Max-SAT Evaluation 2007

Unweighted and Weighted Max-SAT Categories

Josep Argelich
Computer Science Department
Universitat de Lleida
jargelich@diei.udl.es

Chu Min Li
LaRIA
Université de Picardie Jules Verne
chu-min.li@u-picardie.fr

Felip Manyà
Computer Science Department
Universitat de Lleida
felip@diei.udl.es

Jordi Planes
School of Electronics and Computer Science
University of Southampton
jp3@ecs.soton.ac.uk

Solvers:

- **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.

- **LB-SAT (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 and Maxsatz, 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 “Exploiting Inference Rules to Compute Lower Bounds for MAX-SAT Solving” by Han Lin and Kaile Su at IJCAI'07.

- **MaxSatz (Chu Min Li, Felip Manyà, and Jordi Planes)**
  MaxSatz applies resolution style inference rules, and incorporates a lower bound computation method that increments the lower bound by one when it detects an inconsistent subset using unit propagation and failed literal detection. It adapts to Max-SAT the technology of the SAT solver Satz. For further details see "Detecting Disjoint Inconsistent Subformulas for Computing Lower Bounds for Max-SAT" at AAAI'06, and “New Inference Rules for Max-SAT” at Journal of Artificial Intelligence Research, by Li, Manyà and Planes.

- **MaxSatz14 (Sylvain Darras, Gilles Dequen, Laure Devendeville, and Chu Min Li)**
  This solver is based on the last release of Maxsatz solver, built and improved by Chu Min Li, Felip Manyà and Jordi Planes. Our 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.

- **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.

- **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 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 "Partial Max-SAT Solvers with Clause Learning" by Argelich and Manyà at SAT'07.

- **SAT4Jmaxsat (Daniel Le Berre)**
  SAT4Jmaxsat translates a Max-SAT instance S = {C1, C2, ... Cm} with n variables into the following optimization problem: For each clause Ci in the original problem, a new variable Vi is created and added. Some people call those variables selector variables because satisfiying such variable disable a clause. So solving Max-SAT on the original problem is equivalent to solve the optimization problem: Minimize the number of Vi satisfied in S' = {C1 v V1, C2 v V2 ... Cn v Vn}. Since SAT4J supports cardinality constraints, we simply ask a SAT solver to solve S', and each time a model M is found, we try to find a better one, by adding a cardinality constraints SUM(Vi) < number_of_Vi_satisfied_in_M. Once S' and all the cardinality constraints are inconsistent, the latest model is the optimal solution.

- **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−DNNF compiler by Darwiche and the state−of−the−art SAT solver MiniSAT 2.0.

- **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 [Larrosa and Heras 2005: Resolution rule for Max-SAT and its relation to local consistency for WCSP]. 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 v y,2), (¬x v y,1) are replaced by (x v y,1), (y,1) (This is detected by AC* in WCSP).
  - clauses (x,1), (¬x v y,2), (¬y v z,1), (¬z,1) are replaced by (□,1),(¬x v y,1),(x v ¬y,1),(y v ¬z,1) where (□, 1) represents an increment of the lower bound (This is detected by EDAC* in WCSP).

- **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 add an remove clauses without a significant overhead in CPU time. W-MaxSatz has a dynamic variable selection heuristic, advanced inference rules and lower bound computation based on unit propagation and failed literals detection.

Benchmarks:

Benchmarks contributed by Han Li:
- **Random Max-SAT**
  Max-3-SAT instances with 40,50, 60, and 70 variables.
- **Max-Cut**
  Spin-glass

Benchmarks contributed by de Givry, Heras, Larrosa, and Schiex:
- **Max-Cut**
  Random graphs, Dimacs simplified graphs and spinglass instances.
- **Random Max-SAT**
  Max-2-SAT instances with 60, 100 and 140 variables.
  Max-3-SAT instances with 40, 60 and 80 variables.
  In both cases, clauses range from 200 to 1000 (step 200).
- **Ramsey Number Problems**

Benchmarks contributed by the organizers:
- **Random Max-SAT**
  Max-2-SAT instances with 100 variables.
  Max-3-SAT instances with 70 variables.
- **Random Weighted Max-SAT**
  Weighted Max-2-SAT instances with 100 variables.
  Weighted Max-3-SAT instances with 70 variables.

Computational resources:

- **The experiments were performed on a Linux Cluster with 160 processors where each processor is a 2GHz AMD Opteron with 1Gb of RAM.**

The cluster was provided by the Universitat de Lleida.

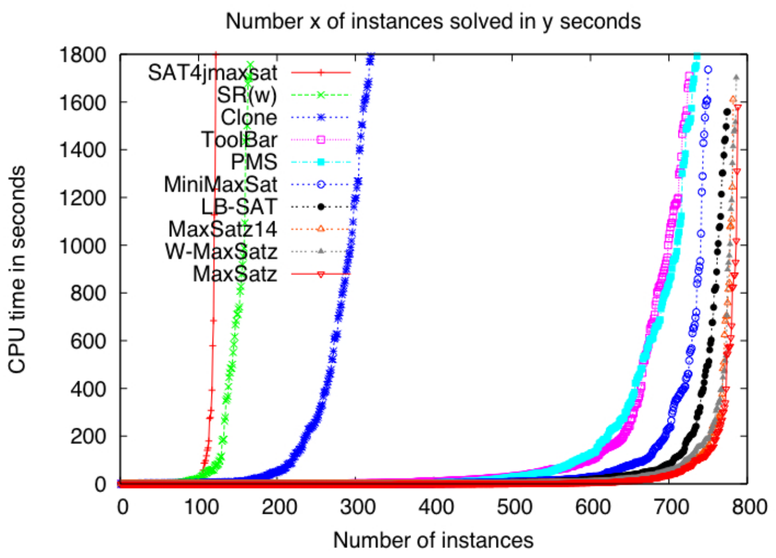
The Max-SAT Evaluation has been organized by:

Universitat de Lleida · LaRIA · University of Southampton

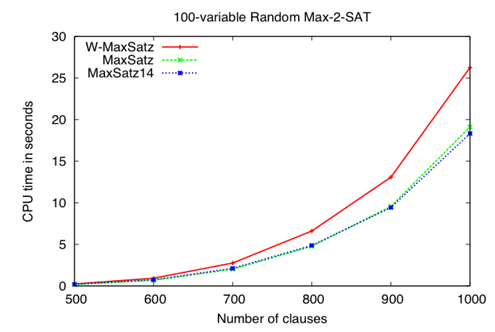
Unweighted Max-SAT Category

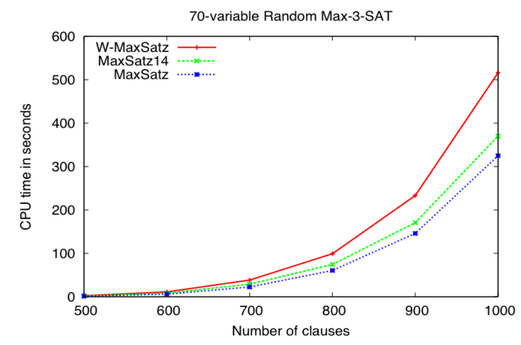
Set Name	NI	Clone	LB-SAT	MaxSatz14	MaxSatz	MiniMaxSat	PMS	SAT4jmaxsat	SR(w)	ToolBar	W-MaxSatz
MAX3SAT/40VARS	40	376.02(28)	1.23 (40)	1.02 (40)	1.05 (40)	3.34 (40)	9.48 (40)	1462.17(2)	629.93(9)	7.20 (40)	1.48 (40)
MAX3SAT/50VARS	40	492.35(16)	7.84 (40)	6.04 (40)	5.90 (40)	25.79(40)	58.45 (40)	480.68(3)	1287.91(2)	57.64 (40)	8.60 (40)
MAX3SAT/60VARS	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)
MAX3SAT/70VARS	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)	-	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)
MAX2SAT/100VARS	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)
MAX2SAT/140VARS	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)
MAX2SAT/60VARS	110	329.83(51)	0.11 (110)	0.03 (110)	0.03 (110)	0.19 (110)	0.23 (110)	-	140.84(21)	0.69 (110)	0.04 (110)
MAX3SAT/40VARS	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)
MAX3SAT/60VARS	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)
MAX3SAT/80VARS	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)
MAXCUT/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)
MAXCUT/RANDOM	40	-	10.66(40)	5.43 (40)	5.58 (40)	15.88(40)	683.22 (34)	-	-	55.54(40)	8.43 (40)
MAXCUT/SPINGLASS	5	2.67 (2)	7.60 (3)	25.99(3)	44.96(3)	1.62 (3)	0.41 (2)	-	9.96 (2)	4.75 (2)	54.07 (3)

- For each solver and for each set of instances, we display the mean time of the solved instances and the number of solved instances (in brackets). Time in seconds. Timeout: 30 minutes.
- The best performing solver: it solves the maximum number of instances in minimum time
- Solvers solving the same number of instances as the best performing solver



Random instances



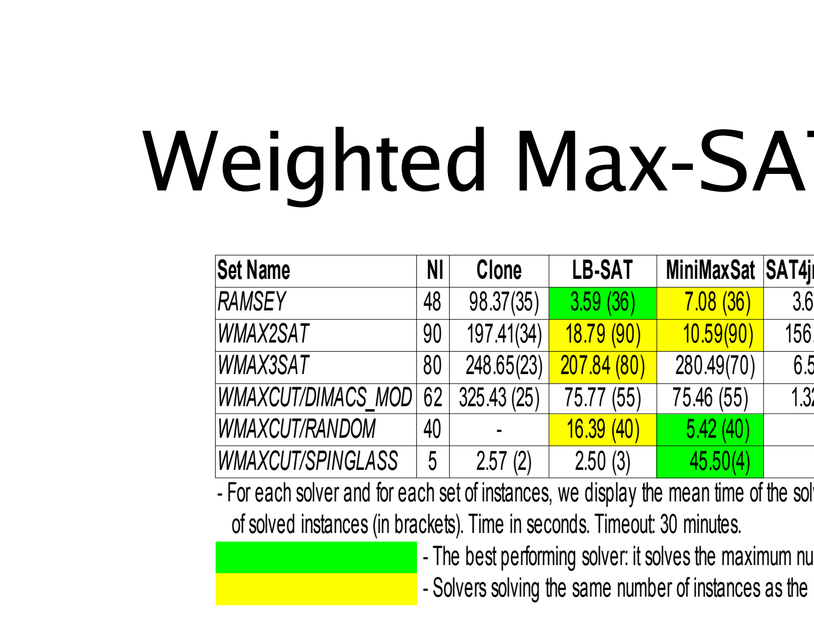


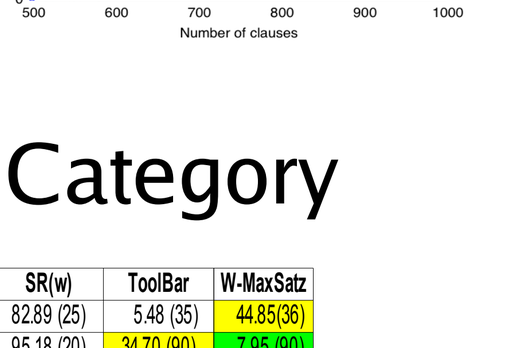
Weighted Max-SAT Category

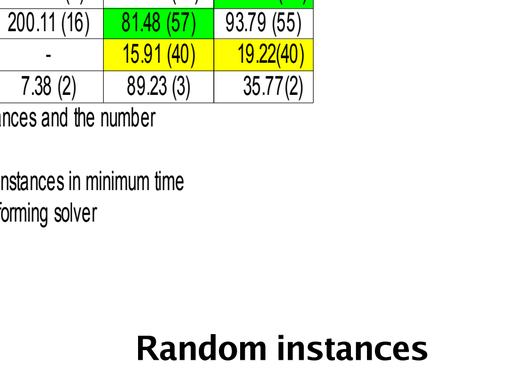
Set Name	NI	Clone	LB-SAT	MiniMaxSat	SAT4jmaxsat	SR(w)	ToolBar	W-MaxSatz
RAMSEY	48	98.37(35)	3.59 (36)	7.08 (36)	3.60 (32)	82.89 (25)	5.48 (35)	44.85(36)
WMAX2SAT	90	197.41(34)	18.79 (90)	10.59(90)	156.50(10)	95.18 (20)	34.70 (90)	7.95 (90)
WMAX3SAT	80	248.65(23)	207.84 (80)	280.49(70)	6.52 (9)	414.35 (9)	242.76 (51)	191.40(80)
WMAXCUT/DIMACS_MOD	62	325.43 (25)	75.77 (55)	75.46 (55)	1.32 (2)	200.11 (16)	81.48 (57)	93.79 (55)
WMAXCUT/RANDOM	40	-	16.39 (40)	5.42 (40)	-	-	15.91 (40)	19.22(40)
WMAXCUT/SPINGLASS	5	2.57 (2)	2.50 (3)	45.50(4)	-	7.38 (2)	89.23 (3)	35.77(2)

- For each solver and for each set of instances, we display the mean time of the solved instances and the number of solved instances (in brackets). Time in seconds. Timeout: 30 minutes.
- The best performing solver: it solves the maximum number of instances in minimum time
- Solvers solving the same number of instances as the best performing solver

Random instances







Max-SAT Evaluation 2007

Unweighted and Weighted Partial Max-SAT Categories

Josep Argelich
Computer Science Department
Universitat de Lleida
jargelich@diei.udl.es

Chu Min Li
LaRIA
Université de Picardie Jules Verne
chu-min.li@u-picardie.fr

Felip Manyà
Computer Science Department
Universitat de Lleida
felip@diei.udl.es

Jordi Planes
School of Electronics and Computer Science
University of Southampton
jp3@ecs.soton.ac.uk

Solvers:

- **ChaffBS & ChaffLS (Zhaohui Fu, and Sharad Malik)**
  Both ChaffBS and ChaffLS are implemented on top of the SAT solver zChaff. In order to translate a Max-SAT instance into a SAT one, it appends a distinct slack variable to every Max-SAT clause. A true slack variable essentially means 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 slack 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 slack 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 slack variables. In order to find the minimum k, i.e. the minimum number of true slack variables, we 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 "On Solving the Partial Max-SAT Problem" by Fu and Malik at SAT'06.

- **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.

- **LB-SAT (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 and Maxsatz, 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 “Exploiting Inference Rules to Compute Lower Bounds for MAX-SAT Solving” by Han Lin and Kaile Su at IJCAI'07.

- **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.

- **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 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 "Partial Max-SAT Solvers with Clause Learning" by Argelich and Manyà at SAT'07.

- **SAT4Jmaxsat (Daniel Le Berre)**
  SAT4Jmaxsat translates a Max-SAT instance S = {C1, C2, ... Cm} with n variables into the following optimization problem: For each clause Ci in the original problem, a new variable Vi is created and added. Some people call those variables selector variables because satisfiying such variable disable a clause. So solving Max-SAT on the original problem is equivalent to solve the optimization problem: Minimize the number of Vi satisfied in S' = {C1 v V1, C2 v V2 ... Cn v Vn}. Since SAT4J supports cardinality constraints, we simply ask a SAT solver to solve S', and each time a model M is found, we try to find a better one, by adding a cardinality constraints SUM(Vi) < number_of_Vi_satisfied_in_M. Once S' and all the cardinality constraints are inconsistent, the latest model is the optimal solution.

- **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-DNNF compiler by Darwiche and the state-of-the-art SAT solver MiniSAT 2.0.

- **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 [Larrosa and Heras 2005: Resolution rule for Max-SAT and its relation to local consistency for WCSP]. 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 v y,2), (¬x v y,1) are replaced by (x v y,1), (y,1) (This is detected by AC* in WCSP).
  - clauses (x,1), (¬x v y,2), (¬y v z,1), (¬z,1) are replaced by (□,1),(¬x v y,1),(x v ¬y,1),(y v ¬z,1) where (□, 1) represents an increment of the lower bound (This is detected by EDAC* in WCSP).

- **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 add an remove clauses without a significant overhead in CPU time. W-MaxSatz has a dynamic variable selection heuristic, advanced inference rules and lower bound computation based on unit propagation and failed literals detection. The current version does not exploit the fact that some clauses are declared as hard.

Benchmarks:

Benchmarks contributed by de Givry, Heras, Larrosa and Schiex in Unweighted Partial Max-SAT:

- **Max-Clique**
  Random graphs, and Dimacs graphs.
- **Max-Ones**
  Random instances, and SAT-2002 Competition structured instances.
- **Max-CSP (WCSP)**
  Random Binary Max-CSP in four distributions: Dense loose, dense tight, sparse loose, and sparse tight.
- **Weighted Queens (WCSP), Garden (Pseudo), Logic synthesis (Pseudo), Minimum prime implicant (Pseudo), and Routing (Pseudo)**

Benchmarks contributed by de Givry, Heras, Larrosa, and Schiex in Weighted Partial Max-SAT:

- **Combinatorial auctions**
  Random combinatorial auctions in three different distrubutions: paths, schedulling, and regions.
- **Planning (WCSP), Spot (WCSP), Quasigroup Completion Problem, Factor (Pseudo), and MIPLib (Pseudo)**

Problems labeled with **(WCSP)** were obtained from a Weighted CSP repository, and problems labeled with **(Pseudo)** were obtained from the Pseudo-Boolean Evaluation 2005 repository.

Benchmarks contributed by the organizers:

- **Random Partial Max-SAT**
  Partial Max-2-SAT instances with 150 variables.
  Partial Max-3-SAT instances with 100 variables.
- **Random Weighted Partial Max-SAT**
  Weighted Partial Max-2-SAT instances with 150 variables.
  Weighted Partial Max-3-SAT instances with 100 variables.

Computational resources:

- **The experiments were performed on a Linux Cluster with 160 processors, where each processor is a 2GHz AMD Opteron with 1Gb of RAM.**

The cluster was provided by the Universitat de Lleida.

The Max-SAT Evaluation has been organized by:

Universitat de Lleida — LaRIA — University of Southampton

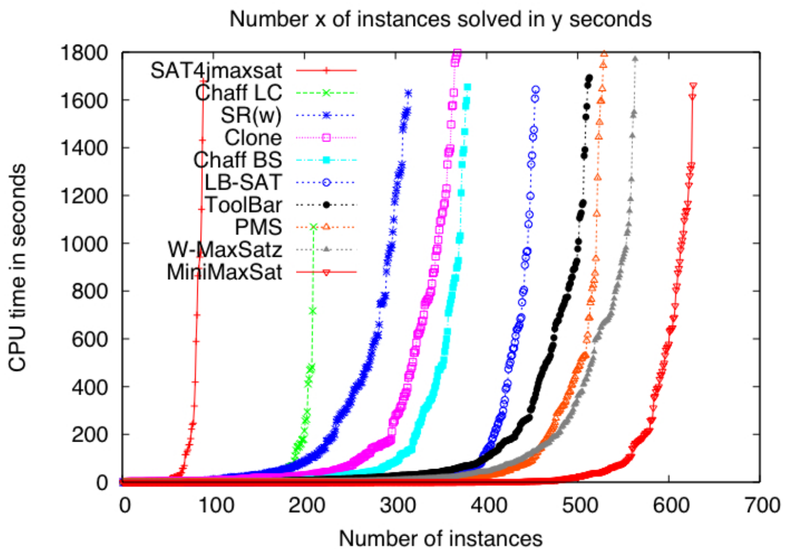
Unweighted Partial Max-SAT Category

Set Name	NI	Chaff_BS	Chaff_LC	Clone	LB-SAT	MiniMaxSat	PMS	SAT4jmaxsat	SR(w)	ToolBar	W-MaxSatz
RANDOM/PMAX2SAT/	90	-	-	8.20 (1)	305.93 (59)	221.57(83)	220.27 (44)	-	-	149.86(89)	40.42(90)
RANDOM/PMAX3SAT/	60	40.25(24)	22.44(22)	251.62(19)	52.42 (59)	156.47(58)	80.83 (59)	4.57 (20)	327 (16)	172.31(47)	59.01(60)
MAXCLIQUE/RANDOM/	96	146.24(54)	-	189.65(79)	9.89 (96)	2.39 (96)	68.19 (96)	-	225.38 (55)	11.39(96)	49.34(80)
MAXCLIQUE/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)
MAXONE/3SAT/	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)
MAXONE/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)	-	865.73 (3)	216.28(2)	2.55 (1)	-	-	-	-
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)	-	93.88(14)	25.98 (5)	-	-	-	143.94(5)
WCSP/MAXCSP/DENSE_LOOSE/	20	324.93(14)	143.86(6)	831.09(1)	1.16 (20)	0.65 (20)	2.03 (20)	-	588.37 (1)	336.71(15)	7.19 (20)
WCSP/MAXCSP/DENSE_TIGHT/	20	65.83(20)	106.81(18)	25.90(20)	2.87 (20)	0.68 (20)	2.25 (20)	-	199.93 (18)	461.84(20)	10.53(20)
WCSP/MAXCSP/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)
WCSP/MAXCSP/SPARSE_TIGHT/	20	28.87(20)	16.23(19)	29.58(20)	7.14 (20)	0.85 (20)	2.19 (20)	-	219.99 (19)	20.36(10)	26.04(20)
WCSP/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)

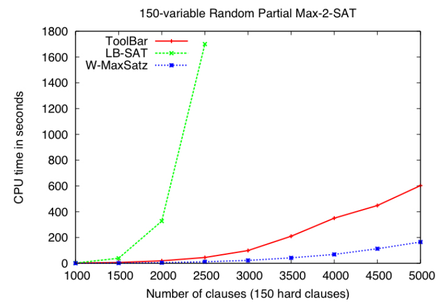
- For each solver and for each set of instances, we display the mean time of the solved instances and the number of solved instances (in brackets). Time in seconds. Timeout: 30 minutes.

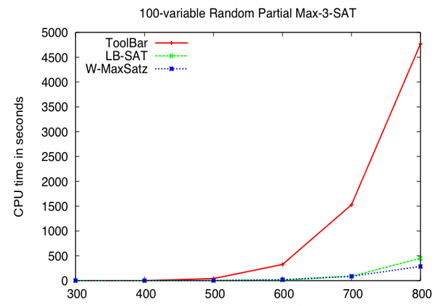
- The best performing solver: it solves the maximum number of instances in minimum time

- Solvers solving the same number of instances as the best performing solver



Random instances





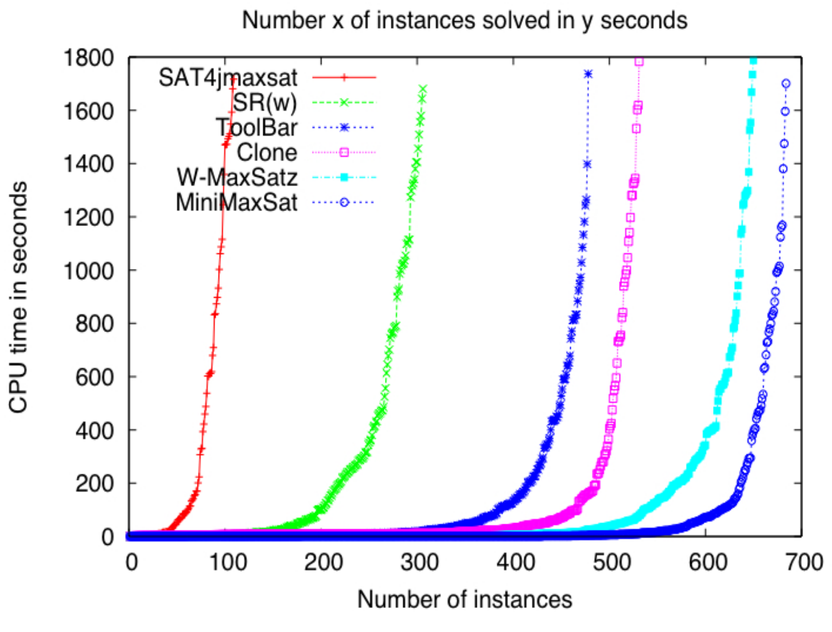
Weighted Partial Max-SAT Category

Set_Name	NI	Clone	MiniMaxSat	SAT4jmaxsat	SR(w)	ToolBar	W-MaxSatz
RANDOM/WPMAX2SAT/	90	-	246.28(81)	-	-	213.24(88)	196.30(88)
RANDOM/WPMAX3SAT/	60	136.28(21)	186.63(58)	6.41 (20)	275.44 (17)	188.75(47)	91.81(60)
AUCTIONS/AUC_PATHS/	88	50.78(88)	31.55(88)	-	163.45 (77)	48.68(88)	243.98(70)
AUCTIONS/AUC_REGIONS/	84	30.51(84)	1.61 (84)	-	130.20 (82)	6.45 (84)	6.70 (84)
AUCTIONS/AUC_SCHEDULING/	84	228.16(74)	46.22(84)	-	231.83 (55)	74.11(82)	103.85(82)
PSEUDO/factor/	186	9.85 (186)	1.17 (186)	598.29(55)	-	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)
QCP/	25	-	25.01(20)	377.01(14)	652.50 (5)	191.07(12)	37.54(11)
WCSP/PLANNING/	71	261.15(62)	9.97 (71)	73.22(16)	365.48 (52)	22.82(52)	101.49(59)
WCSP/SPOT5/DIR/	21	9.32 (6)	3.83 (3)	0.56 (1)	2.92 (6)	128.04(5)	17.35(2)
WCSP/SPOT5/LOG/	21	7.27 (5)	9.18 (4)	0.54 (1)	14.91 (6)	111.41(4)	640.86(4)

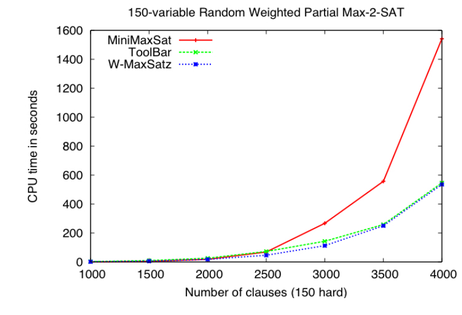
- For each solver and for each set of instances, we display the mean time of the solved instances and the number of solved instances (in brackets). Time in seconds. Timeout: 30 minutes.

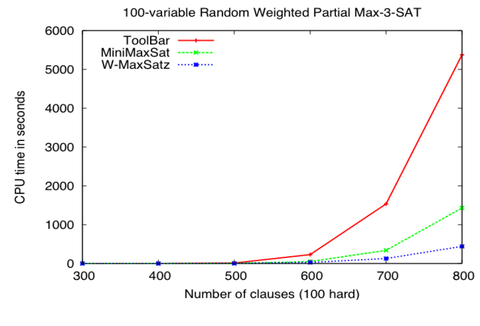
- The best performing solver: it solves the maximum number of instances in minimum time

- Solvers solving the same number of instances as the best performing solver



Random instances





Computational resources:

- **The experiments were performed on a Linux Cluster with 160 processors, where each processor is a 2GHz AMD Opteron with 1Gb of RAM.**

The cluster was provided by the Universitat de Lleida.

The Max-SAT Evaluation has been organized by:

Universitat de Lleida — LaRIA — University of Southampton