingly, the Random SAT track is dominated by local search solvers, and hence the minimum solver sets consist entirely of local search solvers[31]. We believe that developing an improved understanding of the fact that all solvers had a unique solver contribution in the Random track might prove to be fruitful in light of developing new, perhaps more complex local search heuristics based on the SC 2012 solver submissions.

6.6. Impact of the Computing Environment

Before running the competition, we measured the variance in runtime when simultaneously running two ($(2/8)$ scenario), four ($(4/8)$ scenario) or eight ($(8/8)$ scenario) solvers on a node of the cluster used for the competition. The computing nodes have two sockets, each with a quad-core CPU that, in turn, consists of two dual-core dies. Note that, at least in principle, the $(1/8)$ scenario is identical to the $(2/8)$ scenario due to each node having two independent sockets (we have also confirmed this experimentally). The topology of the CPUs is represented in Figure 6.6[32].

The EDACC computation client is designed to perform a CPU architecture topology scan prior to starting the

[30]Note, however, that the SATzilla2012 portfolio includes a local search solver component.

[31]Note again, however, that the portfolio solvers might have used complete solvers.

[32]The CPU and cache topology of a machine can be displayed with the *lstopo* command provided within the *hwloc* package. See `http://www.open-mpi.de/projects/hwloc/` for more details.

solvers (using Portable Hardware Locality *hwloc*) in an attempt to minimize the number of resource (cache and memory) collisions. When we start two solvers per node on the system (see Figure 6.6), the client will guarantee to start the solvers on disjunctive sockets (to avoid memory and cache collisions). If we would start four solvers on a node, the client will start the solvers on cores that do not share L2 cache. While multi-core CPUs allow for running several solver instantiations on a single CPU / computing node in parallel, this may have a non-deterministic effect on the runtime behavior of the solvers. In the following, we provide results on experiments on the influence of the number of solvers per node on the runtimes of both CDCL and SLS solvers. These experiments were run *before* the SC 2012 benchmark selection. Based on the results, we decided to limit the number of simultaneous solver runs to two per node.[33]

6.6.1. CDCL Solvers

CDCL solvers are relatively memory intensive when compared to other solving techniques, and when it comes to cache sharing their performance is expected to drop. We evaluated two of the best performing CDCL solvers **glucose** and **lingeling** (the binary versions submitted to the 2011 SAT Competition) on the SAT-Race 2010 instances by running every solver on every instance five times. The multiple runs per instance are needed to measure the runtime variability on the same instance. The memory limit was set to 3.5 GB for the $(2/8)$ and $(4/8)$ scenario and 2 GB for the $(8/8)$ scenario. The 2-GB limit resulted in a memory problem on 2 instances that were very large.

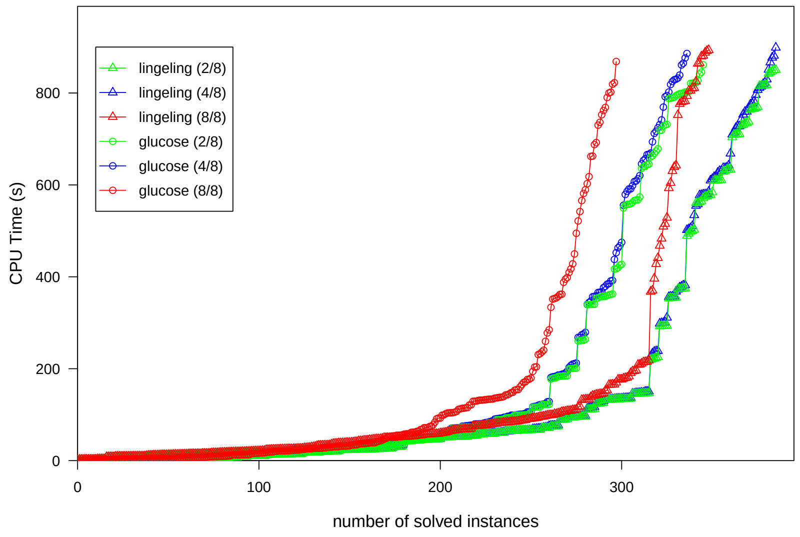


Figure 11: Difference in performance between the execution conditions of the CDCL solvers **lingeling** and **glucose** when two $(2/8)$ (green), four $(4/8)$ (blue) and eight $(8/8)$ (red) solvers are executed per node.

Figure 11 shows a cactus plot for the CDCL solvers executed under the three different scenarios. Table 21 lists the detailed results. There is almost no difference between executing two or four solvers per node (the green and blue curves). The variation in runtime per instance though is larger for **glucose**, whereas **lingeling** shows almost no difference, which is probably due to better memory management. When using all eight cores, the performance of both solvers drops significantly, likely due to sharing of the same L2 cache. In the $(2/8)$ scenario we can see small conglomeration blocks in the runtime curves, which represent the five runs we have performed per instances.

In the $(4/8)$ scenario these conglomerations loosen up, and in the $(8/8)$ scenario they are almost not identifiable.

[33] An *a posteriori* analysis using SC 2012 solver and benchmarks was unfortunately not possible due to hardware updates to the computing cluster used for SC 2012.

Table 20: Average coefficient of variation over all instances on the different execution scenarios.

	$(2/8)$	$(4/8)$	$(8/8)$
lingeling	0.0035	0.0121	0.0373
glucose	0.0045	0.0159	0.0463

Table 21: The ranking of the solvers **lingeling** and **glucose** when executed in the different scenarios.

#	Solver	# of successful runs	total time (sec.)	median time (sec.)
1	lingeling (2/8)	385	157525.12	315.05
2	lingeling (4/8)	385	158654.54	317.30
3	lingeling (8/8)	348	177518.27	355.03
4	glucose (2/8)	345	191395.34	382.79
5	glucose (4/8)	336	194150.19	388.30
6	glucose (8/8)	297	217558.40	435.11

To analyze the variability in runtime, we have computed the average coefficient of variation which is ten times larger for the $(8/8)$ scenario than for the $(2/8)$ scenario independent of the solver (see Table 20 for more details).

6.6.2. SLS Solvers

SLS solvers are less memory intensive than CDCL-based solvers (since there is no clause learning involved), but still highly cache intensive due to frequent non-localized memory accesses. We ran the two best performing SLS solvers **Sparrow2011** and **sattime2011** from the 2011 SAT Competition on a set of 19 randomly generated k–SAT instances for $k = 3, 5, 7$, executing each solver four times on the same instance using different seeds. As can be seen from Figure 12, there is almost no difference between the $(2/8)$ (green curve) and $(4/8)$ (blue curve), whereas the $(8/8)$ scenario shows significant degradation in the performance for both solvers.

Performance degradation as a consequence of too many solvers per node appears to occur only when solvers have to share the cache hierarchy. Our analysis will also hold when the solvers get close to the memory limit but do not exceed it. Running four solvers per node would have been possible, but then the amount of memory available per solver would have dropped below 4 GB, which for some CDCL solvers could have been critical. Hence we ran only two solvers per node at a time, with a memory limit of 6 GB.

7. Lessons Learned

In this section we discuss issues which were realized either during organizing SC 2012 or after the competition was run, and propose possible ways of dealing with these issues in forthcoming SAT and related solver competitions.

7.1. Progress with Respect to Previous Competitions

Since the first SAT Competition in 2002, considerable progress has been made in improving both SAT solver performance and stability (see also [33] for a comparison of the best solvers from 2002 to 2011). Today, CDCL-based solvers are not only used within the SAT community, but are also widely employed as black-box solvers in many different projects and application areas (see, e.g., references mentioned in Section 1).

Over the last years, and especially in 2012, we could see a continuing trend towards parallel and portfolio solvers. Whereas until 2009 Satzilla [63] was the only participating portfolio solver, and the first parallel solvers entered competitions only in 2007, in SC 2012 we had 19 solvers in the Parallel Track and 14 participating multi-engine or portfolio solvers. Multi-engine or portfolio solvers dominated the standard single-engine ones in both the Application and the Hard Combinatorial track in terms of runtime performance. Their success can be explained by their ability to overcome the shortcomings of a single, particular solving heuristic used in the single-engine solvers.

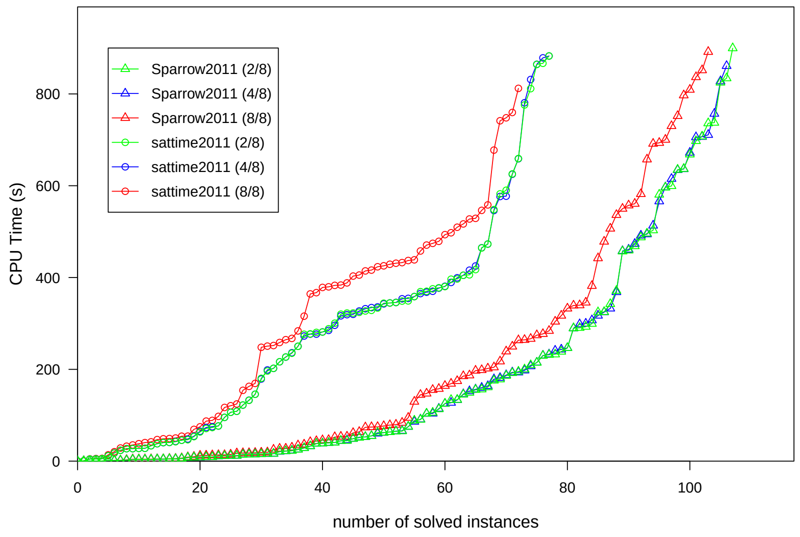


Figure 12: Difference in performance between the execution conditions of the SLS solvers **Sparrow2011** and **satttime2011** when executed two (2/8) (green), four (4/8) (blue) and eight (8/8)(red) solvers per node.

However, considerable improvements could also be observed on basic algorithms for single-engine solvers. Especially the success of **CCASat** in the Random SAT Track, which implements a new local-search algorithm is remarkable. But also in the Application SAT+UNSAT Track, progress can be registered, for example in the case of **glucose**, the best single-engine solver, which has shown a considerably better performance than its 2011 version.

To further illustrate the progress, let us take a closer look at two instances of the Application Track, which both had already been used in SAT-Race 2010. The problem `bitverif/minxorminand128` (containing 153,834 variables and 459,965 clauses) could not be solved by any of the 20 solvers participating in SAT-Race 2010. In 2012, it was solved for the first time in less than 900 seconds (in 233 seconds by **linge_dyphase**). Similarly, the instance `md5gen/gus-md5-10`, encoding a cryptographic problem, and also not solved by any solver in 2010, was solved by six solvers in 2012.[34]

7.2. Benchmark Selection

In Section 3.4 we pointed out a potential pitfall of the benchmark rating procedure that evaluates the empirical hardness of benchmarks using a limited set of SAT solvers: the selected set might be biased towards a particular solver used during evaluation, and thus a newer version of this solver used in the competition. Although increasing the number of evaluation solvers decreases this bias, it cannot fully eliminate it. In addition, such increase may be infeasible due to resource restrictions. Thus, the benchmark selection process must maintain a balance between the performance of the evaluation solvers on the selected benchmark set *explicitly* during the construction of the set. However, it is not entirely clear how possible attempts to “correct” bias towards particular solvers might affect the results of the competition. For example, eliminating bias towards a recently introduced, exceptionally well performing solver, might also lead to favoring already more established solver techniques implemented by many over new and highly potential techniques implemented by few. This in turn may lead to the competition results masking away potentially very important new solver techniques from becoming more standard ones.

[34]Both `bitverif/minxorminand128` and `md5gen/gus-md5-10` are unsatisfiable.

7.3. Timeout and the Number of Benchmark Instances

Enforcing a fixed limit on the computational resources (especially, the timeout in the case of SC 2012) poses challenges for any solver competition, and has connections to the number of benchmark instances used. Using a small timeout enables the use of more benchmark instances, and favors solvers which are geared towards solving relatively easy instances very fast but which may not perform well on harder instances. A large timeout favors solvers with robust performance in terms of solving instances of varying hardness and characteristics, but at the same time are likely to exhibit a higher average runtime on a smaller set of instances. In SC 2012 a somewhat small timeout of 900 seconds was enforced, while the number of benchmark instances per track per track was quite large (600). In contrast, in the past SAT Competitions a larger timeout (5000 seconds) and fewer instances (around 200-300) were used.

Based on the analysis of the results presented in Section 6, at least in "short-timeout" competitions like SC 2012, the ranking of the top-performing sequential solvers can be established rather accurately using a smaller number of benchmark instances (300 instead of 600, cf. Section 6.3), and also with a somewhat smaller timeout value (450 seconds instead of 900, cf. Section 6.2). This is somehow surprising and puts the relatively high timeout of the SAT Competition (5000 seconds) into question.

However, this is not the case for parallel solvers, whose ranking is affected to a very large degree by both the number of selected benchmarks and the cutoff. This is most likely due to the fact that the performance of these solvers is significantly less deterministic—due to nondeterminism caused by parallel execution, at the very least—than the performance of sequential solvers. Thus, it seems that resource allocation in future competitions should definitely favor parallel solvers to a large degree. Finally, elimination of a single fixed timeout as the basis of solver ranking could be another—perhaps more ambitious—goal, taking advantage of the fact that running experiments with a fixed timeout t easily produces results for any timeout $t' < t$.

7.4. Benchmark Categorization

Specific benchmark instances may have both Application and Hard Combinatorial characteristics, with one notable example being the class of SAT encoded cryptographic problems, such as attacks against the block ciphers AES and DES (see discussion in Section 6.3), but other examples are abound — consider, for example, factoring vs. equivalence checking of multipliers. This means that the classification of benchmarks between the two classes is far from straightforward. Furthermore, as demonstrated by the results of the experiment described in Section 6.3, the changes in classification may have a dramatic effect on the rankings of the solvers. This is clearly an important issue that must be addressed in future competitions.

7.5. Solver Categorization

SAT Competitions have had to deal with different categories of solvers from the beginning. Initially, the main distinction was between complete DPLL/CDCL solvers, look-ahead solvers, and incomplete local-search algorithms. Each type of solver was specialized on certain benchmark instances, and thus the three categories of Application, Hard Combinatorial and (satisfiable) Random benchmarks emerged.

Later, with the rise of multi-core CPUs, parallel SAT algorithms turned up, and in 2008 a special track for parallel solvers was added to the competitions. In 2007, a portfolio solver (SATzilla) won a medal for the first time, by combining different solvers (even different solver types), and applying machine learning techniques to detect the most suitable solver. The number of portfolio solvers submission has increased in the subsequent competitions. Lately, solvers combining ideas from different approaches and algorithms (which we called multi-engine solvers in SC 2012) also surfaced. With this growing diversity of SAT solving approaches, it becomes more and more complicated for competitions to choose the right tracks and benchmark sets to accommodate all solvers suitably, and to categorize solvers in the right way.

One may wonder why solver categorization is needed at all. Why not have just one track with the union of all benchmarks on which the "globally best" solver is determined? One reason might be that a taxonomy of solvers is of scientific interest by itself (cf. Section 6.4). A more important rationale, in our opinion, is that by having different tracks, research on particular solver classes can be stimulated. Having, for example, a separate track for satisfiable random instances, gives stochastic local search solvers a chance to win a prize and thus furthers research in this area.

In SC 2012, we mainly adopted the traditional categorization of solvers as laid down by previous competitions (CDCL vs. look-ahead vs. local search). We also kept the special track for Parallel Solvers introduced in 2008.

For the first time, we introduced a special track for Sequential Portfolio Solvers, as these solvers had grown more important over the last years. This track was poorly accepted, though. Authors of solvers that we classified as "portfolio" pushed to get into the main tracks. We thus opened the main tracks for such solvers, too, at the same time adding a classification to the main tracks. We distinguished between (i) "traditional" solvers, which employ one core SAT algorithm (*single-engine*); (ii) simple combinations of existing solvers (*portfolio*), in which a focus is put on selecting the right solvers, e.g., by using machine-learning techniques; and (iii) more complex approaches using interacting combinations of different solvers (*multi-engine*), as described in Section 4. For SC 2012, we considered this distinction a good compromise between simplicity, historical evolution of categories, and separation of different solving techniques.

As determination of solver categories allows, to some extent, to give a stimulus to particular research directions, we believe that it will remain a controversial topic in future competitions.

7.6. Native Implementations vs. Building on Existing Solver Source Code

A recurring theme in recent SAT solver competitions has been that many competitors submit solvers which are relative small modifications of already available state-of-the-art open-source SAT solvers. In SC 2012, we saw a number of such solver submissions that were modifications or built on top of the successful Glucose solver. Here one should notice that Glucose itself is based on the successful Minisat solver. This raises the question of whom to give credit for a successful solver. As many SAT solvers are open source, it is quite easy to take existing solvers, and to either just modify them slightly or combine different such solvers into a new tool. One can argue whether the act of combining (or somewhat modifying) them is as significant a contribution to SAT solving technology as the ideas that went into the original solvers.

This problem is aggravated by preprocessors (such as **SatELite** [86]) and similar tools, which are used as "add-on components" in existing solvers. Again, the question arises, how to handle such component-based solvers and whom to give credit for competition entrants that just replace or add a component.

To partially deal with this problem, a *MiniSat Hack Track* was initiated in 2009, in which solvers that modify the source code of MiniSat by at most 5% could participate.[35] By this construct, small contributions (with respect to implementation effort) to the state of the art in CDCL solvers were encouraged, while still giving credit to the original MiniSat authors.

In SC 2012, there was no separation between native implementations of solvers and solver submissions that were heavily based on already available solver source code. We believe that more incentives should be provided for solver developers to actually implement their solvers from scratch, hence contributing to the maintenance of a heterogeneous set of publicly available SAT solvers. Apart from recognizing the overall best-performing solvers, giving formal recognition to solvers with large unique contributions to the VBS—of high interest to, e.g., portfolio solver developers—would be a good option to consider. This idea has also been suggested in [83].

The MiniSAT Hack Track of the 2009 and 2011 SAT Competitions is one way of organizing a separate track for solvers heavily building on the source code of others. However, even irrespectively of whether the solvers are required to be submitted in open source to a competition, we foresee some difficulties to objectively evaluate to what extent a submitted solver relies on existing solver source code.

7.7. Participation of Portfolio Solvers

The SAT Competition series was introduced to promote the practical development of SAT solvers and to provide a way to measure the utility of solving techniques and modifications proposed in the literature. The heterogeneity of the used benchmarks and the results produced by different solvers provide a perfect testbed for portfolio SAT solvers, which have shown remarkable performance in the last competitions. However, allowing portfolio solvers to compete in the same competition tracks as other solvers poses certain problems.

One could argue that it is beneficial to enter portfolio solvers to the main competition tracks in order to measure the performance of the actual current state of the art in SAT solvers, often exhibited by portfolio solvers. However, portfolio SAT solvers use outdated core SAT solvers by default. This is due to the fact that the portfolio submitted

[35]In the 2009 SAT Competition, the rules state that a solver participating in the MiniSat Hack Track is only allowed to change or add 125 lines of C code, which is approximately 5% of the total lines of code of MiniSat.

to a competition cannot contain actual core solvers submitted to the same solver competition, since the core solver developers generally do not make their solvers available before the competition. Thus the performance of portfolio solvers submitted to a specific competition could likely be improved simply by including the best-performing new solvers, submitted to the same competition, to the portfolios.[36] A fundamental question is what actually constitutes "state–of–the–art", in case core solvers are made to directly compete with portfolio approaches. Assume, for example, that a portfolio solver using solvers from over two years back ends up winning a track. In our opinion, this does not directly imply that the improvements to the core solvers within the last two years have not been significant. However, we do believe that it is valuable to know how well the best portfolio solvers perform in comparison to the current state–of–the–art core SAT solvers.

A more competition-oriented issue, arising from allowing portfolio solvers (and other types of multi-engine approaches using external core solvers in a black-box manner) to compete for the same awards as the core solvers, is fairness from the side of the core solver developers. This is due to the fact that the portfolio approaches directly capitalize on the advancements in the core SAT solvers by employing the core solvers in a black-box manner.

A separate track for portfolio solvers would seem to be the correct way to measure the advances in portfolio solvers. The aim would be to objectively evaluate the core part of intelligent portfolio approaches, namely, the algorithms applied within the portfolio to select the solver that is run on a given input instance. In such a track, the set of solvers available, as well as the set of training instances, should be fixed by the organizers a priori. Entrants submitted to the track would consist of the solver selection methods only, without any a priori training. The solver selection methods would then be trained using the fixed set of training instances. The performance of the submitted solver selection methods should then be measured on a separate test set. This approach to evaluating portfolios would allow to more objectively measure the merits of the most important methods implemented within portfolios, namely, the SAT solver selection algorithms. Ideally, such a portfolio competition would be run immediately after the main core SAT solver competition of the same year, so that the most recent best-performing core solvers could be included in the set of solvers available in the portfolio.

7.8. Ranking Schemes

Our analysis of the correlation between the rankings established with different ranking schemes (see Section 6.1) suggests that the more elaborate careful ranking (CR) scheme would produce very similar results to the currently used solution count ranking (SCR) as well as the PARx ranking scheme family. On the other hand, SCR benefits from transitivity and, perhaps, from being more intuitive and accessible to researchers outside the SAT community. As such, in our opinion, SCR is an adequate scheme for future SAT competitions.

8. Conclusions

We provided a detailed overview of SAT Challenge 2012, the main SAT solver competition organized in 2012. We covered various aspects of the competition, including the rules, tracks, ranking scheme, and benchmark selection and generation process. Furthermore, we presented an in-depth analysis of the results of the competition—given that the number of participants and the distribution of their respective research groups is very high (recall Section 4)—the results of the competition arguably provide a snapshot of the state of the art in SAT solvers in 2012. Finally, we suggested a number of improvements to future SAT, and other similar, competitions.

SAT solver competitions have been one of the driving forces for progress achieved over the last one and a half decade in implementing SAT solvers. Competitions can be seen as a community-wide *algorithm engineering* approach, in which different proposed algorithms and data structures are evaluated based on clear standards, and the results of the evaluation are passed back to the community to further improve their solvers.

Acknowledgements: We would like to emphasize that a solver competition cannot be successful without the active involvement and participation from the community at large. We would especially like to thank all those who contributed to SC 2012 by submitting solvers or benchmarks. We are also very grateful to the anonymous reviewers for their insightful comments which helped us to improve this article considerably.

[36]Some analysis on this, using the SATzilla framework and the SAT Competition 2011 results, was suggested by Xu et al. and is available at `http://www.cril.univ-artois.fr/SAT11/xu/revised/SATzilla_2011.html`.

References

[1] A. Biere, M. Heule, H. van Maaren, T. Walsh (Eds.), Handbook of Satisfiability, Vol. 185 of Frontiers in Artificial Intelligence and Applications, IOS Press, 2009.

[2] S. A. Cook, The complexity of theorem-proving procedures, in: Proc. STOC, 1971, pp. 151–158.

[3] A. Biere, A. Cimatti, E. M. Clarke, Y. Zhu, Symbolic model checking without BDDs, in: Proc. TACAS, Vol. 1579 of LNCS, Springer, 1999, pp. 193–207.

[4] E. M. Clarke, D. Kroening, F. Lerda, A tool for checking ANSI-C programs, in: Proc. TACAS, Vol. 2988 of LNCS, 2004, pp. 168–176.

[5] A. R. Bradley, SAT-based model checking without unrolling, in: Proc. VMCAI, Vol. 6538 of LNCS, Springer, 2011, pp. 70–87.

[6] H. A. Kautz, B. Selman, Planning as satisfiability, in: Proc. ECAI, 1992, pp. 359–363.

[7] J. Rintanen, Planning as satisfiability: Heuristics, Artificial Intelligence 193 (2012) 45–86.

[8] C. Barrett, R. Sebastiani, S. A. Seshia, C. Tinelli, Satisfiability modulo theories, in: Handbook of Satisfiability, IOS Press, 2009, Ch. 26, pp. 825–885.

[9] R. Nieuwenhuis, A. Oliveras, C. Tinelli, Solving SAT and SAT modulo theories: From an abstract Davis–Putnam–Logemann–Loveland procedure to DPLL(*T*), Journal of the ACM 53 (6) (2006) 937–977.

[10] R. Sebastiani, Lazy satisfiability modulo theories, Journal of Satisfiability, Boolean Modeling and Computation 3 (3-4) (2007) 141–224.

[11] M. Janota, W. Klieber, J. Marques-Silva, E. M. Clarke, Solving QBF with counterexample guided refinement, in: Proc. SAT, Vol. 7317 of LNCS, Springer, 2012, pp. 114–128.

[12] M. Janota, J. P. Marques-Silva, Abstraction-based algorithm for 2QBF, in: Proc. SAT, Vol. 6695 of LNCS, Springer, 2011, pp. 230–244.

[13] F. Lin, Y. Zhao, ASSAT: computing answer sets of a logic program by SAT solvers, Artif. Intell. 157 (1-2) (2004) 115–137.

[14] E. Giunchiglia, Y. Lierler, M. Maratea, Answer set programming based on propositional satisfiability, J. Autom. Reasoning 36 (4) (2006) 345–377.

[15] M. Gebser, B. Kaufmann, T. Schaub, Conflict-driven answer set solving: From theory to practice, Artif. Intell. 187 (2012) 52–89.

[16] C. Drescher, M. Gebser, T. Grote, B. Kaufmann, A. König, M. Ostrowski, T. Schaub, Conflict-driven disjunctive answer set solving, in: Proc. KR, AAAI Press, 2008, pp. 422–432.

[17] J. Davies, F. Bacchus, Solving MAXSAT by solving a sequence of simpler SAT instances, in: Proc. CP, Vol. 6876 of LNCS, Springer, 2011, pp. 225–239.

[18] F. Heras, A. Morgado, J. Marques-Silva, Core-guided binary search algorithms for maximum satisfiability, in: Proc. AAAI, AAAI Press, 2011.

[19] Z. Fu, S. Malik, On solving the partial MAX-SAT problem, in: Proc. SAT, Vol. 4121 of LNCS, Springer, 2006, pp. 252–265.

[20] C. Ansótegui, M. L. Bonet, J. Levy, A new algorithm for weighted partial MaxSAT, in: Proc. AAAI, AAAI Press, 2010.

[21] E. Grégoire, B. Mazure, C. Piette, On approaches to explaining infeasibility of sets of boolean clauses, in: ICTAI, 2008, pp. 74–83.

[22] J. Marques-Silva, Minimal unsatisfiability: Models, algorithms and applications, in: Proc. ISMVL, IEEE Computer Society, 2010, pp. 9–14.

[23] A. Belov, I. Lynce, J. Marques-Silva, Towards efficient MUS extraction, AI Communications 25 (2) (2012) 97–116.

[24] S. Wieringa, Understanding, improving and parallelizing MUS finding using model rotation, in: Proc. CP, Vol. 7514 of LNCS, Springer, 2012, pp. 672–687.

[25] E. M. Clarke, A. Gupta, O. Strichman, SAT-based counterexample-guided abstraction refinement, IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems 23 (7) (2004) 1113–1123.

[26] E. M. Clarke, O. Grumberg, S. Jha, Y. Lu, H. Veith, Counterexample-guided abstraction refinement for symbolic model checking, Journal of the ACM 50 (5) (2003) 752–794.

[27] M. Janota, R. Grigore, J. Marques-Silva, Counterexample guided abstraction refinement algorithm for propositional circumscription, in: Proc. JELIA, Vol. 6341 of LNCS, Springer, 2010, pp. 195–207.

[28] C. M. Wintersteiger, Y. Hamadi, L. de Moura, Efficiently solving quantified bit-vector formulas, in: Proc. FMCAD, IEEE, 2010, pp. 239–246.

[29] L. de Moura, H. Ruess, M. Sorea, Lazy theorem proving for bounded model checking over infinite domains, in: Proc. CADE-18, Vol. 2392 of LNCS, Springer, 2002, pp. 438–455.

[30] C. W. Barrett, D. L. Dill, A. Stump, Checking satisfiability of first-order formulas by incremental translation to SAT, in: Proc. CAV, Vol. 2404 of LNCS, Springer, 2002, pp. 236–249.

[31] C. Flanagan, R. Joshi, X. Ou, J. B. Saxe, Theorem proving using lazy proof explication, in: Proc. CAV, Vol. 2725 of LNCS, Springer, 2003, pp. 355–367.

[32] W. Dvořák, M. Järvisalo, J. P. Wallner, S. Woltran, Complexity-sensitive decision procedures for abstract argumentation, Artificial Intelligence 206 (2014) 53–78.

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

[34] M. Buro, H. K. Büning, Report on a SAT competition, Bulletin of the European Association for Theoretical Computer Science 49 (1993) 143–151.

[35] D. Johnson, M. Trick (Eds.), Second DIMACS implementation challenge: Cliques, coloring and satisfiability, Vol. 26 of DIMACS Series in Discrete Mathematics and Theoretical Computer Science, American Mathematical Society, 1996.

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

[37] D. Le Berre, L. Simon, The essentials of the SAT 2003 Competition, in: Proc. SAT 2003, Vol. 2919 of LNCS, Springer, 2004, pp. 452–467.

[38] D. Le Berre, L. Simon, Fifty-five solvers in Vancouver: The SAT 2004 Competition, in: SAT 2004 Selected Paper, Vol. 3542 of LNCS, Springer, 2005, pp. 321–344.

[39] A. Balint, A. Belov, D. Diepold, S. Gerber, M. Järvisalo, C. Sinz (Eds.), Proceedings of SAT Challenge 2012: Solver and Benchmark Descriptions, Vol. B-2012-2 of Department of Computer Science Series of Publications B, University of Helsinki, 2012, iSBN 978-952-10-8106-4.

[40] G. S. Tseitin, On the complexity of derivation in propositional calculus, in: J. Siekmann, G. Wrightson (Eds.), Automation of Reasoning 2: Classical Papers on Computational Logic 1967–1970, Springer, 1983, pp. 466–483.

[41] D. A. Plaisted, S. Greenbaum, A structure-preserving clause form translation, Journal of Symbolic Computation 2 (3) (1986) 293–304.

[42] M. Heule, M. Dufour, J. van Zwieten, H. van Maaren, March_eq: Implementing additional reasoning into an efficient look-ahead SAT solver, in: SAT 2004 Revised Selected Papers, Vol. 3542 of LNCS, Springer, 2005, pp. 345–359.

[43] bwGRiD (http://www.bw grid.de/), Member of the german d-grid initiative, funded by the ministry of education and research (bundesministerium für bildung und forschung) and the ministry for science, research and arts baden-wuerttemberg (ministerium für wissenschaft, forschung und kunst baden-württemberg), Tech. rep., Universities of Baden-Württemberg (2007-2010).

[44] O. Roussel, Controlling a solver execution: the runsolver tool, Journal on Satisfiability, Boolean Modeling and Computation 7 (2011) 139–144.

[45] A. Balint, D. Gall, G. Kapler, R. Retz, Experiment design and administration for computer clusters for SAT-solvers (EDACC), Journal on Satisfiability, Boolean Modeling and Computation 7 (2-3) (2010) 77–82.

[46] A. Balint, D. Diepold, D. Gall, S. Gerber, G. Kapler, R. Retz, EDACC - an advanced platform for the experiment design, administration and analysis of empirical algorithms, in: Proc. LION 5, Vol. 6683 of LNCS, Springer, 2011, pp. 586–599.

[47] D. P. Anderson, BOINC: A system for public-resource computing and storage, in: Proc. 5th IEEE/ACM Intl. Workshop on Grid Computing (GRID'04), IEEE Computer Society, 2004, pp. 4–10.

[48] L. Simon, P. Chatalic, SatEx: A web-based framework for SAT experimentation, Electronic Notes in Discrete Mathematics 9 (2001) 129–149.

[49] G. Sutcliffe, C. B. Suttner, Evaluating general purpose automated theorem proving systems, Artif. Intell. 131 (1-2) (2001) 39–54.

[50] C. Barrett, M. Deters, L. M. de Moura, A. Oliveras, A. Stump, 6 years of SMT-COMP, J. Autom. Reasoning 50 (3) (2013) 243–277.

[51] M. Gebser, L. Liu, G. Namasivayam, A. Neumann, T. Schaub, M. Truszczynski, The first answer set programming system competition, in: Proc. LPNMR, Vol. 4483 of LNCS, Springer, 2007, pp. 3–17.

[52] M. Denecker, J. Vennekens, S. Bond, M. Gebser, M. Truszczynski, The second answer set programming competition, in: Proc. LPNMR, Vol. 5753 of LNCS, Springer, 2009, pp. 637–654.

[53] F. Calimeri, G. Ianni, F. Ricca, The third open answer set programming competition, CoRR abs/1206.3111.

[54] C. Suttner, G. Sutcliffe, The design of the CADE-13 ATP system competition, Journal of Automated Reasoning 30 (1997) 1–1.

[55] G. Sutcliffe, Proceedings of the 6th IJCAR ATP System Competition (CASC-J6), `http://www.cs.miami.edu/~tptp/CASC/J6/Proceedings.pdf` (2012).

[56] M. Järvisalo, D. L. Berre, O. Roussel, The SAT Competition 2011, Results of Phase 1, slides, `http://www.cril.univ-artois.fr/SAT11/phase1.pdf` (2011).

[57] B. Selman, D. G. Mitchell, H. J. Levesque, Generating hard satisfiability problems, Artif. Intell. 81 (1-2) (1996) 17–29.

[58] I. P. Gent, T. Walsh, The SAT phase transition, in: Proc. ECAI, John Wiley and Sons, 1994, pp. 105–109.

[59] D. Pham, C. Gretton, gNovelty^{+}, `http://www.satcompetition.org/2007/gNovelty+.pdf`. (2007).

[60] R. Bruttomesso, D. R. Cok, A. Griggio, Satisfiability modulo theories competition (SMT-COMP 2012): Rules and procedures, `http://smtcomp.sourceforge.net/2012/rules12.pdf` (2012).

[61] G. Sutcliffe, Private communication. (2013).

[62] H. Hoos, B. Kaufmann, T. Schaub, M. Schneider, Robust benchmark set selection for boolean constraint solvers, in: Proc. LION 7, Vol. 7997 of LNCS, Springer, 2013, pp. 138–152.

[63] L. Xu, F. Hutter, H. Hoos, K. Leyton-Brown, SATzilla: Portfolio-based algorithm selection for SAT, Journal of Artificial Intelligence Research 32 (2008) 565–606.

[64] S. Mertens, M. Mézard, R. Zecchina, Threshold values of random *K*-SAT from the cavity method, Random Struct. Algorithms 28 (3) (2006) 340–373.

[65] R. Monasson, R. Zecchina, S. Kirkpatrick, B. Selman, L. Troyansky, 2+p-SAT: Relation of typical-case complexity to the nature of the phase transition, Random Struct. Algorithms 15 (3-4) (1999) 414–435.

[66] O. Kullmann, The SAT 2005 solver competition on random instances, Journal on Satisfiability, Boolean Modeling and Computation 2 (1-4) (2006) 61–102.

[67] P. L'Ecuyer, R. Simard, Testu01: A C library for empirical testing of random number generators, ACM Trans. Math. Softw. 33 (4).

[68] A. Braunstein, M. Mézard, R. Zecchina, Survey propagation: An algorithm for satisfiability, Random Struct. Algorithms 27 (2) (2005) 201–226.

[69] A. Balint, U. Schöning, Choosing probability distributions for stochastic local search and the role of make versus break, in: Proc. SAT, LNCS, Springer, 2012, pp. 16–29.

[70] D. A. Tompkins, H. H. Hoos, UBCSAT: An implementation and experimentation environment for SLS algorithms for SAT & MAX-SAT, in: Revised Selected Papers of SAT 2004, Vol. 3542, Springer, 2004, pp. 306–320.

[71] J. Marques-Silva, I. Lynce, S. Malik, Conflict-driven clause learning SAT solvers, in: Handbook of Satisfiability, IOS Press, 2009, Ch. 4, pp. 131–153.

[72] A. Darwiche, K. Pipatsrisawat, Complete algorithms, in: Handbook of Satisfiability, IOS Press, 2009, Ch. 3, pp. 99–130.

[73] N. Eén, N. Sörensson, An extensible SAT-solver, in: SAT 2003 Selected Revised Papers, Vol. 2919 of LNCS, Springer, 2004, pp. 502–518.

[74] J. M. Silva, K. Sakallah, GRASP: A search algorithm for propositional satisfiability, IEEE Transactions on Computers 48 (5) (1999) 506–521.

[75] M. W. Moskewicz, C. F. Madigan, Y. Zhao, L. Zhang, S. Malik, Chaff: Engineering an efficient SAT solver, in: DAC, ACM, 2001, pp. 530–535.

[76] M. J. Heule, H. van Maaren, Look-ahead based SAT solvers, in: Handbook of Satisfiability, IOS Press, 2009, Ch. 5, pp. 155–184.

[77] H. Kautz, A. Sabharwal, B. Selman, Incomplete algorithms, in: Handbook of Satisfiability, IOS Press, 2009, Ch. 6, pp. 185–203.

[78] G. Audemard, L. Simon, Predicting learnt clauses quality in modern SAT solvers, in: Proc. IJCAI, 2009, pp. 399–404.

[79] S. Cai, K. Su, Configuration checking with aspiration in local search for SAT, in: Proc. AAAI, AAAI Press, 2012.

[80] A. Van Gelder, Careful ranking of multiple solvers with timeouts and ties, in: Proc. SAT, Vol. 6695 of LNCS, Springer, 2011, pp. 317–328.

[81] M. Nikolik, Statistical methodology for comparison of SAT solvers, in: Proc. SAT, Vol. 6175 of LNCS, Springer, 2010, pp. 209–222.

[82] F. Hutter, H. H. Hoos, K. Leyton-Brown, T. Stützle, ParamILS: An automatic algorithm configuration framework, J. Artif. Intell. Res. 36 (2009) 267–306.

[83] L. Xu, F. Hutter, H. Hoos, K. Leyton-Brown, Evaluating component solver contributions to portfolio-based algorithm selectors, in: Proc. SAT, Vol. 7317 of LNCS, Springer, 2012, pp. 228–241.

[84] J. Chen, Phase selection heuristics for satisfiability solvers, CoRR abs/1106.1372.

[85] A. Kügel, Natural Max-SAT encoding of Min-SAT, in: Proc. LION 6, Vol. 7219 of LNCS, Springer, 2012, pp. 431–436.

[86] N. Eén, A. Biere, Effective preprocessing in SAT through variable and clause elimination, in: Proc. SAT, Vol. 3569 of LNCS, Springer, 2005, pp. 61–75.

Appendix A. Values for Generating the Random SAT Track Benchmarks

Table A.22: The values for clause density α (at top in a cell) and the number of variables n (at bottom) used for generating each subset of random benchmarks.

Set	$k = 3$	$k = 4$	$k = 5$	$k = 6$	$k = 7$
1	4.2 40000	9 10000	20 1600	40 400	85 200
2	4.208 35600	9.121 8800	20.155 1420	40.674 360	85.558 180
3	4.215 31400	9.223 7800	20.275 1280	41.011 340	85.837 170
4	4.223 27200	9.324 6800	20.395 1140	41.348 320	86.116 160
5	4.23 23000	9.425 5800	20.516 1000	41.685 300	86.395 150
6	4.237 18800	9.526 4800	20.636 860	42.022 280	86.674 140
7	4.245 14600	9.627 3800	20.756 720	42.359 260	86.953 130
8	4.252 10400	9.729 2800	20.876 580	42.696 240	87.232 120
9	4.26 6200	9.83 1800	20.997 440	43.033 220	87.511 110
10	4.267 2000	9.931 800	21.117 300	43.37 200	87.79 100

Appendix B. Results of SC 2012: Full Rankings

Table B.23: Full results: Application SAT+UNSAT main track

Solver type	Rank	T-Rank	Solver	#solved	%solved	time (cum.)	time (med.)
vbs	-	-	*Virtual Best Solver (VBS)*	568	94.7	56528	30.3
portfolio	1	1	SATzilla2012 APP	531	88.5	85194	114.0
portfolio	2	2	SATzilla2012 ALL	515	85.8	86638	122.2
multi-engine	3	1	Industrial SAT Solver	499	83.2	93705	160.2
reference	-	-	*lingeling (SAT Comp. 2011 Bronze)*	488	81.3	84715	135.3
multi-engine	4	2	interactSAT	480	80.0	87676	152.5
single-engine	5	1	glucose	475	79.2	71501	114.4
single-engine	6	2	SINN	472	78.7	86302	146.4
single-engine	7	3	ZENN	468	78.0	74019	124.7
single-engine	8	4	Lingeling	467	77.8	91973	185.5
single-engine	9	5	linge_dyphase	458	76.3	90192	204.4
single-engine	10	6	simpsat	453	75.5	95737	222.0
single-engine	11	7	glue_dyphase	452	75.3	67412	126.4
reference	-	-	*glueminisat (SAT Comp. 2011 Silver)*	452	75.3	68818	145.7
multi-engine	12	3	CCCneq	452	75.3	94956	224.7
reference	-	-	*glucose (SAT Comp. 2011 Gold)*	451	75.2	62424	77.8
single-engine	13	8	TENN	451	75.2	82154	173.1
multi-engine	14	4	CCCeq	446	74.3	91896	230.9
reference	-	-	*CryptoMiniSat*	442	73.7	95035	240.6
portfolio	15	3	ppfolio2012	423	70.5	97819	293.0
portfolio	16	4	pfolioUZK	404	67.3	98418	348.6
reference	-	-	*minisat*	399	66.5	65633	189.5
single-engine	17	9	relback	393	65.5	75842	285.6
single-engine	18	10	Glucose++	389	64.8	75377	300.4
single-engine	19	11	satUZKs	387	64.5	70910	263.7
single-engine	20	12	caglue	387	64.5	78154	323.6
single-engine	21	13	contrasat12	383	63.8	63645	243.6
single-engine	22	14	satUZK	379	63.2	68621	282.3
single-engine	23	15	relback_m	358	59.7	65364	397.3
single-engine	24	16	riss	351	58.5	67549	434.3
multi-engine	25	5	Clingeling	278	46.3	76576	900.0
single-engine	26	17	Sat4j	249	41.5	61334	900.0
multi-engine	27	6	Flegel	231	38.5	40190	900.0
single-engine	28	18	march	37	6.2	9689	900.0

Table B.24: Full results: Hard Combinatorial SAT+UNSAT main track

Solver type	**Rank**	**T-Rank**	**Solver**	**#solved**	**%solved**	**time (cum.)**	**time (med.)**
vbs	-	-	*Virtual Best Solver (VBS)*	529	88.2	24848	1.3
portfolio	1	1	SATzilla2012 COMB	476	79.3	38108	45.4
portfolio	2	2	SATzilla2012 ALL	473	78.8	41765	45.2
portfolio	3	3	ppfolio2012	422	70.3	35784	50.5
multi-engine	4	1	interactSAT_c	417	69.5	40313	56.6
portfolio	5	4	pfolioUZK	401	66.8	34187	77.7
portfolio	6	5	aspeed-crafted	370	61.7	49239	269.3
single-engine	7	1	clasp-crafted	367	61.2	49317	277.0
reference	-	-	*MPhaseSAT (SAT Comp. 2011)*	361	60.2	35006	172.6
portfolio	8	6	claspfolio-crafted	352	58.7	42522	296.7
reference	-	-	*clasp (SAT Comp. 2011 #1 Non-portfolio)*	347	57.8	41038	322.2
single-engine	9	2	Lingeling	333	55.5	27313	291.0
multi-engine	10	2	CCCneq	329	54.8	36311	454.6
multi-engine	11	3	CCCeq	329	54.8	36943	494.3
multi-engine	12	4	Flegel	326	54.3	42999	596.2
multi-engine	13	5	Clingeling	326	54.3	47136	599.8
reference	-	-	*glucose (SAT Comp. 2011 #3 Non-portfolio)*	322	53.7	34546	515.4
single-engine	14	3	ZENN	314	52.3	27878	490.8
single-engine	15	4	simpsat	314	52.3	30999	582.3
single-engine	16	5	relback	314	52.3	32182	642.5
single-engine	17	6	SINN	313	52.2	33730	647.6
single-engine	18	7	caglue	311	51.8	30109	647.1
reference	-	-	*CryptoMiniSat*	307	51.2	32414	682.9
single-engine	19	8	satUZKs	306	51.0	40401	801.5
single-engine	20	9	contrasat12	306	51.0	40540	778.0
reference	-	-	*lingeling*	305	50.8	29095	801.4
single-engine	21	10	relback_m	304	50.7	36940	806.0
reference	-	-	*minisat*	304	50.7	39055	843.9
single-engine	22	11	satUZK	300	50.0	35028	869.6
single-engine	23	12	TENN	287	47.8	26183	900.0
single-engine	24	13	linge_dyphase	279	46.5	29183	900.0
single-engine	25	14	riss	273	45.5	31599	900.0
single-engine	26	15	Sat4j	271	45.2	26382	900.0
single-engine	27	16	sattime2012	243	40.5	30541	900.0
single-engine	28	17	sattime2011	238	39.7	28198	900.0
single-engine	29	18	gNovelty+PCL	231	38.5	12791	900.0
reference	-	-	*Sparrow2011*	217	36.2	19972	900.0
single-engine	30	19	march	217	36.2	21817	900.0
single-engine	31	20	BossLS	190	31.7	15034	900.0
single-engine	32	21	CCASat	184	30.7	9052	900.0
single-engine	33	22	sparrow2011-PCL	150	25.0	15330	900.0
reference	-	-	*EagleUP (SAT Comp. 2011)*	34	5.7	997	900.0

Table B.25: Full results: Parallel Application track

Solver type	**Rank**	**Solver**	**#solved**	**%solved**	**time (cum.)**	**time (median)**
vbs	-	*Virtual Best Solver (VBS)*	576	96.0	39670	19.6
parallel	1	pfolioUZK	531	88.5	72390	69.1
parallel	2	PeneLoPe	530	88.3	62967	54.4
parallel	3	ppfolio2012	525	87.5	78833	91.4
parallel	4	Cellulose	521	86.8	53705	42.0
parallel	5	ppfolio	509	84.8	75400	91.3
parallel	6	Sucrose	503	83.8	76120	80.7
parallel	7	ParaCIRMiniSAT	496	82.7	63497	86.7
parallel	8	clasp	490	81.7	62424	77.8
parallel	9	Glycogen	489	81.5	76241	97.1
parallel	10	ZENNfork	485	80.8	73808	89.1
parallel	11	CryptoMiniSat	482	80.3	90543	166.4
parallel	12	Plingeling	467	77.8	87632	169.2
parallel	13	CCCneq	467	77.8	87970	159.5
parallel	14	CCCeq	466	77.7	92724	167.1
parallel	15	SATX10	464	77.3	73527	124.3
parallel	16	Treengeling	457	76.2	82929	180.2
parallel	17	Minifork	428	71.3	55864	106.5
reference	-	*claspmt*	362	60.3	56435	352.3
parallel	18	splitter	273	45.5	35325	900.0