MaxSAT Evaluation 2018:
New Developments and Detailed Results

Fahiem Bacchus fbacchus@cs.toronto.edu
Department of Computer Science, University of Toronto, Canada

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

Ruben Martins[†] rubenm@andrew.cmu.edu
Department of Computer Science, Carnegie Mellon University, USA

Abstract

The series of MaxSAT Evaluations, organized yearly since 2006, has been the main forum for evaluating the state of the art in solvers for the Boolean optimization paradigm of maximum satisfiability (MaxSAT). This article provides an overview of the 2018 MaxSAT Evaluation, including a description of the main changes made in 2017 under a new organizing team, an overview of the solvers and benchmarks submitted in 2018, and detailed results of the 2018 evaluation.

Keywords: *MaxSAT, maximum satisfiability, empirical evaluation, solvers, benchmarks*

Submitted October 2018; revised March 2019; published ?

1. Introduction

Maximum Satisfiability (MaxSAT) is an optimization version of the propositional satisfiability problem (SAT). In both cases, the problem is specified by a propositional formula expressed in conjunctive normal form (CNF). In MaxSAT each clause additionally has a positive integral weight given as part of the input.[1] Unlike SAT, which is the problem of finding a truth assignment that satisfies all clauses of the CNF, MaxSAT is the problem of finding a truth assignment that maximizes the sum of weights of satisfied clauses. Many practical problems involve some component of optimization, and thus MaxSAT can be used to formalize problems from a range of application areas. For further details about MaxSAT see, e.g., [28].

Solvers for the MaxSAT problem have been evolving and improving for over a decade, and during that time our ability to solve complex MaxSAT instances has made tremendous advances. The annual MaxSAT Evaluations aims to both monitor and facilitate that progress. The main activity of the evaluation is to run as wide a set of current solvers as possible against a fixed set of benchmarks in a uniform computational environment. Since

[*] Financially supported by Academy of Finland (grants 276412 and 312662).

[†] Financially supported by NSF (grant 1762363) and CMU-Portugal (grant CMU/AIR/0022/2017).

[1] The evaluation was restricted to integral weights, but some solvers can handle rational weights.

none of the solver developers have prior knowledge of the benchmark set, the evaluation provides trustworthy and unbiased data about the performance of current solvers on the selected benchmark set. Computational limitations restrict the size of the benchmark set, so there are of course limits to what we can conclude from the data. Nevertheless, the data does provide a useful snapshot of current progress in MaxSAT solving and of the current capabilities of different MaxSAT solving algorithms and solver implementations.

The MaxSAT evaluation series has two main purposes: support for continued progress in MaxSAT solving, and promotion of MaxSAT as a viable formalism for solving a wide range of NP-hard optimization problems. Support for the research community working on improving MaxSAT solvers come from the data collected which can be used in various ways to analyze and measure the performance of current solvers; the motivation it provides through friendly competition among solver developers; the knowledge transfer and support for new developers made possible by the recent instituted open source rules; and the large and growing collection of benchmarks that are made available to the community. Promotion of MaxSAT comes mainly from disseminating the results and activities of the evaluation, and from the solicitation and collection of a diverse set of benchmarks that represent problems from a wide range of application areas. The benchmarks also serve as examples of the different problems that can be profitably represented in MaxSAT and then solved with state of the art MaxSAT solvers.

This article provides an overview of the 2018 edition of the MaxSAT Evaluations. The organizing team of the evaluations changed in 2017, and several changes were made to the way the evaluation is run (for overviews of and analysis related to some of the previous editions, see [2, 16, 3]). Since these changes have not previously been reported, an overview of the main changes incorporated in the 2017–2018 evaluations is also provided. Considering the 2018 evaluation, in particular, this article also provides an overview of the solvers and benchmarks submitted in 2018, as well as a detailed overview of the results of the 2018 evaluation. Complete benchmark and solver data are available through the MaxSAT Evaluation 2018 webpages at

https://maxsat-evaluations.github.io.

Furthermore, similarly as for the 2017 evaluation, a published evaluation proceedings are available [6]. These proceedings contain more detailed contributed descriptions of the benchmarks and solvers submitted to the 2018 evaluation.

In the rest of the article, we describe the organizational details of the 2018 MaxSAT evaluation (Section 2), and overview the participating solvers (Section 3), submitted benchmarks (Section 4), and the evaluation results (Section 5). Before that, however, for completeness we give a more formal description of the MaxSAT problem.

Maximum Satisfiability. Recall that a propositional clause is a disjunction of literals, i.e., positive and negative Boolean variables. Satisfaction of a clause by a truth assignment τ is defined in the standard way as for Boolean satisfiability. An instance $\mathcal{I} = (\varphi_h, \varphi_s, w)$ of the MaxSAT problem consists of a set φ_h of *hard* clauses, a set φ_s of *soft* clauses, and a function w which assigns a weight to each soft clause in φ_s. An assignment τ that satisfies every clause in φ_h is a *feasible solution* to $\mathcal{I}$. The *cost* of a solution τ to $\mathcal{I}$ is the sum of the weights of soft clauses falsified by τ. A feasible solution τ is an *optimal solution* if it has the smallest cost among all feasible solutions to φ. The MaxSAT problem is to find an optimal solution to a given instance $\mathcal{I}$.

2. Overview of MaxSAT Evaluation 2018

We continue by describing the main organizational details of the 2018 MaxSAT Evaluation, including new changes implemented for the 2017–2018 evaluations (Section 2.1), the evaluation tracks (Section 2.2), the ranking schemes used in evaluating the performance of the solvers (Section 2.3), disqualification of solvers (Section 2.4), the input and output formats (Section 2.5), and the computing environment used for the evaluation (Section 2.6).

2.1 New Changes and Requirements

We provide an overview of the main changes and new requirements implemented in the 2017–2018 MaxSAT evaluations and motivate these changes.

Open-source Requirement on Solvers. Before 2017, solvers participating in MaxSAT evaluations were submitted as compiled binaries; no requirements for their source availability were imposed. Starting from 2017, it is required that all solvers participating in the evaluations have their source available after the evaluation (open-source). Note that this requirement does not impose any restrictions on the software license under which the source is released. In particular, the authors of the solvers are free to apply for restricted-use licenses if they want. All that the evaluation requires is that others can read the solver's source code.

As with the SAT Competitions, this new requirement has three main motivations. Firstly, source availability promotes openness: it is important that the community has the option of analyzing the evaluation results in terms of implementation-level decisions, which has not been previously possible. Secondly, the organizers wish to make it easier for researchers to enter the world of MaxSAT solver development, with the potential of speeding up the discovery of new performance-improving techniques for MaxSAT. Thirdly, open-source solver technology is important for application domain specialists who want to tweak MaxSAT solvers to improve their performance on a specific problem domain.

Solver and Benchmark Descriptions. Starting from 2017, submitting a short, 1–2 pages written solver description is mandatory for each solver participating in the evaluations. The description must give a reasonable description of the main techniques implemented in the solver, reflecting on the differences between the new and older solver versions when applicable. The main motivation for this requirement, complementing the open-source requirement, is to ensure that key features of participating solvers can be understood (on a high level of abstraction) without the need to an in-depth understanding of the solver source codes. This also ensures the availability of a written reference to each of the participating solvers, as well as a historical account of the development of MaxSAT solvers for future purposes.

While not mandatory, all benchmark submitters are also encouraged to submit 1–2 pages written benchmark descriptions, motivated by the fact that knowledge of the underlying problem domains of evaluation benchmarks is easily lost without descriptions linking this knowledge with the individual benchmark files.

These solver and benchmark descriptions have been gathered together and made available in a published evaluation proceedings 2018 [6]. In the current paper, we provide only brief descriptions of the solvers and submitted benchmarks, so the proceedings can be consulted for further details.

Removal of the Crafted-Industrial Benchmark Distinction. Starting from 2017, no distinction is made between what were called "crafted" and "industrial" benchmarks before 2017. This change is motivated by the fact that the distinction between "crafted" and "industrial" is vague and open to subjective interpretation. This does, of course, mean that by merging these categories there will tend to be a greater variety of instances types used in the evaluation, with some instances being more "crafted" than "industrial" and others more "industrial" than "crafted". Nevertheless, our aim is to have a sufficient range of instances so that no single type of instance dominates. In fact, the main motivation behind developing declarative solver technology is its general applicability; merging of the "crafted" and "industrial" track into one "main" track encourages researchers to develop solvers that can handle a wider variety of instances.

2.2 Evaluation Tracks

The evaluation of MaxSAT solvers in MaxSAT Evaluation 2018 was divided into the following tracks. Each track is characterized by the types of benchmarks instances used within the track to evaluate the participating solvers. We note that MaxSAT Evaluation 2018 did not include a track for randomly generated MaxSAT instances due to low interest from the community to participate in such a track.

Main Tracks.

Unweighted: Combines the industrial and crafted unweighted and unweighted partial MaxSAT categories from previous MaxSAT evaluations. All benchmarks are truly unweighted, i.e., all soft clauses have unit weights. Purely randomly generated instances are not included.

Weighted: Combines the industrial and crafted weighted and weighted partial MaxSAT categories from previous MaxSAT evaluations. All benchmarks will be truly weighted, i.e., contain soft clauses with different weights. Purely randomly generated instances are not included.

Special Tracks for Incomplete Solvers. In addition to the two main tracks, focusing on complete MaxSAT solvers that provide provably optimal solutions, two special tracks (unweighted and weighted, following the categorization of the main tracks) for incomplete solvers were organized. Compared to the main tracks, noticeably smaller per-instance time limits were enforced in these tracks on the participating solvers. Furthermore, the ranking of the solvers is based on the cost of solutions provided, not requiring optimal solutions. Every incomplete solver participating in one or both of the tracks were evaluated using two distinct per-instance time limits: 60 seconds and 300 seconds.

No-restrictions Track. The aim of this special track was to allow solvers not adhering to the open-source requirements of MaxSAT Evaluation 2018 to take part in the evaluation.

Portfolios applying instance-specific algorithm selection technique to select a third-party MaxSAT solver to run on a per-instance basis were also encouraged to participate primarily in this track. As in the other tracks, solvers were required to make use of only one processor core. The benchmark set for the no-restrictions track was planned to consist of a mixture of the benchmarks in the unweighted and weighted main tracks. However, there were no solver submissions to this track, and so the track was not run.

2.3 Ranking Schemes

Ranking of solvers in the main tracks and the special tracks for incomplete solvers were based on different schemes, following the aims of the tracks.

2.3.1 Main Tracks

Following the tradition of the previous MaxSAT evaluations, solvers participating in the main tracks were ranked based on the number of solved instances within predefined per-instance resource limitations. In the main tracks, "solving an instance" requires finding an optimal solution.

2.3.2 Incomplete Solver Tracks

Solvers participating in the incomplete tracks were ranked using an incomplete score that is computed by the sum of the ratios between the best solution found by a given solver and the best solution found by any solver. More precisely, the following *incomplete score* was adopted.

$$\sum_i \frac{(\text{cost of best solution for instance } i \text{ found by any solver} + 1)}{(\text{cost of solution for instance } i \text{ found by solver} + 1)}$$

For each instance, we considered the best solution found by all incomplete solvers within 300 seconds. For an instance i the score is 0 if no solution was found by that solver. Note that, for each instance, the incomplete score is a value in $[0, 1]$.

2.4 Disqualification of Solvers

A solver was considered buggy under the following circumstances.

- The solver (in any track) reports a truth assignment in a `v` line that does not satisfy the hard clauses.
- The solver in the main track reports `s OPTIMUM FOUND` but the truth assignment reported in its `v` line has cost higher than another known solution.
- The solver (in any track) reports a cost on its `o` line that does not match the actual cost of the truth assignment reported on the `v` line.
- The solver crashes on a significant number of instances.

A somewhat relaxed approach was taken to solvers that did not provide correct answers on every benchmark—the event is designed to be an evaluation, not a competition. In particular, a solver would not be immediately disqualified if it displayed a wrong solution during the execution of the evaluation. The organizers allowed all participants a fair chance of submitting bug fixes to their solvers in a timely manner based on feedback on wrong results.

2.5 Input and Output Format

The input-output format has been slightly revised for uniformity for the 2017–2018 MaxSAT Evaluations. We will here specify the current format for completeness; the 2017–2018 benchmark sets are in the new format.

2.5.1 Input Format

Starting in 2017 the same input format is used for all benchmark instances. The parameters line is `p wcnf nbvar nbclauses top`, where `nbvar`, `nbclauses` and `top` are the highest variable number, the number of clauses (soft and hard), and a weight indicating that a clause is hard, respectively. We associate a weight with each clause, specified by the first integer in the clause. The remaining numbers specify the positive and negative literals in the clause, and each clause is terminated by 0.

Hard clauses have weight `top`, soft clauses have a weight smaller than `top`, and `top` is always greater than the sum of the soft clause weights. Weights must be greater than or equal to 1, and the sum of the weights of the soft clauses must be less than $2^{64} - 1$ (so that `top` is at most 2^{64}).

For example, the following specifies a legal MaxSAT instance.

```
c
c comment line
c
p wcnf 4 5 16
16 1 -2 4 0
16 -1 -2 3 0
8 -2 -4 0
4 -3 2 0
3 1 3 0
```

2.5.2 Output Format

The solvers were required to output messages to standard output. The output format is inspired by the DIMACS output specification of the SAT competition. The solvers were not allowed to write to files, only standard output, and standard error. Solver output necessary for parsing the evaluation results were required to be sent to standard output. The output lines are as follows.

Comments (`c` lines): Comment lines start with the two characters: lower case `c` followed by a space. These lines are optional and may appear anywhere in the solver output. They contain any information that authors want to emphasize.

Solution Status (`s` line): This line starts with the two characters: lower case `s` followed by a space. Only one such line is allowed. This line gives the answer of the solver. It must be one of the following answers:

`s OPTIMUM FOUND` This line is to output when the solver has checked that the last `o` and `v` lines (see the following) specify an optimal solution.

`s UNSATISFIABLE` This line is to be output when the solver has checked that the set of hard clauses are unsatisfiable.[2.]

`s UNKNOWN` This line must be output in any other case. Note that solvers in the incomplete track will typically output `s UNKNOWN` unless they can verify that the best solution they have found is optimal, in which case they can output `s OPTIMUM FOUND`. However, no extra score was associated with the incomplete track for proving optimality. Exact solvers in the main track can output `s UNKNOWN` if they are unable to find an optimal solution but such an answer receives no score. (Note that not outputting an `s` line is the same as outputting `s UNKNOWN`)

Solution Cost Line (`o` lines): These lines start with the two characters, lower case `o` followed by a space, and then followed by a number representing the cost of the best solution found, i.e., the sum of the weights of clauses falsified by the solution.

Incomplete solvers have to catch the SIGTERM signal so that they can output an `o` (and `v`) line specifying the best solution found just before termination. There is no need for exact solvers in the main track to output any solution other than an optimal one. When they find an optimal solution they can output an `o` and `v` line specifying the solution they found.

Solution Values (Truth Assignment) (`v` lines): These lines start with the two characters: lower case `v` followed by a space. More than one `v` line is allowed but the evaluation environment will act as if their content was merged. When the solver reports a solution it must provide `s`, `o` and `v` lines. The `v` line provides a truth assignment to the variables of the instance that will be used to check the correctness of the answer, i.e., it must provide a list of non-complementary literals which, when interpreted as true, achieves a sum of weights of unsatisfied clauses as specified in the `o` line. The solution line must define the value of each variable. The order of literals does not matter. If the solver does not output a value line, or if the value line does not match the specification, then UNKNOWN will be assumed.

For example, a valid output of a solver that has found an optimal solution might be as follows.

```
c --------------------------
c My MaxSAT Solver
c --------------------------
o 143
s OPTIMUM FOUND
v -1 2 3 -4 -5 6 -7 8 9 10 -11 -12 13 -14 -15 16 -17 18 19 20
```

[2.] To the best of our knowledge the hard clauses of all instances used were satisfiable, and no solver returned `UNSATISFIABLE` as an answer in the evaluation. However, in future evaluations we plan to verify the SAT status of the hard clauses.

2.6 Computing Environment

MaxSAT Evaluation 2018 was run on the StarExec cluster (https://www.starexec.org/) using computing nodes with the following technical specifications: Intel(R) Xeon(R) CPU E5-2609 0 @ 2.40GHz with 10240-KB cache and 128-GB main memory. To minimize the effect of other processes affecting the runtimes of the solver, at most two solvers were run on any node at the same time. For the complete tracks, a time limit of 3600 seconds and a memory limit of 32 GB was enforced on each solver run. For the incomplete track, the memory limit was also 32 GB, and two different time limits were enforced: 60 seconds and 300 seconds.

3. Participating Solvers

We proceed with an overview of the solvers that participated in MaxSAT Evaluation 2018.

3.1 Complete Solvers

Nine different solvers participated in the complete tracks (Section 2.2), with eight of the solvers participating in both the weighted and unweighed complete tracks, and one participating only in the weighted track (see Table 1).

The complete solvers used three different algorithmic approaches to solving MaxSAT (with some significant differences in the way each of these approaches were realized).

Two solvers used the implicit hitting set approach originating from [10]. In this approach, cores are computed by a SAT solver and minimum-cost hitting sets of those cores computed by an integer programming solver, in a loop until a truth assignment is found that falsifies a weight of clauses equal to the weigh of the minimum-cost hitting set. Such a truth assignment must be optimal.

Five of the solvers used a core based UNSAT to SAT approach. In particular, these solvers solve a sequence of unsatisfiable SAT instances where the first satisfiable instance gives a MaxSAT solution. These approaches build on the original idea of Fu and Malik [15] of exploiting the cores, discovered when the SAT instance is UNSAT, to construct the next SAT instance to solve.

Finally, two solvers use a SAT to UNSAT sequence of SAT instances, where each SAT instance is constrained to find an improved solution, and the final UNSAT instance indicates that the last model found was optimal (i.e., no better solutions exist). This is an old idea that in the SAT context seems first to have been used in MiniSAT+ [13].

A few more details about the submitted solvers are given in the following (see [6] for more details and citations to relevant publications).

LMHS. This is an implicit hitting set solver by Paul Saikko (main developer), Jeremias Berg, and Matti Järvisalo all from the University of Helsinki, Finland. It was originally developed in [32] and has been extended in various ways [33], including MaxSAT specific preprocessing techniques implemented by Tuukka Korhonen [20]. It uses MiniSAT 2.2 as the underlying SAT solver and CPLEX 12.8 [3.] as the underlying integer programming solver.

[3.] Available for academics at https://www.ibm.com/developerworks/community/blogs/jfp/entry/cplex_studio_in_ibm_academic_initiative?lang=en

Table 1: Solvers participating in the complete tracks—approach and coverage.

Solver	Weighted	Unweighed	Approach
LMHS	✓	✓	Implicit Hitting Set
MaxHS	✓	✓	Implicit Hitting Set
maxino	✓	✓	UNSAT to SAT core based
Open-WBO-Glucose	✓	✓	UNSAT to SAT core based
Open-WBO-Riss	✓	✓	UNSAT to SAT core based
RC-B	✓	✓	UNSAT to SAT core based
RC-A	✓	✓	UNSAT to SAT core based
Pacose	✓		SAT to UNSAT
QMaxSAT	✓	✓	SAT to UNSAT

MaxHS. This solver was originally developed in Davies [10] and was authored by Fahiem Bacchus from University of Toronto, Canada. The solver utilizes a number of techniques to improve the effectiveness of the implicit hitting set approach. As the underlying SAT and IP engines, it uses MiniSAT 2.2 and CPLEX 12.8, respectively.

maxino. This solver was authored by Mario Alviano from the University of Calabria, Italy. It employs a special technique for converting the set of found cores into cardinality constraints originally described in [1]. It uses Glucose 4.1 [5] as the underlying SAT solver.

Open-WBO-Glucose and Open-WBO-Riss. This solver was authored by Ruben Martins from CMU, USA; Norbert Manthey from Germany; and Miguel Terra-Neves, Vasco Manquinho, and Ineŝ Lynce, from INESC-ID/IST, Portugal, building on [26]. It uses partition-based techniques [30] and incremental constraints that can be quickly updated when a new core is found [25]. The two different versions -Glucose and -Riss differ only by which SAT solver backend is used (Glucose 4.1 or RISS).

Pacose. This solver was authored by Tobias Paxian, Sven Reimer, and Bernd Becker from Albert-Ludwigs-Universität, Germany. It uses the QMaxSat solver and adds a new way to encode the pseudo-Boolean constraints [31]. Since the new encoding is only for weighted problems the solver did not participate in the unweighted competition. It uses Glucose 3.0 as the backend SAT solver.

QMaxSAT. This solver was authored by Aolong Zha from Kyushu University, Japan, building on [21]. It handles weighted problems by encoding a pseudo-boolean constraint capturing the sum of the weights of the falsified soft clauses and modifies this constraint to force the SAT solver to find improving solutions following the linear SAT/UNSAT (LSU) algorithm [7, 13]. It uses Glucose 3.0 as the underlying SAT solver.

RC-B and RC-A. This solver was authored by Alexey Ignatiev, Antonio Morgado, and Joao Marques-Silva from the University of Lisbon, Portugal building on the PySAT

Table 2: Solvers participating in the incomplete tracks and the main techniques employed.

Solver	LSU	Clustering	Weight Simp.	local search	Other Technique
LinSBPS	✓		✓		✓
MaxRoster	✓			✓	✓
Open-WBO-Inc-MSC	✓				✓
Open-WBO-Inc-OBV	✓	✓			
Open-WBO-Inc-Cluster	✓	✓	✓		
Open-WBO-Inc-BMO	✓	✓	✓		
Open-WBO-(Gluc/Riss)	✓				
SatLike				✓	
SatLike-c	✓			✓	

framework [17]. It is a new implementation of the OLL MaxSAT algorithm [27] with the addition of a number of other enhancements. The two different versions -A and -B differ only in the policy for core minimization (-A uses core minimization whereas -B does not). It uses Glucose 3.0 as the underlying SAT solver.

3.2 Incomplete Solvers

The incomplete track, in which the solvers return the best solution they can find within the time-bound, attracted nine different solvers (see Table 2).

The most common technique used in these incomplete solvers was the linear SAT/UNSAT (LSU) algorithm [7, 13] in which a satisfying solution for the hard clauses is first found, and then, via a CNF-encoded cardinality (for unweighted instances) or a CNF-encoded pseudo-Boolean constraint (for weighted instances), the SAT solver is then forced to find a better solution. The solvers did, however, vary in the techniques they used to encode these constraints.

One advantage of LSU in the incomplete track is that it appears to often find the first few improving solutions quite rapidly. The other advantage is that it can be used to try to improve the best solution found via other techniques, one can simply give it the best solution found so far as a starting point and LSU will force the SAT solver to try to find a better solution.

Additional techniques that were often used in the incomplete solvers include: **clustering** techniques to partition the set of soft clause so that some of the softs could be ignored at various stages of the computation, **weight simplification** techniques (on the weighted instances) so that a simpler problem with fewer distinct weights could be solved, and **local search**.

A few more details about the submitted solvers are given in the following (see [6] for more detail about these solvers and more citations to relevant publications).

LinSBPS. This solver is by Emir Demirović and Peter Stuckey from the University of Melbourne, Australia. It makes direct use of LSU with the added technique of solution based phase saving [11]. It also simplifies the weights by dividing them by a constant (which via integer division sets some soft clauses to weight zero, effectively removing them). The best solution found with the most simplified weights are then used as the starting point for a new LSU computation with less simplified weights. The solutions found for the problem with simplified weights are also solutions for the original problem and thus the best solution found can be tracked and returned when the time limit is reached. It is built on top of Open-WBO and uses Glucose 4.1 as the underlying SAT solver.

MaxRoster. This solver is by Takayuki Sugawara from Sugawara Systems, Japan. It first runs a local search procedure for a short amount of time to try to find a good solution. It then uses either LSU or an unsat-based algorithm based on the ratio of the cost of initial solution and the sum of weights of the soft clauses. It uses MapleSAT as the SAT solver for the LSU and unsat-based algorithms.

Open-WBO-Inc-MSC. All four of the Open-WBO-Inc solvers are authored by Saurabh Joshi, Prateek Kumar and Sukrut Rao, from IIT Hyderabad, India; Vasco Manquinho from INECS-ID, Portugal; Ruben Martins from CMU, USA; and Alexander Nadel from Intel Corporation, Israel. This particular solver participated in the **unweighted** tracks only. It utilizes an algorithm for enumerating minimum correction sets (MCSs). Each MCS is a set of soft clauses whose removal renders the remaining hard and soft clauses satisfiable, and thus yields a potentially reasonably low cost solution. After enumerating MCS's under some fixed resource bounds the best solution found is used as a starting point for running LSU.

Open-WBO-Inc-OBV. This solver participated in the **unweighted** tracks only. It uses a number of different orderings of the soft clauses and attempts to find a solution satisfying the first k soft clauses in that ordering, for increasing values of k stopping when the first k softs cannot be simultaneously satisfied. Various techniques are used for picking the ordering [29]. After doing this under some fixed resource bounds the best solution found is used as a starting point for running LSU.

Open-WBO-Inc-Cluster. This solver participated in the **weighted** tracks only. It first clusters the soft clauses into different groups and gave all of the clauses in each group the same weight. It then solves this weight simplified version of the problem with LSU with special techniques for taking advantage of the regularity of the simplified weights [19]. If LSU successfully solved the weight simplified version, the solution found is used as a starting point for running LSU again on the original unsimplified problem.

Open-WBO-Inc-BMO. This solver participated in the **weighted** tracks only. It clusters the soft clauses by weight so that each set of soft clauses with the same weight are in a separate cluster, and then sorts the clusters by decreasing weight. LSU is then employed to find optimal solutions to the first k clusters for increasing k, under the restriction that the number of falsified soft clauses in clusters 1 to $k-1$ cannot be increased [19]. After this, if there is time left, LSU is called on the original unsimplified problem with the best solution found as its starting point.

Table 3: Unweighted benchmark families submitted to the 2018 evaluation. The # column show the number of instances in that family. The Clauses, Variables, and %Softs columns show the minimum and maximum of the number of clauses, number of variables and percentage of softs among the instances in that family.

Family	#	Clauses	Variables	%Softs
drmx-atmostk	36	421–4,523	181–3,013	1.55–9.50
drmx-cryptogen	40	9,044–102,000	1,600–22,880	16.94–22.45
optic	65	349–171,411	153–20,294	0.31–64.64
uaq	97	1,664–8,831	516–6,050	2.47–7.81
vpa	67	604–559,809	55–5,356	0.96–9.11
xai-mindset2	45	864–22,324,591	264–76,107	0.031–7.353

Open-WBO-Glucose and Open-WBO-Riss. This solver is an incomplete version of Open-WBO-Glucose and Open-WBO-Riss solvers used in the complete track (described above) and is written by the same authors. It uses the LSU algorithm and also employed solution phase saving.

SatLike. This solver is authored by Zhendong Lei and Shaowei Cai from Institute of Software Chinese Academy of Sciences, Beijing, China. It employes a local search technique designed to handle both hard and soft clauses [22].

SatLike-c. This solver is by the same authors as SatLike, and it runs SatLike under some fixed resource bound after which it runs LSU. It relies on the Open-WBO framework for the LSU algorithm.

4. Benchmarks

In this section, we will briefly describe the new MaxSAT benchmarks submitted to the 2018 evaluation, and then discuss the set of benchmarks that the submitted solvers were run against in each of the evaluation categories.

4.1 Submitted Benchmarks

Nine new benchmark families were submitted to the 2018 evaluation, as summarized in the Tables 3 and 4. All of these new families are described in more detail in [6], but we provide a brief description for each in the following.

drmx-atmostk. These are a set of benchmark instances each of which encodes a formula saying that we would like to make all variables true, along with a hard cardinality constraint saying that at most k can be true. More precisely, each instance is a formula $F(m,k) = H_{m,k} \wedge S_m$, where $H_{m,k}$ is a CNF encoding of the cardinality constraint $\sum_{i=1}^{m} x_i \leq k$ (using different encodings) and S_m is the set of unit soft clauses $\{(x_1), \ldots, (x_n)\}$. These formulas all have an obvious optimal solution—simply pick any k variables and make them all true and the rest false. This will satisfy the maximum number k of soft clauses. In the weighted

Table 4: Weighted benchmark families submitted to the 2018 evaluation. The columns are as in Table 3 and in addition the #Weights column shows the minimum and maximum number of different weights among the instances in each family. If all instances have the same value, only one number is given.

Family	#	Clauses	Variables	#Weights	%Softs
cluster-expansion	21	60,375	125	71	100
drmx-atmostk	36	421–4,523	181–3,013	2	1.55–9.50
drmx-cryptogen	40	9,044–102,000	1,600–22,880	2	100
max-realizability	87	1,436–743,834	203–171,619	1–3	0.001–0.341
tcp	60	131,943–243,546	128,513–238,313	3	0.10–0.22

version of these instances, the clauses of $H_{m,k}$ are given a weight of $m+1$ while the clauses of S_m have weight 1. Hence, it is better to falsify all the clauses of S_m rather than one of the clauses of $H_{m,k}$. Many solvers can exploit this fact to quickly conclude that the clauses of H can be hardened.

On the other hand, it can also be noted that Davies analyzed such instances in her thesis [10] and showed that implicit hitting set solvers like MaxHS and LMHS must exhibit exponential runtimes on these instances.

drmx-cryptogen. These are a set of relatively easy SAT formulas encoding cryptanalysis of some stream cipher generators that have been converted to MaxSAT formulas using the so-called dual rail encoding. The dual rail encoding has been shown to translate some hard SAT formulas into easy MaxSAT instances (notably the pigeonhole principle formulas) [8]. In this case, however, the dual rail encoding to MaxSAT was expected to make the instances harder. Nevertheless, in the evaluation some solvers were able to solve all instances included in the test suite. In the weighted instances all of the soft clauses were given weight 1 while the hard clauses were given a weight large enough to make it better to falsify all soft clauses rather than a single hard clause.

optic. These are a set of instances for automatically computing good CNF encodings of various building block constraints. That is, in modeling real problems certain types of constraints might frequently appear, e.g., cardinality constraints. The approach is to construct a MaxSAT instance whose solution specifies a CNF encoding that optimally trades of size for the unit-propagating power the encoding achieves [14]. The benchmark set consists of a number of these MaxSAT instances.

uqa. These are a set of instances encoding the solving of various user authorization queries. The user wants to perform an activity (typically on database system) that requires some set of permissions be granted. Control of these permissions is organized by granting the user a number of roles each of which carries some set of permissions, subject to some constraints on the set of roles that can be simultaneously assigned. The MaxSAT instances encode the problem of computing an optimal set of roles that grants the user the permissions they need [4].

vpa. These are a set of instances encoding the minimization of visibly pushdown automata. In the MaxSAT encoding each satisfied soft clause indicates that a pair of states in the automata can be merged, and the hard clauses ensure that the minimized automaton accepts the same language.

xai-mindset2. These are a set of instances encoding the problem of minimizing the size of the decision rules in a set of k decision rules used to classify data items. In particular, for explainable decisions as to why each data item was classified the way it was, the rules used to make that classification should be as short as possible. So once the initial set of rules have been determined the MaxSAT instances encode the optimization problem of removing as many conditions from those rules as possible while still maintaining the accuracy of the decisions [18].

cluster-expansion. These are a set of benchmark instances encoding generalized Ising models, see the description in [6] for more details. Finding a most probable assignment in a (non-generalized) binary Ising model is known to correspond to computing a maximum cut in a corresponding graph, and previous evaluations have used Max-Cut instances that correspond to solving binary Ising models.

max-realizability. These are a set of instances involving the synthesis of systems satisfying a collection of linear temporal logic (LTL) formulas (specifications) [12]. If the synthesized system satisfies one of the LTL formulas f, then f is said to be realized. These instances encode the synthesis problem as a MaxSAT problem whose solution synthesizes a system that realizes a maximum number the LTL formulas. There were two different domains encoded among the instances. A small number of instances encoded a robotic navigation problem, while the majority of the instances encoded power distribution network problems.

tcp. These are a set of instances for finding optimal assignments of students to desks. In these problems the desks have different sizes and the students have preferences as to who they would like to sit with [24].

4.2 Benchmarks Used in the Evaluation

The benchmark sets used in the evaluation were constructed by randomly picking instances from the set of all new benchmarks submitted to the evaluation and the set of all instances from all previous evaluations. As in 2017, we set a predefined upper limit on the number of benchmarks selected from each benchmark family. This was done to ensure that no particular problem domains would be over-represented in the resulting benchmark sets, which has been an issue in the pre-2017 benchmark sets. In particular, to put more emphasis on new submitted benchmarks, we enforced an upper limit of 40 benchmarks from each new submitted benchmark domain, and an upper limit of 25 benchmarks for the old benchmark domains.

For the incomplete track, we selected a subset of the benchmarks used in the main tracks that were not optimally solved by any complete solver in less than 300 seconds. This ensures that the incomplete benchmarks are challenging for complete approaches.

More detailed information about the composition of the benchmark test suite used in each track is given in Tables 5 to 8.

Table 5: Unweighted benchmark used in the complete unweighted track of the 2018 evaluation. The columns show the same information as Table 3. In addition, the column Year shows the first year this benchmark family appeared in the annual evaluations.

Family	#	Year	Clauses	Variables	%Softs
aes	7	2011	387–201,533	147–136,253	7.18–67.61
aes-key-recovery	20	2015	372,403–372,649	21,368–21,368	0.044–0.11
atcoss/mesat	15	2014	969,655–1,605,464	321,140–467,197	0.019–0.031
bcp-fir	20	2008	9,206–583,176	3,041–203,287	1.03–17.75
bcp-syn	25	2008	111–14,421	67–25,167	22.53–98.17
bcp-msp	25	2008	1,340–476,372	360–14,500	3.04–32.07
circuitDebuggingProblems	3	2009	621,323–2,223,029	399,591–1,304,121	100.0
circuitTraceCompaction	4	2009	38,700–2,452,560	15,760–913,940	0.0008–0.0518
close_solutions	20	2013	137,496–4,810,148	2,888–314,455	1.30–27.03
des	20	2013	240,293–619,960	53,176–136,160	1.20–1.22
drmx-cryptogen	20	2018	9,044–102,000	1,600–22,880	16.94–22.45
drmx-atmostk	20	2018	421–4,523	181–3,013	1.55–9.50
extension-enforcement	25	2017	105,506–1,120,160	16,000–76,190	3.43–21.27
fault-diagnosis	25	2016	659,998–1,360,750	130,048–167,004	3.66–7.54
frb	15	2009	5,821–42,485	220–760	1.79–3.78
gen-hyper-tw	25	2017	11,754–2,413,400	3,852–1,225,430	0.006–0.121
hs-timetabling	2	2014	528,321–557,113	69,783–160,572	0.18–0.64
jobshop	3	2009	359,683–1,151,201	35,772–110,641	0.01–0.04
kbtree	10	2009	1,297–1,309	280–280	71.94–72.19
maxclique/structured	10	2011	630–378,247	70–1,035	0.26–34.33
maxcut/dimacs_mod	8	2011	348–1,836	40–64	100.0
maxcut/spinGlass	2	2007	162–1,296	27–216	100.0
min-fill	15	2017	2,570–3,451,298	594–278,608	0.33–1.36
optic	40	2018	843–171,411	153–20,294	0.95–64.65
protein_ins	12	2008	12,967–3,848,496	171–3,016	0.002–0.100
reversi	20	2014	4,072–334,992	836–55,432	0.02–0.79
scheduling	5	2013	47,065–1,171,279	12,617–289,958	0.16–0.57
SeanSafarpour	24	2008	140,056–8,812,799	45,552–2,785,108	100.0
set-covering/crafted/scpclr	4	2014	721–4,810	210–715	100.0
set-covering/crafted/scpcyc	6	2014	432–39,424	192–11,264	100.0
tpr/multiple_path	10	2013	149,783–933,162	49,688–312,598	0.35–0.37
treewidth-computation	20	2015	30,519–3,803,650	6,025–341,053	0.003–0.0721
uaq	40	2018	1,664–8,831	516–6,050	2.47–7.81
vpa	40	2018	604–559,809	55–5,356	0.96–9.11
xai-mindset2	40	2018	864–22,324,591	264–76,107	0.031–7.35
All	600		111–22,324,591	27–2,785,108	0.0008–100.0

Table 6: Weighted benchmark used in the complete weighted track of the 2018 evaluation. The columns show the same information as Table 4. In addition, the column Year shows the first year this benchmark family appeared in the annual evaluations.

Family	#	Year	Clauses	Variables	#Weights	%Softs
abstraction-refinement	10	2017	5,142,998–22,435,229	2,254,256–9,046,260	2	0.03–0.08
af-synthesis	25	2017	75,792–432,346	11,550–25,385	9	0.06–0.09
auctions/auc_paths	15	2012	1,766–3,016	130–172	43–64	5.67–8.23
BTBNSL	25	2015	2,150–634,165	620–110,096	28–3,098	0.073–2.46
causal-discovery	25	2015	46,116–3,138,850	12,733–868,916	151–9,409	0.37–0.52
cluster-expansion	20	2018	60,375	125	71	100.0
correlation-clustering	25	2015	67,838–2,664,958	16,135–571,739	428–15,295	1.92–2.38
CSG	10	2013	29,699–1,654,241	791–11,325	22–144	0.005–0.07
css-refactoring	11	2017	7,459–399,473	1,901–23,379	71–195	2.91–10.86
dalculus	25	2017	1,971–77,089	873–21,118	7	0.71–3.25
drmx-atmostk/weighted	20	2018	519–4,523	194–3,013	2	100.0
drmx-cryptogen/weighted	20	2018	9,044–102,000	1,600–22,880	2–2	100.0
frb	20	2008	661–42,499	60–760	2	100.0
haplotyping-pedigrees	25	2012	276,866–3,700,524	62,161–215,039	2	0.51–1.63
hs-timetabling	13	2014	25,262–2,448,118	7,520–669,064	2–30	0.14–2.15
lisbon-wedding	20	2017	104,116–232,451	29,663–33,626	3–4	0.15–0.32
max-realizability	43	2018	2,267–743,834	246–171,619	3	0.001–0.34
maxcut/dimacs_mod	17	2011	348–3,648	28–64	10–10	100.0
maxcut/spinGlass	3	2006	162–2,058	27–343	81–1,024	100.0
metro	15	2017	126,470–863,825	44,375–158,766	9–19	0.02–0.10
min-width	20	2017	5,083–374,950	3,880–192,509	174–4,700	1.96–9.16
miplib	10	2007	238–99,188	123–24,776	9–227	0.07–16.81
railway-transport	5	2015	719,175–17,367,985	71,809–392,516	2–1,748	0.07–0.98
relational-inference	8	2017	119,999–23,364,255	40,810–19,264,629	3–36	7.31–99.57
rna-alignment	25	2017	9,239–1,414,209	857–6,685	2–7	0.14–2.44
shiftdesign	15	2017	679,154–16,584,658	335,687–6,972,354	3	0.038–0.32
spot5/log	25	2012	479–37,639	96–3,004	2–5	2.48–15.31
staff-scheduling	10	2017	17,506–2,308,011	4,750–558,350	4	0.37–2.59
tcp	40	2018	132,073–243,215	128,530–238,260	3	0.10–0.22
timetabling	20	2010	1,602,794–51,338,353	423,364–7,515,310	20–173	0.12–0.26
upgradeability/wpms	25	2011	112,632	18,169	3–3	16.22
warehouses	10	2009	1,955–12,648	836–5,200	106–2,482	20.15–20.46
All	600		162–51,338,353	27–19,264,629	2–15,295	0.001–100.0

Table 7: Unweighted benchmark used in the incomplete unweighted track of the 2018 evaluation. The columns show the same information as Table 5.

Family	#	Year	Clauses	Variables	%Softs
aes	5	2011	73,120–201,533	7,840–136,253	10.72–67.61
aes-key-recovery	1	2015	372,403–372,403	21,368–21,368	0.04
atcoss/mesat	7	2014	969,655–1,605,464	321,140–465,243	0.019–0.031
bcp-msp	13	2008	4,350–476,372	1,200–14,500	3.04–32.07
bcp-syn	2	2008	6,552–13,034	6,136–25,167	78.10–88.75
close_solutions	1	2013	1,298,432	289828	22.32
des	2	2013	560,857–587,549	12,4024–12,9928	1.19817–1.19820
extension-enforcement	13	2017	105,506–188,290	42,742–76,190	21.19–21.27
fault-diagnosis	3	2016	1,005,071–1,111,264	146,219–157,161	4.48–4.95
gen-hyper-tw	23	2017	75,405–2,413,400	46,941–1,225,430	0.006–0.121
hs-timetabling	1	2014	557,113	69,783	0.64
maxclique/structured	3	2011	112,234–378,247	800–1,000	0.264–0.713
maxcut/dimacs_mod	6	2011	1,132–1,836	40–64	100.0
maxcut/spinGlass	1	2007	1,296	216	100.0
min-fill	8	2017	53,140–1,209,120	9,036–125,992	0.39–0.64
optic	11	2018	7,643–171,411	1,044–20,294	2.22–64.65
reversi	6	2014	62,928–136,176	10,452–22,562	0.05–0.11
scheduling	3	2013	47,065–1,171,279	12,617–289,958	0.16–0.57
SeanSafarpour	4	2008	1,759,150–8,812,799	463,080–2,785,108	100.0
set-covering/crafted/scpclr	3	2014	1,353–4,810	330–715	100.0
set-covering/crafted/scpcyc	6	2014	432–39,424	192–11,264	100.0
treewidth-computation	5	2015	38,856–1,023,459	7,274–118,369	0.007–0.062
uaq	9	2018	4,276–5,759	2,046–2,046	2.60–3.51
xai-mindset2	17	2018	31,470–22,324,591	6,412–76,107	0.03–2.74
All	153		432–22,324,591	40–2,785,108	0.006–100.0

Table 8: Weighted benchmark used in the incomplete weighted track of the 2018 evaluation. The columns show the same information as Table 6.

Family	#	Year	Clauses	Variables	#Weights	%Softs
abstraction-refinement	2	2017	12,681,081–13,561,252	6,422,659–6,536,173	2–2	0.026–0.047
af-synthesis	19	2017	75,792–432,346	11,550–25,385	9	0.05–0.09
BTBNSL	16	2015	31,112–634,165	7,067–110,096	268–3,098	0.25–1.77
causal-discovery	14	2015	145,910–3,138,850	40,290–868,916	494–9,409	0.37–0.46
cluster-expansion	20	2018	60,375	125	71	100.0
correlation-clustering	12	2015	156,237–2,664,958	35,940–571,739	914–15,295	1.92–2.38
hs-timetabling	13	2014	25,262–2,448,118	7,520–669,064	2–30	0.14–2.14
lisbon-wedding	12	2017	110,267–232,451	29,748–33,626	3	0.16–0.33
max-realizability	5	2018	301,030–743,834	25,185–171,619	3	0.001–0.010
maxcut/dimacs_mod	10	2011	1,092–3,648	40–64	10	100.0.0
maxcut/spinGlass	1	2006	2058	343	1024	100.0
min-width	16	2017	18,005–374,950	11,558–192,509	285–4,700	1.96–9.16
miplib	5	2007	19,630–99,188	5,519–24,776	26–227	0.07–3.23
railway-transport	4	2015	2,285,655–17,367,985	85,796–392,516	2–1,748	0.07–0.98
relational-inference	2	2017	22,951,677–23,364,255	17,997,976–19,264,629	36	53.0–64.7
spot5/log	3	2011	15,294–29,921	701–1,335	5	2.86–3.40
staff-scheduling	10	2017	17,506–2,308,011	4,750–558,350	4	0.37–2.59
tcp	7	2018	132,268–242,852	128,562–238,175	3	0.11–0.18
timetabling	1	2010	10,632,939	1,908,942	103	0.18
All	172		1,092–23,364,255	40–19,264,629	2–15,295	0.001–100.0

5. Evaluation Results

Finally, we turn to the evaluation results. We provide an overview of the results by first focusing on the main tracks and then on the special tracks for incomplete solvers.

5.1 Main Tracks

5.1.1 Unweighted

Table 9 shows the number of solved instances and average solving time in seconds for incomplete solvers on unweighted instances. RC2-B was the best-performing solver in the unweighted track, closely followed by RC2-A and maxino. Recall that the difference between RC2-B and RC2-A lies in core minimization, RC2-B performs a limited form of core minimization while RC2-A does not.

Figure 1 shows a cactus plot with the per-solver, per-instance running times in ascending order. We can observe three distinct clusters with similar performance. The first cluster is formed by RC2-B, RC2-A and maxino. These solvers use similar unsat-based algorithms and solving techniques. The second cluster is formed by MaxHS, Open-WBO-Gluc and Open-WBO-Riss and the third cluster by LMHS and QMaxSAT. Note that LMHS and MaxHS implement similar algorithms but MaxHS enhancements allows it to clearly outperform LMHS. QMaxSAT is the only sat-unsat solver and performs poorly on unweighted benchmarks.

The virtual best solver (VBS) is equivalent to run all solvers in parallel and stop when the first solver finds an optimal solution. There is a clear gap between the VBS and RC2-B which shows that solvers use different solving techniques that can solve a wide range of benchmarks.

Table 10 shows the number of instances solved by each solver including the VBS for each family. To improve readability, we excluded RC2-A and Open-WBO-Riss from the table since they are similar (in both implementation and performance) to RC2-B and Open-WBO-Gluc. Open-WBO-Gluc is denoted by Open-WBO in Table 10. While there are

Table 9: Ranking of complete solvers on unweighted instances including the VBS.

Solver	#Solved	Time, average (s)
VBS	472	92.73
RC2-B	421	126.32
RC2-A	416	138.98
maxino	405	137.5
MaxHS	386	178.06
Open-WBO-Gluc	382	171.54
Open-WBO-Riss	371	154.23
LMHS	323	261.75
QMaxSAT	292	257.03

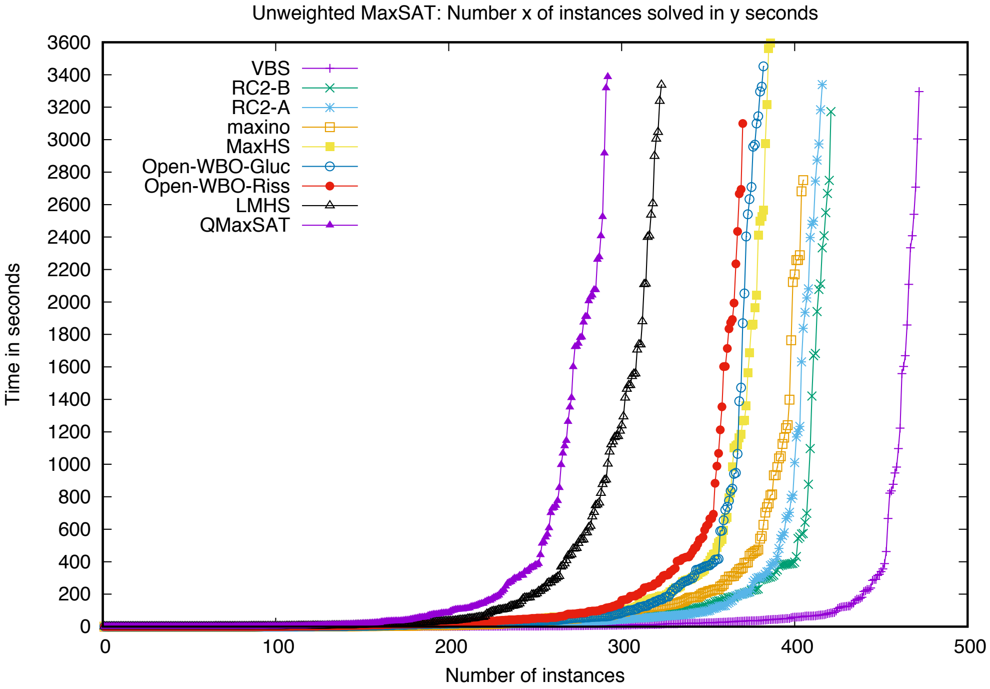


Figure 1: Cactus plot for complete solvers on unweighted instances including the VBS.

families where the performance of most MaxSAT solvers is similar, there are other families on which the performance differs greatly. We highlight some of these families that exhibit some interesting results.

Analysis of Sat-Unsat approaches. The only sat-unsat solver submitted to the unweighted track was QMaxSAT. We can observe that it performs poorly on some families such as **aes-key-recovery** and **extension-enforcement**. The poor performance can be partially explained by a large number of soft clauses for some of these benchmarks and by the optimal value being small. For example, the **aes-key-recovery** family contains on average 111,940 soft clauses per benchmark with optimal values of 1 or 2 for most benchmarks. The **extension-enforcement** also has similar properties with 32,924 soft clauses on average per benchmark and optimal solutions unsatisfying a small number of soft clauses (on average 50 or less).

Complementarity between Hitting Set and Unsat-based approaches. These approaches often complement each other for several families. For example, hitting set approaches perform well for the families **optic** and **kbtree**, whereas unsat-based approaches perform poorly on these families. In contrast, unsat-based approaches perform well for the families **fault-diagnosis** and **protein_ins** whereas hitting set approaches exhibit poor performance on these families.

Mixed behavior. Some families exhibit mixed behavior. An interesting example is the **drmx-atmostk** family for which some solvers are able to find an optimal solution for all instances in that family while others were unable to find a single optimal solution. This

Table 10: Results per family for the complete unweighted track of the 2018 evaluation.

Family	#	VBS	RC2-B	maxino	MaxHS	Open-WBO	LMHS	QMaxSAT
aes	7	2	1	1	**2**	1	**2**	1
aes-key-recovery	20	19	**19**	18	17	**19**	13	5
atcoss/mesat	15	9	8	**9**	**9**	8	**9**	**9**
bcp-fir	20	20	19	**20**	17	**20**	17	17
bcp-syn	25	13	12	12	12	**13**	12	5
bcp-msp	25	23	17	18	**23**	16	**23**	4
circuitDebuggingProblems	3	3	**3**	**3**	**3**	**3**	2	2
circuitTraceCompaction	4	4	**4**	**4**	2	**4**	2	3
close_solutions	20	20	**19**	**19**	**19**	**19**	14	15
des	20	20	18	18	19	**20**	10	**20**
drmx-atmostk	20	20	**20**	19	7	**20**	7	**20**
drmx-cryptogen	20	20	**20**	**20**	**20**	0	0	0
extension-enforcement	25	15	**12**	**12**	11	7	11	0
fault-diagnosis	25	24	**24**	22	2	21	2	16
frb	15	15	**15**	5	**15**	**15**	7	**15**
gen-hyper-tw	25	5	4	**5**	3	3	**5**	2
hs-timetabling	2	1	**1**	**1**	**1**	**1**	**1**	**1**
jobshop	3	3	**3**	**3**	**3**	**3**	**3**	**3**
kbtree	10	10	0	0	**10**	2	4	0
maxclique/structured	10	7	6	5	6	**7**	**7**	5
maxcut/dimacs_mod	8	3	1	2	**3**	1	2	1
maxcut/spinGlass	2	1	**1**	**1**	**1**	**1**	**1**	**1**
min-fill	15	7	5	4	**7**	2	**7**	4
optic	40	30	19	18	**30**	11	28	2
protein_ins	12	12	**12**	**12**	3	**12**	3	**12**
reversi	20	14	**14**	**14**	**14**	**14**	**14**	**14**
scheduling	5	2	1	**2**	1	1	0	**2**
SeanSafarpour	24	22	**21**	20	15	**21**	10	10
set-covering/crafted/scpclr	2	2	0	0	**2**	0	**2**	0
set-covering/crafted/scpcyc	6	0	0	0	0	0	0	0
tpr/multiple_path	10	10	**10**	**10**	**10**	**10**	**10**	**10**
treewidth-computation	20	16	**16**	**16**	14	14	**16**	**16**
uaq	40	33	**33**	31	26	31	21	26
vpa	40	40	**40**	**40**	**40**	**40**	**40**	37
xai-mindset2	40	27	**23**	21	19	22	18	14
All	600	472	**421**	405	386	382	323	292

Table 11: Ranking of complete solvers on weighted instances including the VBS.

Solver	#Solved	Time, average (s)
VBS	499	141.48
RC2-B	421	256.02
RC2-A	416	267.55
MaxHS	390	274.87
Pacose	390	348.98
QMaxSAT	381	320.78
maxino	373	250.98
Open-WBO-Gluc	371	292.56
Open-WBO-Riss	338	315.47
LMHS	300	345.49

family would require further analysis to understand this behavior since we have solvers that are based on hitting sets (MaxHS) and unsatisfiable cores (RC2-B and maxino) that can solve these benchmarks but at the same time we also have other solvers of the same type (LMHS for hitting sets and Open-WBO-Gluc for unsat-based) that are not able to solve them.

5.1.2 Weighted

Table 11 shows the number of solved instances and average solving time in seconds for complete solvers on weighted instances. RC2-B is the best-performing solver in the weighted track with similar performance to RC2-A.

Figure 2 shows a cactus plots with all weighted solvers. For weighted benchmarks, we can observe that RC2-B and RC2-A have similar performance but there is a gap between these solvers and the remaining. MaxHS, Pacose, QMaxSAT, maxino and Open-WBO-Gluc have similar performance with Open-WBO-Riss and LMHS trailing behind.

In contrast to the unweighted track, we can see that linear sat-unsat solvers perform much better for weighted than unweighted problems. This can be surprising since to solve weighted problems these solvers encode to CNF a pseudo-Boolean constraint that may have thousands of literals and very large coefficients. To cope with this challenge, these solvers often use encodings that are not arc consistent, such as the Adder encoding [35] that adds a linear number of variables and clauses to the formula and prevent the solver from quickly reaching the memory out. Even though Open-WBO-Gluc and Open-WBO-Riss only differ in the underlying SAT solver, the difference in their performance is significant. This shows that the underlying SAT solver can have a large impact on the performance of MaxSAT solvers. Similarly to the unweighed track, LMHS and MaxHS have a large gap between their performances even though they implement similar algorithms. The gap between the VBS and RC2-B is even wider for weighted benchmarks than for unweighted benchmarks which shows the diversity of MaxSAT solving techniques used in the different solvers.

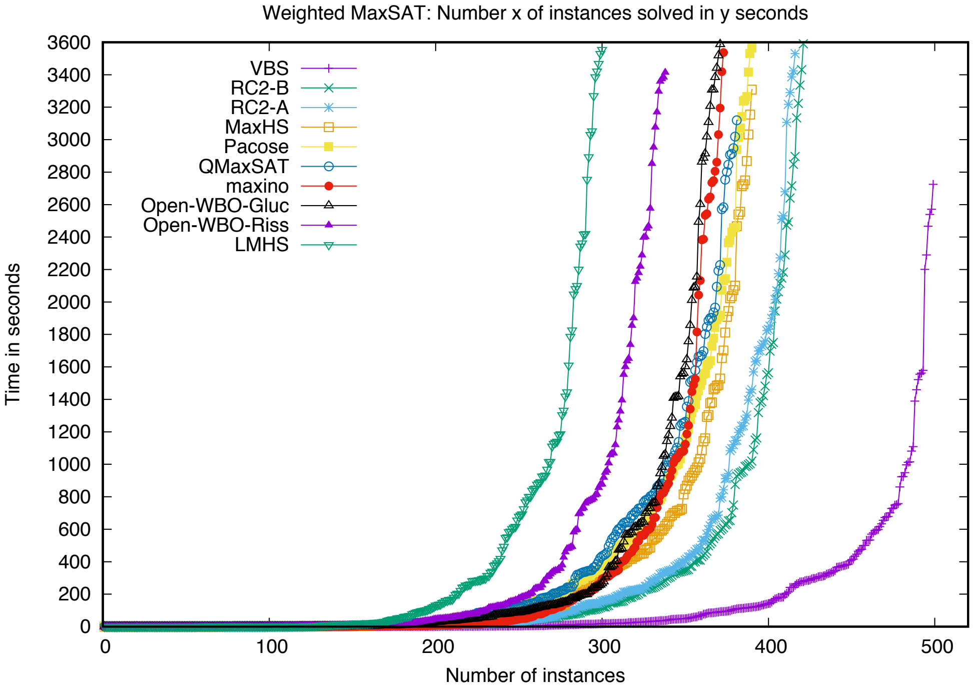


Figure 2: Cactus plot for complete solvers on weighted instances including VBS.

Table 12 shows the number of instances solved by each solver including the VBS for each family of weighted benchmarks. To improve readability of the table, we omit RC2-A and Open-WBO-Riss since they are similar to their counterparts RC2-B and Open-WBO-Gluc. Open-WBO-Gluc is denoted by Open-WBO in Table 12. Some solver types are particularly effective for some families whereas ineffective for others. We highlight those cases in the following discussion.

Analysis of Sat-Unsat approaches. Solver based on sat-unsat approaches (Pacose and QMaxSAT) are particularly effective for the **spot5/log** benchmark family. This family has benchmark characteristics that are beneficial for a sat-unsat approach, namely the optimal solution is close to the upper bound (on average larger than 70% of the sum of the weights of soft clauses are unsatisfied) and the number of soft clauses is small (on average 450 soft clauses per benchmark). In contrast, the performance of the sat-unsat approaches is very poor for benchmark families with a very large number of soft clauses which is the case of **relational-inference** (on average 3,721,201 soft clauses per benchmark).

Complementarity between Hitting Set and Unsat-based approaches. These approaches often complement each other for several families. For example, hitting set approaches perform well for the **BTBNSL** and **correlation-clustering** families, whereas unsat-based approaches perform poorly on these benchmarks. In contrast, unsat-based approaches perform well for the **af-synthesis**, **drmx-atmostk/weighted**, **lisbon-wedding** and **shiftdesign** families, whereas hitting set approaches perform poorly.

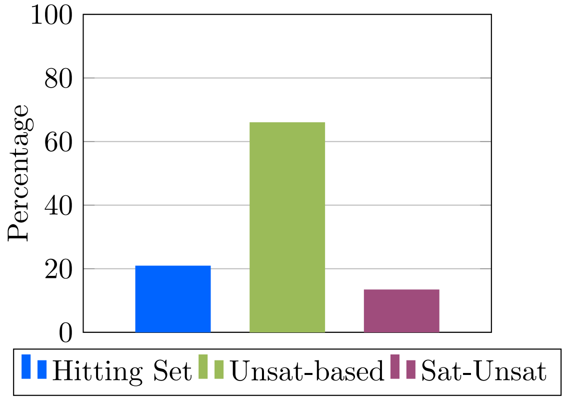
Table 12: Results per family for the complete weighted track of the 2018 evaluation.

Family	#	VBS	RC2-B	MaxHS	Pacose	QMaxSAT	maxino	Open-WBO	LMHS
abstraction-refinement	10	10	**10**	**10**	3	4	**10**	9	6
af-synthesis	25	25	15	5	**25**	21	12	12	1
auctions/auc_paths	15	15	**15**	**15**	**15**	12	**15**	**15**	**15**
BTBNSL	25	18	5	**18**	9	9	6	5	8
causal-discovery	25	21	14	15	**19**	17	18	6	15
cluster-expansion	20	0	0	0	0	0	0	0	0
correlation-clustering	25	21	5	19	2	3	5	3	**21**
CSG	10	10	**10**	**10**	**10**	**10**	**10**	9	**10**
css-refactoring	11	11	**11**	5	10	9	10	10	2
dalculus	25	25	25	**25**	22	20	**25**	**25**	**25**
drmx-atmostk/weighted	20	20	**20**	6	**20**	**20**	19	14	6
drmx-cryptogen/weighted	20	20	**20**	**20**	0	0	0	**20**	0
frb	20	20	**20**	**20**	**20**	**20**	10	6	14
haplotyping-pedigrees	25	25	**25**	22	22	**25**	**25**	23	15
hs-timetabling	13	2	**2**	1	1	1	1	1	**2**
lisbon-wedding	20	9	**8**	0	1	0	7	6	0
max-realizability	43	39	38	**39**	**39**	**39**	37	38	37
maxcut/dimacs_mod	17	9	2	**9**	2	3	1	2	7
maxcut/spinGlass	3	3	1	**3**	1	1	1	1	2
metro	15	15	**15**	**15**	**15**	**15**	**15**	**15**	12
min-width	20	5	3	3	**5**	**5**	4	4	3
miplib	10	5	**5**	**5**	**5**	**5**	**5**	**5**	3
railway-transport	5	2	**2**	1	**2**	**2**	1	1	1
relational-inference	8	8	5	**7**	0	0	**7**	6	6
rna-alignment	25	25	**25**	18	**25**	**25**	18	18	8
shiftdesign	15	15	**15**	8	13	**15**	**15**	**15**	6
spot5/log	25	25	16	11	22	**25**	15	15	7
staff-scheduling	10	2	0	0	1	**2**	1	1	0
tcp	40	39	34	26	38	37	34	**39**	15
timetabling	20	20	**20**	19	**20**	**20**	**20**	**20**	18
upgradeability/wpms	25	25	**25**	**25**	18	11	**25**	**25**	**25**
warehouses	10	10	**10**	**10**	5	5	1	2	**10**
All	600	499	**421**	390	390	381	373	371	300

Mixed behavior. There are also a few families where the behavior of solvers cannot be categorized by using its type. For example, the **drmx-cryptogen/weighted** family has solvers implementing hitting set (MaxHS) and unsat-based (RC2-B and Open-WBO) solving all benchmarks in this family while other hitting set based solvers (LMHS) and unsat-based solvers (maxino) cannot solve any of these benchmarks.

Solver	# Contributions
maxino	144
Open-WBO-Gluc	64
QMaxSAT	63
MaxHS	62
RC2-A	49
RC2-B	41
LMHS	36
Open-WBO-Riss	13
Total	472

(a) Contributions per solver

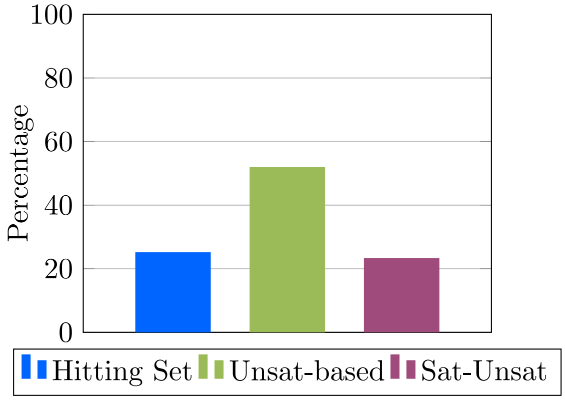


(b) Contributions per solver type

Figure 3: Contribution of solvers to the VBS on unweighted instances.

Solver	# Contributions
maxino	91
MaxHS	83
Open-WBO-Gluc	75
Pacose	67
QMaxSAT	49
LMHS	42
RC2-A	41
Open-WBO-Riss	31
RC2-B	20
Total	472

(a) Contributions per solver



(b) Contributions per solver type

Figure 4: Contributions of solvers to the VBS on weighted instances.

Hard benchmarks. The **cluster-expansion** family had no benchmarks solved by any of the MaxSAT solvers submitted to the 2018 Evaluation.

5.2 Analysis of Virtual Best Solver

Since the gap between the VBS and the best solver for both unweighted and weighted is very large, we perform a deeper analysis of the constitution of the VBS.

Figures 3 and 4 show the contribution of each solver and solver type to the VBS. We group the solvers into the following types:

- Hitting set: MaxHS and LMHS;
- Unsat-based: maxino, Open-WBO-Gluc, Open-WBO-Riss, RC2-A and RC2-B;
- Sat-Unsat: Pacose and QMaxSAT.

Table 13: Ranking of incomplete solver on unweighted instances.

(a) 60-second per-instance time limit

Solver	Score (avg)
SATLike-c	0.735
LinSBPS	0.705
SATLike	0.675
Open-WBO-Inc-OBV	0.654
Open-WBO-Inc-MCS	0.631
Open-WBO-Gluc	0.612
Open-WBO-Riss	0.564
maxroster	0.541

(b) 300-second per-instance time limit

Solver	Score (avg)
SATLike-c	0.854
maxroster	0.829
LinSBPS	0.782
SATLike	0.718
Open-WBO-Inc-OBV	0.713
Open-WBO-Inc-MCS	0.687
Open-WBO-Gluc	0.670
Open-WBO-Riss	0.633

Figures 3b and 4b show that unsat-based solvers have a larger contribution to the VBS, followed by hitting set solvers and sat-unsat solvers. However, these results should be taken with a grain of salt since a majority of the solvers are unsat-based. Figures 3a and 4a break the contribution to the VBS by each solver. The maxino solver makes the largest contribution on both unweighted and weighted benchmarks. Even though Open-WBO-Gluc did not perform as well as other solvers, its contribution to the VBS is significant on both unweighted and weighted benchmarks. On the other hand, RC2-B is the best-performing solver for both unweighted and weighted but its contribution to the VBS is rather small when compared to other solvers. This suggests that RC2-B is a robust solver but it is not the fastest solver per benchmark, possibly due to its Python implementation.

5.3 Incomplete Tracks

5.3.1 Unweighted

Table 13 shows the average score of each incomplete solver on the 153 unweighted benchmarks that were not optimally solved by any complete solver in less than 300 seconds. These benchmarks are challenging to complete solvers and are a good target for incomplete approaches that may not provide any guarantees of optimality. SATLike-c dominates both 60 and 300 seconds time limits with a better average score than the remaining solvers. Note that the best-performing solver of MaxSAT Evaluation 2017 in the unweighted track with 60 seconds was the previous version of Open-WBO-Gluc and for 300 seconds was maxroster. We can observe that all new submissions of incomplete MaxSAT solvers outperform Open-WBO-Gluc. When considering a time limit of 300 seconds, SATLike-c still outperforms maxroster. This shows that there was a significant improvement of incomplete approaches for unweighted MaxSAT. SATLike-c pushes these boundaries by combining a stochastic approach (SATLike) with a complete solver (previous version of Open-WBO-Gluc). This combination outperforms any of the individual solvers and opens new research directions.

Figure 5 shows a cactus plot with the scores in ascending order with 300 seconds time limit. This plot shows the distribution of the benchmark scores. For example, we can observe that there are only around 40 benchmark for which SATLike-c has a score lower than 0.8 and for the remaining 116 benchmarks the score is higher or equal to 0.8.

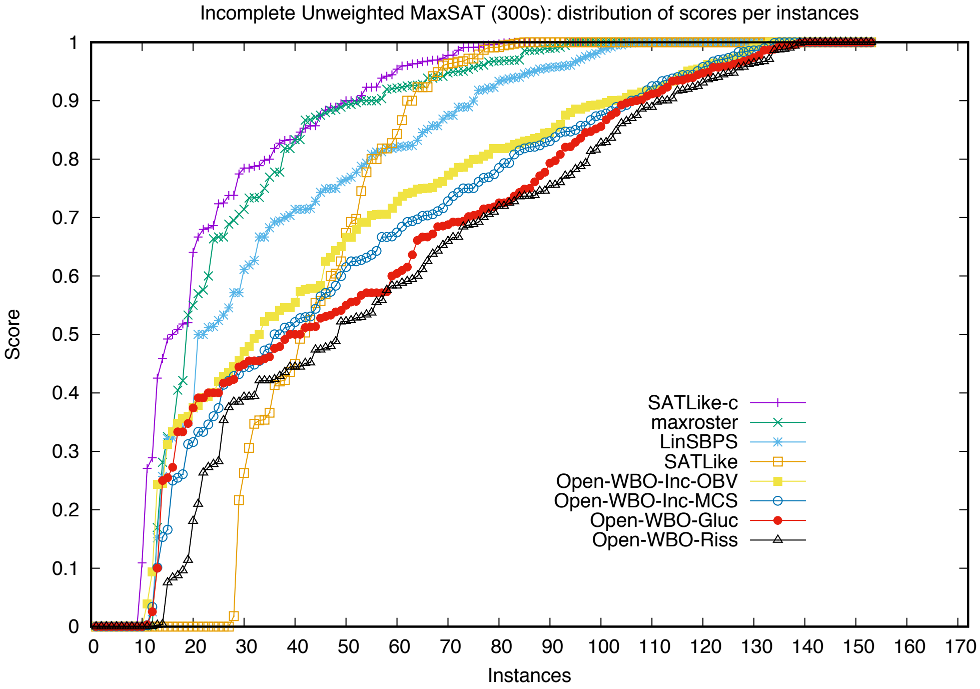


Figure 5: Cactus plot for incomplete solvers on unweighted instances (300 seconds).

Table 14: Ranking of incomplete solver on weighted instances.

(a) 60-second per-instance time limit

Solver	Score (avg)
Open-WBO-Inc-BMO	0.810
LinSBPS	0.799
maxroster	0.773
Open-WBO-Inc-Cluster	0.743
SATLike-c	0.696
Open-WBO-Gluc	0.669
SATLike	0.661
Open-WBO-Riss	0.638

(b) 300-second per-instance time limit

Solver	Score (avg)
LinSBPS	0.900
Open-WBO-Inc-BMO	0.842
maxroster	0.804
Open-WBO-Inc-Cluster	0.762
SATLike-c	0.747
SATLike	0.702
Open-WBO-Gluc	0.68
Open-WBO-Riss	0.663

5.3.2 Weighted

Table 14 shows the average score of each incomplete solver on the 172 weighted benchmarks that were not optimally solved by any complete solver in less than 300 seconds. Open-WBO-Inc-BMO is the best-performing solver for 60 seconds with LinSBPS coming as a close second. For 300 seconds, LinSBPS clearly outperforms the other incomplete MaxSAT solvers. Note that maxroster was the best-performing solver under both 60-second and 300-second time limits in the MaxSAT Evaluation 2017 and is outperformed by both LinSBPS and Open-WBO-Inc-BMO in 2018. This shows the significant improvement of incomplete

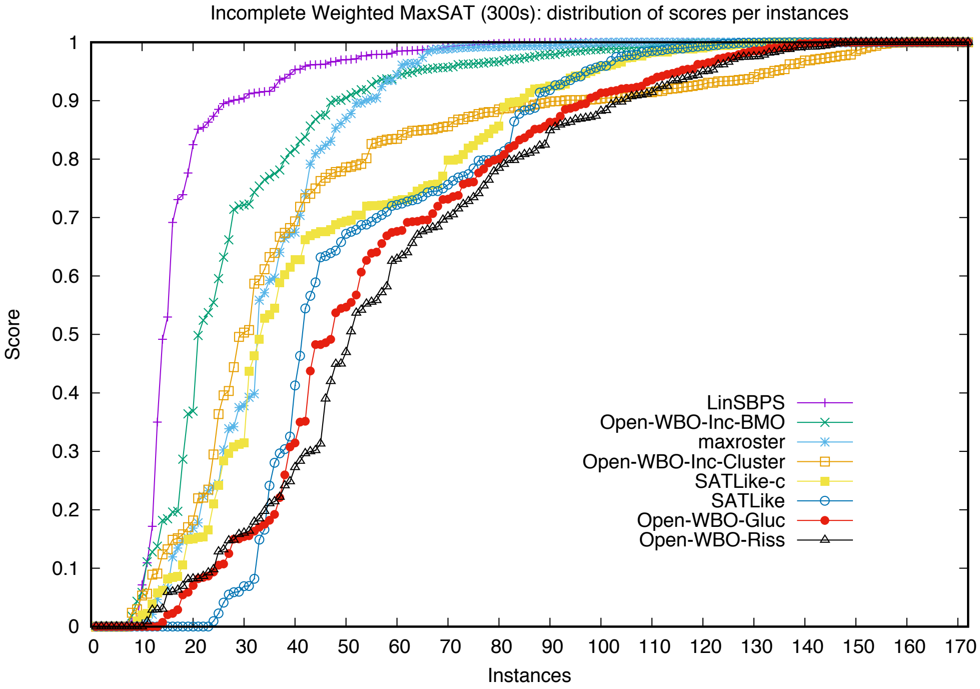


Figure 6: Cactus plot for incomplete solvers on weighted instances (300 seconds).

MaxSAT solvers for weighted benchmarks. Figure 6 shows a cactus plot with ascending scores for each solver and we can observe that for 300 seconds LinSBPS has a high score for the majority of the benchmarks since there are only around 20 out of 172 benchmarks with a score lower than 0.85.

6. Conclusions

With this edition of the evaluation, 2018 marked the 12th year of the series of MaxSAT Evaluations. In this article, we provided an overview of the main organizational details, the participating solvers and submitted benchmarks and the results of the evaluation. The evaluations witnessed concrete changes in 2017 with a new organizing team. This also resulted in some major changes in submission requirements, perhaps most significantly the requirement for open-source solver implementations. Overall, the changes implemented in 2017-2018 have not been reported before this, which further motivates the present article.

As for lessons learned in 2017–2018, we identify some avenues for potential further improvements for forthcoming MaxSAT Evaluations. As with any solver evaluation, the ranking scheme plays a significant role in terms of representing and providing overviews of the evaluation results. One downside of the current scoring mechanism for the incomplete track is that the solvers are only scored at the end of the timeout (60 and 300 sec.) Hence, solvers that generate an equally good solution within these bounds get equal scores even if one solver generated its solution much earlier. The Area Scoring Procedure used in the

MiniZinc Challenges (see https://www.minizinc.org/challenge.html) tries to address this issue and might offer a more accurate score for the incomplete track.

Incrementality has been identified as an important direction of MaxSAT solver development, with various real-world use cases, including data-oriented problem settings where fully grounding a high-level problem representation can result in extremely large MaxSAT instances [23], as well as applications of MaxSAT solvers within counterexample-guided abstraction refinement procedures for optimization problems [9]. While there are some exceptions to the rule, MaxSAT solvers do not generally offer interfaces for incremental use. An incremental track in future MaxSAT Evaluations could provide further incentives for solver developers to extend their solvers towards incrementality.

Finally, over its 12 year existence the evaluations have collected a large number of benchmarks instances, and this collection is growing. Hence, curating these instances and making them more easily available is a task that would be well worth undertaking.

Acknowledgments

We would like to thank everyone who contributed to MaxSAT Evaluation 2018 by submitting solvers or benchmarks: Mario Alviano, Alessandro Armando, Shaowei Cai, Emir Demirović, Rayna Dimitrova, Rüdiger Ehlers, Giorgia Gazzarata, Matthias Heizmann, Wenxuan Huang, Alexey Ignatiev, Saurabh Joshi, Zhendong Lei, Norbert Manthey, Joao Marques-Silva, Tobias Paxian, Paul Saikko, Christian Schilling, Peter J. Stuckey, Takayuki Sugawara, Oleg Zaikin, and Aolong Zha. Furthermore, we thank StarExec [34] and Aaron Stump for enabling running MaxSAT Evaluation 2018 by offering the needed computing resources.

References

[1] Mario Alviano, Carmine Dodaro, and Francesco Ricca. A MaxSAT algorithm using cardinality constraints of bounded size. In Qiang Yang and Michael Wooldridge, editors, *Proc. IJCAI*, pages 2677–2683. AAAI Press, 2015.

[2] Josep Argelich, Chu Min Li, Felip Manyà, and Jordi Planes. The first and second Max-SAT Evaluations. *Journal of Satisfiability, Boolean Modeling and Computation*, **4**(2-4):251–278, 2008.

[3] Josep Argelich, Chu Min Li, Felip Manyà, and Jordi Planes. Analyzing the instances of the MaxSAT evaluation. In Karem A. Sakallah and Laurent Simon, editors, *Proc. SAT*, **6695** of *Lecture Notes in Computer Science*, pages 360–361. Springer, 2011.

[4] Alessandro Armando, Silvio Ranise, Fatih Turkmen, and Bruno Crispo. Efficient runtime solving of RBAC user authorization queries: pushing the envelope. In Elisa Bertino and Ravi S. Sandhu, editors, *Proc. CODASPY*, pages 241–248. ACM, 2012.

[5] Gilles Audemard and Laurent Simon. Predicting learnt clauses quality in modern SAT solvers. In Craig Boutilier, editor, *Proc. IJCAI*, pages 399–404. AAAI Press, 2009.

[6] Fahiem Bacchus, Matti Järvisalo, and Ruben Martins, editors. *MaxSAT Evaluation 2018: Solver and Benchmark Descriptions*, **B-2017-2** of *Department of Computer Science Series of Publications B*. University of Helsinki, 2018.

[7] Daniel Le Berre and Anne Parrain. The sat4j library, release 2.2. *Journal of Satisfiability, Boolean Modeling and Computation*, **7**(2-3):59–6, 2010.

[8] Maria Luisa Bonet, Sam Buss, Alexey Ignatiev, Joao Marques-Silva, and António Morgado. MaxSAT resolution with the dual rail encoding. In Sheila A. McIlraith and Kilian Q. Weinberger, editors, *Proc. AAAI*, pages 6565–6572. AAAI Press, 2018.

[9] Arun Tejasvi Chaganty, Akash Lal, Aditya V. Nori, and Sriram K. Rajamani. Combining relational learning with SMT solvers using CEGAR. In Natasha Sharygina and Helmut Veith, editors, *Proc. CAV*, **8044** of *Lecture Notes in Computer Science*, pages 447–462. Springer, 2013.

[10] Jessica Davies. *Solving MAXSAT by Decoupling Optimization and Satisfaction*. PhD thesis, University of Toronto, 2013. http://www.cs.toronto.edu/~jdavies/Davies_Jessica_E_201311_PhD_thesis.pdf.

[11] Emir Demirovic, Geoffrey Chu, and Peter J. Stuckey. Solution-based phase saving for CP: A value-selection heuristic to simulate local search behavior in complete solvers. In John N. Hooker, editor, *Proc. CP*, **11008** of *Lecture Notes in Computer Science*, pages 99–108. Springer, 2018.

[12] Rayna Dimitrova, Mahsa Ghasemi, and Ufuk Topcu. Maximum realizability for linear temporal logic specifications. In Shuvendu K. Lahiri and Chao Wang, editors, *Proc. ATVA*, **11138** of *Lecture Notes in Computer Science*, pages 458–475. Springer, 2018.

[13] Niklas Eén and Niklas Sörensson. Translating pseudo-boolean constraints into SAT. *Journal of Satisfiablity, Boolean Modeling and Computation*, **2**(1-4):1–26, 2006.

[14] Rüdiger Ehlers and Francisco Palau Romero. Approximately propagation complete and conflict propagating constraint encodings. In Olaf Beyersdorff and Christoph M. Wintersteiger, editors, *Proc. SAT*, **10929** of *Lecture Notes in Computer Science*, pages 19–36. Springer, 2018.

[15] Zhaohui Fu and Sharad Malik. On solving the partial MAX-SAT problem. In Armin Biere and Carla P. Gomes, editors, *Proc. SAT*, **4121** of *Lecture Notes in Computer Science*, pages 252–265. Springer, 2006.

[16] Federico Heras, Javier Larrosa, Simon de Givry, and Thomas Schiex. 2006 and 2007 Max-SAT evaluations: Contributed instances. *Journal of Satisfiability, Boolean Modeling and Computation*, **4**(2-4):239–250, 2008.

[17] Alexey Ignatiev, António Morgado, and Joao Marques-Silva. PySAT: A Python Toolkit for Prototyping with SAT Oracles. In Olaf Beyersdorff and Christoph M. Wintersteiger, editors, *Proc. SAT*, **10929** of *Lecture Notes in Computer Science*, pages 428–437. Springer, 2018.

[18] Alexey Ignatiev, Filipe Pereira, Nina Narodytska, and Joao Marques-Silva. A SAT-Based Approach to Learn Explainable Decision Sets. In Didier Galmiche, Stephan Schulz, and Roberto Sebastiani, editors, *Proc. IJCAR*, **10900** of *Lecture Notes in Computer Science*, pages 627–645. Springer, 2018.

[19] Saurabh Joshi, Prateek Kumar, Ruben Martins, and Sukrut Rao. Approximation strategies for incomplete MaxSAT. In John N. Hooker, editor, *Proc. CP*, **11008** of *Lecture Notes in Computer Science*, pages 219–228. Springer, 2018.

[20] Tuukka Korhonen, Jeremias Berg, Paul Saikko, and Matti Järvisalo. MaxPre: An extended MaxSAT preprocessor. In Serge Gaspers and Toby Walsh, editors, *Proc. SAT*, **10491** of *Lecture Notes in Computer Science*, pages 449–456. Springer, 2017.

[21] Miyuki Koshimura, Tong Zhang, Hiroshi Fujita, and Ryuzo Hasegawa. QMaxSAT: A partial Max-SAT solver. *Journal of Satisfiability, Boolean Modeling and Computation*, **8**(1/2):95–100, 2012.

[22] Zhendong Lei and Shaowei Cai. Solving (Weighted) Partial MaxSAT by Dynamic Local Search for SAT. In Jérôme Lang, editor, *Proc. IJCAI*, pages 1346–1352. ijcai.org, 2018.

[23] Ravi Mangal, Xin Zhang, Aditya Kamath, Aditya V. Nori, and Mayur Naik. Scaling relational inference using proofs and refutations. In Dale Schuurmans and Michael P. Wellman, editors, *Proc. AAAI*, pages 3278–3286. AAAI Press, 2016.

[24] Felip Manyà, Santiago Negrete, Carme Roig, and Joan Ramon Soler. A maxsat-based approach to the team composition problem in a classroom. In Gita Sukthankar and Juan A. Rodríguez-Aguilar, editors, *Proc. AAMAS*, **10643** of *Lecture Notes in Computer Science*, pages 164–173. Springer, 2017.

[25] Ruben Martins, Saurabh Joshi, Vasco Manquinho, and Inês Lynce. Incremental cardinality constraints for maxsat. In Barry O'Sullivan, editor, *Proc. CP*, **8656** of *Lecture Notes in Computer Science*, pages 531–548. Springer, 2014.

[26] Ruben Martins, Vasco Manquinho, and Inês Lynce. Open-WBO: A modular MaxSAT solver. In Carsten Sinz and Uwe Egly, editors, *Proc. SAT*, **8561** of *Lecture Notes in Computer Science*, pages 438–445. Springer, 2014.

[27] António Morgado, Carmine Dodaro, and Joao Marques-Silva. Core-guided maxsat with soft cardinality constraints. In Barry O'Sullivan, editor, *Proc. CP*, **8656** of *Lecture Notes in Computer Science*, pages 564–573. Springer, 2014.

[28] António Morgado, Federico Heras, Mark H. Liffiton, Jordi Planes, and João Marques-Silva. Iterative and core-guided MaxSAT solving: A survey and assessment. *Constraints*, **18**(4):478–534, 2013.

[29] Alexander Nadel. Solving MaxSAT with bit-vector optimization. In Olaf Beyersdorff and Christoph M. Wintersteiger, editors, *Proc. SAT*, **10929** of *Lecture Notes in Computer Science*, pages 54–72. Springer, 2018.

[30] Miguel Neves, Ruben Martins, Mikolás Janota, Inês Lynce, and Vasco M. Manquinho. Exploiting Resolution-Based Representations for MaxSAT Solving. In Marijn Heule and Sean Weaver, editors, *Proc. SAT*, **9340** of *Lecture Notes in Computer Science*, pages 272–286. Springer, 2015.

[31] Tobias Paxian, Sven Reimer, and Bernd Becker. Dynamic polynomial watchdog encoding for solving weighted MaxSAT. In Olaf Beyersdorff and Christoph M. Wintersteiger, editors, *Proc. SAT*, **10929** of *Lecture Notes in Computer Science*, pages 37–53. Springer, 2018.

[32] Paul Saikko. Re-implementing and extending a hybrid SAT-IP approach to maximum satisfiability. Master's thesis, University of Helsinki, 2015. http://hdl.handle.net/10138/159186.

[33] Paul Saikko, Jeremias Berg, and Matti Järvisalo. LMHS: A SAT-IP hybrid MaxSAT solver. In Nadia Creignou and Daniel Le Berre, editors, *Proc. SAT*, **9710** of *Lecture Notes in Computer Science*, pages 539–546. Springer, 2016.

[34] Aaron Stump, Geoff Sutcliffe, and Cesare Tinelli. Starexec: A cross-community infrastructure for logic solving. In Stéphane Demri, Deepak Kapur, and Christoph Weidenbach, editors, *Proc. IJCAR*, **8562** of *Lecture Notes in Computer Science*, pages 367–373. Springer, 2014.

[35] Joost P. Warners. A Linear-Time Transformation of Linear Inequalities into Conjunctive Normal Form. *Information Processing Letters*, **68**(2):63–69, 1998.