

Minimum Satisfiability and its Applications*

Chu-Min Li^{1,2}

¹Huazhong University of Science and Technology, Wuhan, China
chu-min.li@u-picardie.fr

Zhu Zhu²

Université de Picardie Jules Verne
Amiens, France
zhu.zhu@u-picardie.fr

Felip Manyà³

³IIIA-CSIC
Bellaterra, Spain
felip@iiia.csic.es

Laurent Simon⁴

⁴LRI/CNRS/INRIA & Université Paris 11
Orsay, France
Laurent.Simon@lri.fr

Abstract

We define solving techniques for the Minimum Satisfiability Problem (MinSAT), propose an efficient branch-and-bound algorithm to solve the Weighted Partial MinSAT problem, and report on an empirical evaluation of the algorithm on Min-3SAT, MaxClique, and combinatorial auction problems. Techniques solving MinSAT are substantially different from those for the Maximum Satisfiability Problem (MaxSAT). Our results provide empirical evidence that solving combinatorial optimization problems by reducing them to MinSAT may be substantially faster than reducing them to MaxSAT, and even competitive with specific algorithms. We also use MinSAT to study an interesting correlation between the minimum number and the maximum number of satisfied clauses of a SAT instance.

1 Introduction

Solving NP-complete decision problems by reducing them to the propositional satisfiability problem (SAT) is a powerful solving strategy that is widely used to tackle both academic and industrial problems. Recently, the success of SAT has led to explore MaxSAT formalisms such as Weighted MaxSAT and Weighted Partial MaxSAT [Li and Manyà, 2009]. The results achieved so far indicate that they are becoming promising and competitive generic approaches for solving practical optimization problems.

In this paper, contrarily to MaxSAT formalisms, which focus on maximizing the cost of satisfied clauses, we will take the opposite direction and focus on minimizing that cost. Specifically, we focus on the Weighted Partial MinSAT problem, where instances are formed by a set of clauses, each clause is declared to be either hard or soft, and each clause has an associated weight. Solving a Weighted Partial MinSAT instance amounts to finding an assignment that satisfies all the hard clauses, and minimizes the sum of the weights

of satisfied soft clauses. MinSAT is interesting for two reasons: (1) Some problems admit more natural encodings if we look at them as minimization instead of maximization problems, (2) Minimization allows us to consider novel and powerful upper bounding techniques which cannot be applied to branch-and-bound MaxSAT solvers. Indeed, although MinSAT and MaxSAT are both natural extension of SAT, their solving techniques are very different.

Concretely, we propose in this paper a branch-and-bound algorithm for Weighted Partial MinSAT equipped with original bounding techniques, called *MinSatz*, and report on an empirical investigation. Our experiments include Min-3SAT, MaxClique, and combinatorial auction problems, and the obtained results provide empirical evidence that solving such problems by reducing them to MinSAT may be substantially faster than reducing them to MaxSAT, and even competitive with specific algorithms. MinSAT also allows us to study an interesting correlation between the minimum number and the maximum number of satisfied clauses of a SAT instance.

To the best of our knowledge, this is the first genuine exact solver for Weighted Partial MinSAT, as well as for its variants MinSAT, Weighted MinSAT, and Partial MinSAT. It is also the first time that the use of MinSAT formalisms has been proposed for solving practical optimization problems. The closest work to our approach was proposed in [Li *et al.*, 2010b]: A number of encodings were defined for reducing (unweighted) MinSAT to Partial MaxSAT, and experiments were limited to random Min-2SAT and Min-3SAT instances. One drawback of that work is that the defined encodings do not generalize to Weighted Partial MinSAT. Interestingly, (unweighted) MinSAT has been studied in the area of approximation algorithms (see e.g. [Avidor and Zwick, 2005; Kohli *et al.*, 1994; Marathe and Ravi, 1996] and the references therein).

The paper is structured as follows. Section 2 contains basic definitions about MinSAT and graphs. Section 3 describes the MinSAT solver we have implemented, including a description of its bounding techniques. Section 4 presents the benchmarks and solvers used in the empirical investigation, as well as a discussion of the experimental results. Section 5 concludes and points out future research directions.

2 Preliminaries

A literal is a propositional variable or a negated propositional variable. A clause is a disjunction of literals. A weighted

*This work is supported by French ANR UNLOC project: ANR-08-BLAN-0289-03, Spanish projects Consolider-Ingenio10 CSD2007-0022, TIN2010-20967-C04-01, SGU P. N. de Movilidad de Recursos Humanos, and GenCat 2009-SGR-1434, and National Natural Science Foundation of China (NSFC) grant No. 61070235.

clause is a pair (c, w) , where c is a clause and w , its weight, is a natural number or infinity. A clause is hard if its weight is infinity; otherwise it is soft. A Weighted Partial MinSAT (MaxSAT) instance is a multiset of weighted clauses $\phi = \{(h_1, \infty), \dots, (h_k, \infty), (c_1, w_1), \dots, (c_m, w_m)\}$, where the first k clauses are hard and the last m clauses are soft. For simplicity, in what follows, we omit infinity weights, and write $\phi = \{h_1, \dots, h_k, (c_1, w_1), \dots, (c_m, w_m)\}$.

A truth assignment is a mapping that assigns to each propositional variable either 0 or 1. The cost of a truth assignment I for ϕ is the sum of the weights of the clauses satisfied by I . The Weighted Partial MinSAT problem for an instance ϕ consists in finding an assignment with minimum cost that satisfies all the hard clauses (i.e., an optimal assignment), while the Weighted Partial MaxSAT problem consists in finding an assignment with maximum cost that satisfies all the hard clauses. The Weighted MinSAT (MaxSAT) problem is the Weighted Partial MinSAT (MaxSAT) problem when there are no hard clauses. The Partial MinSAT (MaxSAT) problem is the Weighted Partial MinSAT (MaxSAT) problem when all soft clauses have weight 1. The (Unweighted) MinSAT (MaxSAT) problem is the Partial MinSAT (MaxSAT) problem when there are no hard clauses. The SAT problem is the Partial MaxSAT problem when there are no soft clauses.

A *clique* in an undirected graph $G = (V, E)$, where V is the set of vertices and E is the set of edges, is a subset C of V such that, for every two vertices in C , there exists an edge connecting them. This is equivalent to saying that the subgraph induced by C is complete. A maximum clique is a clique of the largest possible size. The *maximum clique problem* (*MaxClique*) for an undirected graph G consists in finding a maximum clique in G . A *clique partition* for an undirected graph $G = (V, E)$ is a partition of V into disjoint subsets V_1, \dots, V_k such that, for $1 \leq i \leq k$, the subgraph induced by V_i is a complete graph. Let $\chi(G)$ be the minimum number of colors needed to color the vertices of G in such a way that adjacent vertices have different colors, and $\omega(G)$ the size of a maximum clique of G . G is perfect if $\chi(G') = \omega(G')$ for any induced subgraph G' of G .

3 An Exact MinSAT Solver

We define, to the best of our knowledge, the first genuine exact Weighted Partial MinSAT solver, called MinSatz, which was implemented from Satz [Li and Anbulagan, 1997] as MaxSatz [Li *et al.*, 2007]. For the sake of clarity, we start by presenting the (unweighted) Partial MinSAT case. MinSatz implements the branch-and-bound scheme, and the search space is formed by a tree representing all the possible interpretations. Contrarily to a branch-and-bound MaxSAT solver like MaxSatz, which solves a MaxSAT instance by minimizing the number of unsatisfied clauses, MinSatz solves a MinSAT instance by maximizing the number of unsatisfied clauses.

At every node, MinSatz starts by applying unit propagation using only hard unit clauses (i.e., given an existing or newly derived hard unit clause l , it satisfies and removes all the clauses containing the literal l , and removes all the occurrences of $\neg l$; soft unit clauses are not propagated because

they are not necessarily satisfied). If any hard clause becomes empty, MinSatz backtracks. Otherwise, it computes an upper bound of the maximum number of soft clauses that are falsified (UB) if the current partial assignment is extended to a complete one. This number UB is then compared with the number of clauses falsified in the best assignment found so far (LB). If $UB \leq LB$, a better solution cannot be found from the current node, and MinSatz backtracks. Otherwise, a variable is selected and instantiated. This process continues until all the search space has been explored, and MinSatz returns the best solution found. Algorithm 1 shows the pseudo-code of MinSatz.

Algorithm 1: MinSatz(ϕ, LB)

```

 $\phi \leftarrow \text{hardUnitPropagation}(\phi)$ ;
if  $\phi$  contains a hard empty clause then return -1;
if  $\phi = \{\}$  or  $\phi$  only contains empty soft clauses
then return #empty( $\phi$ );
 $UB \leftarrow \text{\#empty}(\phi) + \text{overestimation}(\phi)$ ;
if ( $UB \leq LB$ ) then return  $LB$ ;
 $x \leftarrow \text{select}(\phi)$ ;
 $LB \leftarrow \text{MinSatz}(\phi_x, LB)$ ;
 $LB \leftarrow \text{MinSatz}(\phi_{\neg x}, LB)$ ;
return  $LB$ .

```

We execute MinSatz($\phi, 0$) to solve an input instance ϕ . If MinSatz returns -1, the hard part of ϕ is unsatisfiable, and there is no feasible solution. Function #empty(ϕ) returns the number of empty soft clauses in ϕ , overestimation(ϕ) returns an overestimation of the number of soft clauses that will be unsatisfied if the current partial assignment is extended to a complete one, select(ϕ) returns the most occurring variable in ϕ . The instance ϕ_x ($\phi_{\neg x}$) is ϕ in which all clauses containing x ($\neg x$) are satisfied and removed, and the literal $\neg x$ (x) is removed from the remaining clauses. The MinSAT value of the input instance ϕ (i.e., the minimum number of satisfied clauses of ϕ) is #soft(ϕ) - MinSatz($\phi, 0$), where #soft(ϕ) is the number of soft clauses in ϕ .

We now describe how UB is computed. Assume that we are in a node of the search space and, after applying unit propagation using only hard unit clauses, we have an instance formed by the hard clauses $\{h_1, \dots, h_k\}$, and the not yet decided soft clauses $\{c_1, \dots, c_m\}$. We build an undirected graph $G = (V, E)$, where V contains a vertex for every clause in $\{c_1, \dots, c_m\}$, say $V = \{v_1, \dots, v_m\}$. We add an edge between vertex v_i and vertex v_j if clause c_i contains a literal l , and clause c_j contains $\neg l$. Moreover, we add an edge between vertex v_i , corresponding to clause $c_i = \{l_1^i, \dots, l_p^i\}$, and vertex v_j , corresponding to clause $c_j = \{l_1^j, \dots, l_q^j\}$, if the set of clauses $\{\neg l_1^i, \dots, \neg l_p^i, \neg l_1^j, \dots, \neg l_q^j, h_1, \dots, h_k\}$ may be declared to be unsatisfiable using unit propagation. The idea behind the graph G is that the soft clauses associated to the two vertices of an edge cannot be simultaneously falsified. In the first case, there is at least one satisfied soft clause because either l or $\neg l$ is satisfied by any assignment. In the second case, if the soft clauses associated to the two vertices of an edge are falsified, then a hard clause is falsified.

Once the graph G is built, a clique partition is created using the heuristic algorithm described in [Li and Quan, 2010b; 2010a]. By construction of G , there is at most one unsatisfied clause in every clique, and at least all the clauses in a clique except one are satisfied by any complete assignment. Therefore, the number of cliques in the partition, say s , is an overestimation of the number of soft clauses that may be falsified if the current partial assignment is completed. Taking into account this fact, we define $UB = e + s$, where e is the number of empty soft clauses in the current MinSAT instance.

Observe that the graph could also be defined in such a way that there is an edge between v_i and v_j if the set of clauses $\{-l_1^i, \dots, -l_p^i, -l_1^j, \dots, -l_q^j, h_1, \dots, h_k\}$ may be declared to be unsatisfiable using a complete SAT solver instead of unit propagation. This way, we could obtain better quality bounds. Nevertheless, for efficiency reasons, we restrict to contradictions that may be detected by unit propagation.

Example 1 Assume that we are in a node in which we have the hard clauses $\neg x_1 \vee \neg x_2, \neg x_2 \vee \neg x_3, \neg x_3 \vee \neg x_4, \neg x_4 \vee \neg x_5, \neg x_1 \vee \neg x_5$, and the soft clauses $\neg x_1, \neg x_2, \neg x_3, \neg x_4, \neg x_5$. Also assume that no clause has yet become empty. We build a graph G with vertices v_1, v_2, v_3, v_4, v_5 , where vertex v_i is associated with the soft clause $\neg x_i$, for $1 \leq i \leq 5$. The edges of G are $\{(v_1, v_2), (v_2, v_3), (v_3, v_4), (v_4, v_5), (v_1, v_5)\}$, corresponding to the graph in Figure 1. Assume that the algorithm finds the following clique partition of G : $\{\{v_1, v_2\}, \{v_3, v_4\}, \{v_5\}\}$. Then, at most 3 soft clauses among the 5 soft clauses can become empty, and $UB = 3$.

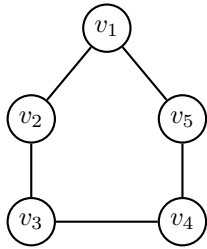


Figure 1: A simple imperfect graph ($\chi(G)=3$ and $\omega(G)=2$)

Nevertheless, the partition cannot give the optimal upper bound if G is not perfect, as the following example adapted from the one given in [Li and Quan, 2010b] shows.

The graph in Figure 1 can be partitioned into three cliques $\{(v_1, v_2), (v_3, v_4), (v_5)\}$, giving an upper bound 3 for the number of falsified clauses. However, a deep analysis can show that only two clauses can be falsified (instead of 3), so that the upper bound can be improved to 2. This analysis can be achieved by adapting the approach proposed in [Li and Quan, 2010b; 2010a] to transform the clique partition of G into a partial MaxSAT instance. Then, MaxSAT technologies can be used to improve the upper bound by performing a propositional reasoning to show that the three cliques cannot have each a falsified clause for any complete assignment. The reasoning is as follows. Assume that every clique contains a falsified clause under some complete assignment. Then v_5 should be falsified, but v_1 and v_4 cannot be falsified, because v_1 and v_4 are connected to v_5 . So, the only possibility for the

first and the second cliques to have a falsified clause is that v_2 and v_3 are both falsified, but this is impossible, because v_2 and v_3 are connected. So, $\{(v_1, v_2), (v_3, v_4), (v_5)\}$ is a subset of cliques in which not all cliques can have a falsified clause.

In the general case, we use MaxSAT technologies to detect subsets of cliques in which not all cliques can have a falsified clause in a similar way as in [Li and Quan, 2010b; 2010a]. Each of such subsets allows to improve the upper bound by one. We refer the reader to [Li and Quan, 2010b; 2010a] for more details of this approach. This detection takes $O(m^2)$ time, as the graph partition, for a graph of m soft clauses, so that the time complexity of the overestimation(ϕ) function is $O(m^2)$. The graph is stored in $O(m^2)$ space and the obtained cliques are stored in $O(m)$ space.

In the weighted case, where all weights are positive integers, we define the weight of a vertex v in G to be the weight of the corresponding soft clause c . Let \mathcal{P} be a set of cliques of G , where two cliques may share some vertices and each clique cl_{q_i} is associated with a weight $w_{cl_{q_i}}$. We call \mathcal{P} a *weighted clique partition* of G if for any vertex v of G , its weight equals $\sum_{v \in cl_{q_i}} w_{cl_{q_i}}$. By definition, each vertex of G belongs to at least one clique in \mathcal{P} .

Example 2 Refer to Figure 1, let the five weighted soft clauses corresponding to the five vertices be $\{(c_1, 2), (c_2, 3), (c_3, 4), (c_4, 5), (c_5, 6)\}$. $\mathcal{P}_1 = \{\{(v_1, v_2), 2\}, \{(v_3, v_4), 4\}, \{(v_5), 6\}, \{(v_2), 1\}, \{(v_4), 1\}\}$ and $\mathcal{P}_2 = \{\{(v_1, v_2), 2\}, \{(v_2, v_3), 1\}, \{(v_3, v_4), 3\}, \{(v_4, v_5), 2\}, \{(v_5), 4\}\}$ are both weighted clique partitions of the graph.

Given a MinSAT instance ϕ , a trivial upper bound for the sum of weights of unsatisfied soft clauses is $UB = \sum_{c_i \in \phi} w_i$, where w_i is the weight of the soft clause c_i . Each clique $\{(v_{i_1}, \dots, v_{i_k}), w\}$ of weight w allows to improve the trivial upper bound by $(k-1)w$, since the clique implies at least $k-1$ satisfied clauses. In other words, each clique allows the weight w of the $k-1$ soft clauses not to be counted in the upper bound. In Example 2, the weighted clique partition \mathcal{P}_1 gives an upper bound 14, and \mathcal{P}_2 gives an upper bound 12.

Generating a weighted clique partition of a weighted graph G consists in partitioning the weights of all vertices into a set of cliques of G . Algorithm 2 used in MinSatz presents such a partition. It first partitions G into cliques without considering the weights of vertices. Then it associates to each clique $\{v_{i_1}, \dots, v_{i_k}\}$ the weight $w = \min(w_{i_1}, \dots, w_{i_k})$, where w_{i_j} is the weight of v_{i_j} in G for $1 \leq j \leq k$, and modifies the weight of v_{i_j} in G to $w_{i_j} - w$. After each clique has a weight, a new graph G' is constructed from G by removing all vertices with weight 0. The process continues for G' . Note that if there are t cliques in the partition of G , G' has at least t vertices fewer than G .

In Example 2, \mathcal{P}_1 is obtained using Algorithm 2. Note that \mathcal{P}_2 is better than \mathcal{P}_1 . In fact, efficiently finding weighted clique partitions of better quality deserves future research.

We also use MaxSAT technologies to improve the upper bound given by the above partition in the same way as in the unweighted case. Every time we detect a subset of cliques in which not all cliques can have a falsified clause, we improve the upper bound by w , where w is the minimum weight among all the cliques in the subset.

Algorithm 2: partition(ϕ)

Construct a weighted graph G from ϕ ;
 $\mathcal{P} \leftarrow \{\}$;
repeat
 Find a clique partition of G , and add the cliques into \mathcal{P} ;
 Construct G' from the cliques and G ;
 $G \leftarrow G'$;
until G becomes empty
return \mathcal{P} .

4 Empirical Evaluation

We conducted five experiments to evaluate the performance and the usefulness of MinSatz. We used a Macpro with a 2.8Ghz intel Xeon processor and 4 Gb memory with MAC OS X 10.5, unless otherwise stated.

4.1 Benchmarks

MaxClique. The Partial MaxSAT encoding of MaxClique for a graph $G = (V, E)$ used in the MaxSAT Evaluation is as follows [Heras *et al.*, 2008]: There is one propositional variable associated to each vertex of V . Variable x_i is true iff vertex v_i belongs to the clique. For each pair v_i, v_j of non-adjacent vertices, there is a hard clause $\neg x_i \vee \neg x_j$. For each vertex v_i , there is a soft unit clause $(x_i, 1)$. Solving the resulting instance amounts to maximize the number of vertices belonging to the clique.

We obtain a Partial MinSAT encoding using the same hard clauses and, for each vertex v_i , we add a soft unit clause $(\neg x_i, 1)$. Solving the resulting instance amounts to minimize the number of vertices not belonging to the clique.

Combinatorial Auctions. Remind that a combinatorial auction is defined by a set of goods G , and a set of bidders that bid for indivisible subsets of goods. Each bid b_i is defined by the subset of requested goods $G_i \subseteq G$ and the amount of money offered. The auctioneer, who wants to maximize its revenue, must decide which bids are to be accepted, knowing that two bids sharing common goods cannot be jointly accepted.

The Weighted Partial MaxSAT encoding of combinatorial auctions used in the MaxSAT Evaluation is as follows [Heras *et al.*, 2008]: There is one propositional variable associated to each bid. Variable x_i is true iff the bid b_i is accepted. For each pair of bids (b_i, b_j) containing common goods, there is a hard clause $\neg x_i \vee \neg x_j$ indicating that both bids cannot be jointly accepted. For each bid b_i , there is a soft unit clause (x_i, w_i) indicating that there is a profit w_i if bid b_i is accepted. Solving the resulting instance amounts to maximize the profit.

We obtain a Partial MinSAT encoding using the same hard clauses and, for each bid b_i , we add a soft unit clause $(\neg x_i, w_i)$ indicating that there is a loss of profit w_i if bid b_i is not accepted. Solving the resulting instance amounts to minimize the loss of profit.

Min-3SAT. Remind that (Unweighted) Min-3SAT consists in solving a MinSAT instance whose clauses have exactly 3 literals. MinSAT may also be reduced to Partial MaxSAT. Three MaxSAT encodings (E1, E2, and E3) are presented in [Li *et*

al., 2010b], giving the first exact unweighted MinSAT solving approach by using a MaxSAT solver.

4.2 Solvers

We compared MinSatz with the following solvers in our empirical investigation :

- akmaxsat, akmaxsat_ls [Kuegel, 2010], IncMaxSatz [Lin *et al.*, 2008], MaxSatz, Wmaxsatz+ [Li *et al.*, 2007; 2010a]: We used the versions of these branch-and-bound Weighted Partial MaxSAT solvers that competed in the 2010 MaxSAT Evaluation.
- MaxCLQ [Li and Quan, 2010a] and MaxCliqueDyn (MCQDyn for short) [Konc and Janezic, 2007]: These are the two best specific MaxClique solvers in our knowledge.
- CASS [Fujishima *et al.*, 1999]: It is a state-of-the-art specific solver for combinatorial auctions.
- CPLEX: We used the version 8.0 of this well-known commercial linear integer programming solver.

4.3 Analysis of Empirical Results

In all the tables we will give the mean runtimes in seconds for the instances solved within a cutoff time, followed by the number of these solved instances (in brackets).

In the first experiment, we solved the random Min-3SAT instances from [Li *et al.*, 2010b], using a cutoff time of 3 hours as in [Li *et al.*, 2010b], and compared the performance of MinSatz with the Partial MaxSAT encodings proposed in [Li *et al.*, 2010b] for these instances on the same computer. Results are shown in Table 1. These instances were generated uniformly using 40 to 100 variables, and the number of clauses was determined by the clause-to-variable ratios (C/V) 4, 4.25 and 5. At each point, 50 instances were generated. The best runtimes of the three encodings were obtained by using MaxSatz [Li *et al.*, 2010b]. Results for the instances of 20 and 30 variables of Min-3SAT and the Min-2SAT instances from [Li *et al.*, 2010b] are not shown because they are easy for both MaxSatz using encoding E3 and MinSatz.

Table 1 indicates that a genuine MinSAT solver clearly outperforms the best performing MaxSAT solvers. Notice that MinSatz solved all the instances within the cutoff time, which is not the case for MaxSatz.

In the second experiment we solved the 96 random MaxClique instances from the the 2010 MaxSAT Evaluation¹ and the 66 DIMACS MaxClique instances (62 of them are also in the evaluation under the name: structured)², using a cutoff time of 1800 seconds as in the Evaluation. We compared MinSatz with the two best performing MaxSAT solvers on those instances: IncMaxSatz and akmaxsat, as well as with two of the best performing specific algorithms for MaxClique: MaxCLQ and MCQDyn. This experiment was conducted on a 3.33 Ghz intel core 2 duo CPU with Linux and 4 Gb memory, because the executables of IncMaxSatz and akmaxsat we obtained run under Linux. The results are shown in Table 2. MinSatz outperforms both MaxSAT solvers and

¹<http://www.maxsat.udl.cat/10/introduction/index.html>

²available at <http://cs.hbg.psu.edu/txn131/clique.html>

instance		MinSatz	MaxSatz		
#var	C/V		E1	E2	E3
40	4.00	0.01(50)	3.28(50)	507.8(50)	0.19(50)
50	4.00	0.03(50)	49.30(50)	- (0)	0.92(50)
60	4.00	0.12(50)	742.4(50)	- (0)	4.96(50)
70	4.00	0.41(50)	5735(34)	- (0)	23.21(50)
80	4.00	1.97(50)	- (0)	- (0)	100.7(50)
90	4.00	8.87(50)	- (0)	- (0)	381.5(50)
100	4.00	30.59(50)	- (0)	- (0)	1658(33)
40	4.25	0.01(50)	4.67(50)	992.5(50)	0.28(50)
50	4.25	0.04(50)	75.6(50)	- (0)	1.57(50)
60	4.25	0.14(50)	1153(50)	- (0)	8.31 (50)
70	4.25	0.69(50)	5989(5)	- (0)	42.77(50)
80	4.25	3.02(50)	- (0)	- (0)	186.3(50)
90	4.25	10.52(50)	- (0)	- (0)	760.4(50)
100	4.25	61.85(50)	- (0)	- (0)	2819(26)
40	5.00	0.02(50)	10.87(50)	5693(48)	0.80(50)
50	5.00	0.07(50)	226.8(50)	- (0)	5.24(50)
60	5.00	0.47(50)	3803(48)	- (0)	39.3(50)
70	5.00	1.59(50)	- (0)	- (0)	243.4(50)
80	5.00	14.68(50)	- (0)	- (0)	1512(50)
90	5.00	75.57(50)	- (0)	- (0)	5167(39)
100	5.00	380.72(50)	- (0)	- (0)	- (0)

Table 1: Mean runtimes in seconds for random Min-3SAT instances solved within 3 hours, followed by number of the solved instances (in brackets)

equals specific algorithms for MaxClique. Taking into account that MaxClique solvers have been investigated for a long time, these results provide evidence that reducing problems to MinSAT is a viable alternative for solving optimization problems.

instances	MinSatz	IncMaxsatz	akmaxsat	MaxCLQ	MCQDyn
random(96)	0.01(96)	1.6(96)	3(96)	0.01(96)	0.01(96)
DIMACS(66)	115(53)	88(39)	84(40)	69(54)	57(52)

Table 2: Mean runtimes (in seconds on a 3.33 Ghz intel core 2 duo under Linux) for MaxClique instances solved within 1800 seconds, followed by number of solved instances (in brackets)

In the third experiment we solved the combinatorial auction instances from the 2010 MaxSAT Evaluation. They were generated with the Combinatorial Auction Test Suite (CATS) [Leyton-Brown *et al.*, 2000], which is a random generator inspired from real-world scenarios. The used distributions were Paths (88 instances) and Scheduling (84 instances). We compared MinSatz with the results obtained in the 2010 MaxSAT Evaluation with akmaxsat, akmaxsat_ls, and Wmaxsatz+, the three best performing MaxSAT solvers on those instances. The cutoff time is 1800 seconds as in the Evaluation. Runtimes of akmaxsat, akmaxsat_ls, and Wmaxsatz+ are taken from the Evaluation which were obtained on a 2 GHz AMD Opteron with 512 MB memory under Linux. Results are shown in Table 3. Despite that our computer for MinSatz is slightly faster than the one used in the Evaluation, it is clear that MinSatz is several orders of magnitude faster than the MaxSAT solvers. Since those instances were very easy for MinSatz, we decided to compare MinSatz with specific algorithms for combinatorial auctions, reported in the literature, on harder instances.

instances	MinSatz	akmaxsat_ls	akmaxsat	Wmaxsatz+
path (88)	0.01(88)	22(88)	23(88)	192(71)
scheduling (84)	0.01(84)	267(75)	230(73)	63(83)

Table 3: Combinatorial Auctions (time in seconds)

In the fourth experiment, we compared the performance of MinSatz with CASS and CPLEX on the combinatorial auction benchmarks available at <http://people.cs.ubc.ca/~kevinlb/downloads.html>. These instances are generated with CATS 2.0 for 9 distributions (5 legacy and 4 real application distributions). We used the suite “variable problem size” with 40-400 goods, 50-2000 non-dominated bids. There are 800 instances in each distribution, but we arbitrarily only used the first 100 instances (numbered from 000001 to 000100). Results are shown in Table 4, using a cutoff time of 3600 seconds. The runtimes of CASS and CPLEX for these instances are taken from the same web page, which were obtained on a XEON 2.4 GHz. To make a fair comparison, we divide the taken runtimes of CASS and CPLEX by 1.17 (remind that we used a XEON 2.8 Ghz under Mac OS X). MinSatz is substantially faster than CASS, but slower than CPLEX. A more detailed analysis shows that MinSatz indeed solved a number of instances that CPLEX failed to solve in less than 3600 seconds. For example, MinSatz is better than CPLEX for the L7 distribution.

Instances	MinSatz	CASS	CPLEX
Arbitrary(100)	108(52)	133(39)	34(68)
L2(100)	0.01(100)	0.01(100)	1(100)
L3(100)	164(43)	165(32)	161(50)
L4(100)	131(52)	84 (46)	32(97)
L6(100)	170(44)	90(38)	0.01(100)
L7(100)	56(93)	6(88)	154(89)
Matching(100)	98(79)	148(31)	0.01(100)
Region(100)	186(50)	37(35)	17(96)
Scheduling(100)	112(82)	28(71)	2(100)
Total(900)	98(595)	44(480)	49(800)

Table 4: Hard Combinatorial Auctions (time in seconds)

Recall that CPLEX is a highly optimized and mature commercial IP solver, as well as its combinatorial auctions encoding. However, MinSatz is the first dedicated algorithm for MinSAT. We are confident that MinSatz can be significantly improved for combinatorial auctions by improving its upper bound using better weighted graph partitions and other techniques such as virtual arc consistency [Cooper *et al.*, 2010].

In the fifth experiment, we investigated the relationships between the MaxSAT value of a given random formula and its MinSAT value. This was our original motivation to study MinSAT. We generated uniform random 3SAT instances of 120 variables at the threshold (i.e. with 511 clauses), and used their MaxSAT value to partition them so that each partition contains all instances having the same MaxSAT value. Figure 2 shows that each of the 4 partitions has a significantly distinct Cumulative Distribution Function (CDF) shape, suggesting that the distribution of MinSAT values is distinct. Note that a point (x, y) in a curve means that the fraction

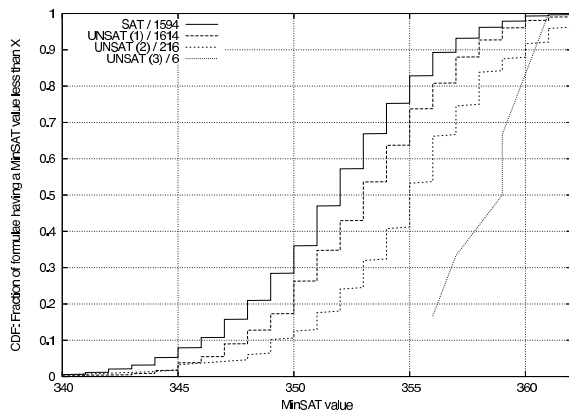


Figure 2: CDF plot for 511 clauses and 120 variables (3430 trials). The curves are respectively based on 1594 instances (MaxSAT=511, i.e. all the 511 clauses in each of these instances can be satisfied), 1614 instances (MaxSAT=510, i.e. 510 of the 511 clauses in each of these instances can be satisfied), 216 instances (MaxSAT=509) and only 6 instances for MaxSAT=508.

of instances having a MinSAT value $\leq x$ in the corresponding partition is y . In other words, y roughly corresponds to the probability of an instance with a given MaxSAT value to have a MinSAT value smaller or equal to a given number. In order to confirm that the plot reports statistically significant differences between partitions, we performed a pairwise two-sample Kolmogorov-Smirnov test between all couples of CDF. The Null hypothesis was rejected with high confidence.

What can be concluded here is somehow surprising. Random 3SAT formulae with higher MaxSAT values significantly tend to have smaller MinSAT values, i.e., their probability to have a MinSAT value smaller or equal to a given number is significantly higher than those formulae with lower MaxSAT values. We also confirmed this phenomenon at higher clause/variable ratio (up to $r = 8$) and also for formulae of 80 and 100 variables at different clause/variable ratios.

5 Concluding Remarks

We have investigated, for the first time, MinSAT formalisms from the problem solving viewpoint, and developed the first branch-and-bound solver for Weighted Partial MinSAT called MinSatz. Although MinSAT and MaxSAT are two natural extensions of SAT, the usefulness of MinSAT was not clear before our investigation. Despite that the MaxSAT and MinSAT encodings of the instances used in our investigation are similar, it turns out that the performance profile of MinSAT is extremely competitive w.r.t. MaxSAT. This is because our solver incorporates original bounding techniques combining graph partitioning and propositional reasoning, which are very different from the ones applied in MaxSAT solvers and very powerful to exploit graph structures. Our experiments also show that our approach is a viable alternative for optimization problems because it was able to beat specific solvers for Max-Clique and combinatorial auctions. The efficiency of MinSatz also allows us to investigate a correlation between the MinSAT and MaxSAT values of a random 3SAT instance.

As future work, we plan to improve the bounding tech-

niques using a better partition of a graph, and evaluate the performance of MinSatz on other combinatorial problems.

References

- [Avidor and Zwick, 2005] Adi Avidor and Uri Zwick. Approximating min 2-sat and min 3-sat. *Theory of Computing Systems*, 38(3):329–345, 2005.
- [Cooper et al., 2010] M. Cooper, S. de Givry, M. Sanchez, T. Schiex, M. Zytynicki, and T. Werner. Soft arc consistency revisited *Artificial Intelligence*, (7-8):449-478, 2010
- [Fujishima et al., 1999] Yuzo Fujishima, Kevin Leyton-Brown, and Yoav Shoham. Taming the computational complexity of combinatorial auctions: Optimal and approximate approaches. In *IJ-CAI*, pages 548–553, 1999.
- [Heras et al., 2008] Federico Heras, Javier Larrosa, Simon de Givry, and Thomas Schiex. 2006 and 2007 Max-SAT evaluations: Contributed instances. *JSAT*, 4(2-4):239–250, 2008.
- [Kohli et al., 1994] Rajeev Kohli, Ramesh Krishnamurti, and Prakash Mirchandani. The minimum satisfiability problem. *SIAM J. Discrete Mathematics*, 7(2):275–283, 1994.
- [Konc and Janezic, 2007] J. Konc and D. Janezic. An improved branch and bound algorithm for the maximum clique problem. *Communications in Mathematical and in Computer Chemistry*, 58:569–590, 2007.
- [Kuegel, 2010] Adrian Kuegel. Improved exact solver for the Weighted MAX-SAT problem. In *Workshop Pragmatics of SAT*, 2010.
- [Leyton-Brown et al., 2000] Kevin Leyton-Brown, Mark Pearson, and Yoav Shoham. Towards a universal test suite for combinatorial auction algorithms. In *ACM Conference on Electronic Commerce*, pages 66–76, 2000.
- [Li and Anbulagan, 1997] Chu Min Li and Anbulagan, Heuristics Based on Unit Propagation for Satisfiability Problems. In Proceedings of IJCAI, pages 366–371,
- [Li and Manyà, 2009] Chu Min Li and F. Manyà. MaxSAT, hard and soft constraints. In Armin Biere, Hans van Maaren, and Toby Walsh, editors, *Handbook of Satisfiability*, pages 613–631. IOS Press, 2009.
- [Li and Quan, 2010a] Chu Min Li and Zhe Quan. Combining graph structure exploitation and propositional reasoning for the maximum clique problem. In *ICTAI*, pages 344–351, 2010.
- [Li and Quan, 2010b] Chu Min Li and Zhe Quan. An efficient branch-and-bound algorithm based on maxsat for the maximum clique problem. In *AAAI*, pages 128–133, 2010.
- [Li et al., 2007] Chu Min Li, Felip Manyà, and Jordi Planes. New inference rules for Max-SAT. *Journal of Artificial Intelligence Research*, 30:321–359, 2007.
- [Li et al., 2010a] Chu Min Li, Felip Manyà, Nouredine Ould Mohamedou, and Jordi Planes. Resolution-based lower bounds in MaxSAT. *Constraints*, 15(4):456–484, 2010.
- [Li et al., 2010b] Chu Min Li, Felip Manyà, Zhe Quan, and Zhu Zhu. Exact MinSAT solving. In *SAT*, pages 363–368, 2010.
- [Lin et al., 2008] Han Lin, Kaile Su and Chu Min Li. Within-Problem Learning for Efficient Lower Bound Computation in Max-SAT Solving. In *AAAI*, pages 351–356, 2008.
- [Marathe and Ravi, 1996] M. V. Marathe and S. S. Ravi. On approximation algorithms for the minimum satisfiability problem. *Information Processing Letters*, 58:23–29, 1996.