Journal on Satisfiability, Boolean Modeling and Computation 4 (2008) 251-278

The First and Second Max-SAT Evaluations

Josep Argelich josep@sat.inesc-id.pt
INESC-ID Lisboa
Rua Alves Redol 9, 1000-029 Lisboa, Portugal

Chu-Min Li chu-min.li@u-picardie.fr
Université de Picardie
33 Rue St. Leu, 80039 Amiens Cedex 1, France

Felip Manyà felip@iiia.csic.es
Artificial Intelligence Research Institute (IIIA, CSIC)
Campus UAB, 08193 Bellaterra, Spain

Jordi Planes jp3@ecs.soton.ac.uk
University of Southampton
Highfield, Southampton SO17 1BJ, United Kingdom

Abstract

We describe the organization and report on the results of the First and Second Max-SAT Evaluations, which were organized as affiliated events of the 2006 and 2007 editions of the International Conference on Theory and Applications of Satisfiability Testing (SAT-2006 and SAT-2007), discuss the insights gained and point out new directions for forthcoming evaluations. The main objectives of both evaluations were assessing the advancements in the field of Max-SAT solvers through a comparison of their performances, identifying successful solving techniques and encouraging researchers to develop new ones, and creating a publicly available collection of challenging Max-SAT benchmarks.

KEYWORDS: *Max-SAT, Weighted Max-SAT, Partial Max-SAT, Weighted Partial Max-SAT, empirical evaluation, solvers, benchmarks*

Submitted October 2007; revised March 2008; published September 2008

1. Introduction

Satisfiability testing is a well-established research field that focus on the propositional satisfiability problem of Boolean CNF formulas (SAT), as well as in related problems such as Pseudo-Boolean Solving and Optimization, Satisfiability of Quantified Boolean formulas, Satisfiability of Many-Valued formulas, and Max-SAT.

Taking into account the growing interest of the SAT community in developing fast exact Max-SAT solvers, the lack of a good collection of benchmarks, and the good experiences gathered from the SAT competitions [8], we decided to organize the First Max-SAT Evaluation [3] as an affiliated event of the SAT-2006 conference with the following objectives: assess the advancements in the field of Max-SAT solvers through a comparison of their performances, identify successful solving techniques and encourage researchers to develop new ones, and create a publicly available collection of challenging Max-SAT benchmarks. In the 2006 Max-SAT Evaluation participated 6 solvers and there were two categories: Unweighted

Max-SAT and Weighted Max-SAT. Due to the success of that event, we organized the 2007 Max-SAT Evaluation [4] as an affiliated event of the SAT-2007 conference, which counted with the participation of 12 solvers and two additional categories: Partial Max-SAT and Weighted Partial Max-SAT.

We concentrated on exact solvers because, for the time being, there is a big gap between the size and type of instances that can be solved with exact solvers and the instances that can be solved with approximation and heuristic solvers. On the other hand, there is no guarantee that the solutions computed by the latter solvers are optimal.

In our opinion, the first Max-SAT evaluations have provided a quite accurate snapshot of the current state-of-the-art of exact Max-SAT solvers, have contributed to increase the interest and activity of the research community on Max-SAT, have allowed to identify a number of good performing solving techniques, and have promoted the creation of a publicly available collection of challenging Max-SAT benchmarks.

Most of the benchmarks used in the first two evaluations were contributed by F. Heras, J. Larrosa, S. de Givry and T. Schiex, who have written a paper for this special issue containing a description of the encodings used for solving problems such as max-cut, max-clique and combinatorial auctions, as well as a description of how they have translated benchmarks from the Pseudo-Boolean and CSP communities into Max-SAT. In the sequel, we omit details about benchmarks and refer the reader to [14]. Besides, the organizers added random Max-2-SAT and Max-3-SAT instances, changing the clause–to–variable ratio, for each category in order to test more instances. In future evaluations, it is not planned to add instances by the organizers because the number of submitted instances has grown significantly.

The present paper is structured as follows: In Section 2 we recall some basic definitions and define the problems solved in the evaluations (Max-SAT, Weighted Max-SAT, Partial Max-SAT, and Weighted Partial Max-SAT). In Section 3 we describe the input and output formats that the solvers should, respectively, read and write. In Section 4 we describe the main features of all the solvers that participated in the evaluations. In Section 5 we report on and discuss the experimental results of both evaluations. In Section 6 we present the conclusions and point out new directions for forthcoming evaluations.

2. Preliminaries

In propositional logic, a variable x_i may take values 0 (for false) or 1 (for true). A literal l_i is a variable x_i or its negation $\bar{x}_i$. A clause is a disjunction of literals, and a CNF formula is a multiset of clauses. A weighted clause is a pair (C_i, w_i), where C_i is a disjunction of literals and w_i, its weight, is a positive number, and a weighted CNF formula is a multiset of weighted clauses. Note that we define CNF formulas as multisets of clauses because, in contrast to SAT, duplicated clauses cannot be collapsed into one clause in Max-SAT. For instance, the multiset $\{a, \neg a, \neg a, a \vee b, \neg b\}$, where a clause is repeated, has a minimum of two unsatisfied clauses. In Weighted Max-SAT, two clauses of the form $(C, w_i), (C, w_j)$ can be replaced with $(C, w_i + w_j)$.

An assignment of truth values to the propositional variables satisfies a literal x_i if x_i takes the value 1 and satisfies a literal $\bar{x}_i$ if x_i takes the value 0, satisfies a clause if it satisfies at least one literal of the clause, and satisfies a CNF formula if it satisfies all the clauses

of the formula. Given a CNF formula ϕ, an assignment is said to be complete with respect to ϕ if every variable in ϕ is assigned a truth value, otherwise the assignment is said to be partial.

The *(Unweighted) Max-SAT problem* for a CNF formula ϕ is the problem of finding an assignment of values to propositional variables that maximizes the number of satisfied clauses. Max-SAT is called Max-k-SAT when all the clauses have k literals per clause. In the sequel we often use the term of Max-SAT meaning Min-UNSAT. This is because, with respect to exact computations, finding an assignment that minimizes the number of unsatisfied clauses is equivalent to finding an assignment that maximizes the number of satisfied clauses.

Max-SAT instances ϕ_1 and ϕ_2 are *equivalent* if ϕ_1 and ϕ_2 have the same number of unsatisfied clauses for every complete assignment of ϕ_1 and ϕ_2.

We also consider three extensions of (Unweighted) Max-SAT which are more well-suited for representing and solving over-constrained problems: Weighted Max-SAT, Partial Max-SAT, and Weighted Partial Max-SAT.

The *Weighted Max-SAT problem* for a weighted CNF formula ϕ is the problem of finding an assignment of values to propositional variables that minimizes the sum of weights of unsatisfied clauses (or equivalently, that maximizes the sum of weights of satisfied clauses).

A *Partial Max-SAT* instance is a CNF formula in which some clauses are *relaxable* or *soft* and the rest are *non-relaxable* or *hard*. Solving a Partial Max-SAT instance amounts to find an assignment that satisfies all the hard clauses and the maximum number of soft clauses.

The *Weighted Partial Max-SAT* problem is the combination of Weighted Max-SAT and Partial Max-SAT: Every soft clause in a Weighted Partial Max-SAT instance has a weight, and solving this instance amounts to find an assignment that satisfies all the hard clauses and minimizes the sum of weights of unsatisfied soft clauses.

Notice that Max-SAT can also be defined as Weighted Max-SAT restricted to formulas whose clauses have weight 1, and as Partial Max-SAT in the case that all the clauses are declared to be soft.

3. Input and Output Formats

We define the input and output file formats that solvers should, respectively, read and write. The input formats of each category are adaptations of the DIMACS format, and the output format, which is common for all the categories, is inspired by the output specification of the 2006 Pseudo Boolean Evaluation [24] and the SAT Competitions [8].

3.1 Input Format

The input file format for Unweighted Max-SAT instances is the standard DIMACS format. The file may start with comments; i.e., lines beginning with the character "c". Right after the comments, there is the parameters line "p cnf nbvar nbclauses", where cnf indicates that the instance is in CNF format; nbvar is the number of variables appearing in the file, and nbclauses is the exact number of clauses contained in the file. Then, the clauses follow. Each clause is a sequence of distinct non-null numbers between $-$nbvar and nbvar ending with a 0 on the same line; 0 is a terminator of clauses. Positive numbers denote positive

literals of the corresponding variables, and negative numbers denote negative literals of the corresponding variables.

The following example corresponds to an Unweighted Max-SAT instance, the left hand side corresponds to the instance in the common notation, and the right hand side to the same instance in DIMACS format.

$x_1 \vee \bar{x}_2$
$\bar{x}_1 \vee x_2 \vee \bar{x}_3$
$\bar{x}_3 \vee x_2$
$x_1 \vee x_3$

```
c
c Unweighted Max-SAT instance
c
p cnf 3 4
1 -2 0
-1 2 -3 0
-3 2 0
1 3 0
```

In Weighted Max-SAT, the parameters line is "p wcnf nbvar nbclauses". The weights of each clause will be identified by the first integer in each clause line. The weight of each clause is an integer greater than or equal to 1, and smaller than 2^{20}. Big integers have not been considered so far in Weighted Max-SAT, but they are necessary to solve certain types of real-world problems. This is an open question that has to be discussed in the near future with the participants of the forthcoming evaluations.

The following example corresponds to a Weighted Max-SAT instance, the left hand side corresponds to the instance in the common notation, and the right hand side to the same instance in the evaluation file format.

$(x_1 \vee \bar{x}_2,\ 10)$
$(\bar{x}_1 \vee x_2 \vee \bar{x}_3,\ 3)$
$(\bar{x}_3 \vee x_2,\ 8)$
$(x_1 \vee x_3,\ 5)$

```
c
c Weighted Max-SAT instance
c
p wcnf 3 4
10  1 -2 0
 3 -1 2 -3 0
 8 -3 2 0
 5  1 3 0
```

In Weigthed Partial Max-SAT, the parameters line is "p wcnf nbvar nbclauses top", where top is an integer that should be greater than any solution of the input instance. We associate a weight with each clause, which is the first integer in the clause. Weights must be greater than or equal to 1, and smaller than 2^{20}. Hard clauses have weight top and soft clauses have a weight smaller than top. The weight of top is an upper bound. Initially, top, which was of common use in the Weighted CSP (WCSP) community but not in the Max-SAT community, was introduced to facilitate the participation of WCSP solvers in the evaluation. It may not be useful for certain Partial Max-SAT solvers; in this case, they just have to treat the clauses having a top weight as hard clauses. However, nowadays, there exist Partial Max-SAT solvers that use the top to transform soft clauses into hard clauses [18].

The following example corresponds to a Weighted Partial Max-SAT instance, the left hand side corresponds to the instance in the common notation, and the right hand side to

the same instance in the evaluation file format. Note that the hard clauses are represented between square brackets.

	`c`
	`c Weighted Partial Max-SAT instance`
$[x_1 \vee \bar{x}_2 \vee x_4]$	`c`
$[\bar{x}_1 \vee \bar{x}_2 \vee x_3]$	`p wcnf 4 5 16`
$(\bar{x}_2 \vee \bar{x}_4,\ 8)$	`16  1 -2 4 0`
$(\bar{x}_3 \vee x_2,\ 4)$	`16 -1 -2 3 0`
$(x_1 \vee x_3,\ 3)$	` 8 -2 -4 0`
	` 4 -3 2 0`
	` 3  1 3 0`

Partial Max-SAT instances are represented as Weighted Partial Max-SAT instances in which soft clauses have weight 1.

Finally, we would like to emphasize that we have described the current input formats, but we believe that these formats will be probably modified in the future.

3.2 Output Format

Solvers must output messages on the standard output that are used to check the results. Solvers cannot write to any files other than standard output and standard error (only standard output will be parsed for results, but both output and error will be memorized during the whole evaluation process, for all executions). The messages of the output format are:

- Comments ("c " lines): These lines start by the lower case 'c' followed by a space. They are optional and may appear anywhere in the solver output, and contain any information that authors want to register.
- Current optimal solution ("o " lines): These lines, which are mandatory, start by lower case "o" followed by a space and then by an integer which represents the lowest number of clauses falsified so far by an assignment. The evaluation environment will take as optimal number of unsatisfied clauses the last "o " line in the output stream.
- Solution ("s " line): This line, which is mandatory, starts by lower case 's' followed by a space. It gives the answer of the solver, which must be one of the following answers:
  - OPTIMUM FOUND. This line must be output when the solver has traversed all the search space and checked that the last "o " line is the optimal solution.
  - UNKNOWN. This line must be output in any other case; i.e., when the solver is not able to give the optimal solution for any reason.

  If the solver does not display a solution line (or if the solution line is not valid), then UNKNOWN will be assumed.
- Values ("v " lines): These lines start by lower case 'v' followed by a space. If the solver finds an optimal solution (it outputs "s OPTIMUM FOUND"), it must provide the

minimum number of unsatisfied clauses as well as an optimal truth assignment that is used to check the correctness of the answer. The truth assignment is represented by a list of literals, where each literal containts a distinct variable, and the value that it assigns to the variable of a literal is 1 if the literal is positive and is 0 if the literal is negative. If the solver does not output a value line, or if the value line is misspelled, then UNKNOWN will be assumed.

An example of output format is the following file:

```
c --------------------------
c My Weighted Max-SAT Solver
c --------------------------
o 481
o 245
o 146
o 145
o 144
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
```

A solver is considered buggy in the following cases:

- It outputs OPTIMUM FOUND but provides an assignment that falsifies a number of clauses different from the number of clauses in the last "o " line (or the sum of weights of unsatisfied clauses in the case of Weighted Max-SAT).
- It outputs OPTIMUM FOUND but the obtained assignment is not optimal.

As for the input formats, we would like to emphasize that we have described the current output formats, but we believe that in future evaluations new rules have to be defined to validate the optimal solutions provided by the solvers. We decide the *correct* optimal solution by consensus. In the definitive experiments of both evaluations, we did not detect two solvers that answered OPTIMUM FOUND and provided a different minimum number of unsatisfied clauses.

4. Solvers

We now give a description of the main features of the solvers that participated in the evaluations. Such descriptions were provided by their authors. For each solver, we also give the existing publications in which the reader can find more detailed descriptions of the solvers and their solving techniques.

4.1 Solvers of the 2006 Max-SAT Evaluation

ChaffBS and ChaffLS (Zhaohui Fu and Sharad Malik): Both ChaffBS and ChaffLS are implemented on top of the SAT solver zChaff [25]. In order to translate a Max-SAT

instance into a SAT one, it appends a distinct selector variable to every Max-SAT clause. A true selector variable essentially means that the corresponding Max-SAT clause can be left unsatisfied. It then constructs a hierarchical tree adder using three-at-a-time adders (i.e., full adders). The hierarchical tree adder sums up the number of true selector variables and presents the summation in binary format to a logic comparator, which returns true if and only if the binary number is less than or equal to any given number k. At this point, the Max-SAT instance can be translated into a SAT instance, which consists of the Max-SAT clauses with selector variables and the SAT clauses correspond to the hierarchical tree adder and the logic comparator for a given k value. Obviously, k is greater than or equal to 0 and less than or equal to the total number of selector variables. In order to find the minimum k, i.e., the minimum number of true selector variables, one can either do Binary Search (ChaffBS) or Linear Search (ChaffLS) within the possible range of k. For ChaffLS, it starts with $k = 0$ and increase k by one until it finds the translated SAT instance satisfiable. As a side effect, ChaffLS is not able to produce any sub-optimal solution as the first solution it finds is the optimal solution. For further details see [13].

Lazy (Teresa Alsinet, Felip Manyà and Jordi Planes): It is a branch and bound solver for both Unweighted Max-SAT and Weighted Max-SAT that uses very simple lazy data structures and a static variable selection heuristic. It applies, as preprocessing, the almost common clause rule: $x \vee D$ and $\neg x \vee D$ is replaced with D. The initial upper bound is computed with a local search algorithm. Lazy applies the complementary unit clause rule at each node of the proof tree, and applies unit propagation whenever the difference between the lower bound and the upper bound is one. It implements the star rule as lower bound computation method: The lower bound is incremented by one for every detected disjoint subset either of the form $x, \neg x$ or of the form $x, y, \neg x \vee \neg y$. For further details see [1, 2].

Maxsatz (Chu Min Li, Felip Manyà and Jordi Planes): It is a branch and bound solver for Unweighted Max-SAT that incorporates into a Max-SAT solver some of the technology developed for Satz [19]. At each node of the proof tree, it transforms the formula into an equivalent formula that preserves the number of unsatisfied clauses by applying some efficient refinements of unit resolution that the authors have defined for Max-SAT (e.g., it replaces $x, y, \neg x \vee \neg y$ with $\square, x \vee y$, it replaces $x, \neg x \vee y, \neg x \vee z, \neg y \vee \neg z$ with $\square, \neg x \vee y \vee z, x \vee \neg y \vee \neg z$). MaxSatz implements a lower bound computation method that increments the lower bound by one for every disjoint inconsistent subset that can be detected by applying unit propagation, or unit propagation enhanced with failed literal detection. The variable selection heuristics takes into account the number of positive and negative occurrences in binary and ternary clauses. For further details see [20, 21, 22].

SAT4Jmaxsat (Daniel Le Berre): SAT4Jmaxsat translates a Max-SAT instance $S = \{C_1, C_2, \ldots, C_m\}$ with n variables into the following optimization problem: For each clause C_i in the original problem, a new variable V_i is created and added. Some people call those variables selector variables because satisfying such a variable disables a clause. So solving Max-SAT on the original problem is equivalent to solve the optimization problem: Minimize the number of V_i's satisfied in $S' = \{C_1 \vee V_1, C_2 \vee V_2, \ldots, C_m \vee V_m\}$. Since SAT4Jmaxsat supports cardinality constraints, it simply asks a SAT solver to solve S', and each time a model M is found, it tries to find a better one, by adding a cardinality constraints $SUM(V_i) < number\_of\_V_i\_satisfied\_in\_M$. Once S' and all the cardinality

constraints are inconsistent, the latest model is the optimal solution. For further details see [7].

ToolBar (Simon de Givry, Federico Heras, Javier Larrosa, and Thomas Schiex): A DPLL-like algorithm is used to find a better solutions or proving optimality. After each assignment the current subproblem is transformed to an equivalent (and simpler) one. The transformations are based on the resolution rule for Max-SAT [17]. Note that these transformations can be explained as different levels of local consistency for WCSP. It is easy to see that a Max-SAT instance can be represented as a WCSP problem where all variables have two values (Boolean variables) and forbidden tuples represent weighted clauses. Examples of such transformations are (and its related WCSP local consistencies): - clauses $(x \vee y, 2), (\neg x \vee y, 1)$ are replaced by $(x \vee y, 1), (y, 1)$ (This is detected by AC* in WCSP). - clauses $(x, 1), (\neg x \vee y, 2), (\neg y \vee z, 1), (\neg z, 1)$ are replaced by $(\square, 1), (\neg x \vee y, 1), (x \vee \neg y, 1), (y \vee \neg z, 1)$ where $(\square, 1)$ represents an increment of the lower bound (this is detected by EDAC* in WCSP [11]). For further details see [18].

4.2 Solvers of the 2007 Max-SAT Evaluation

The solvers from the 2006 Max-SAT Evaluation that also participated in the 2007 Max-SAT Evaluation are: ChaffBS & ChaffLS , MaxSatz, SAT4Jmaxsat and ToolBar. We show below the description of all the new solvers and only the description of the solvers from the 2006 edition that have been changed by their authors in the 2007 edition.

Clone (Knot Pipatsrisawat, Mark Chavira, Arthur Choi, and Adnan Darwiche): Clone is an exact Max-SAT solver that uses branch-and-bound search to find optimal solutions. The method for computing bounds used by Clone is rather different from those of contemporary Max-SAT solvers. Clone relaxes some constraints in the original CNF and turns it into an approximate formula, which is then compiled into a d-DNNF (Deterministic Decomposable Negation Normal Form). The approximate formula's Max-SAT solution, which can be computed very efficiently, can be used as a bound on the solution of the original problem. Once every variable involved in the relaxation is assigned a value, the solution of the conditioned approximate formula is no longer a bound -it becomes exact. Thus, Clone only needs to perform branch-and-bound search on the search space of those variables involved in the relaxation of constraints, resulting in a smaller search space. For further details see [26, 27].

LB-SAT and LB-PSAT (Han Lin, and Kaile Su): LB-SAT is a two-stage solver for MAX-SAT. At the first stage, it invokes a local search procedure to calculate an approximate optimal solution. At the second stage, taking the approximate value as an initial upper bound, a branch and bound routine is called to find the exact solution. At each search node, like UP [20] and Maxsatz [22], LB-SAT exploits unit propagation to compute a lower bound. The lower bound is computed in an incremental style, i.e., at each node, instead of computing the lower bound from scratch, LB-SAT reuses the information from the previous search nodes to boost the computation and improve the lower bound. Other techniques incorporated into LB-SAT can be found in [23]. LB-PSAT is LB-SAT for Weighted and Partial Max-SAT.

MaxSatz14 (Sylvain Darras, Gilles Dequen, Laure Devendeville, and Chu Min Li): This solver is based on the last release of Maxsatz [22], built and improved by Chu

Min Li, Felip Manyà and Jordi Planes. The main contribution has been to speed up the two look-ahead functions by selecting and storing useful conflictual subformulas in order to avoid their recomputation at each node of the search tree. For further details see [9].

MiniMaxSat (Federico Heras, Javier Larrosa, Albert Oliveras, and Simon de Givry): MiniMaxSat incorporates the best current SAT and Max-SAT techniques. It can handle hard clauses (clauses of mandatory satisfaction as in SAT), soft clauses (clauses whose falsification is penalized by a cost as in Max-SAT) as well as pseudo-boolean objective functions and constraints. Its main features are: learning and backjumping on hard clauses; resolution-based and subtraction-based lower bounding; and lazy propagation with the two-watched literals scheme. For further details see [15, 16].

PMS (Josep Argelich, and Felip Manyà): PMS is a branch and bound solver which incorporates efficient data structures, a dynamic variable selection heuristic, inference rules and a good quality lower bound based on unit propagation. PMS exploits the fact that some clauses are hard to increase the efficiency of its heuristics and its techniques. PMS also incorporates a clause learning schema for hard clauses; this learning is similar to the learning incorporated into modern SAT solvers. For further details see [6].

SR(w) (Miquel Ramírez, and Héctor Geffner): SR(w) is a MinCostSAT solver which uses explicit structural relaxation to derive lower bounding functions that allow a Branch & Bound DLL-style search procedure to potentially prune vast tracts of the search space. SR(w) is built on top of two off-the-shelf tools: the d-SDNNF compiler by Darwiche [10] and the state-of-the-art SAT solver MiniSAT 2.0 [12]. For further details see [28].

W-MaxSatz (Josep Argelich, Chu Min Li, and Felip Manyà): W-MaxSatz is the weighted version of MaxSatz. It is a branch and bound Weighted Max-SAT solver that incorporates all the features of MaxSatz adapted to deal with weights. This implies the modification of the data structures to dynamically add and remove clauses without a significant overhead in CPU time. W-MaxSatz implements a dynamic variable selection heuristic, advanced inference rules, and a lower bound computation method based on unit propagation and failed literals detection.

5. Empirical Evaluation

The empirical evaluation of both the 2006 Max-SAT Evaluation and the 2007 Max-SAT Evaluation started with the submission of benchmarks and solvers. After a few days of the submission, a representative sample of benchmark instances were provided to the authors of solvers with the aim of allowing them to correct possible bugs and tune their solvers. After that, authors had the opportunity of submitting a new version of their solver. Only one version per solver was allowed. In parallel, the organizers programmed the scripts needed to perform the evaluation.

The evaluations were conducted on a cluster with machines with the following specification:

- Operating System: Rocks Cluster 4.0.0 Linux 2.6.9
- Processor: AMD Opteron 248 Processor, 2 GHz
- Memory: 1 GB

Table 1. Results in the Unweighted Max-SAT category of the 2006 Max-SAT Evaluation. Mean time in seconds.

Set Name	#Ins.	MaxSatz	ToolBar	Lazy	ChaffBS	ChaffLS	SAT4J maxsat
Max-Cut (brock)	12	**13.35(12)**	57.50(12)	178.48(12)	— (0)	— (0)	— (0)
Max-Cut (c-fat)	7	**0.07(5)**	21.05(5)	151.13(5)	0.01(2)	0.01(2)	0.85(2)
Max-Cut (hamming)	6	**180.12(3)**	575.52(3)	42.06(2)	— (0)	— (0)	— (0)
Max-Cut (johnson)	4	**45.39(3)**	134.68(3)	2.45(2)	— (0)	— (0)	— (0)
Max-Cut (keller)	2	**6.12(2)**	17.25(2)	69.86(2)	— (0)	— (0)	— (0)
Max-Cut (p_hat)	12	**15.84(12)**	61.86(12)	192.05(12)	— (0)	— (0)	— (0)
Max-Cut (san)	11	**275.05(11)**	65.02(7)	249.83(7)	— (0)	— (0)	— (0)
Max-Cut (sanr)	4	**71.98(4)**	266.86(4)	80.78(3)	— (0)	— (0)	— (0)
Max-Cut (random)	40	**5.58(40)**	34.67(40)	752.34(25)	— (0)	— (0)	— (0)
Max-Cut (spinglass)	5	**44.92(3)**	4.96(2)	48.21(2)	9.97(1)	6.19(1)	— (0)
Max-One	45	**0.02(45)**	5.44(45)	81.34(40)	1.00(45)	0.20(45)	2.31(41)
Ramsey	48	**8.99(34)**	53.14(33)	81.70(28)	53.39(34)	7.36(33)	2.86(32)
Max-2-SAT (60 vars)	50	**0.03(50)**	0.62(50)	3.27(50)	13.74(10)	25.69(10)	— (0)
Max-2-SAT (100 vars)	50	**1.40(50)**	17.57(50)	235.83(31)	0.70(10)	1.08(10)	24.37(10)
Max-2-SAT (140 vars)	50	**7.02(50)**	105.61(49)	204.10(23)	272.77(12)	99.86(11)	47.26(11)
Max-2-SAT(discarded)	180	**16.79(180)**	99.34(175)	141.39(107)	262.04(18)	172.67(14)	59.87(4)
Max-3-SAT (40 vars)	50	**1.50(50)**	8.09(50)	6.94(50)	0.31(10)	0.28(10)	50.05(11)
Max-3-SAT (60 vars)	50	**23.31(50)**	264.98(50)	266.70(43)	84.76(11)	68.55(11)	1.96(10)
Solved instances		**604 (626)**	592 (626)	444 (626)	153 (626)	147 (626)	121 (626)

- Cache: 1024 KB
- Compilers: GCC 3.4.3, javac JDK 1.5.0

The time limit was set to 1800 seconds for each instance and solver. No memory limit was imposed to solvers and no swap problem was detected. Assessment of solvers was based on the number of successfully solved instances and the time needed to solve them.

5.1 Empirical Results in the 2006 Max-SAT Evaluation

Table 1 and Table 2 show the experimental results of the two categories of the 2006 Max-SAT Evaluation. The instances were grouped into sets. The instances in each set were decided by the authors of the instances. There were 18 sets of instances in the Unweighted Max-SAT category and 31 sets of instances in the Weighted Max-SAT category. We display the number of instances in each set (#Ins.) and then, for each solver and for each set of instances, the mean time needed to solve an instance and the number of solved instances (in brackets) using a cutoff of 1800 seconds. The last line of each table displays the total number of instances that were solved by each solver. The best results are displayed in bold. There are instances (e.g. instances in spinglass) that were not solved by any solver with the current cutoff.

Figure 1 and Figure 2 globally compare the performance of the solvers in the two categories of the 2006 Max-SAT Evaluation on the instances in Table 1 and Table 2, respectively. Each point (x, y) of a curve shows the number x of instances that the corresponding solver is able to solve within y seconds. In other words, each of the x instances is solved within y seconds (the total run time for these x instances may be larger than y seconds), y being limited to 1800 seconds. In the Unweighted Max-SAT category, MaxSatz and ToolBar are the best performing solvers, followed by Lazy. In the Unweighted Max-SAT category, ToolBar is the best performing solvers, followed by Lazy.

Table 2. Results in the Weighted Max-SAT category of the 2006 Max-SAT Evaluation. Mean time in seconds.

Set Name	#Ins.	ToolBar	Lazy	SAT4Jmaxsat
Auction (paths)	30	**249.77(26)**	81.24(20)	— (0)
Auction (regions)	30	**8.16(30)**	2.03(28)	926.99(6)
Auction (scheduling)	30	132.15(30)	**63.33(30)**	518.41(8)
Max-Clique (brock)	12	**96.76(4)**	104.69(4)	— (0)
Max-Clique (c-fat)	7	25.19(7)	**17.36(7)**	346.68(4)
Max-Clique (hamming)	6	**134.04(5)**	195.05(5)	6.32(2)
Max-Clique (johnson)	4	53.91(3)	**38.64(3)**	61.73(2)
Max-Clique (keller)	2	**34.12(1)**	43.38(1)	— (0)
Max-Clique (mann_a)	4	**45.62(3)**	0.31(1)	726.50(2)
Max-Clique (p_hat)	12	**325.70(8)**	254.14(6)	— (0)
Max-Clique (san)	11	**25.01(3)**	10.88(1)	— (0)
Max-Clique (sanr)	4	**821.98(3)**	790.55(2)	— (0)
Weighted Max-Cut (brock)	12	**12.37(12)**	18.01(12)	— (0)
Weighted Max-Cut (c-fat)	7	**7.80(7)**	25.99(7)	1.07(2)
Weighted Max-Cut (hamming)	6	105.22(4)	**88.93(4)**	— (0)
Weighted Max-Cut (johnson)	4	**71.39(3)**	74.29(3)	— (0)
Weighted Max-Cut (keller)	2	**13.46(2)**	17.43(2)	— (0)
Weighted Max-Cut (mann_a)	4	1155.02(2)	**1015.62(4)**	— (0)
Weighted Max-Cut (p_hat)	12	11.42(12)	**10.92(12)**	— (0)
Weighted Max-Cut (san)	11	82.66(11)	**57.33(11)**	— (0)
Weighted Max-Cut (sanr)	4	42.26(4)	**25.90(4)**	— (0)
Weighted Max-Cut (random)	40	**11.57(40)**	246.96(40)	— (0)
Weighted Max-Cut (spinglass)	5	**40.50(3)**	0.26(2)	— (0)
Max-One	45	**122.37(45)**	343.82(27)	472.66(9)
Quasigroup Completion	25	112.33(10)	94.58(6)	**22.04(24)**
Ramsey	48	**19.04(35)**	54.79(29)	32.54(33)
Weighted CSP (dense-loose)	40	**236.81(34)**	527.02(32)	— (0)
Weighted CSP (dense-tight)	60	**30.68(30)**	— (0)	— (0)
Weighted CSP (sparse-loose)	40	**94.13(32)**	392.13(27)	299.68(25)
Weighted CSP (sparse-tight)	40	**40.96(20)**	— (0)	— (0)
Weighted CSP (spot)	42	**75.77(12)**	15.37(5)	35.34(4)
Solved instances		**441 (599)**	335 (599)	121 (599)

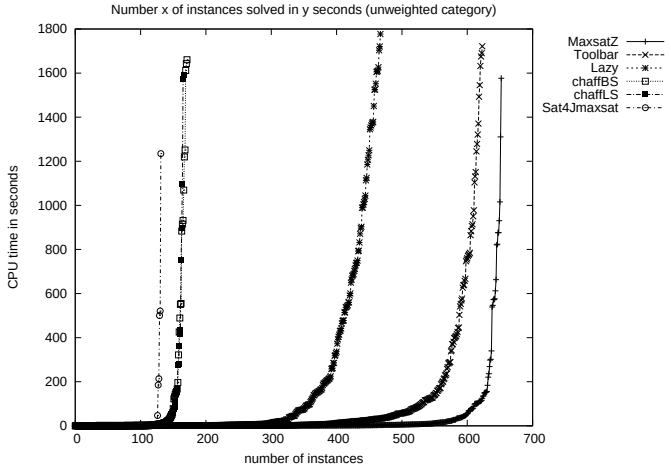

Figure 1. Comparison of the solvers in the Unweighted Max-SAT category of the 2006 Max-SAT Evaluation. A point (x, y) means number x of instances solved in y seconds.

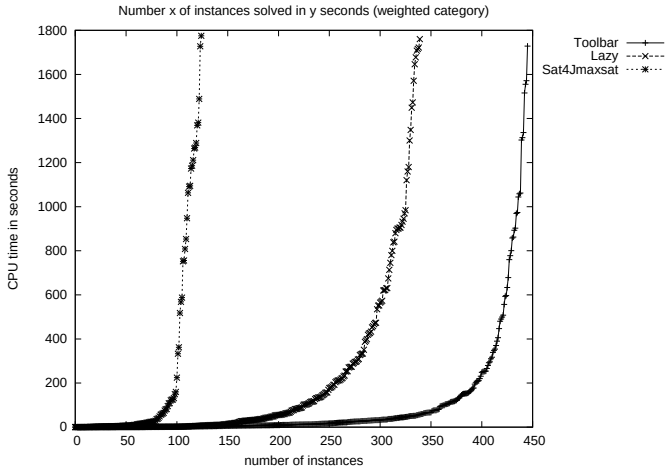

Figure 2. Comparison of the solvers in the Weighted Max-SAT category of the 2006 Max-SAT Evaluation. A point (x, y) means number x of instances solved in y seconds.

Figure 3 and Figure 4 show the scalability of the three fastest solvers on random unweighted Max-2-SAT instances with 100 variables and a number of clauses ranging from 200 to 1200, and on random unweighted Max-3-SAT instances with 70 variables and a number of clauses ranging from 300 to 1000. A point (x, y) in a curve represents the mean time y, in seconds, needed to solve an instance with x clauses by the corresponding solver. 100 instances are solved at each point of the plot. These instances were randomly generated using the generator `mwff` developed by Bart Selman, which allows for duplicated clauses. We observe that MaxSatz outperforms significantly the rest of solvers. We notice that we

consider the three fastest solvers for these instances, independently of the results shown in Table 1.

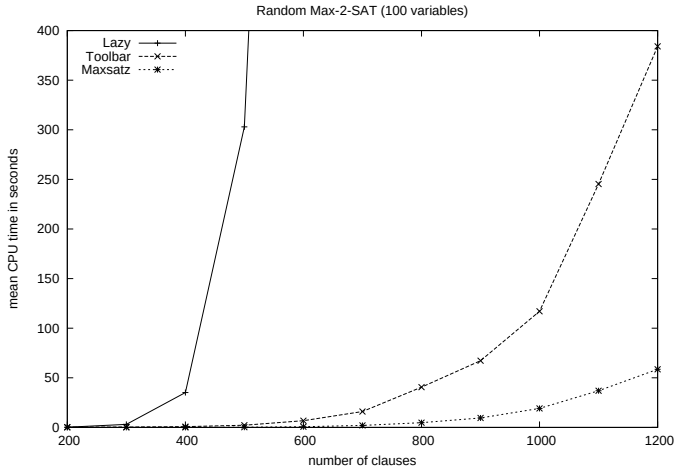

Figure 3. Scalability of the three fastest solvers in the Unweighted Max-SAT category of the 2006 Max-SAT Evaluation on random Max-2-SAT instances with 100 variables and a number of clauses ranging from 200 to 1200.

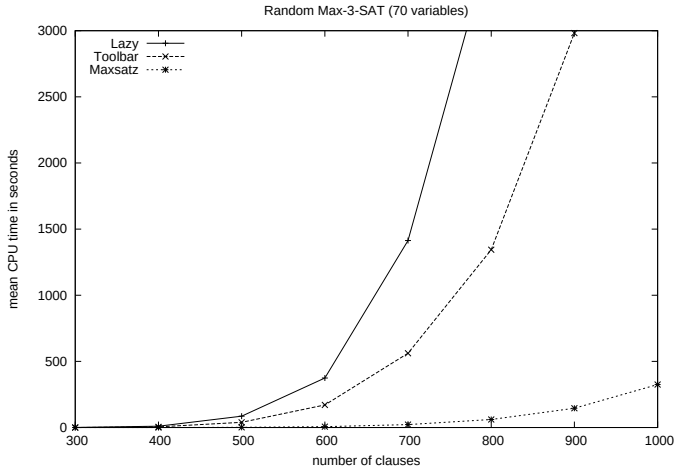

Figure 4. Scalability of the three fastest solvers in the Unweighted Max-SAT category of the 2006 Max-SAT Evaluation on random Max-3-SAT instances with 70 variables and a number of clauses ranging from 300 to 1000.

Figure 5 and Figure 6 show the scalability of the three Weighted Max-SAT solvers of the 2006 Max-SAT Evaluation on weighted random Max-2-SAT instances with 100 variables and number of clauses ranging from 200 to 1000, and on weighted random Max-3-SAT instances

with 70 variables and number of clauses ranging from 300 to 1000. The displayed mean time is computed by solving 100 instances at each point of the plot. These instances were generated using a modified version of the generator `mwff`. The generator associates to each clause, randomly, a weight between 1 and 10. We observe that Toolbar outperforms significantly the rest of solvers on Max-2-SAT, and Toolbar and Lazy are the best performing solvers on Max-3-SAT.

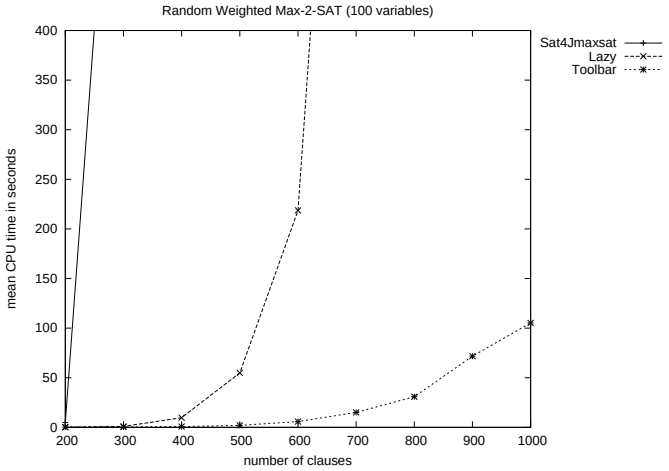

Figure 5. Scalability of the solvers in the Weighted Max-SAT category of the 2006 Max-SAT Evaluation on random Max-2-SAT instances with 100 variables and a number of clauses ranging from 200 to 1000.

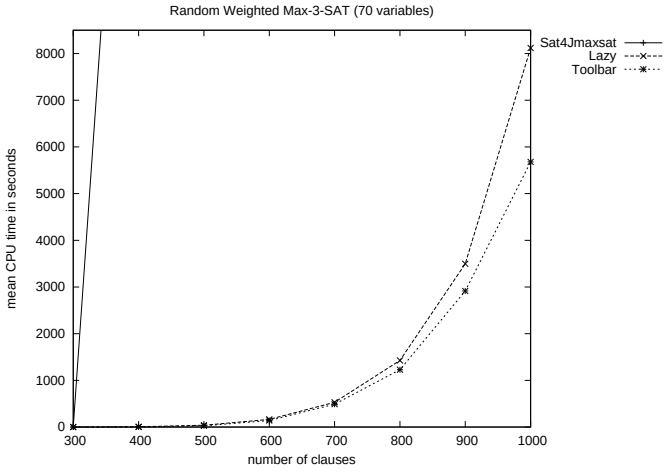

Figure 6. Scalability of the solvers in the Weighted Max-SAT category of the 2006 Max-SAT Evaluation on random Max-3-SAT instances with 70 variables and a number of clauses ranging from 300 to 1000.

5.2 Empirical Results in the 2007 Max-SAT Evaluation

The 2007 Max-SAT Evaluation was organized with the same rules and conditions of the 2006 Max-SAT Evaluation, but there were two additional categories: Partial Max-SAT and Weighted Partial Max-SAT. The instances were also grouped into sets by their authors. There were 15 sets of instances in the Unweighted Max-SAT category, 6 sets of instances in the Weighted Max-SAT category, 15 sets of instances in the Partial Max-SAT category, and 11 sets of instances in the Weighted Partial Max-SAT category. Table 3, Table 4, Table 5, and Table 6 show the results of the four categories of the 2007 Max-SAT Evaluation. We first display the number of instances in each set (#Ins.) and then, for each solver and for each set of instances, the mean time needed to solve an instance and the number of solved instances (in brackets) using a cutoff of 1800 seconds. The last line of each table displays the total number of instances that were solved by each solver. The best results are displayed in bold.

Figure 7, Figure 8, Figure 9, and Figure 10 globally compare the performance of the solvers in the four categories on the instances in Table 3, Table 4, Table 5, and Table 6, respectively. Each point (x, y) of a curve shows the number x of instances that the corresponding solver is able to solve within y seconds. In other words, each of the x instances is solved within y seconds (the total run time for these x instances may be larger than y seconds), y being limited to 1800 seconds (30 minutes) to solve an instance. The best performing solvers of the 2007 Max-SAT Evaluation are MiniMaxSat and the differents variants of MaxSatz. The solvers Clone, LB-SAT, LB-PSAT, PMS, SR(w), Toolbar are competitive on some classes of benchmarks. Solvers ChaffBS, ChaffLS and SAT4Jmaxsat are just competitive on a reduced number of benchmarks.

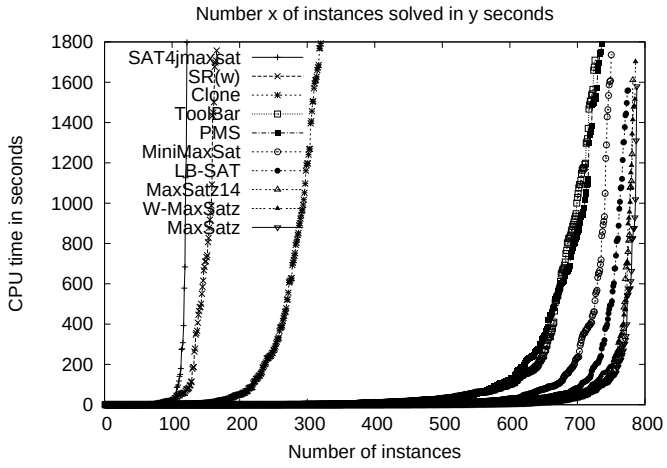

Figure 7. Comparison of the solvers in the Unweighted Max-SAT category of the 2007 Max-SAT Evaluation. A point (x, y) means number x of instances solved in y seconds.

Figure 11 and Figure 12 show the scalability of the three fastest Unweighted Max-SAT solvers of the 2007 evaluation on random Max-2-SAT instances with 100 variables and a number of clauses ranging from 500 to 1000, and on random Max-3-SAT instances with

Table 3. Results in the Unweighted Max-SAT category of the 2007 Max-SAT Evaluation. Mean time in seconds.

Set Name	#Ins.	Clone	LB-SAT	MaxSatz14	MaxSatz	MiniMaxSat	PMS	SAT4J maxsat	SR(w)	ToolBar	W-MaxSatz
Max-3-SAT (40 Vars)	40	376.02(28)	1.23(40)	**1.02**(40)	1.05(40)	3.34(40)	9.48(40)	1462.17(2)	629.93(9)	7.20(40)	1.48(40)
Max-3-SAT (50 Vars)	40	492.35(16)	7.84(40)	6.04(40)	**5.90(40)**	25.79(40)	58.25(40)	480.68(3)	1287.91(2)	57.64(40)	8.60(40)
Max-3-SAT (60 Vars)	40	356.90(13)	24.13(40)	15.61(40)	**14.24(40)**	77.53(38)	128.38(40)	68.05(9)	650.58(4)	272.90(40)	21.63(40)
Max-3-SAT (70 Vars)	40	7.79(10)	124.68(40)	57.85(40)	**48.82(40)**	207.90(35)	191.93(37)	2.24(10)	891.70(8)	334.33(29)	77.88(40)
Spinglass	20	6.19(10)	11.83(20)	43.01(20)	69.40(20)	**4.56(20)**	3.29(10)	— (0)	24.51(10)	24.02(10)	80.76(20)
Ramsey	48	103.20(33)	21.15(35)	12.27(29)	8.99(34)	29.81(34)	29.99(35)	2.88(33)	55.88(23)	**20.40(35)**	16.57(34)
Max-2-SAT (100 Vars)	110	138.34(31)	10.53(110)	1.84(110)	**1.78(110)**	9.62(110)	40.82(110)	17.83(10)	97.45(20)	29.02(110)	2.54(110)
Max-2-SAT (140 Vars)	110	112.22(31)	156.54(103)	**26.83(110)**	29.57(110)	121.54(99)	155.06(93)	37.74(15)	4.77(20)	235.40(96)	39.48(110)
Max-2-SAT (60 Vars)	110	329.83(51)	0.11(110)	**0.03(110)**	**0.03(110)**	0.19(110)	0.23(110)	— (0)	140.84(21)	0.69(110)	0.04(110)
Max-3-SAT (40 Vars)	50	373.87(34)	1.74(50)	**1.43(50)**	1.50(50)	5.53(46)	15.09(50)	5.40(10)	17.02(10)	9.54(50)	2.13(50)
Max-3-SAT (60 Vars)	50	134.16(20)	36.05(50)	25.22(50)	**23.33(50)**	111.81(50)	214.28(50)	1.61(10)	5.48(10)	339.96(48)	35.40(50)
Max-3-SAT (80 Vars)	50	151.15(20)	170.41(42)	210.89(48)	**197.58(49)**	230.82(37)	253.57(41)	111.81(18)	0.45(10)	241.94(28)	245.23(47)
Max-Cut (dimacs_mod)	62	123.42(21)	156.01(52)	**83.66(52)**	83.86(52)	100.06(48)	333.28(44)	0.93(2)	305.10(16)	127.82(48)	145.06(52)
Max-Cut (random)	40	— (0)	10.66(40)	**5.43(40)**	5.58(40)	15.88(40)	683.22(34)	— (0)	— (0)	55.54(40)	8.43(40)
Max-Cut (spinglass)	5	2.67(2)	7.60(3)	25.99(3)	44.96(3)	**1.62(3)**	0.41(2)	— (0)	9.96(2)	4.75(2)	54.07(3)
Solved instances		320 (815)	775 (815)	782 (815)	**788 (815)**	750 (815)	736 (815)	122 (815)	165 (815)	726 (815)	786 (815)

Table 4. Results in the Weighted Max-SAT category of the 2007 Max-SAT Evaluation. Mean time in seconds.

Set Name	#Ins.	Clone	LB-PSAT	MiniMaxSat	SAT4J maxsat	SR(w)	ToolBar	W-MaxSatz
Ramsey	48	98.37(35)	**3.59(36)**	7.08(36)	3.60(32)	82.89(25)	5.48(35)	44.85(36)
Weighted Max-2-SAT	90	197.41(34)	18.79(90)	10.59(90)	156.50(10)	95.18(20)	34.70(90)	**7.95(90)**
Weighted Max-3-SAT	80	248.65(23)	207.84(80)	280.49(70)	6.52(9)	414.35(9)	242.76(51)	**191.40(80)**
Weighted Max-Cut (dimacs_mod)	62	325.43(25)	75.77(55)	77.46(55)	1.32(2)	200.11(16)	**81.48(57)**	93.79(55)
Weighted Max-Cut (random)	40	— (0)	16.39(40)	**5.42(40)**	— (0)	— (0)	15.91(40)	19.22(40)
Weighted Max-Cut (spinglass)	5	2.57(2)	2.50(3)	**45.50(4)**	— (0)	7.83(2)	89.23(3)	35.77(2)
Solved instances		119 (325)	**304 (325)**	295 (325)	53 (325)	72 (325)	276 (325)	303 (325)

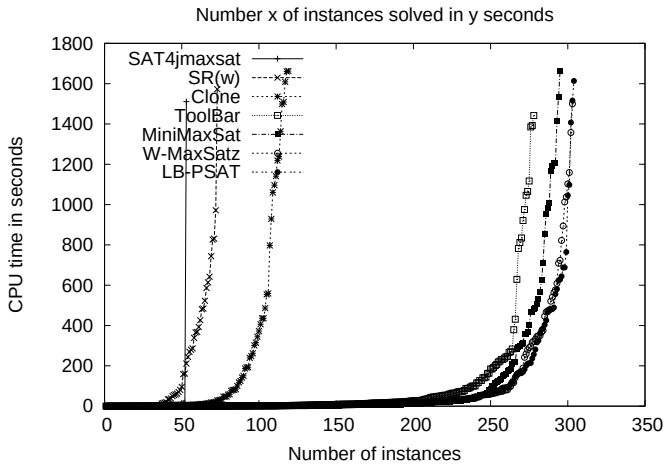

Figure 8. Comparison of the solvers in the Weighted Max-SAT category of the 2007 Max-SAT Evaluation. A point (x, y) means number x of instances solved in y seconds.

70 variables and number of clauses ranging from 500 to 1000. A point (x, y) in a curve represents the mean time y, in seconds, needed to solve an instance with x clauses by the corresponding solver. 100 instances are solved at each point of the plot. The best performing solvers are MaxSatz and MaxSatz14. The same generator was used in both evaluations. We notice that we consider the three fastest solvers for these instances, independently of the results shown in Table 3.

Figure 13 and Figure 14 show the scalability of the three fastest Weighted Max-SAT solvers of the 2007 Max-SAT Evaluation on random Max-2-SAT instances with 100 variables and a number of clauses ranging from 200 to 1000, and on random Max-3-SAT instances with 70 variables and number of clauses ranging from 300 to 1000. A point (x, y) in a curve represents the mean time y, in seconds, needed to solve an instance with x clauses by the corresponding solver. 100 instances are solved at each point to compute the displayed mean

Table 5. Results in the Partial Max-SAT category of the 2007 Max-SAT Evaluation. Mean time in seconds.

Set Name	#Ins.	Chaff_BS	Chaff_LS	Clone	LB-PSAT	MiniMaxSat	PMS	SAT4J maxsat	SR(w)	ToolBar	W-MaxSatz
Partial Max-2-SAT	90	— (0)	— (0)	8.20(1)	305.93(59)	221.57(83)	220.27(44)	— (0)	— (0)	149.86(89)	**40.42(90)**
Partial Max-3-SAT	60	40.25(24)	22.44(22)	251.62(19)	52.42(59)	156.47(58)	80.83(59)	4.57(20)	327.62(16)	172.31(47)	**59.01(60)**
Max-Clique (random)	96	146.24(54)	— (0)	189.65(79)	9.89(96)	**2.39(96)**	68.19(96)	— (0)	225.38(55)	11.39(96)	49.34(80)
Max-Clique (structured)	62	282.83(19)	54.44(9)	308.72(16)	128.34(32)	**85.26(36)**	171.13(27)	13.16(1)	19.35(9)	202.68(33)	153.30(22)
Max-One (3-SAT)	80	402.14(23)	11.67(41)	420.67(54)	62.18(76)	**1.30(80)**	4.23(80)	1013.93(5)	273.87(70)	102.34(80)	199.16(77)
Max-One (structured)	60	52.98(57)	81.21(2)	258.19(32)	2.29(2)	**31.04(60)**	176.71(37)	412.66(3)	443.59(22)	221.31(44)	385.89(54)
Pseudo (garden)	7	1.34(5)	0.78(5)	2.59(5)	**0.47(5)**	7.13(5)	0.55(5)	1.42(3)	2.55(5)	1.82(4)	2.16(4)
Pseudo (logic-synthesis)	17	39.42(2)	**32.16(4)**	— (0)	865.73(3)	216.28(2)	2.55(1)	— (0)	— (0)	— (0)	— (0)
Pseudo (primes-dimacs-cnf)	148	72.92(99)	41.25(46)	89.72(99)	82.68(35)	**88.15(107)**	124.09(88)	82.11(45)	67.03(77)	68.71(60)	129.97(85)
Pseudo (routing)	15	**180.33(15)**	0.22(14)	19.08(5)	— (0)	93.88(14)	25.98(5)	— (0)	— (0)	— (0)	143.94(5)
Weighted CSP (dense-loose)	20	324.93(14)	143.86(6)	831.09(1)	1.16(20)	**0.65(20)**	2.03(20)	— (0)	588.37(1)	336.71(15)	7.19(20)
Weighted CSP (dense-tight)	20	65.83(20)	106.81(18)	25.90(20)	2.87(20)	**0.68(20)**	2.25(20)	— (0)	199.93(18)	461.84(20)	10.53(20)
Weighted CSP (sparse-loose)	20	19.16(20)	41.80(19)	122.28(13)	1.86(20)	**0.35(20)**	1.42(20)	222.86(10)	264.08(16)	4.18(10)	25.55(20)
Weighted CSP (sparse-tight)	20	28.87(20)	16.23(19)	29.58(20)	7.14(20)	**0.85(20)**	2.19(20)	— (0)	219.99(19)	20.36(10)	26.04(20)
Weighted CSP (wqueens)	7	13.94(7)	18.94(5)	80.49(4)	**5.26(7)**	0.52(6)	12.95(7)	11.17(2)	45.17(6)	12.73(5)	85.10(6)
Solved instances		379 (722)	210 (722)	368 (722)	454 (722)	**627 (722)**	529 (722)	89 (722)	314 (722)	513 (722)	563 (722)

Table 6. Results in the Weighted Partial Max-SAT category of the 2007 Max-SAT Evaluation. Mean time in seconds.

Set Name	#Ins.	Clone	MiniMaxSat	SAT4J maxsat	SR(w)	ToolBar	W-MaxSatz
Weighted Partial Max-2-SAT	90	— (0)	246.28(81)	— (0)	— (0)	213.24(88)	**196.30(88)**
Weighted Partial Max-3-SAT	60	136.28(21)	186.63(58)	6.41(20)	275.44(17)	188.75(47)	**91.81(60)**
Auctions (paths)	88	50.78(88)	**31.55(88)**	— (0)	163.45(77)	48.68(88)	243.98(70)
Auctions (region)	84	30.51(84)	**1.61(84)**	— (0)	130.20(82)	6.45(84)	6.70(84)
Auctions (scheduling)	84	228.16(74)	**46.22(84)**	— (0)	231.83(55)	74.11(82)	103.85(82)
Pseudo (factor)	186	9.85(186)	1.17(186)	598.29(55)	— (0)	246.39(12)	**0.43(186)**
Pseudo (miplib)	16	132.42(5)	41.66(5)	6.74(3)	**244.85(6)**	2.92(4)	1.50(4)
Quasigroup Completion	25	— (0)	**25.01(20)**	377.01(14)	652.50(5)	191.07(12)	37.54(11)
Weighted CSP (planning)	71	261.15(62)	**9.97(71)**	73.22(16)	365.48(52)	22.82(52)	101.49(59)
Weighted CSP (spot5-dir)	21	9.32(6)	3.83(3)	0.56(1)	**2.92(6)**	128.04(5)	17.35(2)
Weighted CSP (spot5-log)	21	7.27(5)	9.18(4)	0.54(1)	**14.91(6)**	111.41(4)	640.86(4)
Solved instances		531 (746)	**684 (746)**	110 (746)	306 (746)	478 (746)	650 (746)

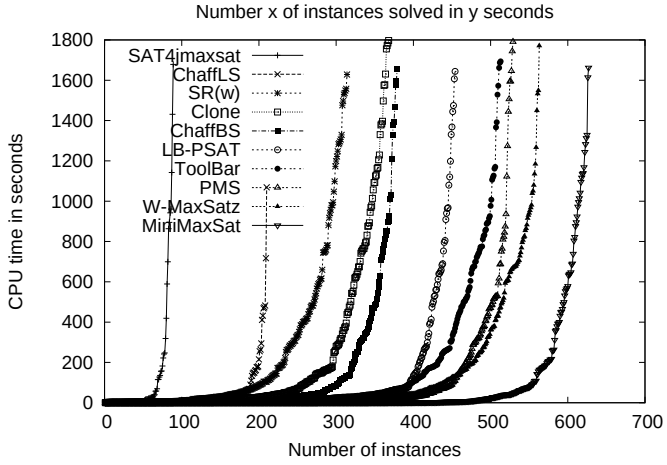

Figure 9. Comparison of the solvers in the Partial Max-SAT category of the 2007 Max-SAT Evaluation. A point (x, y) means number x of instances solved in y seconds.

time. The best performing solver is W-MaxSatz. MiniMaxSat has a good performance profile on Max-2-SAT, and LB-SAT has a good performance profile on Max-3-SAT. We

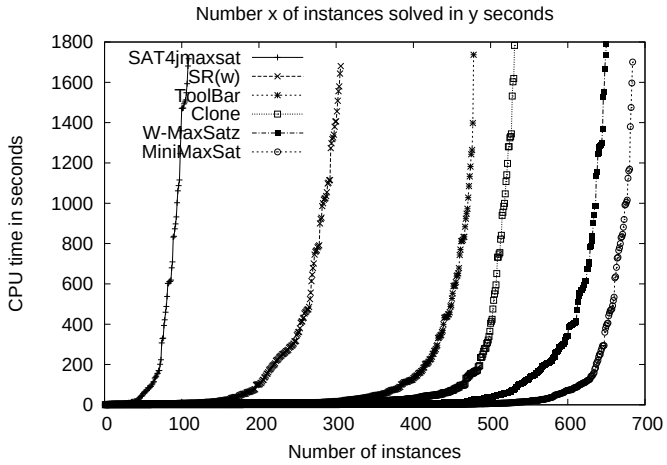

Figure 10. Comparison of the solvers in the Weighted Partial Max-SAT category of the 2007 Max-SAT Evaluation. A point (x, y) means number x of instances solved in y seconds.

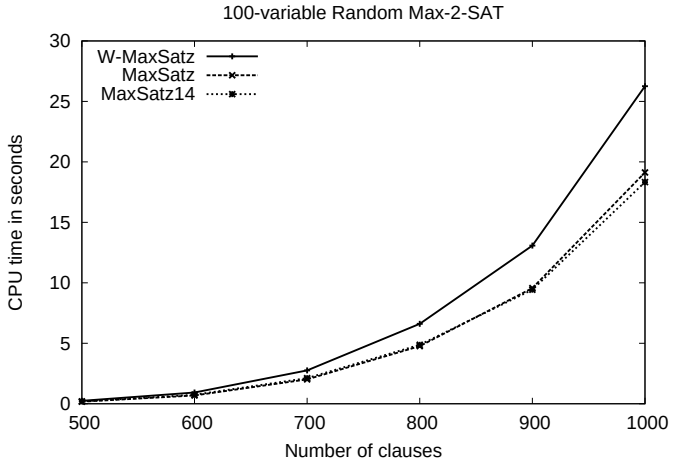

Figure 11. Scalability of the three fastest solvers in the Unweighted Max-SAT category of the 2007 Max-SAT Evaluation on random Max-2-SAT instances with 100 variables and number of clauses ranging from 500 to 1000.

notice that we consider the three fastest solvers for these instances, independently of the results shown in Table 4.

Figure 15 and Figure 16 show the scalability of the three fastest Partial Max-SAT solvers of the 2007 Max-SAT Evaluation on random Max-2-SAT instances with 150 variables, 150 hard clauses and a number of soft clauses ranging from 850 to 4850, and on random Max-3-SAT instances with 100 variables, 100 hard clauses and a number of soft clauses ranging from 200 to 700. 100 instances are solved at each point of the plot. The generator of the

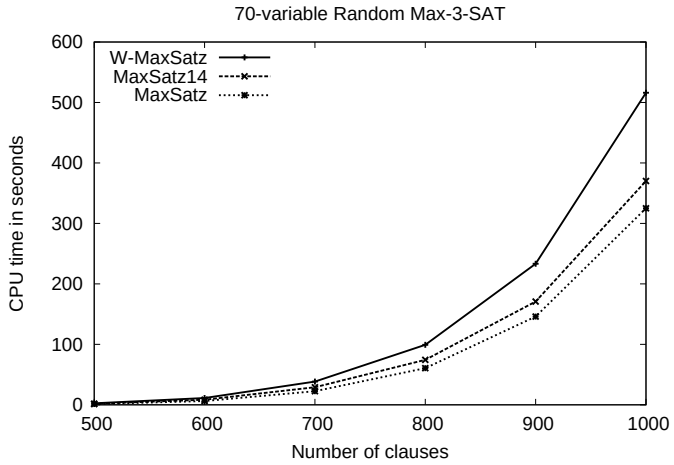

Figure 12. Scalability of the three fastest solvers in the Unweighted Max-SAT category of the 2007 Max-SAT Evaluation on random Max-3-SAT instances with 70 variables and number of clauses ranging from 500 to 1000.

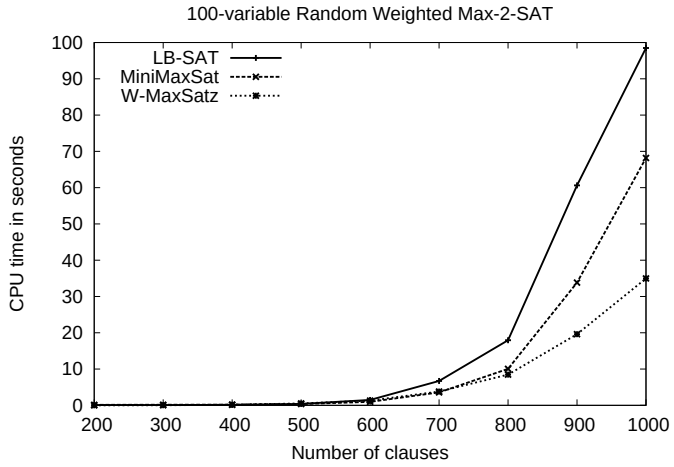

Figure 13. Scalability of the three fastest solvers in the Weighted Max-SAT category of the 2007 Max-SAT Evaluation on random Max-2-SAT instances with 100 variables and number of clauses ranging from 200 to 1000.

2006 edition was modified in order to create Partial Max-SAT instances. A Partial Max-SAT instance is generated as a weighted instance with hard clauses having weight equal to the number of its soft clauses. The number of hard clauses in an instance corresponds to its number of variables. The soft clauses have weight 1. The best performing solver is W-MaxSatz; and LB-SAT has a good performance profile, particularly on Max-3-SAT. We

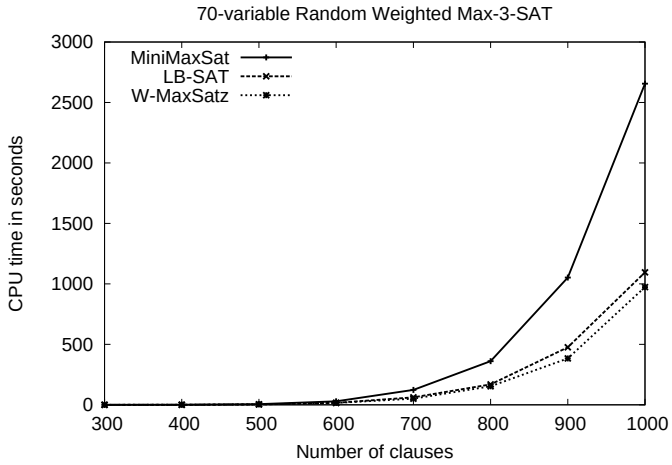

Figure 14. Scalability of the three fastest solvers in the Weighted Max-SAT category of the 2007 Max-SAT Evaluation on random Max-3-SAT instances with 70 variables and number of clauses ranging from 300 to 1000.

notice that we consider the three fastest solvers for these instances, independently of the results shown in Table 5.

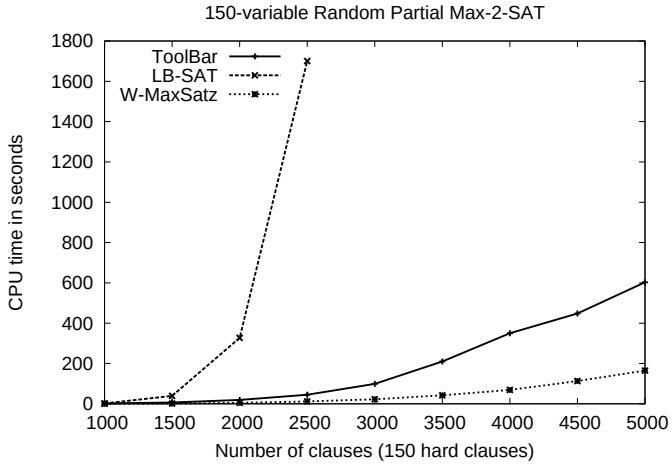

Figure 15. Scalability of the three fastest solvers in the Partial Max-SAT category of the 2007 Max-SAT Evaluation on random Max-2-SAT instances with 150 variables, 150 hard clauses and number of soft clauses ranging from 850 to 4850. The total number of clauses ranges from 1000 to 5000.

Figure 17 and Figure 18 show the scalability of the three fastest Weighted Partial Max-SAT solvers of the 2007 Max-SAT Evaluation on random Max-2-SAT instances with 150

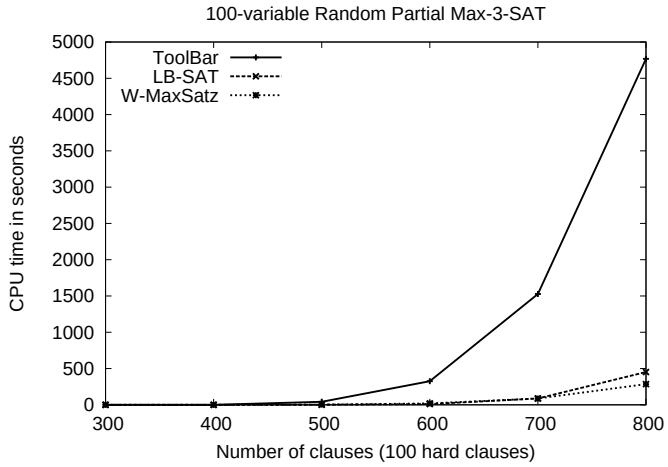

Figure 16. Scalability of the three fastest solvers in the Partial Max-SAT category of the 2007 Max-SAT Evaluation on random Max-3-SAT instances with 100 variables, 100 hard clauses and number of soft clauses ranging from 200 to 700. The total number of clauses ranges from 300 to 800.

variables, 150 hard clauses and a number of soft clauses ranging from 850 to 3850, and on random Max-3-SAT instances with 100 variables, 100 hard clauses and number of soft clauses ranging from 200 to 700. 100 instances are solved at each point of the plot. The generator of the 2006 edition was modified in order to create Weighted Partial Max-SAT instances. A Weighted Partial Max-SAT instance is generated as a weighted instance with hard clauses having weight equal to the sum of the weights of soft clauses (randomly generated between 1 and 10). The number of hard clauses in an instance corresponds to its number of variables. On Max-2-SAT, the best performing solvers are ToolBar and W-MaxSatz. On Max-3-SAT, the best performing solver is W-MaxSatz followed by MiniMaxSat. We notice that we consider the three fastest solvers for these instances, independently of the results shown in Table 6.

6. Conclusions

We believe that the first Max-SAT evaluations have provided a quite accurate snapshot of the current state-of-the-art of exact Max-SAT solvers, have contributed to increase the interest and activity of the research community on Max-SAT, have allowed to identify a number of good performing solving techniques, and have promoted the creation of a publicly available collection of challenging Max-SAT benchmarks.

The solvers participating in the evaluation can be classified into three classes: (i) solvers like ChaffBS, ChaffLS and SAT4Jmaxsat that solve Max-SAT using a SAT encoding and a state-of-the-art SAT solver; (ii) solvers like LB-SAT, MaxSatz, MaxSatz14, MiniMaxsat, and PMS that implement a branch and bound scheme and apply inference rules and compute unit propagation-based underestimations of the lower bound at each node of the proof tree; and (iii) solvers like Clone and SR(w) that implement a branch and bound scheme and

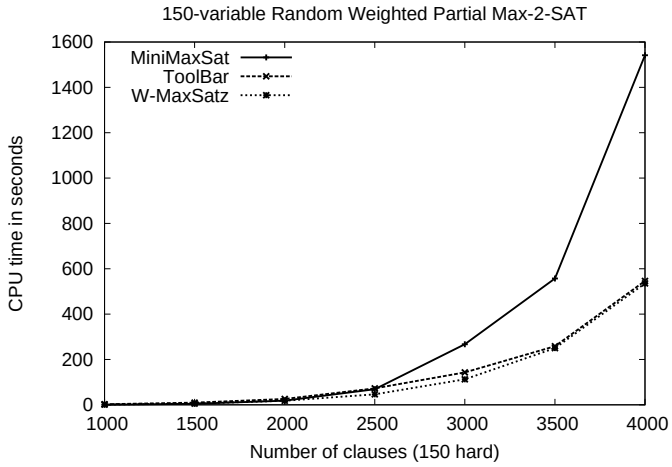

Figure 17. Scalability of the three fastest solvers in the Weighted Partial Max-SAT category of the 2007 Max-SAT Evaluation on random Max-2-SAT instances with 150 variables, 150 hard clauses and number of soft clauses ranging from 850 to 3850. The total number of clauses ranges from 1000 to 4000.

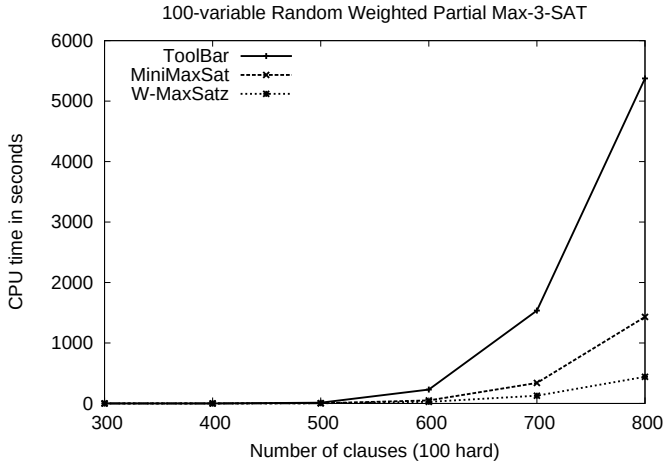

Figure 18. Scalability of the three fastest solvers in the Weighted Partial Max-SAT category of the 2007 Max-SAT Evaluation on random Max-3-SAT instances with 100 variables, 100 hard clauses and number of soft clauses ranging from 200 to 700. The total number of clauses ranges from 300 to 800.

compute an underestimation by solving a relaxation of a d-DNNF compiled translation of the Max-SAT instance into a Minimum Cardinality instance. Most of the solvers solving Partial Max-SAT, independently of the class to which they belong, incorporate learning of

hard clauses. Solvers of the second class were the ones with better performance profile in both evaluations.

The solving techniques that we have identified as powerful and promising are:

- Resolution-style inference rules that transform Max-SAT instances into equivalent Max-SAT instances have a dramatic impact on the the performance profile of solvers. Solvers implementing powerful inference rules include MaxSatz, MaxSatz14, MiniMaxsat, PMS and ToolBar.

- Despite the dramatic improvements achieved by applying inference rules, the computation of good quality underestimations of the lower bound is decisive to speed up solvers. The two more powerful techniques that have been identified are the detection of disjoint inconsistent subsets of clauses via unit propagation and failed literal detection (LB-SAT, MaxSatz, MaxSatz14, MiniMaxsat, PMS), and transforming the Max-SAT instance into a Minimum Cardinality instance and solving a relaxation of this new instance after compiling it with a d-DNNF compiler (Clone, SR(w)).

- Learning of hard clauses produces significant performance improvements on several types of Partial Max-SAT instances.

- The selection of suitable data structures is decisive for producing fast implementations. Solvers incorporating lazy data structures include ChaffBS, ChaffLS, Lazy, MiniMaxsat and SAT4Jmaxsat.

- The formalism used to encode problems has a remarkable impact on performance. When there are hard and soft constraints, the Partial Max-SAT formalism allows to exploit structural information.

As a side effect, most authors of solvers had the opportunity of correcting a significant number of bugs in their code. Since the number of solvers was moderate, the errors detected during the course of the experimentation were notified to the authors. They had the opportunity of correcting the reported errors and submitting a new version of their code. This was very time consuming for the organizers but was very helpful to improve the robustness of the participating solvers.

Future directions of the 2006 and 2007 Max-SAT Evaluations include to solve optimization problems with big integers, promote the participation of solvers from the Operations Research community, promote a collection of industrial benchmarks, and transform the evaluation into a competition. As a first step, in the 2008 Max-SAT Evaluation [5], the benchmarks, in each category, are divided into 3 types: random, crafted and industrial.

7. Acknowledgments

We would like to thank all the people who contributed solvers and benchmarks. Without their effort, these evaluations could not exist. We would also like to thank the *Universitat de Lleida* for allowing to use its cluster for performing the two evaluations.

This research has been partially supported by projects IEA (TIN2006-15662-C02-02), Agreement Technologies (CONSOLIDER CSD2007-0022, INGENIO 2010) and MULOG

(TIN2007-68005-C04-02) funded by the *Ministerio de Educación y Ciencia*. The second author is partially supported by National 973 Program of China under Grant No. 2005CB321900.

References

[1] Teresa Alsinet, Felip Manyà, and Jordi Planes. Improved exact solver for weighted Max-SAT. In *Proceedings of the 8th International Conference on Theory and Applications of Satisfiability Testing, SAT 2005, St. Andrews, Scotland*, pages 371–377. Springer LNCS **3569**, 2005.

[2] Teresa Alsinet, Felip Manyà, and Jordi Planes. An efficient solver for Weighted Max-SAT. *Journal of Global Optimization*, **41**:61–73, 2008.

[3] Josep Argelich, Chu Min Li, Felip Manyà, and Jordi Planes. 2006 Max-SAT Evaluation. http://www.iiia.csic.es/~maxsat06/.

[4] Josep Argelich, Chu Min Li, Felip Manyà, and Jordi Planes. 2007 Max-SAT Evaluation. http://www.maxsat07.udl.es/.

[5] Josep Argelich, Chu Min Li, Felip Manyà, and Jordi Planes. 2008 Max-SAT Evaluation. http://www.maxsat.udl.cat/08/.

[6] Josep Argelich and Felip Manyà. Partial Max-SAT solvers with clause learning. In João Marques-Silva and Karem A. Sakallah, editors, *Proceedings of 10th International Conference on the Theory and Applications of Satisfiability Testing, SAT 2007*, *LNCS* **4501**, pages 28–40. Springer, 2007.

[7] Daniel Le Berre. SAT4J, a satisfiability library for java. http://www.sat4j.org/.

[8] Daniel Le Berre and Laurent Simon. SAT Competition. http://www.satcompetition.org.

[9] Sylvain Darras, Gilles Dequen, Laure Devendeville, and Chu Min Li. On inconsistent clause-subsets for Max-SAT solving. In Christian Bessiere, editor, *Proceedings of 13th International Conference on Principles and Practice of Constraint Programming, CP 2007*, *LNCS*, pages 225–240, Providence, USA, 2007. Springer.

[10] Adnan Darwiche. c2d compiler. http://reasoning.cs.ucla.edu/c2d/.

[11] Simon de Givry, Matthias Zytnicki, Federico Heras, and Javier Larrosa. Existential arc consistency: Getting closer to full arc consistency in weighted CSPs. In *Proceedings of the International Joint Conference on Artificial Intelligence, IJCAI 2005*, pages 84–89, Edinburgh, Scotland, 2005.

[12] Niklas Eén and Niklas Sörensson. The MiniSat page. http://www.cs.chalmers.se/Cs/Research/FormalMethods/MiniSat/.

[13] Zhaohui Fu and Sharad Malik. On solving the partial MAX-SAT problem. In Armin Biere and Carla P. Gomes, editors, *Proceedings of the 9th International Conference on the Theory and Applications of Satisfiability Testing, SAT 2006*, LNCS, pages 252–265, Seattle, USA, 2006. Springer.

[14] Federico Heras, Javier Larrosa, Simon de Givry, and Thomas Schiex. 2006 and 2007 Max-SAT evaluations: Contributed instances. *submitted to JSAT, Special issue on SAT 2007 competitions and evaluations*, 2007.

[15] Federico Heras, Javier Larrosa, and Albert Oliveras. MiniMaxSat: A new weighted Max-SAT solver. In Joao Marques-Silva and Karem A. Sakallah, editors, *Proceedings of 10th International Conference on the Theory and Applications of Satisfiability Testing, SAT 2007, LNCS* **4501**, pages 41–55, 2007. Springer.

[16] Federico Heras, Javier Larrosa, and Albert Oliveras. MiniMaxSAT: An efficient weighted Max-SAT solver. *Journal of Artificial Intelligence Research*, **31**:1–32, 2008.

[17] Javier Larrosa and Federico Heras. Resolution in Max-SAT and its relation to local consistency in weighted CSPs. In *Proceedings of the International Joint Conference on Artificial Intelligence, IJCAI 2005*, Edinburgh, Scotland, 2005.

[18] Javier Larrosa, Federico Heras, and Simon de Givry. A logical approach to efficient Max-SAT solving. *Artificial Intelligence*, **172**(2–3):204–233, 2008.

[19] Chu Min Li and Anbulagan. Look-ahead versus look-back for satisfiability problems. In *Proceedings of the 3rd International Conference on Principles of Constraint Programming, CP'97, Linz, Austria*, *LNCS* **1330**, pages 341–355, 1997. Springer.

[20] Chu Min Li, Felip Manyà, and Jordi Planes. Exploiting unit propagation to compute lower bounds in branch and bound Max-SAT solvers. In *Proceedings of the 11th International Conference on Principles and Practice of Constraint Programming, CP-2005, Sitges, Spain*, pages 403–414.*LNCS* **3709**, 2005. Springer.

[21] Chu Min Li, Felip Manyà, and Jordi Planes. Detecting disjoint inconsistent subformulas for computing lower bounds for Max-SAT. In *Proceedings of the 21st National Conference on Artificial Intelligence, AAAI 2006*, page 14, Boston, USA, 2006. AAAI Press.

[22] Chu Min Li, Felip Manyà, and Jordi Planes. New inference rules for Max-SAT. *Journal of Artificial Intelligence Research*, **30**:321–359, 2007.

[23] Han Li and Kaile Su. Exploiting inference rules to compute lower bounds for MAX-SAT solving. In *Proceedings of the International Joint Conference on Artificial Intelligence, IJCAI 2007*, page 376, Hyderabad, India, 2007.

[24] Vasco Manquinho and Olivier Roussel. 2006 Pseudo Boolean Evaluation. http://www.cril.univ-artois.fr/PB06/.

[25] M. Moskewicz, C. Madigan, Y. Zhao, L. Zhang, and S. Malik. Chaff: Engineering an efficient SAT solver. In *39th Design Automation Conference*, 2001.

[26] Knot Pipatsrisawat and Adnan Darwiche. Clone: Solving weighted Max-SAT in a reduced search space. In *Proceedings of the Australian Conference on Artificial Intelligence, AI 2007*, LNCS, Queensland, Australia, 2007. Springer.

[27] Knot Pipatsrisawat, Akop Palyan, Mark Chavira, Arthur Choi, and Adnan Darwiche. Solving weighted max-sat problems in a reduced search space: A performance analysis. *Journal on Satisfiability Boolean Modeling and Computation (JSAT)* **4**:191–217, 2008.

[28] Miquel Ramirez and Hector Geffner. Structural relaxations by variable renaming and their compilation for solving MinCostSAT. In Christian Bessiere, editor, *Proceedings of 13th International Conference on Principles and Practice of Constraint Programming, CP 2007*, *LNCS* **4741**, pages 605–619, Providence, USA, 2007. Springer.