Optimizing with minimum satisfiability

Chu Min Li\textsuperscript{a,b,*}, Zhu Zhu\textsuperscript{a}, Felip Manyà\textsuperscript{c}, Laurent Simon\textsuperscript{d}

\textsuperscript{a} MIS, Université de Picardie Jules Verne, 5 Rue du Moulin Neuf, 80000 Amiens, France
\textsuperscript{b} Huazhong University of Science and Technology, Wuhan, China
\textsuperscript{c} Artificial Intelligence Research Institute (IIIA), Spanish Scientific Research Council (CSIC), Campus UAB, 08193 Bellaterra, Spain
\textsuperscript{d} LRI/INRIA Futurs, Parc Club Université, Batiment G, 4, rue Jacques Monod, 91893 Orsay, France

\textbf{Abstract}

MinSAT is the problem of finding a truth assignment that minimizes the number of satisfied clauses in a CNF formula. When we distinguish between hard and soft clauses, and soft clauses have an associated weight, then the problem, called Weighted Partial MinSAT, consists in finding a truth assignment that satisfies all the hard clauses and minimizes the sum of weights of satisfied soft clauses. In this paper we describe a branch-and-bound solver for Weighted Partial MinSAT equipped with original upper bounds that exploit both clique partitioning algorithms and MaxSAT technology. Then, we report on an empirical investigation that shows that solving combinatorial optimization problems by reducing them to MinSAT is a competitive generic problem solving approach when solving MaxClique and combinatorial auction instances. Finally, we investigate an interesting correlation between the minimum number and the maximum number of satisfied clauses on random CNF formulae.

\textcopyright{} 2012 Elsevier B.V. All rights reserved.

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\cite{10} for solving practical optimization problems. Nowadays, MaxSAT formalisms are quite competitive on certain domains, and we believe that the development of new solving techniques and the annual celebration of a MaxSAT Evaluation\cite{1} will act as a driving force to incorporate MaxSAT technology in industrial environments.

In this paper we focus on MinSAT, which is close to MaxSAT but the goal now is to minimize the cost of satisfied clauses instead of maximizing 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 soft 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. Even when Weighted Partial MinSAT can be reduced to MinSAT, encoding optimization problems into Weighted Partial MinSAT produces more compact and expressive encodings. On the one hand,
we distinguish between constraints that are compulsory (hard) and constraints that can be relaxed (soft). On the other hand, we establish a priority among soft constraints by assigning them a weight that represents the significance of the constraint.

Our interest in MinSAT is motivated by the fact that minimization allows us to consider novel and powerful upper bounding techniques which cannot be applied to branch-and-bound MaxSAT solvers. At the same time, minimization problems admit more natural encodings into MinSAT. Indeed, although MinSAT and MaxSAT are both extensions of SAT, their solving techniques are quite different as well as complementary in the sense that problems that are beyond the reach of current MaxSAT solvers can be solved with MinSAT solvers, and vice versa.

Concretely, in this paper we first describe a new branch-and-bound solver for Weighted Partial MinSAT, called MinSat, equipped with original upper bounds that exploit both clique partitioning algorithms and MaxSAT technology. Then, we 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. Finally, we investigate an interesting correlation between the minimum number and the maximum number of satisfied clauses on random CNF formulae, which indicates that random formulae with higher MaxSAT values significantly tend to have smaller MinSAT values, and vice versa.

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 [17]: A number of encodings were defined to reduce (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. Recently, Kügel has proposed a new encoding from Weighted Partial MinSAT to Weighted Partial MaxSAT [8]. Kügel’s encoding also allows to encode Weighted Partial MaxSAT into Weighted Partial MinSAT, opening the door to solving Weighted Partial MinSAT with Weighted Partial MaxSAT solvers, and vice versa, although this approach is not yet competitive. Interestingly, (unweighted) MinSAT has been studied in the area of approximation algorithms (see e.g. [2,5,20] and the references therein).

The paper is structured as follows. In Section 2 we give some basic definitions that will be used in the rest of the paper. In Section 3 we describe the branch-and-bound solver we have developed, and explain in detail its bounding techniques. In Section 4 we report on the empirical evaluation we have conducted in order to evaluate our approach to MinSAT solving, and to investigate the correlation between the minimum number and the maximum number of satisfied clauses on random CNF formulae. We first describe the solved benchmarks, the executed solvers and the experiments performed, and then discuss the experimental results. In Section 5 we give the conclusions. The paper extends the results of [18] by defining and empirically evaluating two new upper bounding methods, called UB2 and UB3 in the text, and providing further details of MinSat and more experimental results.

2. Preliminaries

A literal is a propositional variable or a negated propositional variable. A clause is a disjunction of literals. A weighted 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 \( \phi = \{ (h_1, \infty), \ldots, (h_k, \infty), (c_1, w_1), \ldots, (c_m, w_m) \} \), 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 \( \phi = \{ h_1, \ldots, h_k, (c_1, w_1), \ldots, (c_m, w_m) \} \). Notice that a soft clause \( (c, w) \) is equivalent to having \( w \) copies of the clause \( (c, 1) \), and that \( \{ (c, w_1), (c, w_2) \} \) is equivalent to \( (c, w_1 + w_2) \).

A truth assignment is a mapping that assigns to each propositional variable either 0 or 1. The cost of a truth assignment \( I \) for \( \phi \) is the sum of the weights of the soft clauses satisfied by \( I \). The Weighted Partial MinSAT problem for an instance \( \phi \) 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 MinSAT or 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 \( \pi \) for an undirected graph \( G = (V, E) \) is a partition of \( V \) into disjoint subsets \( V_1, \ldots, V_k \) 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 \( G \). \( \chi(G) \) is usually known as the chromatic number of \( G \).
3. Branch-and-bound MinSAT solvers

We first present a branch-and-bound solver for the (unweighted) Partial MinSAT case, and explain how a good quality upper bound (UB) can be computed. Then, we extend the solver, and especially its UB, to Weighted Partial MinSAT. Our UB methods are based on a clique partition of a graph and on MaxSAT reasoning.

3.1. MinSatz for (unweighted) partial MinSAT

Our solver implements the branch-and-bound scheme, and the search space is formed by a tree representing all the possible truth assignments. At every node, the solver 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 ¬l; soft unit clauses are not propagated because the simplified instance might have a different minimum number of satisfied clauses). If any hard clause becomes empty, then the solver backtracks. Otherwise, it computes an upper bound of the maximum number of soft clauses that will be 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 explored, and the solver returns the best solution found. Algorithm 1 shows the pseudo-code of our Partial MinSAT solver.

Algorithm 1 MinSatz(φ, LB)

\[
\begin{aligned}
\phi & \leftarrow \text{hardUnitPropagation}(\phi); \\
\text{if } \phi \text{ contains a hard empty clause then return } \text{LB}; \\
\text{if } \phi = \emptyset \text{ or } \phi \text{ only contains empty clauses then return } \text{max}(\#\text{empty}(\phi), \text{LB}); \\
\text{UB} & \leftarrow \#\text{empty}(\phi) + \text{overestimation}(\phi); \\
\text{if } (\text{UB} \leq \text{LB}) \text{ then return } \text{LB}; \\
x & \leftarrow \text{select}(\phi); \\
\text{LB} & \leftarrow \text{MinSatz}(\phi_x, \text{LB}); \\
\text{LB} & \leftarrow \text{MinSatz}(\phi_{\neg x}, \text{LB}); \\
\text{return } \text{LB}.
\end{aligned}
\]

For solving an instance \( \phi \), we should call Algorithm 1 with the following parameters: MinSatz(\( \phi, -1 \)). If the algorithm returns \(-1\), then the hard part of \( \phi \) is unsatisfiable, and there is no feasible solution. Function \#\text{empty}(\phi) returns the number of empty soft clauses in \( \phi \), overestimation(\( \phi \)) returns an overestimation of the maximum number of soft clauses that will be falsified if the current partial assignment is extended to a complete one, and select(\( \phi \)) implements the following variable selection heuristic: Let hard(l) (soft(l)) be the number of occurrences of literal \( l \) in hard (soft) clauses, and let score(l) = 2 × hard(l) + soft(l). Function select(\( \phi \)) chooses a variable \( x \) with the highest value of score(\( x \) × score(\( \neg x \) + score(\( \neg x \)) is removed from the remaining clauses. The MinSAT value for the input instance \( \phi \), i.e., the minimum number of satisfied clauses of \( \phi \) is \#soft(\( \phi \))−MinSatz(\( \phi, -1 \)), where \#soft(\( \phi \)) is the number of soft clauses in \( \phi \).

It is worth noticing that a branch-and-bound MaxSAT solver like MaxSatz [15] solves a MaxSAT instance by minimizing the number of falsified clauses, and that our MinSAT solver solves a MinSAT instance by maximizing the number of falsified clauses.

3.2. UB computation for (unweighted) partial MinSAT

A decisive aspect for obtaining fast MinSAT solvers is to equip them with good quality UBs. In this section, we describe original UB computation methods, which are based on first computing a clique partition of the graph built from the current MinSAT instance, and then improving the obtained UB with MaxSAT technology. We describe how UB is computed in the unweighted case in this subsection, and then describe three different methods for computing UB in the weighted case in the next subsection.

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 \( \phi \) formed by the hard clauses \([h_1, \ldots, h_k]\), e empty soft clauses, and the not yet decided soft clauses \([c_1, \ldots, c_m]\). We start by building an undirected graph \( G = (V, E) \), where \( V \) contains an element for every soft clause in \([c_1, \ldots, c_m]\), say \( V = \{v_1, \ldots, v_m\} \). We add an edge between vertex \( v_i \), corresponding to clause \( c_i = [l_{i1}, \ldots, l_{ik}] \), and vertex \( v_j \), corresponding to clause \( c_j = [l_{j1}, \ldots, l_{jk}] \), if there exist a literal \( l_i \) and a literal \( l_j \) such that \( l_i = \neg l_j \), or the set of clauses \([\neg l_{i1}, \ldots, \neg l_{ip}], [\neg l_{j1}, \ldots, \neg l_{jq}] \) may be declared to be unsatisfiable using unit propagation. The idea behind the graph \( G \), called the graph associated to \( \phi \), is that the clauses associated to the two vertices of an edge cannot be simultaneously falsified. Indeed, in the former case, two literals in \( c_i \) and \( c_j \) are complementary, and in the latter case, if \( c_i \) and \( c_j \) are both falsified, then a hard clause is violated.
Once the graph $G$ is built, an overestimation of the maximum number of soft clauses that can be falsified corresponds to a maximum independent set (MIS) of $G$, where there exists no edge between any two vertices of the independent set. We could then compute an UB of the maximum number of soft clauses that can be falsified by computing an UB of the cardinality of a MIS of $G$. For this purpose, we create a clique partition of $G$ using the heuristic algorithm described in [22], and also used in [11,12]: Suppose that the vertices are sorted by increasing order of their degree and that the current partition is $S_1, S_2, \ldots, S_k$ (in this order, and being $k = 0$ at the beginning). The algorithm inserts the first vertex $v$ of the sequence of vertices into the first $S_i$ such that $v$ is connected to all the vertices already in $S_i$. If such $S_i$ does not exist, a new set $S_{k+1}$ is created and $v$ is inserted in $S_{k+1}$. This process is repeated until there are no more vertices left.

By construction of graph $G$, there is at most one falsified clause in every clique. In other words, at least all the clauses in a clique except one are satisfied by any complete truth assignment satisfying all the hard clauses. Hence, the number of cliques in the partition, say $s$, is an overestimation of the number of clauses that can be falsified if the current partial assignment is completed. Taking into account this fact, we define $UB = e + s$. However, $UB$ cannot be tight enough. If $G$ is not perfect, then $UB$ is not tight because the number of cliques in the partition of $G$ is an upper bound of the chromatic number $C(G)$ of the complementary graph of $G$ ($\bar{G}$), which is strictly larger than the cardinality of a MIS of $G$. If $G$ is perfect, then $UB$ can be tight, but it is not guaranteed.

In order to improve $UB$, we use an approach adapted from [11,12]. We derive a Partial MaxSAT instance $\psi$ from the obtained clique partition of graph $G$: For every edge $(v_i, v_j)$ of the graph, we add the hard clause $\neg v_i \lor \neg v_j$ and, for every clique $\{v_{i1}, \ldots, v_{ih}\}$, we add the soft clause $v_{i1} \lor \cdots \lor v_{ih}$. Hard clauses in $\psi$ state that clauses of the original MinSAT instance $\phi$ associated with adjacent vertices of $G$ cannot be simultaneously falsified, while soft clauses in $\psi$ state that at least one soft clause of $\phi$ associated with the vertices of a clique of $G$ is falsified. It turns out that if an optimal solution of the resulting Partial MaxSAT instance $\psi$ has $u$ falsified soft clauses, then we can decrement $UB$ by $u$, because we can conclude that $u$ clauses cannot contain any falsified soft clause in $\phi$ and should not be counted in the UB of the number of falsified soft clauses in $\phi$. Observe that $v_{ih}$ is true if clause $c_{ih}$ in $\phi$ is falsified, and false if it is satisfied. Hence, if a soft clause $v_{i1} \lor \cdots \lor v_{ih}$ is violated, all the clauses of $\phi$ associated with the clique $\{v_{i1}, \ldots, v_{ih}\}$ are satisfied; in other words, that clique does not contain any falsified soft clause.

Since solving Partial MaxSAT is NP-hard, in practice we underestimate $u$ by using the technology developed for MaxSAT solvers. More precisely, we apply lower bound $UB$ enhanced with failed literal detection [13,14] to the derived Partial MaxSAT instance $\psi$. Let us first explain UP, and then how it can be enhanced with failed literal detection.

The underestimation provided by UP [13] is the number of disjoint subsets of soft clauses that, when added to the hard clauses, can be declared to be unsatisfiable by applying unit propagation. UP works as follows: it applies unit propagation to $\psi$ (soft clauses included) until a contradiction is detected. Then, UP identifies, by inspecting the implication graph created by unit propagation [15], a subset of clauses from which a unit refutation can be constructed, and tries to derive new contradictions in the formula resulting of removing the soft clauses involved in that refutation. UP terminates when no more contradictions can be detected.

UP can be enhanced with failed literal detection (UP$_F$) as follows [14]: Given an instance $\psi$ to which we have already applied UP, and a variable $v$, UP$_F$ applies UP to $\psi \land v$ and $\psi \land \neg v$, where $v$ and $\neg v$ are added as hard unit clauses. If UP derives a contradiction from both $\psi \land v$ and $\psi \land \neg v$, then the union of the soft clauses involved in the refutation of $\psi \land v$ and the soft clauses involved in the refutation of $\psi \land \neg v$, when added to the hard clauses of $\psi$, form an unsatisfiable subset of clauses from which a refutation can be constructed. As before, the soft clauses involved in that refutation are removed, and UP$_F$ tries to derive new contradictions in the simplified formula. This process is repeated until no more contradictions can be detected. Since applying failed literal detection to every variable is time consuming, it is applied to a reduced number of variables in practice.

In our MinSAT setting, UB is improved by decrementing the total number of contradictions detected with UP$_F$. 

Example 1. Assume that we are in a node in which we have the hard clauses $\neg x_1 \lor \neg x_2, \neg x_2 \lor \neg x_3, \neg x_3 \lor \neg x_4, \neg x_4 \lor \neg x_5, \neg x_1 \lor \neg x_3$, 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 the graph $G$ associated with the instance, which is shown in Fig. 1. The set of vertices is $\{v_1, v_2, v_3, v_4, v_5\}$, where vertex $v_1$ is associated with the soft clause $\neg x_1$, for $1 \leq i \leq 5$. The set of edges is $\{(v_1, v_2), (v_2, v_3), (v_3, v_4), (v_4, v_3), (v_1, v_3)\}$. 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 5 soft clauses can be falsified, and $UB = 3$.

$G$ is not perfect and, therefore, $UB$ is not tight. A deeper analysis shows that only two clauses of the instance can be falsified (instead of 3), so that $UB$ might be improved to 2. 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 not possible, 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. Therefore, we could decrement UB by one. In order to detect such a situation, we derive the Partial MaxSAT instance formed by the hard clauses:

\[-v_1 \lor \neg v_2, \neg v_1 \lor \neg v_5, \neg v_2 \lor \neg v_3, \neg v_3 \lor \neg v_4, \neg v_4 \lor \neg v_5\]
and the soft clauses:
\[ v_1 \lor v_2 \lor v_3 \lor v_4 \lor v_5 \]

Then, we apply UP_{PL}. Actually, we detect a contradiction applying just UP to the above instance. It corresponds to the refutation that can be constructed from \{¬v_1 \lor ¬v_5, ¬v_2 \lor ¬v_3, ¬v_4 \lor ¬v_5, v_1 \lor v_2, v_3 \lor v_4, v_5\}. Since all the soft clauses are involved in the refutation, we cannot derive additional contradictions. Therefore, we decrement UB by one, and get UB = 2.

### 3.3. Extending MinSatz and its UB to weighted partial MinSAT

In the weighted case, the objective is to find an assignment that satisfies all the hard clauses and minimizes the sum of weights of satisfied soft clauses. To reach this objective, our solver will find an assignment that satisfies all the hard clauses and maximizes the sum of weights of falsified soft clauses. Now, in Algorithm 1, #empty(φ) returns the sum of weights of all the empty soft clauses in φ, overestimation(φ) returns an overestimation of the maximum sum of weights of the soft clauses that will be falsified if the current partial assignment is completed, soft(φ) is the total sum of the weights of the soft clauses in φ, and the MinSAT value for the input instance φ is soft(φ)-MinSat(φ, -1).

The variable selection heuristic works as follows in the weighted case: Let hard(l) be the number of occurrences of literal l in hard clauses, let soft(l) be the sum of weights of the soft clauses containing literal l, let meanWeight be the mean of the weights among all the soft clauses, and let score(l) = 2 \times meanWeight \times hard(l) + soft(l). Function select(φ) choses a variable with the highest value of score(x) \times score(¬x) \times meanWeight + score(x) + score(¬x). This heuristic is also empirical.

In the rest of the section we explain how the computation method described above for UB can be extended to the weighted case. We obtain three different methods, and will refer to them as UB1, UB2 and UB3. In all of them, the graph associated to the MinSAT instance is built as in the unweighted case. The difference lies in that now the graph is weighted, and in how the clique partition is created and the weights are operated.

We define the weight of a vertex of G as the weight of the corresponding soft clause, and the weight w of a clique \{v_1, ..., v_n\} as the minimum weight among the weighted soft clauses (c_1, w_1), ..., (c_n, w_n) (i.e., w = min(w_1, ..., w_n)).

#### 3.3.1. Upper bound UB1

UB1 creates a clique partition in the graph G associated to the current MinSAT instance using the same heuristic algorithm as in the unweighted case. Let (v_i, w_i) be a vertex in a clique of weight w. Then, it constructs a subgraph G' induced by those vertices v_i of G such that w_i - w > 0, and defines the weight of v_i in G' as w_i - w. G' is in turn partitioned into cliques, and a subgraph of G' is constructed in the same way for finding further partitions. This process, whose pseudo-code is shown in Algorithm 2, continues until the empty graph is obtained. Finally, UB1 is computed taking into account the weights of all the obtained cliques: \[ UB1 = \sum_{\text{clique } c_i \text{ is empty} } w_i + \sum_{\text{all cliques } w} \]

**Algorithm 2** Partition(φ)

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

Algorithm 2 returns a partition of G into cliques, in such a way that all vertices in each clique have the same weight in order to compute a tight upper bound UB1. Observe that there are many ways to partition a weighted graph into cliques in which all vertices have the same weight. Some of them risk to give a huge number of cliques, especially when the vertex...
weights are very different and the graph is large. The approach presented in Algorithm 2 does not necessarily give the best upper bound, but guarantees that the number of vertices in $G'$ is at most $|V| - s$, where $V$ is the set of vertices of $G$ and $s$ is the number of cliques in its clique partition, because at least one vertex in each clique has the same weight as the clique before the subtraction and cannot belong to $G'$, which seems a reasonable approach in the implementation of a MinSAT solver.

As in the unweighted case, UP enhanced with failed literal detection is applied to improve the upper bound. Now, for every clique $\{v_1, \ldots, v_k\}$, we add the weighted soft clause $(v_1 \lor \cdots \lor v_k, w)$, being $w$ the weight of the clique. Every time we detect a subset of cliques in which not all cliques can have falsified clauses, we improve the upper bound by $w$, where $w$ is the minimum weight among all the cliques in the subset.

Example 2. Consider the MinSAT instance formed by the hard clauses $\neg x_1 \lor \neg x_2, \neg x_2 \lor \neg x_3, \neg x_3 \lor \neg x_4, \neg x_4 \lor \neg x_5, \neg x_5$, and the soft clauses $(\neg x_1, 2), (\neg x_2, 3), (\neg x_3, 4), (\neg x_4, 5), (\neg x_5, 6)$. The graph associated to that instance is shown in Fig. 1. Algorithm 2 returns a weighted clique partition $\{(v_1, v_2), (v_3, v_4), (v_5), (v_6), (v_2), (v_4), (v_1)\}$, getting $UB_1 = 14$.

To improve $UB_1$, $UP_{FL}$ is applied to the following Weighted Partial MaxSAT instance:

$$
\neg v_1 \lor \neg v_2 \quad \neg v_1 \lor \neg v_5 \quad \neg v_2 \lor \neg v_3 \\
\neg v_2 \lor \neg v_4 \quad \neg v_3 \lor \neg v_5 \\
(v_1 \lor v_2, 2) \quad (v_3 \lor v_4, 4) \quad (v_2, 1) \\
(v_4, 1) \quad (v_5, 6)
$$

As a result, the unsatisfiable subset $\neg v_1 \lor \neg v_5, \neg v_2 \lor \neg v_3, \neg v_4 \lor \neg v_5, (v_1 \lor v_2, 2), (v_3 \lor v_4, 4), (v_5, 6)$ is detected. We decrement $UB_1$ by 2 because it is the minimum among 2, 4, and 6. Therefore, $UB_1 = 12$.

Notice that, depending on the order in which unit clauses were selected by $UP_{FL}$, $UB_1$ could instead detect the unsatisfiable subset $\neg v_4 \lor \neg v_5, (v_4, 1), (v_5, 6)$. In this case, we could decrement $UB_1$ just by 1.

3.3.2. Upper bound $UB_2$

$UB_2$ is an upper bound that builds a clique partition in an incremental way: It starts by selecting a vertex $v$ with minimum degree, breaking ties by selecting a vertex with minimum weight. Then, it computes a maximal clique containing $v$ by considering vertices in increasing order of their degree, and decrements the weight of every vertex of $G$ in the computed clique by the weight of the clique. It removes the vertices with weight zero (so, it removes at least one vertex), and repeats this process until the empty graph is derived. The sum of the weights of all the computed cliques is $UB_2$. Finally, $UB_2$ is improved by applying $UP_{FL}$ to a Weighted Partial MaxSAT instance as in $UB_1$.

Example 3. We consider the same MinSAT instance as in Example 2. In the first step, $UB_2$ selects vertex $v_1$ and finds the clique $\{v_1, v_2\}$ with weight 2. Then, it removes vertex $v_1$ from the graph and the weight of vertex $v_2$ is set to 1. In the second step, $UB_2$ selects vertex $v_2$ and finds the clique $\{v_2, v_3\}$ with weight 1. Then, it removes vertex $v_2$ from the graph and the weight of vertex $v_3$ is set to 3. In the third step, $UB_2$ selects vertex $v_3$ and finds the clique $\{v_3, v_4\}$ with weight 3. Then, it removes vertex $v_3$ from the graph and the weight of vertex $v_4$ is set to 2. In the fourth step, $UB_2$ selects vertex $v_4$ and finds the clique $\{v_4, v_5\}$ with weight 2. Then, it removes vertex $v_4$ from the graph and the weight of vertex $v_5$ is set to 4. In the fifth step, $UB_2$ selects vertex $v_5$ and finds the clique $\{v_5\}$ with weight 4. Then, it removes vertex $v_5$, gets the empty graph and $UB_2 = 12$.

To improve $UB_2$, $UP_{FL}$ is applied to the following Weighted Partial MaxSAT instance:

$$
\neg v_1 \lor \neg v_2 \\
\neg v_3 \lor \neg v_4 \\
(v_1 \lor v_2, 2) \quad (v_2 \lor v_3, 1) \quad (v_3 \lor v_4, 3) \\
(v_4 \lor v_5, 2) \quad (v_5, 4)
$$

As a result, it is detected that the soft clauses $\{v_1 \lor v_2, 2\}, (v_3 \lor v_4, 3\}, (v_5, 4)$, when added to the hard clauses, are unsatisfiable. We decrement $UB_2$ by 2 because it is the minimum among 2, 3, and 4. Therefore, $UB_2 = 10$. Notice that $UB_2$ is tighter than $UB_1$ for the analyzed MinSAT instance. When we remove the soft clauses $\{v_1 \lor v_2, 2\}, (v_3 \lor v_4, 3\}, (v_5, 4)$, we cannot detect any other contradiction. Actually, 10 is the optimal $ UB$ value for the present MinSAT instance.

It is worth noticing that $UB_2$ applied to an unweighted graph is slower than $UB_1$, because $UB_2$ computes a clique at a time while $UB_1$ can compute several cliques at a time (in a single iteration). However, in the weighted case, several
iterations are necessary to partition the graph, and the remaining graph $G'$ after an iteration becomes less connected in
UB1 than in UB2, because more vertices are removed (at least one per clique), so that cliques partitioning $G'$ become
smaller than in UB2. Therefore, UB2 should perform better than UB1 for weighted graphs, because larger cliques produce
better upper bounds. Refer to Example 2 and Example 3, there are three cliques of cardinality 1 in UB1, two of which are
from $G'$, giving $UB1 = 14$ before MaxSAT reasoning, while there is only one clique of cardinality 1 in UB2, getting $UB2 = 12$.

### 3.3.3. Upper bound UB3

UB3 can be seen as an upper bound that improves UB2. It computes incrementally a clique partition as UB2, and the
difference lies in the fact that, in the subsets of soft clauses detected by UPFL, the weights of vertices are taken into account,
in the sense that the cliques with weight greater than the minimum weight can be used to detect additional contradictions.
Of course, we must update the weights of cliques and generate new weighted soft clauses after detecting a contradiction,
which is the overhead of UB3. We illustrate UB3 using another MinSAT instance because UB2 is optimal for the instance in
Example 2.

**Example 4.** Consider the MinSAT instance formed by the hard clauses $\neg x_1 \lor \neg x_2$, $\neg x_2 \lor \neg x_3$, $\neg x_3 \lor \neg x_4$, $\neg x_4 \lor \neg x_5$, $\neg x_1 \lor \neg x_5$, $\neg x_5 \lor \neg x_6$, $\neg x_6 \lor \neg x_7$, $\neg x_7 \lor \neg x_8$, $\neg x_8 \lor \neg x_9$, $\neg x_9 \lor \neg x_3$ and the soft clauses $(\neg x_1, 2)$, $(\neg x_2, 3)$, $(\neg x_3, 4)$, $(\neg x_4, 5)$, $(\neg x_5, 9)$, $(\neg x_6, 3)$, $(\neg x_7, 1)$, $(\neg x_8, 3)$, $(\neg x_9, 3)$. The graph associated to that instance is shown in Fig. 2.

In the first step, UB3 selects vertex $v_7$ and finds the clique $\{v_6, v_7\}$ with weight 1. Then, it removes vertex $v_7$ from the
graph and the weight of vertex $v_6$ is set to 2. In the second step, UB3 selects vertex $v_6$ and finds the clique $\{v_5, v_6\}$ with
weight 2. Then, it removes vertex $v_6$ from the graph and the weight of vertex $v_5$ is set to 7. In the third step, UB3 selects
vertex $v_8$ and finds the clique $\{v_8, v_9\}$ with weight 3. Then, it removes vertices $v_8$ and $v_9$. In the fourth step, UB3 selects
vertex $v_1$ and finds the clique $\{v_1, v_2\}$ with weight 2. Then, it removes vertex $v_1$ from the graph and the weight of vertex $v_2$ is set to 1. In the fifth step, UB3 selects vertex $v_2$ and finds the clique $\{v_2, v_3\}$ with weight 1. Then, it removes vertex $v_2$ from the
graph and the weight of vertex $v_3$ is set to 3. In the sixth step, UB3 selects vertex $v_3$ and finds the clique $\{v_3, v_4\}$ with
weight 3. Then, it removes vertex $v_3$ from the graph and the weight of vertex $v_4$ is set to 2. In the seventh step, UB3 selects
vertex $v_4$ and finds the clique $\{v_4, v_5\}$ with weight 2. Then, it removes vertex $v_4$ from the graph and the weight of vertex $v_5$ is set to 5. In the eighth step, UB3 selects vertex $v_5$ and finds the clique $\{v_5\}$ with weight 5. Then, it removes vertex $v_5$, gets the empty graph and $UB3 = 19$.

To improve UB3, UPFL is applied to the following Weighted Partial MaxSAT instance:

\[
\begin{align*}
\neg v_1 & \lor \neg v_2 \\
\neg v_3 & \lor \neg v_4 \\
\neg v_5 & \lor \neg v_9 \\
\neg v_8 & \lor \neg v_9 \\
(v_1 \lor v_2, 2) & \quad (v_2 \lor v_3, 1) \quad (v_3 \lor v_4, 3) \\
(v_4 \lor v_5, 2) & \quad (v_5, 5) \quad (v_5 \lor v_6, 2) \\
(v_6 \lor v_7, 1) & \quad (v_8 \lor v_9, 3)
\end{align*}
\]

UB3 detects that the soft clauses $(v_1 \lor v_2, 2)$, $(v_3 \lor v_4, 3)$, $(v_5, 5)$, when added to the hard clauses, are unsatisfiable. Now, it removes $(v_1 \lor v_2, 2)$, and replaces $(v_3 \lor v_4, 3)$ with $(v_3 \lor v_4, 1)$ and replaces $(v_5, 5)$ with $(v_5, 3)$. Now, UB3 detects that the soft clauses $(v_6 \lor v_7, 1)$, $(v_8 \lor v_9, 3)$, $(v_5, 3)$, when added to the hard clauses, are unsatisfiable. It removes $(v_6 \lor v_7, 1)$, and replaces $(v_8 \lor v_9, 3)$ with $(v_8 \lor v_9, 2)$ and replaces $(v_5, 3)$ with $(v_5, 2)$. Since no additional contradictions can be detected, $UB3 = 16$. Notice that $UB2 = 17$ for this instance because after detecting the first contradiction the unit soft clause containing $v_5$ is removed. UB2 does not update the weights as UB3.

![Fig. 2. Graph associated to the MinSAT instance of Example 4.](image-url)
4. Experimental results

We conducted four experiments to evaluate the performance and the usefulness of our MinSAT solver for combinatorial optimization, and one experiment to investigate the correlation between the minimum number and the maximum number of satisfied clauses on random CNF formulae.

4.1. Benchmarks

MaxClique. MaxClique is a representative NP-hard combinatorial optimization problem appearing in many real applications. The Partial MaxSAT encoding of MaxClique for a graph \( G = (V, E) \) used in the MaxSAT Evaluation is as follows [4]: There is one propositional variable associated to each vertex of \( V \). Variable \( x_i \) is true if vertex \( v_i \) belongs to the clique; otherwise, it is false. For each pair \((v_i, v_j)\) of non-adjacent vertices, there is a hard clause \( \neg x_i \lor \neg x_j \). For each vertex \( v_i \), there is a soft unit clause \( x_i \). 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 \( \neg x_i \). Solving the resulting instance amounts to minimize the number of vertices not belonging to the clique. Treating each propositional variable \( x_i \) as a binary variable, MaxClique can be formulated as an integer programming problem. The edge formulation asks to maximize the sum of all binary variables subject to \( x_i + x_j \leq 1 \) for each pair \((v_i, v_j)\) of non-adjacent vertices.

We used standard DIMACS Maxclique instances in our experimentation. They are widely used in the literature to evaluate Maxclique algorithms.

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 his 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 [4]: There is one propositional variable associated to each bid. Variable \( x_i \) is true if the bid \( b_i \) is accepted; otherwise, it is false. For each pair of bids \((b_i, b_j)\) containing common goods, there is a hard clause \( \neg x_i \lor \neg x_j \) 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. Treating each propositional variable \( x_i \) as a binary variable, combinatorial auctions can be formulated as an integer programming problem. The usual formulation asks to maximize the sum of all \( w_i x_i \) subject to \( x_k_1 + \cdots + x_k_t \leq 1 \) for each good \( g_k \) such that \( b_i_1, \ldots, b_i_t \) are all bids containing good \( g_k \).

The combinatorial auction problem is also an important combinatorial optimization problem appearing in many applications such as resource allocation and e-commerce. Differently from Maxclique, the soft clauses in combinatorial auction are weighted. We also used standard benchmarks of combinatorial auctions in the literature in our experimentation.

Random 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 [17], giving the first exact unweighted MinSAT solving approach by using a MaxSAT solver.

Random 3SAT is one of the most studied subproblems of SAT because of its hardness and simplicity. The understanding of random 3SAT helped the study of the complexity theory. Minimizing and maximizing the number of satisfied clauses in random 3SAT allow different points of view to this problem.

4.2. Solvers

The solvers used in our empirical investigation are:

- akmaxsat, akmaxsat_ls [7], IncMaxSat [19], MaxSat, Wmaxsatz+ [15,16]: We used the versions of these branch-and-bound Weighted Partial MaxSAT solvers used in the 2011 MaxSAT Evaluation.
- MaxCLQ [11] and MaxCliqueDyn (MCQDyn for short) [6]: These are the two best specific MaxClique solvers to our knowledge.
- CASS [3]: It is a state-of-the-art specific solver for combinatorial auctions.
- CPLEX: We used the last version 12.2 of this well-known commercial linear integer programming solver.
- MinSat: The implementation of our Weighted Partial MinSAT solver used in [18]. It implements lower bound UB1 in an earlier version of our solver which is not as optimized as the current version implementing UB1.
- MinSat(UB1): The implementation of our Weighted Partial MinSAT solver with UB1.
- MinSat(UB2): The implementation of our Weighted Partial MinSAT solver with UB2.
were solved (in brackets).

4.3. Analysis of empirical results for combinatorial optimization

We compared MaxSatz runtimes with our MinSAT solver in Table 1, but do not show results for the Min-2SAT instances.

All the solvers are executed on a 3.33 GHz Intel core 2 duo CPU with Linux and 4 GB memory, unless otherwise stated.

The test instances of each benchmark problem were divided into subsets. In all the tables, we will give the mean runtimes in seconds for the instances solved within a cutoff time in each subset, followed by the number of instances that were solved (in brackets).

In the first experiment, we solved the Min-3SAT instances from [17], using a cutoff time of 3 hours as in [17], and compared the performance of MinSatz with the Partial MaxSAT encodings proposed in [17]. This experiment was performed on a Macpro with a 2.8 GHz Intel Xeon processor with MAC OS X 10.5 and 4 GB memory. The number of variables in the Min-3SAT instances ranged from 20 to 100, and the clause-to-variable ratios (C/V) considered were 4, 4.25 and 5. At each point, 50 instances were solved. The best runtimes of the three encodings were obtained by using MaxSatz [17]. We compared MaxSatz runtimes with our MinSAT solver in Table 1, but do not show results for the Min-2SAT instances from [17] because they are easily solvable, using the improved encoding, by both MaxSAT and MinSAT solvers. The results indicate that a genuine MinSAT solver clearly outperforms the best performing MaxSAT solvers. Notice that MinSAT 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 2011 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 in the evaluation: IncMaxSatz and akmmaxsat, and CPLEX, as well as with two of the best performing specific algorithms for MaxClique: MaxCLQ and MCQDyn. The results are shown in Table 2. Our MinSAT solver outperforms both MaxSAT solvers, as well as CPLEX using the edge formulation of MaxClique, and 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.

2 Available at http://cs.hbg.psu.edu/txn131/clique.html.

Table 1
Mean runtimes in seconds for Min-3SAT, followed by the number of instances solved within 3 hours (in brackets), of MinSat and of MaxSat using three encodings of MinSAT into MaxSAT. Experiments performed on a Macpro with Intel XEON 2.8 GHz CPU under MacOSX 10.5.

<table>
<thead>
<tr>
<th>Instance</th>
<th>#var</th>
<th>MinSatz</th>
<th>MaxSatz</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>E1</td>
<td>E2</td>
</tr>
<tr>
<td>20</td>
<td>4.00</td>
<td>0.01(50)</td>
<td>0.02(50)</td>
</tr>
<tr>
<td>30</td>
<td>4.00</td>
<td>0.01(50)</td>
<td>0.24(50)</td>
</tr>
<tr>
<td>40</td>
<td>4.00</td>
<td>0.01(50)</td>
<td>3.28(50)</td>
</tr>
<tr>
<td>50</td>
<td>4.00</td>
<td>0.03(50)</td>
<td>49.30(50)</td>
</tr>
<tr>
<td>60</td>
<td>4.00</td>
<td>0.12(50)</td>
<td>742.4(50)</td>
</tr>
<tr>
<td>70</td>
<td>4.00</td>
<td>0.41(50)</td>
<td>5735.34</td>
</tr>
<tr>
<td>80</td>
<td>4.00</td>
<td>1.97(50)</td>
<td>– (0)</td>
</tr>
<tr>
<td>90</td>
<td>4.00</td>
<td>8.87(50)</td>
<td>– (0)</td>
</tr>
<tr>
<td>100</td>
<td>4.00</td>
<td>30.59(50)</td>
<td>– (0)</td>
</tr>
</tbody>
</table>

– MinSatz(UB3): The implementation of our Weighted Partial MinSAT solver with UB3.

All the solvers are executed on a 3.33 GHz Intel core 2 duo CPU with Linux and 4 GB memory, unless otherwise stated.
In this section, we report a suspected relationship between the MaxSAT value of a given random formula and its MinSAT value. This observation is reported on uniform random formulae, and a first explanation for this phenomenon is given. Due to the intrinsic hardness of both exact MinSAT and exact MaxSAT problems in random formulae, and the need for a large set of data to analyze (we want to take into account very rare events), we targeted 80,000 runs, on small random formulae, 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.

In the third experiment we solved the combinatorial auction instances from the 2011 MaxSAT Evaluation, also using a cutoff time of 1800 seconds. They were generated with the Combinatorial Auction Test Suite (CATS) [9], which is a random generator inspired from real-world scenarios. The distributions used were Paths (88 instances) and Scheduling (84 instances). We compared MinSatz with akmaxsat, akmaxsat ls, and Wmaxsatz+, which were the three best performing MaxSAT solvers on those instances in the evaluation. Results are shown in Table 3. MinSatz is substantially faster than the three MaxSAT solvers on these instances. Since these instances were very easy for MinSatz, we decided to compare MinSatz with specific algorithms for combinatorial auctions, reported in the literature, on harder instances.

In the fourth experiment, we compared the performance of all our Weighted Partial MinSAT solvers 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 for these instances are taken from the same web page, which were obtained on a XEON 2.4 GHz. To make a fair (but rough) comparison with CASS, we divide the taken runtimes of CASS by 1.3875 (remind time of 3600 seconds).
i.e., 120 variables at the threshold 4.25. Thus, the question whether this relationship holds or not for significantly larger formulae is still open. However, our results are significant enough to suspect a positive answer to this new and intriguing question about random 3SAT formulae, that is worth reporting. The relationship we found is somehow counter-intuitive: the smaller the MaxSAT value is, the larger the MinSAT value is, and vice versa.

For our experiments, we generated 81,461 uniform random formulae with 511 clauses and 120 variables (ratio $r = 4.25$). We then computed, for each formula $f_i$, its exact MinSAT and MaxSAT values (resp. $Min_i$ and $Max_i$). In the first analysis, we used their MaxSAT values to partition them such that each partition contains all generated instances $f_i$ sharing the same MaxSAT value ($\Pi_j$ is the set of $f_i$ such that $Max_i = j$). Each curve shows the cumulative distribution function (CDF) of MinSAT values of $\Pi_{511}$, $\Pi_{510}$, $\Pi_{509}$ and $\Pi_{508}$, respectively ($\Pi_{507}$ is reduced to a singleton, and is not considered here). The CDF of $\Pi_{Max}$ shows the probability (Y axis) that a random formula with a MaxSAT value $Max$ will be found with a MinSAT value less than $X$. The picture clearly shows that each partition has a significantly distinct CDF shape, suggesting that the distribution of MinSAT values are distinct for distinct $\Pi_j$. We confirmed this observation by pairwise two-sample Kolmogorov–Smirnov tests between all couples of MinSAT values sets. The Null hypothesis was rejected with high confidence, confirming that distributions of MinSAT values are indeed statistically distinct.

In Fig. 4, we took the opposite view: we partitioned formulae according to their MinSAT values, and measured, for each partition $\sigma_j$, the percentage of instances that are satisfiable ($\sigma_j$ is the percentage of formulae $f_i$ that are satisfiable in the set of formulae $f_i$ such that $Min_i = j$). The figure clearly shows the relationship between both values, that almost perfectly fit the linear function $y = 2.52x - 838$ if we forget outliers: we obtained 70 distinct MinSAT values ranging from to 329 to 398, but 90% of them are between 346 and 359 (5th and 95th percentiles), as reported on the CDF curve (third curve Fig. 4) of MinSAT values.

If the relationship between MinSAT and MaxSAT values is clear, it is striking to notice that the observed probability values almost perfectly fit a linear curve. However, the CDF curve also suggests that, in most cases, just a weak guess can be made. If we follow the CDF curve, half of the formulae (between the 25th and the 75th percentiles) report MinSAT values between 350 and 356, thus allowing us to bet the satisfiability of the formulae (by reading the fitted curve) only between 46% and 60% chance (which is close to the initial guess of 50% chance at the threshold). And the chances to get a stronger guess are quickly decreasing: there is almost no chance to get a guess like 25% SAT (less than 1% chance) or 75% SAT (less than 5% chance).

We give now a preliminary explanation about the relationship between MaxSAT and MinSAT. In [21], it was suggested to use a second order parameter (in addition to the ratio number of clauses over number of variables) to refine the prediction we can make on the satisfiability of a random instance. The second order parameter $\Delta$ of a random instance is relatively easy to compute:

$$\Delta = \sum_{x \in V} |pos_x - neg_x|$$
Fig. 4. Relationship between the MinSAT value of a formula and the observed probability of its satisfiability. The CDF of MinSAT values somehow reports on the strength of the Y values (which is based on more or less experiments). A total of 81461 formulae were tested. The bottom-right subfigure enlarges the X axis range to report all obtained points, including very rare MinSAT values (where the percentage of SAT instances can be based on 1 or 2 formulae only).

Fig. 5. Regression (linear) fit of MaxSAT values (top figure) and MinSAT values (bottom figure) against their second order $\Delta$ values (over 20000 formulae).

where $V$ is the set of variables in the formula, $pos_x$ is the number of positive (non-negated) occurrences of variable $x$ and $neg_x$ is the number of negated occurrences of variable $x$.

Fig. 5 shows the linear regression of the 20,000 points we gathered here. The asymptotic standard error obtained for each curve, for the gradient of the lines, is less than 1.5%. The curve clearly summarizes the general tendency we discovered: the bigger the $\Delta$ value is, the lower the MinSAT value, and the bigger the MaxSAT value. Note that $\Delta$ was just used to guess SAT/UNSAT in [21], and not as a general indicator on the MaxSAT values. The line for the MaxSAT values is now an upward line, and the line for the MinSAT values a downward line. We think that the relationship between MinSAT and MaxSAT values in uniform, random formulae may be explained by their relationships with the $\Delta$ value. An equivalent observation is that the maximum number of clauses that can be falsified (the total number of soft clauses minus the MinSAT value)
correlates with the maximum number of clauses that can be satisfied (the MaxSAT value), and that both values decrease as the second order parameter increases.

We do not exhaustively report here all our experiments, because this phenomenon is similar at higher clause/variables ratio (until \( r = 8 \) but with fewer runs). Note that for values smaller than 4.25, most generated instances are satisfiable, and any partition according to their MaxSAT values does not make sense.

Our conclusion here is somehow surprising. Random formulae with higher MaxSAT values significantly tend to have smaller MinSAT values, and vice versa. Thus, intuitively, SAT formulae are not simply “more satisfiable” than UNSAT ones. They are more “flexible”. As an additional conclusion, we also want to point out that the MinSAT values might be a more meaningful discriminator for SAT formulae than their MaxSAT values. In fact, the 81461 formulae we used only present 5 different MaxSAT values, but up to 70 different MinSAT values. We also think that studying MinSAT values in random formulae, from a theoretical point of view, may cast new insights on their particular structure. There is a clear relationship between their MinSAT value and their satisfiability, and there is no threshold phenomenon on their MinSAT side, which may open new ways of studying them.

5. Conclusions

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. 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 empirical evaluation 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 which are very different from the ones applied in MaxSAT solvers. Our experiments also show that our approach is a viable alternative for optimization problems because it was able to beat specific solvers for MaxClique and combinatorial auctions. The efficiency of our MinSAT solver has also allowed us to investigate an interesting correlation between the MinSAT and MaxSAT values on random CNF formulae, which indicates that random formulae with higher MaxSAT values significantly tend to have smaller MinSAT values, and vice versa.

References