Journal on Satisfiability, Boolean Modeling and Computation 9 (2015) 89-128

AHMAXSAT: Description and Evaluation of a Branch and Bound Max-SAT Solver

André Abramé andre.abrame@lsis.org
Djamal Habet djamal.habet@lsis.org
Aix Marseille Université, CNRS, ENSAM
Université de Toulon, LSIS UMR 7296, 13397
Marseille, France

Abstract

Branch and bound (BnB) solvers for Max-SAT count at each node of the search tree the number of disjoint inconsistent subsets to compute the lower bound. In the last ten years, important advances have been made regarding the lower bound computation. Unit propagation based methods have been introduced to detect inconsistent subsets and a resolution-like inference rules have been proposed which allow a more efficient and incremental lower bound computation. We present in this paper our solver AHMAXSAT, which uses these new methods and improves them in several ways. The main specificities of our solver over the state of the art are: a new unit propagation scheme which considers all the variable propagation sources; an extended set of patterns which increase the amount of learning performed by the solver; a heuristic which modifies the order of application of the max-resolution rule when transforming inconsistent subset; and a new inconsistent subset treatment method which improves the lower bound estimation. All of them have been published in recent international conferences. We describe these four main components and we give a general overview of AHMAXSAT, with additional implementation details. Finally, we present the result of the experimental study we have conducted. We first evaluate the impact of each component on AHMAXSAT performance before comparing our solver to some of the current best performing complete Max-SAT solvers. The results obtained confirm the ones of the Max-SAT Evaluation 2014 where AHMAXSAT was ranked first in three of the nine categories.

KEYWORDS: *Max-SAT, branch and bound, max-resolution*

Submitted December 2014; revised May 2015; published December 2015

1. Introduction

The Max-SAT problem consists in finding an assignment to the Boolean variables of an input CNF formula which maximizes (minimizes) the number of satisfied (falsified) clauses. Max-SAT is the optimization version of the well-known satisfiability (SAT) problem and it is NP-hard [49]. Max-SAT has numerous applications in real-life domains such as routing [60], bioinformatics [54], scheduling and planning [16, 62], etc. Many academic optimization problems can be formulated as Max-SAT instances (Max-CSP over finite domains [21], Max-Cut, Max-Clique, etc.). In the weighted version of Max-SAT, a positive weight is associated to each clause and the goal is to maximize (minimize) the sum of the weights of the satisfied (falsified) clauses. Two other variants exist: partial Max-SAT and weighted partial Max-

SAT. In these variants, the clauses of the instances are separated in two parts: the hard ones and the soft ones. The solution of a (weighted) partial Max-SAT instance is an assignment which satisfies all the hard clauses while maximizing (minimizing) the number (the sum of the weights) of satisfied (falsified) soft clauses. Unless specified, we consider all instances as weighted partial Max-SAT in the rest of this paper.

Several methods have been proposed to solve the Max-SAT problem. These methods can be divided in two main categories: the complete ones, which give the optimal solution and the incomplete ones which are generally faster but cannot prove the solution optimality. The incomplete methods are mainly based on local search algorithms [1, 14, 24, 52, 61] or approximation methods [23, 57]. Among the complete methods, the ones based on iterative calls to a SAT solver [22, 28, 32, 42, 43, 44, 45, 47] are very efficient on industrial instances. Solvers based on the reformulation in integer linear programming [7, 10] perform particularly good on crafted partial instances. Note that hybrid solvers exist [17, 18, 19], which combine iterative-SAT and ILP-based approaches. Finally, solvers based on a branch and bound (BnB) algorithm outperform the other approaches on random and non-partial crafted instances. It is worth mentioning the existence of portfolio algorithms [11].

A typical branch and bound (BnB) algorithm for Max-SAT works as follows. At each node of the search tree, it computes the lower bound (LB) by counting the disjoint inconsistent subsets (IS) remaining in the formula. Then it compares LB to the best solution found so far (the upper bound, UB). If LB is greater or equal to UB, then no better solution can be found by extending the current assignment and the algorithm performs a backtrack (it goes back to the previous node of the search tree). Otherwise, it selects an unassigned variable and gives it a value. When all the variables have been assigned, a new best solution is found and UB is updated. This process is repeated until the whole search tree has been explored. In the last ten years, many work have been made on BnB solvers for Max-SAT and especially on the lower bound estimation. New methods based on unit propagation have been proposed to detect the disjoint inconsistent subsets [38, 39]. New inference rules have been defined [25, 34, 40] to simplify the formula. These rules are also used to perform a limited form of learning in the sub-part of the search tree, making the LB computation more incremental. Among these inference rules, the max-resolution [13, 25, 34] is especially interesting since all the previously introduced inference rules can be viewed as limited forms of max-resolution [35]. These methods have been implemented in several solvers such as MiniMaxSat [26, 27], wmaxsatz [40, 37] or akmaxsat [33] and lead to significant practical improvements.

In the first part of this paper, we make a review of the techniques used in the most recent BnB solvers. Especially, we detail three of the most critical of their components: (i) the LB computation, (ii) the branching heuristics, and (iii) the inference rules used to extend the assignment. The second part of this paper is dedicated to our solver ahmaxsat, which was ranked first in three of the nine categories of the Max-SAT Evaluation 2014 (Max-SAT random, Partial Max-SAT random and Max-SAT crafted) and second in a fourth one (Weighted Partial Max-SAT random, which is composed of weighted instances and of weighted partial ones). We describe the four main novel components included in ahmaxsat: a new unit propagation scheme [4], extended sets of inference rules [5], a new way to treat the IS detected during unit propagation [3], and a heuristic which makes the max-resolution rule more usable [2]. We give a general overview of our solver and give details

on its implementation. Finally, we present the results of the experimental study we have performed. We first show the improvements brought by each AHMAXSAT new component and we discuss their interactions. Then we compare AHMAXSAT to some of the most recent complete Max-SAT solvers on the benchmark set of the Max-SAT Evaluation 2014 and on an extended set of generated random and crafted instances. We discuss the advantages and weaknesses of the each solver family and we give improvement perspectives for AHMAXSAT and more generally for BnB solvers.

This paper is organized as follows. We give in Section 2 the basic definitions and notations used in this paper. In Section 3, we describe the recent Max-SAT BnB implementations. Section 4 is dedicated to the description of our solver AHMAXSAT. We give the results of the experimental study we have performed in Section 5 before concluding in Section 6.

2. Definitions and Notations

A weighted formula Φ in conjunctive normal form (CNF) defined on a set of n propositional variables $X = \{x_1, \ldots, x_n\}$ is a conjunction of weighted clauses. A weighted clause c_j is a weighted disjunction of literals and a literal l is a variable x_i or its negation $\overline{x}_i$. Alternatively, a weighted formula can be represented as a multiset of weighted clauses $\Phi = \{c_1, \ldots, c_m\}$ and a weighted clause as a tuple $c_j = (\{l_{j_1}, \ldots, l_{j_k}\}, w_j)$ with $\{l_{j_1}, \ldots, l_{j_k}\}$ a set of literals and $w_j > 0$ the clause weight. The minimum value for the weight of the hard clauses is the sum of the weights of the soft clauses plus one. Note that the weighted partial Max-SAT formalism can be used to represent any other of the Max-SAT variants. In non-partial instances, there is no hard clause while in unweighted instances all the weights of the soft clauses are set to 1. We denote the number of clauses of Φ by $|\Phi|$ and the number of literals of c_j by $|c_j|$.

An assignment can be represented as a set I of literals which cannot contain both a literal and its negation. If x_i is assigned to *true* (resp. $false$) then $x_i \in I$ (resp. $\overline{x}_i \in I$). I is a complete assignment if $|I| = n$ and it is partial otherwise. A literal l is said to be satisfied by an assignment I if $l \in I$ and falsified if $\bar{l} \in I$. A variable which does not appear either positively or negatively in I is unassigned. A clause is satisfied by I if at least one of its literals is satisfied, and it is falsified if all its literals are falsified. By convention, an empty clause (denoted by $\square$) is always falsified. A subset ψ of Φ is inconsistent if there is no assignment which satisfies all its clauses. For a unit assignment $I = \{l\}$, we denote by $\Phi|_I$ the formula obtained by applying I on Φ. Formally, $\Phi|_I = \{c_j \mid c_j \in \Phi, \{l, \bar{l}\} \cap c_j = \emptyset\} \cup \{c_j/\{\bar{l}\} \mid c_j \in \Phi, \bar{l} \in c_j\}$. This notation can be extended to any assignment $I = \{l_1, l_2, \ldots, l_k\}$ as follows: $\Phi|_I = (\ldots((\Phi|_{\{l_1\}})|_{\{l_2\}})\ldots|_{\{l_k\}})$. Solving the weighted partial Max-SAT problem consists in finding a complete assignment which satisfies all the hard clauses and which maximizes (minimizes) the sum of the weights of the satisfied (falsified) soft clauses of Φ. Two formulas are equivalent for (weighted) Max-SAT iff they have the same number (sum of weights) of falsified clauses for each partial assignment.

3. State of the Art of Branch and Bound Solvers

A typical branch and bound solver for Max-SAT works as follows (Algorithm 1). It starts by initializing the best solution found so far, the upper bound (UB), to the worst possible value: the sum of the weights of the soft clauses (line 2). Then it explores the space of the possible solutions by constructing a search tree. At each node of this search tree, it first tries to extend the current assignment by applying inference rules (line 6). Then, it computes the lower bound (LB) which is the sum of the weights of the clauses falsified by the current partial assignment I plus an under-estimation of the sum of the clauses which will become falsified if we extend the current assignment (line 7). If $LB \geq UB$, then no better solution can be found by extending the current assignment and the algorithm performs a backtrack (line 8). It undoes the previously made decisions until finding an unexplored branch of the search tree. If the current assignment is complete (i.e. a value is given to all the variables), then a new better solution is found and the algorithm updates the UB before backtracking (lines 9-12). Otherwise, it selects an unassigned variable according to a branching heuristic and it gives it a value (line 14-15). These steps are repeated until the whole search space has been explored.

Algorithm 1: A typical BnB algorithm for Max-SAT

Data: A CNF formula Φ with n the number of variables of Φ.
Result: (UB, I_{UB}) with UB the best solution found and I_{UB} the corresponding assignment.

```
1  begin
2      UB ← ∑_{c_j∈Φ} w_j;
3      I_UB ← ∅;
4      I ← ∅;
5      repeat
6          I ←assignment_extension(Φ,I);
7          LB ← ∑_{c_j=□∈Φ|_I} w_j + count_remaining_conflicts(Φ|_I);
8          if LB ≥ UB then backtrack();
9          else if |I| = n then
10             UB ← ∑_{c_j=□∈Φ|_I} w_j;
11             I_UB ← I;
12             backtrack();
13         else
14             l ←select_new_decision();
15             I ← I ∪ {l};
16     until |I| > 0;
17     return (UB, I_UB)
```

In the rest of this paper, we consider that at each node of the search tree the initial formula Φ is simplified by the current partial assignment I. Thus, the term formula will now refer to $\Phi|_I$ and we will use the term initial formula or original formula to refer to Φ. It implies that other definitions such as unit clause or inconsistent subset must be considered in the simplified formula $\Phi|_I$ and not in the original one Φ.

The efficiency of Max-SAT BnB solvers depends mainly on two parameters: (1) the number of nodes of the search tree they explore (thus their capability to prune the search tree) and (2) the time they spent on each node. The amount of pruning performed depends on the quality of the upper and lower bounds. We consider that UB is of good quality if its value is close to the optimum of the instances. Similarly, at a given node of the search tree, we consider that LB is of good quality if its value is close to the best solution reachable in the current branch of the search tree. Three components have an impact on the UB and LB qualities: the branching heuristic (function `select_new_decision`), the inference rules used to extend the current assignment (function `assignment_extension`) and the under-estimation of the clauses which will become falsified (function `count_remaining_conflicts`). In the rest of this section, we give an overview of the techniques used in these components.

3.1 Branching Heuristic

The branching heuristic in BnB Max-SAT solvers has multiple goals which are not necessarily compatible. On the one hand, to get a good quality upper bound, solvers must choose the assignments which satisfy as many clauses as possible. On the other hand, to get high lower bound values in the early nodes of the search tree solvers should choose the assignments which falsify as much clauses as possible. These assignments should also allow a good estimation of the remaining inconsistencies (thus they should create unit and binary clauses to empower unit propagation). Since both polarities will likely be explored for each chosen variable, solvers should choose the "balanced" variables first, i.e. variables which are likely to give good LB value for both polarities.

The first branching heuristics dedicated to Max-SAT solvers [8, 31, 59] were variants of the SAT branching heuristics MOMS [50] or Jeroslow-Wang (JW) [29, 30]. The main difference of these heuristics over their SAT original versions is that the importance of the unit clauses is reduced while the one of the binary clauses is increased. That way, the heuristics favor the creation of new unit clauses which allow a better LB estimation. However, these rules focus on satisfying clauses rather than falsifying them. Each literal has a score which only depends on the clauses in which it appears. There is no mechanism to ensure a balance between the polarities of the chosen variables.

Recent BnB solvers use more sophisticated branching heuristics. For instance, WMAXSATZ gives a score to each literal based on its occurrence in the clauses and then chooses the variable which maximizes the multiplication of its literal scores plus their sum. It aims at obtaining high LB values in the higher nodes of the search tree while keeping its branches balanced. The branching heuristic of AKMAXSAT is similar, except it takes into consideration the participation of the literals in the inconsistent subsets (IS) detected during the LB computation.

It is worth noticing that to the best of our knowledge only one branching heuristic makes a special treatment for hard clauses. MiniMaxSat applies the VSIDS heuristic [46] as long as there are hard clauses. When there are no hard clauses, it uses a weighted variant of the Jeroslow-Wang heuristic [25].

3.2 Extending the Current Assignment

At each node of the search tree, BnB solvers use inference rules to extend the assignment. This component is critical for the solvers efficiency since it allows to prune the search tree by fixing the value of some variables. The higher in the search tree and the more the variables value can be fixed, the less the number of explored nodes is. Recent solvers use three inference rules for this purpose.

Hard Unit Propagation Unit propagation (UP) is a powerful inference rule used in SAT complete solvers [20, 46]. It consists in iteratively fixing to true the literals which appear in unit clauses. This rule however cannot be applied in the Max-SAT context since it can lead to non-optimal solution [38]. However, when the weight of a unit clause is greater than the difference between UB and LB, falsifying the literal of this clause will necessarily lead to a solution worst than the best one found so far. Thus, solvers can safely satisfy this literal. We call this inference rules hard unit propagation (HUP).

Dominating Unit Clause Another inference rule for SAT which can be used in the Max-SAT context is the dominating unit clause rule (DUC) [46]. If the sum of the weights of the unit clauses containing a literal l_i is greater or equal to the sum of the weights of the clauses which contain $\overline{l_i}$, then satisfying l_i will necessarily lead to a better solution than falsifying it.

Pure Literal The last inference rule used to extend the assignment by BnB Max-SAT solvers is the pure literal rule (PL) [48]. If a variable appears only positively (negatively) in the formula, then it can be fixed to true (false) without falsifying any clause.

3.3 Lower Bound Computation

At each node of the search tree, BnB solvers calculate the lower bound (LB) which is an under-estimation of the sum of the weights of the clauses which will be falsified if we extend the current partial assignment. This component is critical in two ways. Firstly, it is applied very often and its computing time has an important impact on a solver's efficiency. Secondly, its quality determines the number of explored nodes. Thus, a trade-off must be made between the time spent to compute it and its quality.

The simplest way to compute the lower bound consists in counting the sum of the weights of the clauses falsified by the current assignment. In addition, several methods have been proposed to increase LB accuracy by detecting or approximating the disjoint inconsistent subsets (IS) present in the formula under the current partial assignment. The first LB computation method was the inconsistency count due to Wallace and Freuder [58] which counts the number of disjoint IS of the form $\{\{l_i\}, \{\overline{l_i}\}\}$. Shen and Zhang have improved this first method in LB4 [51] by using linear resolution to generate more unit clauses before applying inconsistency count. Alisnet et al. used the star rule [48] to capture disjoint IS of the form $\{\{l_1\}, \ldots, \{l_k\}, \{\overline{l_1}, \ldots, \overline{l_k}\}\}$ [9]. Approximations have also been used to compute a fast lower bound, using integer programming [59] or semi-definite programming [23].

More recently, Li et al. have shown [38, 39] how unit propagation based methods can be used to detect IS. Once detected, they are transformed to ensure their disjointness. We present the existing IS detection and treatment methods in the remaining of this section.

3.3.1 UP-based Detection of the Inconsistent Subsets

Unit propagation (UP) consists in iteratively satisfying the literals which appear in unit clauses until a conflict is found (an empty clause) or no more unit clauses remain. When a conflict is found, the clauses which have led to it by propagation form an IS of the formula (under the current partial assignment).

The propagation steps can be represented by an implication graph $G = (V, A)$ which is a directed acyclic graph where the nodes V are the propagated literals and each arrow of A is tagged with the clause causing the propagation [41].

Simulated Unit Propagation Li et al. have proposed a new LB computation method based on unit propagation [38]. At each node of the search tree, they apply UP on the formula. When a conflict is detected, they analyze the implication graph to build the corresponding IS and apply a treatment (the existing treatments are described below) on it to ensure that it will not be counted more than once. This process is repeated until no more unit clauses remain. Since UP is not sound for Max-SAT (it can lead to non-optimal solution [38]) the propagated variables are unassigned at the end of the lower bound estimation. This method is called *simulated unit propagation* (SUP) to differentiate it from the classical UP usage.

Each soft clause of the formula can only participate in one IS. Thus, the smaller the IS are, the more SUP will be able to detect other IS. Based on this observation, Li et al. have introduced two SUP variants: SUP^S and SUP* [39]. The first one uses a stack instead of a queue to store the unit clauses so the last unit clause inserted is the first one used for propagation. The second one uses two different queues, one for the "real" unit clauses (which does not depend on the simulated propagation steps) and one for the "simulated" unit clauses (which contains literals falsified by the simulated propagation steps). Empirical results suggest that SUP* performs better than the two other variants [39].

Example 1. Let us consider the (unweighted and non-partial) CNF formula $\Phi_1 = \{c_1, \ldots, c_{10}\}$ with $c_1 = \{x_1\}$, $c_2 = \{\overline{x}_1, x_2\}$, $c_3 = \{\overline{x}_1, \overline{x}_2, x_3\}$, $c_4 = \{\overline{x}_2, x_4\}$, $c_5 = \{x_5\}$, $c_6 = \{\overline{x}_5, x_2\}$, $c_7 = \{x_6\}$, $c_8 = \{\overline{x}_6, x_7\}$, $c_9 = \{\overline{x}_6, x_3\}$ and $c_{10} = \{\overline{x}_6, \overline{x}_7, \overline{x}_3\}$. The application of SUP* on Φ_1 leads to the sequence of propagations $\langle x_1@c_1, x_2@c_2, \ldots, x_5@c_5, x_6@c_7, x_7@c_8\rangle$ (meaning that x_1 is propagated by clause c_1, then x_2 by c_2, etc.). The clause c_{10} is empty. Fig. 1 shows the corresponding implication graph. The set of clauses $\psi = \{c_1, c_2, c_3, c_7, c_8, c_{10}\}$ which have led by propagation to the conflict (i.e. to the empty clause c_{10}) is an IS of Φ_1.

Failed Literals The second UP-based method to detect IS, the failed literals technique (FL) [39], is due to Li et al. Let us consider a formula Φ and a partial assignment I. At each node of the search tree and for each unassigned variable x_i such as $I \cap \{x_i, \overline{x}_i\} = \emptyset$, FL applies SUP to $\Phi|_{I\cup\{x_i\}}$ then to $\Phi|_{I\cup\{\overline{x}_i\}}$. If two IS ψ and ψ' are detected in $\Phi|_{I\cup\{x_i\}}$ and $\Phi|_{I\cup\{\overline{x}_i\}}$ respectively, then $\psi \cup \psi'$ is an IS in $\Phi|_I$. FL is generally applied when no more conflict can be reached by SUP. Since FL can be very time consuming, its application is generally restricted to variables which appears both positively and negatively in binary clauses.

This technique has been further refined by Kügel [33] by merging SUP and FL into one single method called generalized unit propagation (GUP). The main idea behind GUP is to

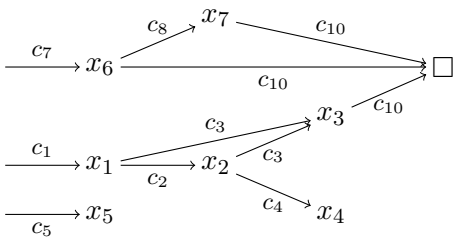


Figure 1: Implication graph of the formula Φ_1 from Example 1. Nodes are the propagated literals and arrows are labeled with the clauses causing the propagations.

reuse the previously detected failed literals. If one IS ψ_1 is detected in $\Phi|_{I\cup\{x_i\}}$, we know that x_i must be set to false to avoid a clause of ψ_1 from being falsified. If no IS is detected in $\Phi|_{I\cup\{\overline{x}_i\}}$, we can still use ψ_1 and continue applying FL in the formula $\Phi|_{I\cup\{\overline{x}_i\}}$.

3.3.2 Transforming Inconsistent Subsets

We give below the existing treatments applied on the clauses of the IS (IS) detected by SUP or FL.

Inconsistent Subset Removal (ISR) A simple way to avoid detecting the same IS several times is to remove its clauses and add an empty clause to the formula. Such operation has the advantages of being fast and it does not increase the size of the formula. Since the removed clauses are restored before the next decision, ISR does not impact the LB estimation in the current sub-tree. However, the formula resulting from the ISR application is not equivalent to the original one and may contain fewer inconsistencies. Indeed, IS clauses are removed without considering their interaction with the rest of the formula. Thus, the quality of the lower bound estimation can be deteriorated and the number of decisions made by a BnB solver increased.

Max-resolution Rule An alternative way to avoid the re-detection of the same conflicts is to use the max-resolution rule [13, 25, 34], which is the Max-SAT version of the SAT resolution. It is defined as follows (on top the original formula and at the bottom the formula after the transformation):

$$\frac{c_i = \{x, y_1, \ldots, y_s\},\ c_j = \{\overline{x}, z_1, \ldots, z_t\}}{cr = \{y_1, \ldots, y_s, z_1, \ldots, z_t\},\ cc_1,\ \ldots,\ cc_t,\ cc_{t+1},\ \ldots,\ cc_{t+s}}$$

with:

$$\begin{aligned} cc_1 &= \{x, y_1, \ldots, y_s, \overline{z}_1, z_2, \ldots, z_t\} & cc_{t+1} &= \{\overline{x}, z_1, \ldots, z_t, \overline{y}_1, y_2, \ldots, y_s\} \\ cc_2 &= \{x, y_1, \ldots, y_s, \overline{z}_2, \ldots, z_t\} & cc_{t+2} &= \{\overline{x}, z_1, \ldots, z_t, \overline{y}_2, \ldots, y_s\} \\ &\vdots & &\vdots \\ cc_t &= \{x, y_1, \ldots, y_s, \overline{z}_t\} & cc_{t+s} &= \{\overline{x}, z_1, \ldots, z_t, \overline{y}_s\} \end{aligned}$$

Where c_i and c_j are the original clauses, cr the resolvent and $cc_1, \ldots, cc_{t+s}$ the compensation clauses added to preserve the number of falsified clauses (keep formula's equivalency).

One should note that c_i and c_j are removed after max-resolution. This definition can be extended to weighted formulas as follows: if m is the minimum weight of the original clauses, m is subtracted from the weight of c_i and c_j and each produced clause (resolvent and compensation ones) takes the weight m.

An IS can be transformed by applying several max-resolution steps between its clauses. These steps are generally applied in reverse propagation order. This process is close to the clause learning mechanism used in modern SAT solvers [41].

The max-resolution is sound but has some disadvantages. Indeed, its application is more time-consuming than the IS removal. Moreover, it increases the number and the size of the clauses and it can push back the detection of inconsistencies in lower nodes of the search tree by removing clauses which could have been used later for propagation.

Example 2. Let us consider again the formula $\Phi_1 = \{c_1, \ldots, c_{10}\}$ from Example 1. We have seen that the IS $\psi = \{c_1, c_2, c_3, c_7, c_8, c_{10}\}$ can be detected by applying SUP*. The transformation of ψ by max-resolution can be done as follows. First, max-resolution is applied between c_8 and c_{10} around the propagated variable x_7. It produces an intermediary resolvent $c_{11} = \{\overline{x_6}, \overline{x_3}\}$ and the compensation clause $c_{12} = \{\overline{x_6}, x_7, x_3\}$. Then, max-resolution is applied between c_{11} and c_7 around the propagated variable x_6 and so forth. The max-resolution steps applied on ψ are shown in Fig. 2 with the compensation clauses in boxes. The original clauses $c_1, c_2, c_3, c_7, c_8, c_{10}$ are removed from the formula and besides the compensation clauses $c_{12}, c_{14}, c_{16}, c_{17}$, the resolvent $c_{19} = \square$ is added to the formula. The intermediary resolvents $c_{11}, c_{13}, c_{15}, c_{18}$ are consumed by the max-resolution steps. We obtain the formula $\Phi' = \{\square, c_4, c_5, c_6, c_9, c_{12}, c_{14}, c_{16}, c_{17}\}$ with $c_{12} = \{\overline{x_6}, x_7, x_3\}$, $c_{14} = \{x_6, x_3\}$, $c_{16} = \{\overline{x_3}, x_1, \overline{x_2}\}$ and $c_{17} = \{\overline{x_3}, x_2\}$.

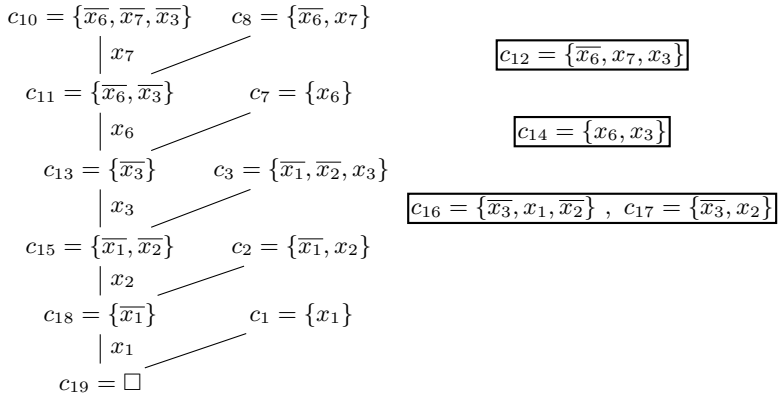


Figure 2: Max-SAT resolution steps applied on the IS ψ in Example 2.

Existing Implementations Recent Max-SAT solvers (e.g. WMAXSATZ [36, 39, 40], AKMAXSAT [33]) apply SUP and FL at each node of the search tree and build an IS for each detected conflict. If the IS corresponds to some patterns (see below), then they apply the max-resolution rule and keep the modifications in the sub-tree. Otherwise, they temporarily

remove all the clauses of the IS from the formula for the duration of the LB estimation. We refer to these two treatments as respectively *sub-tree max-resolution* and *temporary removal* in the rest of this paper. The main patterns are (with on top the clauses of the original formula and on bottom the clauses obtained after transformation):

$$(P1)\ \frac{\{\{x_1, x_2\}, \{x_1, \overline{x}_2\}\}}{\{\{x_1\}\}}, \quad (P2)\ \frac{\{\{x_1, x_2\}, \{x_1, x_3\}, \{\overline{x}_2, \overline{x}_3\}\}}{\{\{x_1\}, \{x_1, x_2, x_3\}, \{\overline{x}_1, \overline{x}_2, \overline{x}_3\}\}},$$

$$(P3)\ \frac{\{\{x_1\}, \{\overline{x}_1, x_2\}, \{\overline{x}_2, x_3\}, \ldots, \{\overline{x}_{k-1}, x_k\}, \{\overline{x}_k\}\}}{\{\square, \{x_1, \overline{x}_2\}, \{x_2, \overline{x}_3\}, \ldots, \{x_{k-1}, \overline{x}_k\}\}.}$$

Note that the number of clauses added is smaller than or equal to the number of original clauses removed. Consequently, the size of the formula does not grow. Moreover, the sizes of the IS corresponding to the patterns are small, and it is less likely that they will hamper the detection of other IS in the sub-part of the search tree. It should be noted that the clauses of an IS are not necessarily all treated by the same method. A part can be transformed by sub-tree max-resolution while the other part is simply removed temporarily.

For the sake of completeness, we also describe the treatment applied on IS by MiniMaxSat [26, 27], which is to the best of our knowledge the only solver which makes a more extensive use of the max-resolution rule. MiniMaxSat detects conflicts by SUP, then it applies sub-tree max-resolution to the IS if at each max-resolution step the resolvent contains less than 4 literals. Otherwise, it temporarily removes the clauses of the IS. As do the aforementioned solvers, MiniMaxSat performs also a limited form of learning in the sub-tree while avoiding or limiting the impact of the max-resolution drawbacks. But rather than specifying a list of patterns, it uses a more general criterion based on the size of the intermediary resolvents. This criterion is less restrictive, i.e. MiniMaxSat makes more learning, but it controls less efficiently the drawbacks cited above.

4. Our BnB Max-SAT Solver: ahmaxsat

We describe in this section our solver ahmaxsat. We first present the four main novel components that it includes: a new unit propagation scheme [4], a heuristic which improves the order of application of the max-resolution steps on IS [2], new sets of patterns to extend the amount of learning performed by the solver [5], and a new transformation method for the parts of the IS which do not match the pattern (IS transformations which are not kept in the sub-part of the search tree) [3]. Then we give an overview of ahmaxsat with an emphasis on its lower bound computation function, its branching heuristic, and the inference rules used to extend the current assignment.

4.1 Handling All Variable Propagation Sources

Multiple Propagation Source Scheme To the best of our knowledge, all the existing implementations of unit propagation (for SAT or Max-SAT) use the *first propagation source* (FPS) scheme: they only consider the first unit clause causing the propagation of the variables (the first propagation source). The next propagation sources encountered are satisfied by the already propagated variables and thus they are simply ignored. When a propagation is undone in this scheme, all the propagations made afterwards must also be

undone to ensure that the previously ignored propagation sources will now be considered. This is satisfactory in the SAT context where the propagations are undone only during the backtracking process (thus in reverse chronological order). In the Max-SAT context, clauses can be removed from the formula during the IS treatment. In such a situation, propagations made afterward must be undone even if they are still valid (i.e. they still have a propagation source). Thus many unnecessary un-propagations and re-propagations may be performed. The following example illustrates this behavior.

Example 3. Let us consider the formula Φ_1 from Example 1. We have seen that the application of SUP* on Φ_1 leads to the sequence of propagations $\langle x_1@c_1, x_2@c_2, \ldots, x_5@c_5, x_6@c_7, x_7@c_8\rangle$. The clause c_{10} is empty and we can compute the inconsistent subset (IS) $\psi = \{c_1, c_2, c_3, c_7, c_8, c_{10}\}$ by analyzing the implication graph. One can note that the clauses c_6 and c_9, which are predecessors (but not the first ones) of respectively x_2 and x_3 are neither considered nor represented in the implication graph (see Figure 1).

If the clauses of ψ are removed from the formula, we obtain $\Phi'_1 = \{c_4, \ldots, c_6, c_9\}$. All the propagations caused by the clauses of ψ must be undone. The less recent ones is $x_1@c_1$ and since the propagations are undone in reverse chronological order in FPS, all the propagations are undone. Thereafter, application of SUP* on Φ'_1 leads to the sequence of propagation $\langle x_5@c_5, x_2@c_6, x_4@c_4\rangle$. Note that these three variables have been consecutively unassigned and reassigned.

To overcome this limitation of the FPS scheme, we have proposed a new scheme called *multiple propagation source* (MPS) scheme [4]. In this scheme, all the propagation sources of each variable are stored rather than only the first one. Propagated variables are undone only when they have no more propagation source. This way, propagations can be undone in a non-chronological order and fewer unnecessary propagation steps are performed. The propagation steps performed in the MPS scheme can be modeled by a full implication graph [4] which is a AND/OR directed graph where the AND nodes are the clauses causing the propagation and the OR nodes the propagated variables. Note that such scheme has been only considered from a theoretical point of view [56] and in a very limited way for improving the backjump level [12] of Conflict Driven Clause Learning SAT solvers [41]. The following example illustrates the MPS scheme behavior.

Example 4. Let us consider again the formula Φ_1 from Example 1. The application of SUP* with MPS leads to the same six first propagation steps done in Example 3: $\langle x_1@c_1, x_2@c_2, \ldots, x_5@c_5, x_6@c_7\rangle$. At this point, the variable x_7 has two propagation sources of opposite polarities, c_8 and c_{10}, which cannot be both satisfied. Fig. 3 shows the full implication graph obtained in a MPS based solver. One can note that the second propagation source of x_2 and x_3 (respectively c_6 and c_9) are represented in the full implication graph. As in the previous example, we can build the IS $\psi = \{c_1, c_2, c_3, c_7, c_8, c_{10}\}$ by taking the first propagation source of each propagated variable which have led to the conflict. If ψ is removed from the formula, the variables x_1, x_3, x_6 and x_7 have no more propagation source and must be unset. Note that x_2, x_4 and x_5 remain propagated since they all have still at least one propagation source.

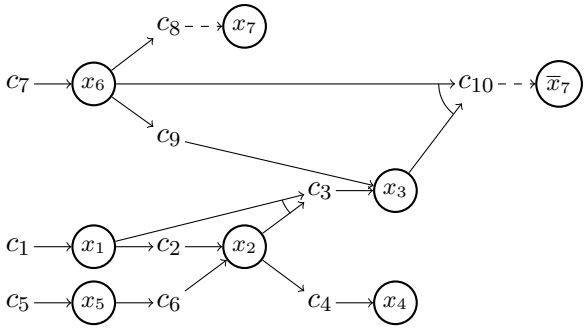


Figure 3: Full implication graph of the formula Φ_1 from Example 4. The circled nodes are the propagated variables while the uncircled nodes are unit clauses. Note that contrary to the implication graph of Fig. 1, the predecessors c_6 of x_2 and c_9 of x_3 are represented.

Reducing the Inconsistent Subset Sizes The MPS scheme has other advantages. It gives information on the structure of the instances which are not available in the FPS scheme. AHMAXSAT uses this information to influence the characteristics of the IS it builds.

When FPS based solvers find a conflict (an empty clause), they build an IS by analyzing the sequence of propagation steps which have led to the conflict. For each propagated variable of this sequence, its propagation source is added to the IS. Since only the first propagation source of each propagated variable is known, only one IS can be built.

To the contrary, MPS based solvers can choose for each propagated variable which has led to a conflict the propagation source they add to the IS. Thus MPS based solvers can influence the structure of the generated IS. We have proposed a simple heuristic to select the propagation source added to the IS [4]. The aim of this heuristic is to reduce the size of the IS produced to let more clauses available for later unit propagation steps. It may improve the estimation of the lower bound and thus reduce the number of explored nodes of the search tree. The heuristic works as follows. For each propagated variable participating in the conflict (taken in reverse propagation order), it selects the propagation source which contains the less new variables. We refer to this method as the *smaller inconsistent subsets* (SIS) heuristic in the rest of this paper.

AHMAXSAT uses the MPS scheme. Each variable keeps two statically allocated lists for its positive and negative propagation sources. When a clause c_j becomes unit, AHMAXSAT identifies the propagated variable x_i (the only unassigned variable of c_j) and adds it to the appropriate propagation source list of x_i. Propagated variables are unset only when they have no more propagation source.

The MPS scheme does not change the overall time complexity of our algorithm. Since there cannot be more propagation sources than the number of clauses of the instance, the worst case time complexity of the IS computation mechanisms (unit propagation, IS building and transformation) remains unchanged. In practice however more clauses will be examined during the unit propagation process and when building the IS. Moreover, the underlying data structures must be adapted to support this new propagation scheme.

4.2 Reducing the Number and the Sizes of the Compensation Clauses

We have seen in Section 3 how IS can be transformed thanks to Max-SAT resolution. When a conflict (an empty clause) is detected by SUP, an IS is built by analyzing the implication graph. This IS is then transformed by applying several max-resolution steps on its clauses. Since max-resolution can be applied on a literal only when it reduces one single clause of the IS (or when all the clauses it reduces have been merged into a single intermediary resolvent), the max-resolution steps are usually applied in reverse propagation order. These transformations produce compensation clauses, and thus the size of the formula can grow quickly.

We have observed that the number and the sizes of the compensation clauses added depend on the order in which the Max-SAT resolution steps are performed. Especially, we have shown that for two adjacent variables in an implication graph, applying max-resolution first around the one producing the smallest intermediary resolvent always produces less compensation clauses and of smaller size [2]. Based on this observation, we have implemented in AHMAXSAT a simple heuristic to choose the order of application of the max-resolution steps. This heuristic works as follows. Let us consider an implication graph $G = (V, A)$. Firstly, the algorithm computes the scores of the variables of V according to the following definition:

$$score(x_i) = \begin{cases} \text{size of the intermediary resolvent} & \text{if } x_i \text{ reduces a single clause } c_k \\ \infty & \text{if } x_i \text{ reduces more than one clause} \end{cases}$$

Then, it selects the variable of V of the smallest score and it applies Max-SAT resolution between its predecessor and successor. It updates the implication graph G by replacing the consumed clauses by the intermediary resolvent produced (duplicated arcs are removed) and it updates the scores of the variables. This process is repeated until the implication graph contains no more variable. The last resolvent produced is added to the formula. We refer to this method as the *smallest intermediary resolvent* (SIR) heuristic in the rest of this paper. It should be noted that this heuristic is not necessarily optimal and, for a given IS, it may exist sequences of application of the max-resolution steps which reduce further the number and sizes of the compensation clauses produced. Nevertheless, experimental results [2] shows that the SIR heuristic reduces significantly both these values. The following example illustrates how this method works.

Example 5. Let us consider a formula $\Phi_2 = \{c_1, c_2, \ldots, c_6\}$ defined on a set of Boolean variables $\{x_1, \ldots, x_5\}$ with $c_1 = \{x_1\}$, $c_2 = \{x_2\}$, $c_3 = \{\overline{x}_2, x_3\}$, $c_4 = \{\overline{x}_2, x_4\}$, $c_5 = \{\overline{x}_1, \overline{x}_3, x_5\}$ and $c_6 = \{\overline{x}_4, \overline{x}_5\}$. The application of unit propagation on Φ_2 leads to the assignments $\langle x_1@c_1, x_2@c_2, \ldots, x_5@c_5\rangle$ (meaning that x_1 is propagated by clause c_1, then x_2 by c_2, etc.). The clause c_6 is falsified and $\psi = \{c_1, c_2, \ldots, c_6\}$ is an IS of Φ_2. Fig. 4 shows the corresponding implication graph. In this example, the intermediary resolvent and the updated implication graph obtained after a sequence of max-resolution steps $\langle x_{i_1}, \ldots, x_{i_k}\rangle$ are denoted respectively $cr_{\langle x_{i_1},\ldots,x_{i_k}\rangle}$ and $G_{\langle x_{i_1},\ldots,x_{i_k}\rangle}$.

a) Classical transformation by max-resolution: The max-resolution steps are typically applied around the literals of the implication graph in reverse propagation order. Thus, max-resolution is applied first between c_6 and c_5. It produces an intermediary resolvent $cr_{\langle x_1\rangle} = \{\overline{x}_1, \overline{x}_3, \overline{x}_4\}$ and three compensation clauses $cc_1 = \{x_1, \overline{x}_3, \overline{x}_4, \overline{x}_5\}$, $cc_2 = \{x_3, \overline{x}_4, \overline{x}_5\}$

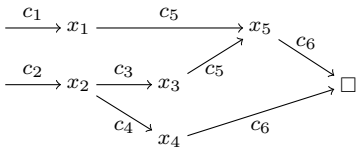


Figure 4: Implication graph for the formula Φ_2 from Example 5.

and $cc_3 = \{\overline{x}_1, \overline{x}_3, x_4, x_5\}$. Then it is applied between $cr_{\langle x_1 \rangle}$ and c_4 and so forth. Fig. 5 shows the max-resolution steps applied on the formula, with the compensation clauses in boxes. The original clauses $c_1, \ldots, c_6$ are removed from the formula and besides the compensation clauses $cc_1, \ldots, cc_8$, the resolvent $cr_{\langle x_1, \ldots, x_5 \rangle} = \square$ is added to the formula. The intermediary resolvents $cr_{\langle x_1 \rangle}, \ldots, cr_{\langle x_1, \ldots, x_4 \rangle}$ are consumed by the Max-SAT resolution steps. We obtain the formula $\Phi'_2 = \{\square, cc_1, \ldots, cc_8\}$. Note that Φ'_2 contains, besides the empty clause $\square$, one clause of size two, three clauses of size three and four clauses of size four.

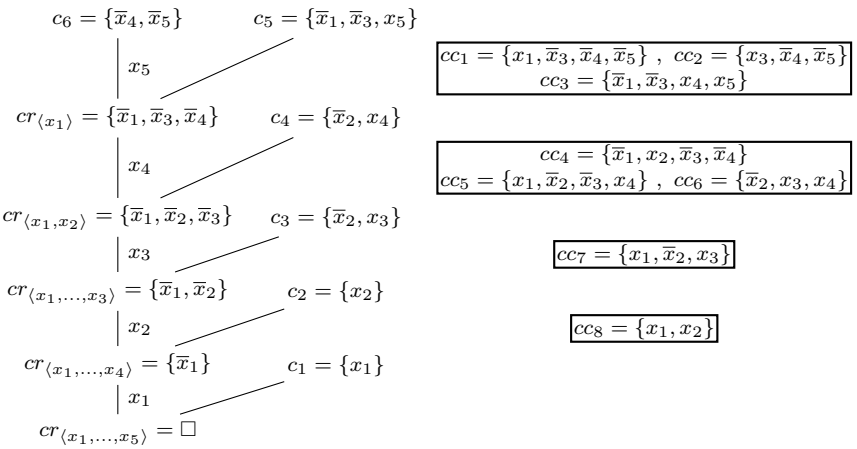


Figure 5: Max-SAT resolution steps applied on the formula Φ_2 in Example 5a.

b) Transformation by max-resolution with the SIR heuristic: If we use the SIR heuristic to transform ψ, a score is given to each literal of G. Fig. 6a shows the implication graph G with in brackets the literal's scores.

Initially, there are two variables with the lowest score 2, x_1 and x_4. The algorithm applies Max-SAT resolution between the predecessor of x_1, c_1, and its successor c_5. It produces the intermediary resolvent $cr_{\langle x_1 \rangle} = \{\overline{x}_3, x_5\}$, then it removes x_1 from the implication graph, it replaces the tag c_5 of the arc (x_3, x_5) by $cr_{\langle x_1 \rangle}$ and it updates the scores of x_3 and x_5. Fig. 6b shows the modified implication graph, with in bold the replaced arcs and the updated scores.

Then, it takes the next variable of score 2, x_4, and it applies Max-SAT resolution between its predecessor and successor (c_4 and c_6 respectively). It updates the implication

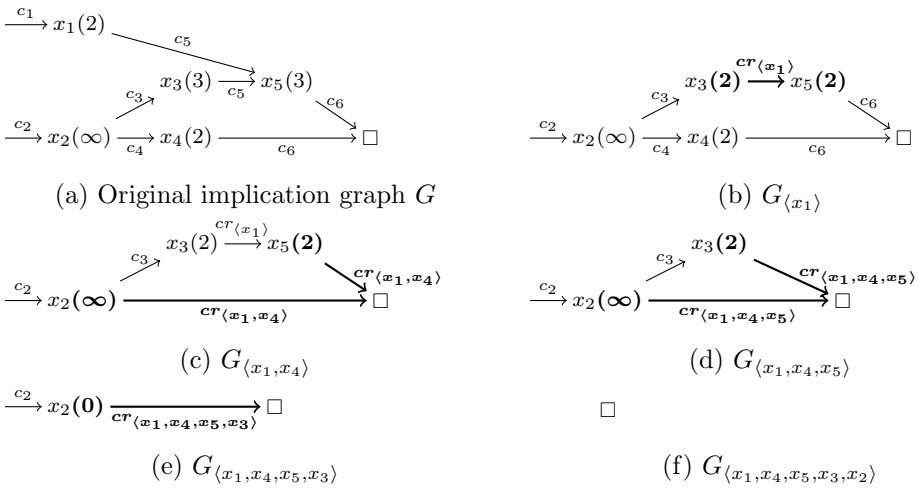


Figure 6: Evolution of the implication graph during the transformation of the formula Φ_2 of Example 5b.

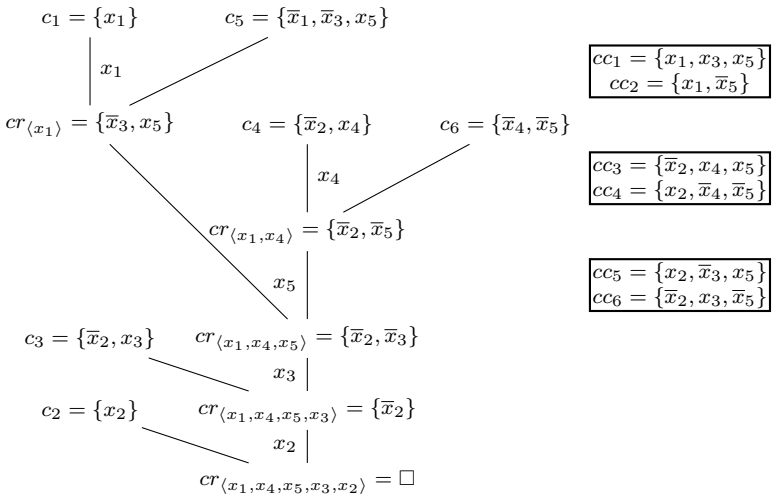


Figure 7: Max-SAT resolution steps applied on the formula Φ_2 in Example 5b.

graph and the variable scores (Fig. 6c). It applies the same treatment on x_5 (Fig. 6d) then x_3 (Fig. 6e). At this point, the two successors of x_2 have been merged and Max-SAT resolution can be applied between its predecessor c_2 and its new successor, the intermediary resolvent $cr_{\langle x_1, x_4, x_5, x_3 \rangle}$. This last resolution step produces an empty clause and, after its updating, the implication graph does not contain any more variable (Fig. 6f). Fig. 7 shows the max-resolution steps applied on the clauses of ψ.

We obtain the transformed formula $\Phi_2'' = \{\square, cc_1, \ldots, cc_6\}$ which contains, besides the empty clause $\square$, six clauses (five of size three and one of size two). There are two clauses

less than with the classical order of application of the max-resolution steps and the clause sizes are smaller. It is also interesting to observe that the max-resolution steps are no longer applied in a topological order of the unit propagation steps.

4.3 Increased Learning

We have seen that the most recent BnB solvers apply a limited form of learning in the sub-part of the search tree on the parts of the IS which match some patterns. Two of these patterns ($P1$ and $P2$) produce, after transformation by max-resolution, a unit resolvent clause. The goal of memorizing such transformations is twofold. On the one hand, it reduces the number of redundant propagations and on the other hand, it may empower the detection of inconsistencies in lower nodes of the search tree since more unit clauses are available for applying unit propagation.

We have proposed to extend the amount of learning performed by BnB solvers by considering more patterns which produce unit resolvent clauses when transformed by max-resolution [5]. These patterns can be formally defined as follows.

Definition 1 (Unit Clause Subset (UCS))**.** Let Φ be a CNF formula. A unit clause subset (UCS) is a set $\{c_{i_1}, \ldots, c_{i_l}\} \subset \Phi$ with $\forall j \in \{1, \ldots, l\}$, $|c_{i_j}| > 1$ such that there exists an order of application of $l-1$ max-resolution steps on $c_{i_1}, \ldots, c_{i_l}$ which produces a unit resolvent clause. We denote the set of the UCS's patterns of size k by k-UCS.

Example 6. Below the patterns of the 3-UCS set:

$$(P2)\ \frac{\{\{x_1,x_2\},\{x_1,x_3\},\{\overline{x}_2,\overline{x}_3\}\}}{\{\{x_1\},\{x_1,x_2,x_3\},\{\overline{x}_1,\overline{x}_2,\overline{x}_3\}\}},\quad (P4)\ \frac{\{\{x_1,x_2\},\{\overline{x}_2,x_3\},\{x_1,\overline{x}_2,\overline{x}_3\}\}}{\{\{x_1\},\{\overline{x}_1,\overline{x}_2,x_3\}\}},$$
$$(P5)\ \frac{\{\{x_1,x_2\},\{x_1,x_3\},\{x_1,\overline{x}_2,\overline{x}_3\}\}}{\{\{x_1\},\{x_1,x_2,x_3\}\}},\quad (P6)\ \frac{\{\{x_1,x_2\},\{x_1,\overline{x}_2,x_3\},\{x_1,\overline{x}_2,\overline{x}_3\}\}}{\{\{x_1\}\}}$$

Since one of the goals of memorizing transformations which produce unit resolvent clauses is to increase the number of assignments made by unit propagation (SUP or FL), we do not consider the subsets of clauses which contain unit clauses. In the best case (if they contain only one unit clause), the transformation of such subsets lets the number of unit clauses of the formula unchanged. In the worst case (if they contain more than one unit clause), the transformed formula contains less unit clauses than the original one. Thus, the number of assignments made by unit propagation and consequently the number of detected IS may be reduced. We also make a distinction between the patterns of the k-UCS sets depending on the size of their clauses. We denote k^b-UCS the subset of k-UCS composed of the patterns which contain only binary clauses and k^t-UCS the subset composed of the patterns which contain at least one ternary clause. It should be noted that the patterns ($P1$) and ($P2$) presented in Section 3.3.2 belong respectively to the sets 2-UCS and 3^b-UCS.

We have shown empirically that our solver AHMAXSAT obtains better performance while memorizing the 2-3^t-4^t-5^t-UCS patterns (i.e. 2-UCS $\cup$ 3^t-UCS $\cup$ 4^t-UCS $\cup$ 5^t-UCS) [5].

The k-UCS patterns are detectable by analyzing the implication graph. Indeed, the clauses which are between the conflict and the *first unit implication point* (FUIP) [41] produce a unit resolvent clause if they transformed by max-resolution. This analysis can be done as follows. The falsified literals of the falsified clause are added to a queue Q. At each step, the most recently propagated literal is removed from Q and the falsified literals of its

propagation source are added to Q. The FUIP is found when Q contains only one literal. Solvers simply have to count the number and the sizes of clauses between the conflict and the FUIP (the falsified clause and the propagation sources encountered during the analysis) to know if they are in presence of a valid UCS. This does not change the complexity of the conflict analysis procedure and the computational overhead is negligible.

4.4 Local Max-Resolution

When they detect an IS, recent BnB solvers apply two kind of treatments depending on the structure of the IS: a max-resolution based transformation or the IS removal (ISR). The first technique keeps the formula equivalency but has some drawbacks: the size of the formula grows and it may reduce the capability to detect IS in the lower nodes of the search tree. The existing solvers use this transformation method to perform learning by keeping the changes in the sub-part of the search tree. To limit the impact of the drawbacks cited above, it is only applied in very specific cases. The other treatment method does not keep the formula equivalency. It is only used to ensure that each detected IS is counted only once during the LB computation. The changes induced by this method are only kept for the duration of the LB estimation (i.e. they are local to each node).

We have measured experimentally that 80% to 90% of the IS were transformed with IS removal. Starting from the assumption that the ISR gives a poor estimation of the inconsistencies of the formulas, we have proposed an alternative treatment which gives a more accurate LB estimation at the cost of a more time-consuming treatment [3]. We have replaced the temporary removal by the max-resolution based treatment, but instead of keeping the changes in the sub-tree, we restore the changes before the next decision. This way, the estimation of the inconsistencies made at each node might be better. The increase in the size of the formula will be limited and it will not hamper the detection of inconsistencies in lower nodes, since the changes are undone before each new decision. We call this new treatment method *local max-resolution*. The following example illustrates the effect of each method of dealing with IS.

Example 7. Let us suppose that we are at the level g of the search tree, and the current unweighted CNF formula is $\Phi_3 = \{c_1, \ldots, c_{14}\}$ with $c_1 = \{x_1\}$, $c_2 = \{\overline{x}_1, x_4\}$, $c_3 = \{\overline{x}_1, \overline{x}_5\}$, $c_4 = \{\overline{x}_4, x_7\}$, $c_5 = \{x_5, \overline{x}_7\}$, $c_6 = \{x_2\}$, $c_7 = \{\overline{x}_2, x_4\}$, $c_8 = \{x_3\}$, $c_9 = \{\overline{x}_3, x_5\}$, $c_{10} = \{\overline{x}_3, x_6\}$, $c_{11} = \{\overline{x}_6, \overline{x}_7\}$, $c_{12} = \{x_2, \overline{x}_8, \overline{x}_3\}$, $c_{13} = \{\overline{x}_2, \overline{x}_8, x_9\}$ and $c_{14} = \{\overline{x}_2, \overline{x}_9\}$.

a) Temporary Removal: The application of SUP* on Φ_3 leads to the following ordered propagation queue: $\langle x_1@c_1, x_4@c_2, \overline{x}_5@c_3, x_7@c_5\rangle$ (meaning x_1 is propagated by unit clause c_1, then x_4 by c_2, etc.). At this point, clause c_5 is empty. The implication graph corresponding to this situation is shown in Fig. 8a. The IS corresponding to this conflict is $\{c_1, c_2, c_3, c_4, c_5\}$. If we remove this subset from Φ_3, we obtain $\Phi'_3 = \{\square, c_6, \ldots, c_{14}\}$ and all the propagated assignments are undone.

The treatment of the remaining unit clauses leads to the following assignment: $\langle x_2@c_6, x_4@c_7, \overline{x}_9@c_{14}, x_3@c_8, x_5@c_9, x_6@c_{10}, \overline{x}_7@c_{11}, \overline{x}_8@c_{13}\rangle$. At this point, no empty clause has been found and there are no more unit clauses to propagate (see Fig. 8b). Then we go to the next decision level $g + 1$. The removed clauses are restored, the propagations undone and a new decision $x_8 = \mathit{true}$ is made. As previously, the variables $\langle x_1@c_1, x_4@c_2, \overline{x}_5@c_3,$

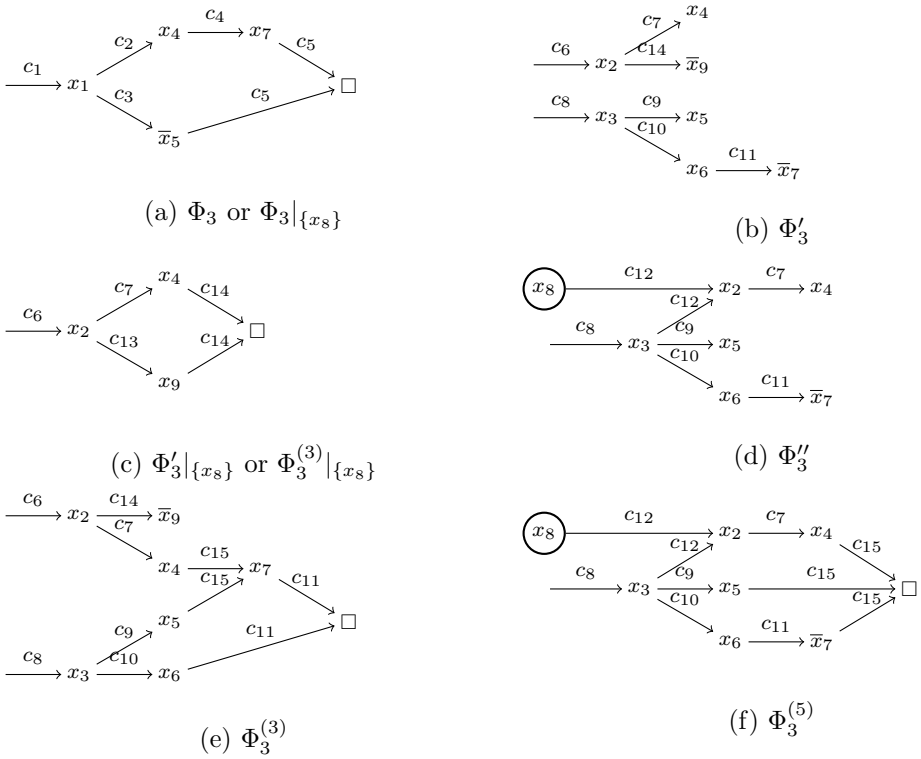


Figure 8: Implication graphs for each formula of the Example 7.

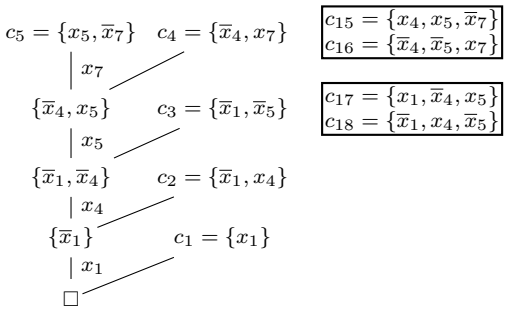


Figure 9: Max-resolution steps applied on the IS $\{c_1, c_2, c_3, c_4, c_5\}$ of the formula Φ_3 in Example 7.

$x_7@c_5\rangle$ are propagated, making c_5 empty (Fig. 8a), and we obtain Φ'_3 after removing the corresponding IS. Then we keep applying unit propagation: $\langle x_2@c_6, x_4@c_7, x_9@c_{13}\rangle$ making c_{14} empty (Fig. 8c). The corresponding IS $\{c_6, c_{13}, c_{14}\}$ is removed from the formula and we obtain $\Phi''_3 = \{\square, \square, c_7, \ldots, c_{12}\}$.

We continue applying unit propagation: $\langle x_3@c_8, x_2@c_{12}, x_5@c_9, x_6@c_{10}, x_4@c_7, \overline{x}_7@c_{11}\rangle$. Formula Φ_3'' contains no more unit clauses (Fig. 8d), and we can go to the next decision. The temporary removal of the IS detects one inconsistency at level g and two at level $g+1$.

b) Sub-tree max-resolution: As in the previous example, the application of unit propagation on Φ_3 leads to the following assignments $\langle x_1@c_1, x_4@c_2, \overline{x}_5@c_3, x_7@c_5\rangle$ (Fig. 8a). The clause c_5 is empty, and the IS $\{c_1, c_2, c_3, c_4, c_5\}$ can be processed. The application of the max-resolution based rule removes the clauses of the IS from the formula and adds an empty clause (the resolvent) and the following compensation clauses to the formula: $c_{15} = \{\overline{x}_4, \overline{x}_5, x_7\}, c_{16} = \{x_4, x_5, \overline{x}_7\}, c_{17} = \{\overline{x}_1, x_4, \overline{x}_5\}$ and $c_{18} = \{x_1, \overline{x}_4, x_5\}$. Fig. 9 shows the max-resolution steps applied during this transformation. We obtain the formula $\Phi_3^{(3)} = \{\square, c_6, \ldots, c_{18}\}$.

The propagations are undone and the treatment of the remaining unit clauses leads to the assignment $\langle x_2@c_6, x_4@c_7, \overline{x}_9@c_{14}, x_3@c_8, x_5@c_9, x_6@c_{10}, x_7@c_{15}\rangle$. At this point, the clause c_{11} is empty (Fig. 8e). The application of the sub-tree max-resolution removes the clauses of the IS $\{c_6, c_7, c_8, c_9, c_{10}, c_{11}, c_{15}\}$ from the formula and adds an empty clause (the resolvent) and the compensation clauses $c_{19} = \{x_4, \overline{x}_5, \overline{x}_6, \overline{x}_7\}$, $c_{20} = \{x_5, \overline{x}_6, \overline{x}_7\}$, $c_{21} = \{\overline{x}_4, \overline{x}_5, x_6, x_7\}$, $c_{22} = \{x_3, \overline{x}_4, \overline{x}_5, \overline{x}_6\}$, $c_{23} = \{\overline{x}_3, x_4, \overline{x}_5, x_6\}$, $c_{24} = \{\overline{x}_3, x_5, x_6\}$, $c_{25} = \{\overline{x}_3, x_4, x_5\}$, $c_{26} = \{x_3, x_4\}$ and $c_{27} = \{x_2, \overline{x}_4\}$. We obtain $\Phi_3^{(4)} = \{\square, \square, c_{12}, \ldots, c_{14}, c_{16}, \ldots, c_{27}\}$. The propagations are undone and no more unit clauses remain.

At the next decision level $g+1$, the formula transformations are kept and a new decision $x_8 = \mathit{true}$ is made. The formula contains no unit clauses, thus the number of inconsistencies stays unchanged (two). The max-resolution based transformation with changes kept in the sub-tree detects two inconsistencies at level g and two at level $g+1$.

c) Local max-resolution: As with the sub-tree max-resolution, two conflicts are detected at level g and after the transformation we obtain the formula $\Phi_3^{(4)} = \{\square, \square, c_{12}, \ldots, c_{14}, c_{16}, \ldots, c_{27}\}$. No more propagation can be made.

At the next decision level $g+1$, the propagations are undone and the original formula is restored. A new decision $x_8 = \mathit{true}$ is made. The first propagations made are similar to the ones of the previous decision level: $\langle x_1@c_1, x_4@c_2, \overline{x}_5@c_3, x_7@c_5\rangle$ (Fig. 8a). The same conflict is detected, which leads to the same transformation of the formula. The clauses c_1, c_2, c_3, c_4, c_5 are removed from the formula and $\square$, c_{15}, c_{16}, c_{17} and c_{18} are added to form $\Phi_3^{(3)} = \{\square, c_6, \ldots, c_{18}\}$.

The propagations are undone, and we can apply propagation again, leading to the following assignments: $\langle x_2@c_6, x_4@c_7, x_9@c_{13}\rangle$. At this point (Fig. 8c), the clause c_{14} is empty. The application of the max-resolution rule removes the clauses c_6, c_{13}, c_{14} and adds an empty clause $\square$ (no compensation clauses are added). The resulting formula is $\Phi_3^{(5)} = \{\square, \square, c_7, \ldots, c_{12}, c_{15}, \ldots, c_{18}\}$.

Again, the propagations are undone and we treat the remaining unit clauses: $\langle x_3@c_8, x_5@c_9, x_6@c_{10}, x_2@c_{12}, \overline{x}_7@c_{11}, x_4@c_7\rangle$. The clause c_{15} is empty (Fig. 8f), and we apply the max-resolution based rule to the corresponding IS $\{c_7, \ldots, c_{12}, c_{15}\}$. These clauses are removed, while an empty clause $\square$ and the following compensation clauses are added $c_{28} = \{x_2, \overline{x}_4, \overline{x}_5, x_7\}$, $c_{29} = \{\overline{x}_2, x_4, x_5, x_7\}$, $c_{30} = \{\overline{x}_2, x_4, \overline{x}_7\}$, $c_{31} = \{\overline{x}_2, \overline{x}_5, x_6, x_7\}$, $c_{32} = \{x_2, \overline{x}_5, \overline{x}_6, \overline{x}_7\}$, $c_{33} = \{x_5, \overline{x}_6, \overline{x}_7\}$, $c_{34} = \{\overline{x}_2, x_3, \overline{x}_5, \overline{x}_6\}$, $c_{35} = \{x_2, \overline{x}_3, x_5, \overline{x}_6\}$, $c_{36} = \{x_2, \overline{x}_3, x_6\}$ and $c_{37} = \{\overline{x}_3, x_5, x_6, \overline{x}_8\}$. We obtain $\Phi_3^{(6)} = \{\square, \square, \square, c_{16}, \ldots, c_{18}, c_{28}, \ldots, c_{37}\}$.

No more propagations are possible. The local max-resolution allows to detect two inconsistencies at level g and three at level $g+1$.

To conclude the example, the local max-resolution detects one more inconsistency than the temporary removal at levels g and $g+1$ and one more than the sub-tree max-resolution at level $g+1$.

4.5 Overview of ahmaxsat

In the remaining of this section, we give an overview of our solver ahmaxsat. Its general structure is similar to the one presented in Algorithm 1. It explores the whole search tree. At each node, it computes the LB and compares it to the UB to decide if it is worth exploring the current branch. As seen previously, ahmaxsat main particularities lie in the LB computation method. We describe this part below, with an emphasis on the interactions between the different methods used. We also detail how it uses HUP, PL and DUC to extend the assignment as well as its branching heuristic.

4.5.1 Inference Rules to Extend the Current Assignment

ahmaxsat uses at each node of the search tree the inference rules presented in Section 3 to extend the assignment. Contrary to the existing solvers (e.g. wmaxsatz, akmaxsat) which apply these rules only once before computing the LB, ahmaxsat dynamically applies HUP, PL and DUC during the LB computation. This way, variable values may be fixed higher in the search tree than in other solvers. Moreover, assigning more variables will reduce clauses which can be used by SUP and FL to detect IS. Thus, the LB estimation may be more accurate. Note that ahmaxsat uses the MPS scheme to apply dynamically HUP. If a literal l_i is such that $LB + \sum_{c_j=(\{l_i\},w_j)} w_j \geq UB$ (where the c_j are the "real" propagation sources of l_i), then l_i can be fixed to true by the HUP rule.

4.5.2 Lower bound computation

The lower bound computation function used by ahmaxsat is presented in Algorithm 2. It starts by trying to extend the current assignment I with the inference rules HUP, PL and DUC (line 5). Then, iteratively, it applies SUP* with the MPS scheme and applies a treatment to the detected IS (lines 7-10). At this point, SUP* cannot detect any conflict. The algorithm selects an unassigned variable from the candidate list (lines 12-13). It tries to detect IS by SUP* and transforms them (lines 14-17) until saturation. If IS have been detected in $\Phi|_{I\cup\{l\}}$, then their treatments have added new clauses which are unit under the assignment I. In this case, the algorithm goes back to the application of SUP* (line 7). Otherwise, it selects a new unassigned variable and applies FL (line 12). If the algorithm has applied FL to all the candidate variables without generating new unit clauses, it goes back to line 4 and checks if the assignment I can be extended again thanks to the inference rules. This whole process is iterated until no more conflict can be detected by SUP*, no more FL candidate remains and no more assignment extension can be made. The algorithm computes the LB value by counting the sum of the weights of the falsified clauses of the formula and it undoes the local transformations (lines 21-22).

The following functions are used by the algorithm. `apply_HUP_PL_DUC()` checks for each unassigned variable if one of the three inference rules HUP, PL and DUC can be applied.

Algorithm 2: AHMAXSAT LB computation

Data: A CNF formula Φ defined on a set of Boolean variables X and a partial assignment I.
Result: The transformed formula Φ and a positive integer LB.

```
1  begin
2      FL_candidate ← X \ I;
3      repeat
4          Φ' ← Φ;
           // Inference rules to extend the assignment
5          I ← apply_HUP_PL_DUC(Φ, X \ I);
6          repeat
               // IS detection by SUP*
7              G = (V, A) ← SUP*(Φ|_I);
8              while □ ∈ V do
9                  Φ ← analyze_and_treat_conflict(Φ, G = (V, A));
10                 G = (V, A) ← SUP*(Φ|_I);
               // Here SUP is saturated, we apply FL
11             repeat
12                 x ← FL_candidate;
13                 l ← literal of x which appear the most in binary clauses;
14                 G = (V, A) ← SUP*(Φ|_{I∪{l}});
15                 while □ ∈ V do
16                     Φ ← analyze_and_treat_conflict(Φ, G = (V, A));
17                     G = (V, A) ← SUP*(Φ|_{I∪{l}});
18             until (Φ|_I contains unit clauses) or (FL_candidate = ∅);
19         until Φ|_I does not contains unit clauses;
20     until Φ|_I = Φ'|_I;
21     LB ← Σ_{c_j=□∈Φ|_I} w_j;
22     Φ ← undo_local_changes(Φ);
23     return (Φ, LB);
```

Algorithm 3: AHMAXSAT conflict treatment

Data: A CNF formula Φ and a conflicting full implication graph $G = (V, A)$.
Result: The transformed formula Φ.

```
1  begin
2      G' = (V', A') ← full_implication_graph_to_implication_graph(G);
3      S ← build_order(G');
4      (ψ_st, ψ_tmp) ← detect_patterns(G');
5      Φ ← apply_max_resolution(Φ,G',S,ψ_st,ψ_tmp);
6      return Φ;
```

If so, it updates the assignment I. `SUP*()` applies simulated unit propagation with two separated queues as described in Section 3 and with the MPS scheme. When it detects a conflict or no more unit clauses remain, it returns the full implication graph. The function `analyze_and_treat_conflict()` is described below. `undo_local_changes()` simply undoes the transformations marked as local and returns the modified formula.

The function `analyze_and_treat_conflict()` (Algorithm 3) takes two parameters: the formula Φ and a conflicting full implication graph. It first converts the full implication graph G into a normal implication graph G' by choosing for each propagated variable participating in the conflict a single propagation source according to the SIS heuristic described in Section 4.1 (line 2). Then it computes the sequence S of the max-resolution steps according to the SIR heuristic presented in Section 4.2 (line 3). It analyzes the implication graph G' and separates the IS in two parts: the one matching the patterns P3 and/or one of the 2-3^t-4^t-5^t-UCS, ψ_{st}, and the other part ψ_{tmp}. Finally, it transforms the clauses of the IS by applying max-resolution steps according to the sequence S. The transformations are stored in two separated queues, one for the local changes (which are restored at the end of the lower bound computation) and one for the sub-tree changes (which are restored during the backtracks). The transformations of the clauses of ψ_{tmp} are kept in the first queue while the transformation of the ones of ψ_{st} are stored in the second queue. The function returns the transformed formula.

4.5.3 Branching Heuristic

ahmaxsat branching heuristic is close to the one of wmaxsatz. It chooses the unassigned variable which maximizes:

$$score(x_i) = lscore(x_i) * lscore(\overline{x_i}) + lscore(x_i) + lscore(\overline{x_i})$$

with

$$lscore(l_i) = 2 * \sum_{\substack{c_j \mid l_i \in c_j \\ \text{and } |c_j|=1}} w_j + 4 * \sum_{\substack{c_j \mid l_i \in c_j \\ \text{and } |c_j|=2}} w_j + 1 * \sum_{\substack{c_j \mid l_i \in c_j \\ \text{and } |c_j|=3}} w_j$$

The value given to the chosen variable x_i depends on the $lscore()$ values of its literals. If $lscore(x_i) \geq lscore(\overline{x_i})$ then x_i is set to true otherwise it is set to false.

4.5.4 Hard Clauses Handling

Except the hard unit propagation inference rule, our solver does not apply any particular treatment on hard clauses. Its behavior on the hard part of the instances can be compared to the one of a classical DPLL solver: it explores the search tree by using (hard) unit propagation to extend the current assignment. When a conflict is detected, a simple backtrack is performed. Note that unsatisfiable formulas are detected when the optimum value obtained is greater than the sum of the soft clause weights.

On partial instances, it would probably be beneficial for ahmaxsat performance to integrate some recent SAT solving techniques. Among the methods which could be applied in a BnB Max-SAT solver, we can cite clause learning, backjumping and activity-based branching heuristic [20, 41]. These methods, which are used in conflict driven clause learning solvers (CDCL), have shown their efficiency for solving SAT crafted and industrial instances.

To the best of our knowledge, no recent BnB solver for Max-SAT includes such techniques (except MiniMaxSat [20, 53], which is build on top of the CDCL solver MiniSAT). We plan to incorporate them in AHMAXSAT in a near future.

5. Experimental Study

We present in this section the results of the experimental study we have performed. This study is divided into two parts. In the first part, we evaluate the impact on our solver of the components presented in Section 4 (except the multiple propagation source scheme). The second part is dedicated to the comparison of AHMAXSAT with some of the best performing complete Max-SAT solvers.

Instance Sets We use in this experimental study two sets of instances. The first one is composed of all the 1776 random and crafted instances of the Max-SAT Evaluation 2014 (MSE 2014) benchmark. These instances are divided into three categories: Max-SAT (ms), partial Max-SAT (pms) and weighted partial Max-SAT (wpms). Note that we do not include industrial instances. Existing branch and bound solvers (including ours and except MiniMaxSat) are notoriously inefficient on these instances. Most of them rely on statically allocated data structures and thus they are not designed to handle such huge instances. Even when they do, their performance are poor compared to the ones of the iterative SAT based solvers[1]. In our opinion, this is mainly due to their inability to exploit the structural properties of the instances to guide the exploration of the search space. For these reasons, we do not include any industrial instance in the experiments presented below.

More than 80% of the random and crafted instances of the Max-SAT Evaluation 2014 are treated successfully by current best performing BnB solvers. The remaining 20% are very hard, and even with a cutoff extended to 7200 seconds, only a handful of them are solved. To strengthen the comparison between solvers, we have generated max2sat and max3sat random instances, both unweighted and weighted[2,3]. We have increased progressively the number of variables and clauses of the instances of the MSE 2014 until reaching the limits of the current solvers (i.e. when no instance is solved by any solver). It gives us 3000 instances with a number of variables and clauses varying respectively from 120 to 200 and from 1200 to 2600 for the max2sat ones and respectively from 70 to 110 and from 700 to 1500 for the max3sat ones. We have also generated maxcut crafted instances using the same methodology. The number of variables and clauses of these last instances varies from 140 to 220 and from 1200 to 1600. We have not found any instance generator which produces partial Max-SAT instances.

Experimental Protocol All the experiments are performed on machines equipped with Intel Xeon 2.4 Ghz processors and 24 Gb of RAM and running under a GNU/Linux operating system. The cutoff time is fixed to 1800 seconds per instance and the maximum amount of memory available for each solver is limited to 3.5 Gb. Note that these values are similar to the ones used in the Max-SAT Evaluation.

1. See for instance the Max-SAT Evaluation 2014 results available from http://www.maxsat.udl.cat/14/results
2. The generators are available from http://web.udl.es/usuaris/m4372594/jair-generators
3. The generated instances are available from http://www.lsis.org/habetd/Djamal_Habet/MaxSAT.html

Our solver ahmaxsat[4] is implemented from scratch in C, and compiled with gcc. The variant of ahmaxsat presented in this paper is numbered 1.68 while the ones presented to the Max-SAT Evaluation 2013 and 2014 are numbered 1.16 and 1.55, respectively. ahmaxsat 1.16 included early versions of the multiple propagation source scheme and of the local max-resolution treatment. Since then, we have rewritten almost entirely our solver. The version 1.55 also includes the two other components presented in this paper: the SIR heuristic and the UCS extended learning scheme. Changes brought by the 1.68 version over the 1.55 one are rather small : some limited code rewriting and memory optimization. Figure 10 compares the number of solved instances and the cumulative solving time of these three versions on the MSE 2014 instances. It shows that the performance gap between versions 1.16 and 1.55 is huge. Thanks to the memory optimization, version 1.68 solves few more instances than the 1.55 one but their performance on the majority of the benchmark instances are similar.

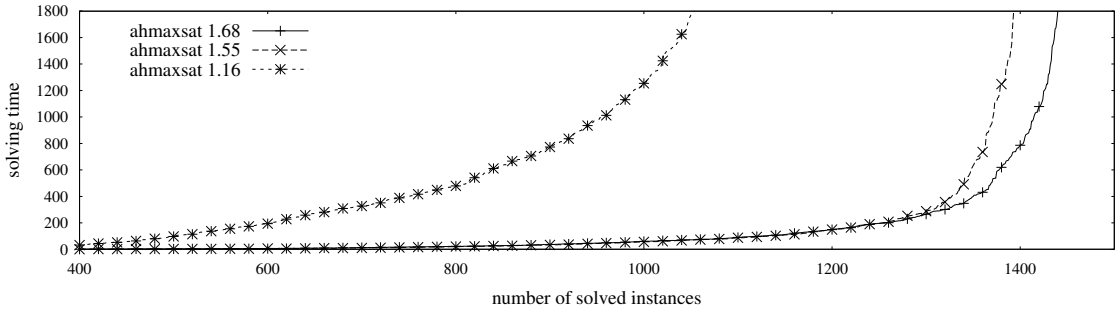


Figure 10: Comparison of the cumulative run times of the different ahmaxsat versions on the random and crafted instances of the Max-SAT Evaluation 2014.

5.1 Impact of ahmaxsat Components

We first study the impact of the new components of ahmaxsat on its behavior. We consider five variants of our solver:

- ahmaxsat-all includes all the components presented in Section 4.
- ahmaxsat-ucs+mrl includes the unit clauses subset extended learning scheme and the local max-resolution treatment but not the SIR heuristic.
- ahmaxsat-mrl+sir includes the local max-resolution treatment and the SIR heuristic but not the unit clause subsets extended learning scheme.
- ahmaxsat-ucs+sir includes the unit clauses subset extended learning scheme and the SIR heuristic but not the local max-resolution treatment.
- ahmaxsat-none includes neither the local max-resolution treatment, nor the unit clauses subset extended learning scheme nor the SIR heuristic.

Note that all these variants include the multiple propagation source scheme as well as the SIS heuristic. The propagation scheme is a core component of our solver, and many data

4. The version 1.55 of ahmaxsat is available from http://www.lsis.org/habetd/Djamal_Habet/MaxSAT.html

structures depend on it. Implementing a variant which uses the first predecessor scheme would have required important changes in the core structure of AHMAXSAT. Moreover, the performance of such a variant would highly depend on the amount of time spent on optimizing it. Thus, we do not present such a variant in this paper. Interested readers can refer to the empirical evaluation of the multiple propagation source scheme presented in [4].

The results obtained by the variants of AHMAXSAT on the MSE 2014 instances are presented in Table 1. For each variant, the number of solved instances is given with in brackets the average solving time. We can first observe that on the whole benchmark, the variant including all the novel components solves more instances in a lower average solving time than the other variants. Especially, AHMAXSAT-ALL solves respectively 34, 2, 49 and 51 instances more than AHMAXSAT-UCS+MRL, AHMAXSAT-MRL+SIR, AHMAXSAT-UCS+SIR and AHMAXSAT-NONE. The average solving time is reduced by respectively 9.1%, 10.2%, 33.9% and 41.2%. Figures 11 and 12 compare for each instance respectively the solving time and the number of nodes of AHMAXSAT-ALL with the ones of the other variants. It shows that the variant AHMAXSAT-ALL explores less nodes than AHMAXSAT-NONE one on almost all the benchmark instances (Figure 11d). Consequently, the solving time is reduced on large majority of them (Figure 12d). Let us now discuss the impact of each component on AHMAXSAT behavior and the interactions between them.

Local Max-resolution The transformations of inconsistent subsets (IS) which do not match the learning criteria by local max-resolution improves the quality of the lower bound estimation and thus the number of explored nodes is reduced on almost every instance of the benchmark (Figure 11c). The impact of this improvement on the solving time is partially compensated by the computational overhead induced by the intensive application of the max-resolution rule and by the increased number of propagation steps performed and IS detected at each node. Nevertheless, the solving time is reduced on a large majority of the benchmark instances (Figure 12c). On the whole benchmark, it is reduced by more than 30% (Table 1, AHMAXSAT-ALL vs. AHMAXSAT-UCS+SIR).

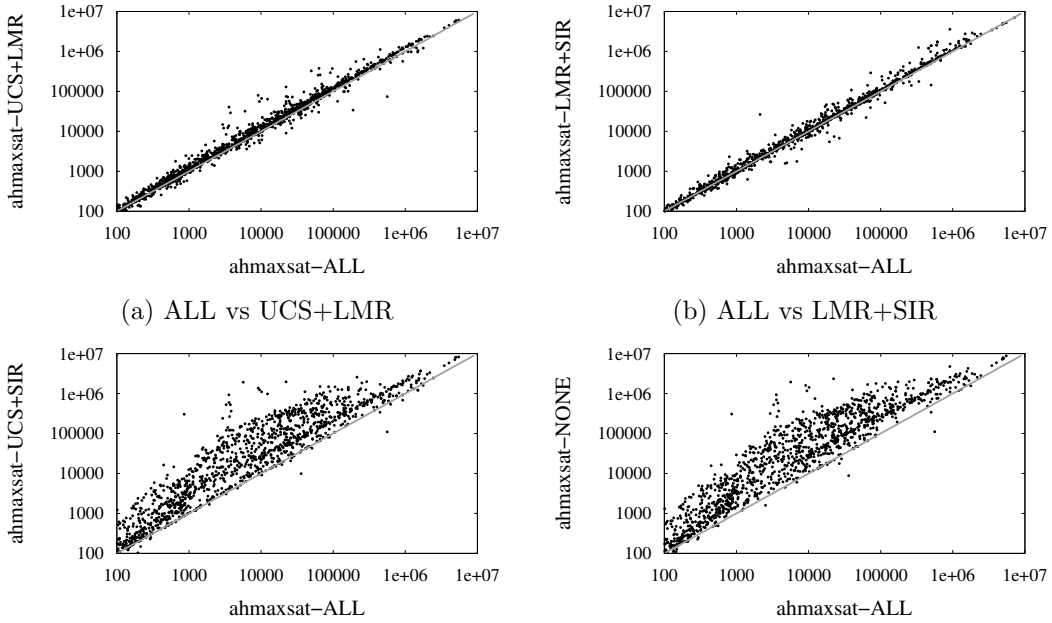
If we look at the results for each instance classes, we can observe some disparities. The variants using the local max-resolution transformation (AHMAXSAT-ALL, AHMAXSAT-UCS+MRL and AHMAXSAT-MRL+SIR) are better on almost all the unweighted instances (ms and pms lines). The results are more mixed on weighted instances (wpms lines). The variants without local max-resolution (AHMAXSAT-UCS+SIR and AHMAXSAT-NONE) give significantly better results on several of these instance classes.

Both the SIR heuristic and the UCS learning scheme impact the efficiency of local max-resolution. The reduction of the number and sizes of the compensation clauses reduces the time dedicated to the application of the max-resolution rule. Similarly, extending the amount of learning performed reduces the redundancy in the IS detection and transformation. In both cases, it reduces the time overhead due to the application of local max-resolution.

SIR heuristic The SIR heuristic reduces the number and sizes of the added compensation clauses by in average respectively 30% and 20% (detailed results are not presented in this paper). It has two effects on the solver behavior. The first one is straightforward: smaller formula induces a faster exploration of the search tree and thus a reduction of the solving time. The second effect is due to the reduction of the compensation clause sizes,

Table 1: Results of the AHMAXSAT variants on the random and crafted MSE 2014 instances. For each variant, the number of solved instances is given with in brackets the average solving time.

Instance classes			#	AHMAXSAT				
				ALL	UCS+MRL	MRL+SIR	UCS+SIR	NONE
ms	crafted	bipartite	100	**100(90.74)**	100(101.07)	100(95.28)	99(373.15)	99(392.22)
		maxcut	67	**56(44.72)**	56(51.12)	56(63.14)	55(43.12)	55(46.65)
		set-covering	10	0(-)	0(-)	0(-)	0(-)	0(-)
	random	highgirth	82	7(1165.99)	6(1238.03)	**7(1139.55)**	7(1140.87)	7(1152.51)
		max2sat	100	100(**89.51**)	100(115.38)	100(107.7)	100(207.54)	100(256.97)
		max3sat	100	100(**231.18**)	100(265.93)	100(284.85)	100(248.6)	100(308.93)
		min2sat	96	96(**2.56**)	96(2.89)	96(2.66)	96(13.56)	96(13.7)
pms	crafted	frb	25	5(**154.16**)	5(156.64)	5(333.46)	5(167.57)	5(383.66)
		job-shop	3	0(-)	0(-)	0(-)	0(-)	0(-)
		maxclique	158	133(30.24)	132(18.84)	**133(28.7)**	132(20.29)	133(35.84)
		maxone	140	109(50.13)	108(44.77)	109(60.77)	110(58.78)	**111(64.43)**
		min-enc/kbtree	42	**34(476.23)**	24(599.54)	34(500.82)	15(718.06)	14(728.78)
		pseudo/miplib	4	2(0.02)	2(0.02)	2(0.02)	2(0.03)	2(0.03)
		reversi	44	**8(129.24)**	7(147.29)	**8(129.95)**	7(122.12)	7(138.47)
		scheduling	5	0(-)	0(-)	0(-)	0(-)	0(-)
	random	min2sat	60	58(186.84)	55(168.83)	**58(184.23)**	44(515.6)	44(518.18)
		min3sat	60	58(**249.93**)	58(311.65)	58(262.63)	52(436.26)	53(464.04)
		pmax2sat	60	60(**3.18**)	60(4.36)	60(3.49)	60(3.88)	60(4.6)
		pmax3sat/hi	30	30(**60.08**)	30(67.17)	30(65.05)	30(68.91)	30(81.02)
wpms	crafted	auction	40	40(115.49)	40(141.31)	40(123.48)	40(**90.56**)	40(98.99)
		CSG	10	4(452.34)	4(435.81)	4(**432.44**)	4(434.74)	4(447.9)
		frb	34	14(**58.74**)	14(64.27)	14(128.57)	14(68.1)	14(142.3)
		min-enc	74	**64(108.83)**	63(125.53)	64(109.06)	58(10.05)	58(9.97)
		pseudo/miplib	12	4(**158.77**)	4(306.37)	4(159.4)	4(293.32)	4(290.82)
		ramsey	15	4(44.79)	4(43.48)	4(44.17)	**5(341.53)**	5(344.78)
		random-net	32	3(572.45)	2(541.23)	1(1069.46)	**3(498.58)**	0(-)
		set-covering	45	25(46.41)	10(136.6)	**25(42.31)**	25(100.62)	25(100.85)
		wmaxcut	48	**46(49.41)**	46(54.37)	46(85.89)	44(38.9)	44(61.39)
	random	wmax2sat	120	**120(45.45)**	120(57.32)	120(51.58)	120(161.24)	119(196.89)
		wmax3sat	40	40(99.52)	40(117.27)	40(115.25)	40(**74.27**)	40(91.86)
		wpmax2sat	90	90(**5.52**)	90(7.5)	90(5.87)	90(5.76)	90(6.44)
		wpmax3sat/hi	30	30(84.09)	30(103.59)	30(88.38)	30(**63.61**)	30(72.14)
Overall			1776	**1440(96.87)**	1406(106.6)	1438(107.88)	1391(146.47)	1389(164.69)



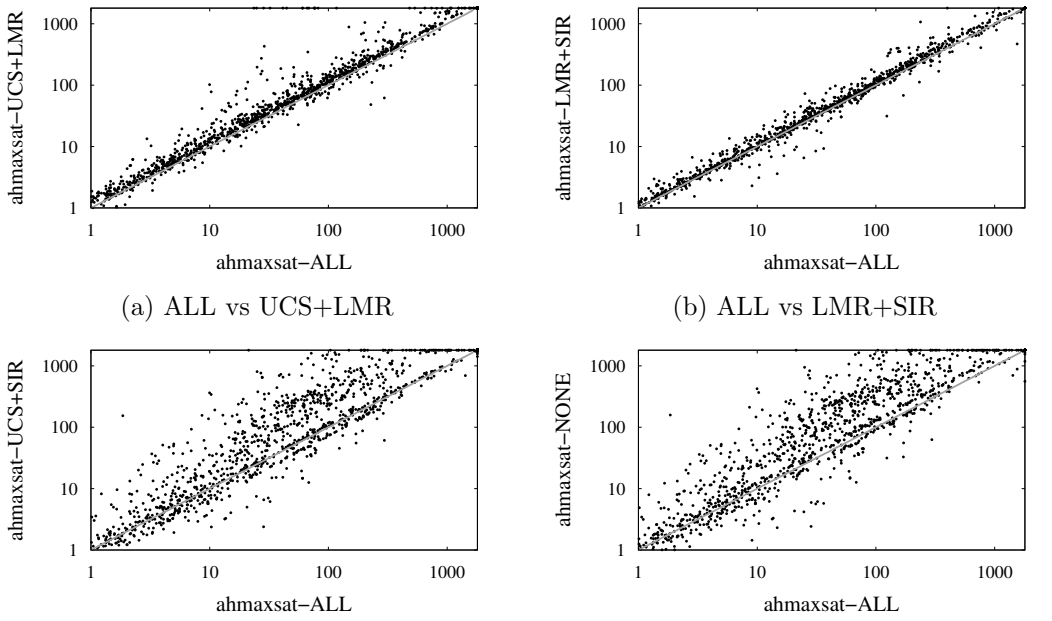
(a) ALL vs UCS+LMR

(b) ALL vs LMR+SIR

(c) ALL vs UCS+SIR

(d) ALL vs NONE

Figure 11: Comparison of the number of nodes explored by the AHMAXSAT variants. Each point represents an instance. All axis are in log scale.



(a) ALL vs UCS+LMR

(b) ALL vs LMR+SIR

(c) ALL vs UCS+SIR

(d) ALL vs NONE

Figure 12: Comparison of the solving time of the AHMAXSAT variants. Each point represents an instance. All axis are in log scale.

which improves the capability of unit propagation to detect IS. The quality of the lower bound estimation is improved, and thus AHMAXSAT explores less nodes of the search tree (Figure 11a). The combination of these two effects reduces the solving time of our solver by in average 9.1%.

One can note that the improvement brought by the SIR heuristic is relatively homogeneous. On almost all the instance classes, the performance of AHMAXSAT are improved by the SIR heuristic (Table 1, AHMAXSAT-ALL vs. AHMAXSAT-UCS+MLR). This observation is confirmed by the solving time comparison instance per instance (Figure 12a). These results show that the impact of the SIR heuristic on the solver behavior is not significantly affected by the discrepancies (in structure, weight or hard/soft clauses) between the instance classes.

Obviously, there is a strong correlation between the frequency of application of the max-resolution rule and the impact of the SIR heuristic on the solver performance. The SIR heuristic shows its full worth only in combination with local max-resolution. Without it, very few max-resolution transformations are performed and the effect of the SIR heuristic is marginal. Conversely, the UCS learning scheme reduces the redundancy in the LB computation, thus less max-resolution transformations are performed and the effect of the SIR heuristic on the solver performance decreases slightly.

Extended Learning Scheme The UCS learning scheme reduces the redundancy in the lower bound computation, thus the time spent on each node of the search tree is reduced. It also lets more unit clauses available for applying unit propagation. More IS are detected at each node and the LB estimation quality is improved. Thus, the number of explored nodes is slightly reduced (Figure 11a). These two effects combined led to a significant reduction of the average solving time by 10.2%.

However, we can observe variations in these results both from the instance per instance solving time comparison (Figure 12a) and from the detailed results (Table 1, AHMAXSAT-ALL vs. AHMAXSAT-MLR+SIR). On some instance classes (e.g. wpms/crafted/ramsey or wpms/crafted/random-net), UCS are frequently found and the performance are drastically improved. On other instance classes, there is a moderate improvement and even a slight performance loss in some cases.

It should be noted that local max-resolution does not impact significantly the behavior of the extended learning scheme. Conversely, the small compensation clauses produced with the SIR heuristic are more likely to appear in UCS patterns, and thus it increases slightly the amount of learning performed with the UCS-based scheme.

5.2 AHMAXSAT vs. State of the Art

In the second part of this experimental evaluation, we compare two variants of our solver (the one including all the novel components, AHMAXSAT-ALL, and the one which does not include any of them, AHMAXSAT-NONE) to some of the best performing complete solvers. We consider the BnB solvers MiniMaxSat [26, 27], two variants of MAXSATZ [37, 40] (WMAXSATZ 2009 and WMAXSATZ 2013) and AKMAXSAT [33]. All have been regularly well ranked during the previous Max-SAT evaluations[5]. We also include in this experiment three iterative SAT solvers, MAXHS [17, 18, 19], EVA500 [47] and OPEN-WBO [44], and

5. Results are available at http://www.maxsat.udl.cat

two solvers based on reformulation in integer linear programming (ILP), ILP-2013 [10] and SCIP-MAXSAT [7]. All these solvers were ranked in the top three on some of the categories of the Max-SAT Evaluation 2014.

5.2.1 MSE 2014 Benchmark

We have run all the solvers on the MSE 2014 random and crafted instances. Results are presented in Table 2, which gives for each solver the number of solved instances and in bracket the average solving time. For each instance class, the best result is presented in bold and the best result among the BnB solvers is underlined.

We can observe that on the whole benchmark, BnB solvers solve more instances than the iterative SAT and ILP-based solvers. They clearly dominate the random instance classes as well as the unweighted, non-partial crafted one. The results are more mixed on the partial crafted (unweighted and weighted) instances, where non-BnB solvers give better results on the majority of the instance classes. It should be noted however that even on these instance classes, BnB solvers perform honorably and the number of instances they solve is not very far from the ones of the non-BnB solvers. Conversely, non-BnB solvers manage to solve only few random instances, which explains the broad difference in the overall results.

The mixed performance of BnB solvers on partial crafted instances may be due to their inefficiency on the partial part of the instances. The difficulty of the majority of the instances from the partial crafted (unweighted and weighted) classes lies in their hard part rather than in their soft one. Thus, to be efficient on such instances, solvers must be able to quickly walk through the assignment satisfying the hard part of the instances. In order to do so, BnB solvers should use recent SAT solving techniques such as clause learning, backjumping or activity-based branching heuristics. To the best of our knowledge, the BnB solvers considered in this paper do not implement such methods (except MiniMaxSat which is build on top of the SAT solver MINISAT [20, 53]).

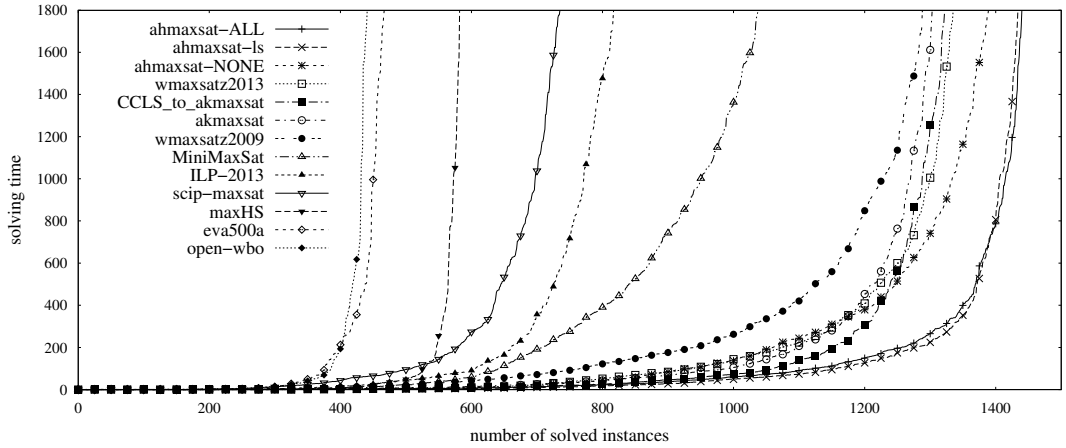
Figure 13a shows the cumulative run-time of the solver of the whole benchmark. We can see that our solver AHMAXSAT-ALL is the more efficient, followed by the other BnB solvers. We also add in this figure AHMAXSAT-LS and CCLS_TO_AKMAXSAT which are variants of AHMAXSAT and AKMAXSAT enhanced with a pre-processing phase where a local search algorithm is used to compute the first UB value. AHMAXSAT-LS uses the local search solver UBCSAT [55] while CCLS_TO_AKMAXSAT uses CCLS [15]. One can note that these variants are slightly faster than the respective original solvers. However, if CCLS_TO_AKMAXSAT manages to solve more instances than AKMAXSAT, it is not the case of AHMAXSAT-LS. This may be due to slight changes in the branching orders induced by the new UB values.

On the crafted instances (Figure 13b), the BnB solver MiniMaxSat solves slightly more instances than AHMAXSAT-ALL even if its solving time is higher. The next best performing solvers are WMAXSATZ 2013 followed by the two ILP-based solvers ILP-2013 and SCIP-MAXSAT. Random instances are dominated by the BnB solvers (Figure 13c). Our solver is the more efficient, followed by AKMAXSAT, WMAXSATZ 2013 and WMAXSATZ 2009. One can note that MiniMaxSat performs poorly on the random instances compared to the other BnB solvers.

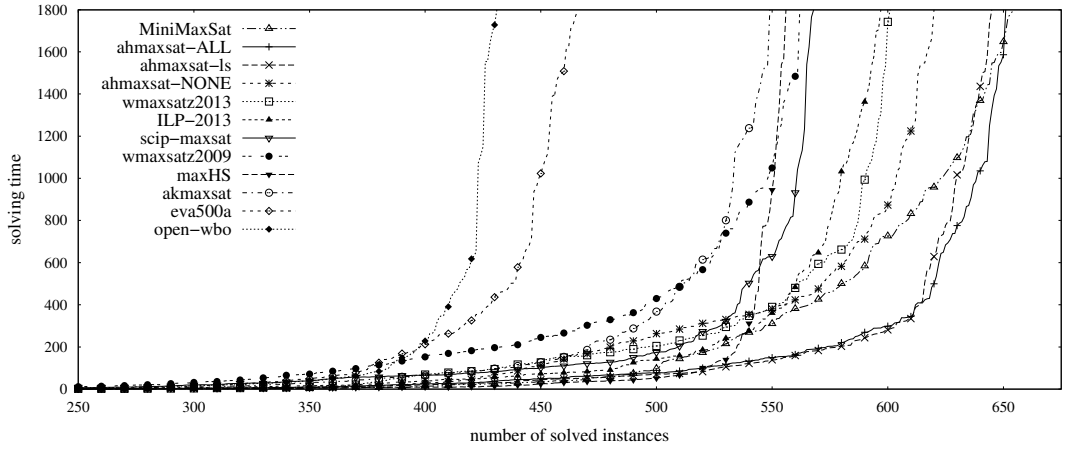
These results are in sync with the ones of the MSE 2014. Branch and bound solvers dominate the random and non-partial crafted categories while ILP-based solvers dominate

Table 2: Solver results on the random and crafted instances of the Max-SAT Evaluation 2014. For each solver, the number of solved instances is given with in brackets the average solving time.

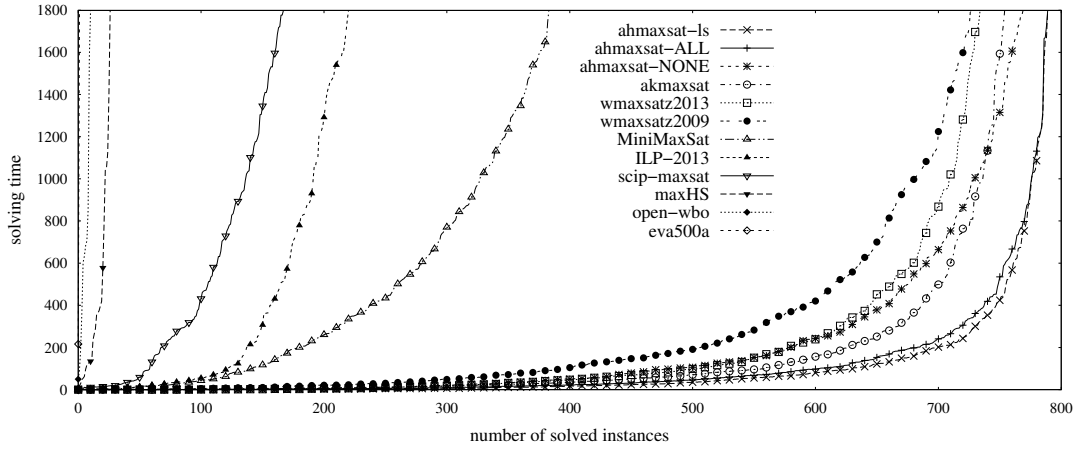
		Instance classes	#	AHMAXSAT-ALL	AHMAXSAT-NONE	WMAXSATZ 2013	AKMAXSAT	WMAXSATZ 2009	MINIMAXSAT	ILP-2013	SCIP-MAXSAT	MAXHS	EVA500A	OPEN-WBO
ms	crafted	bipartite	100	**100(90,7)**	99(392,2)	99(295)	100(105,2)	99(270,2)	85(641)	0(-)	0(-)	0(-)	0(-)	0(-)
		maxcut	67	**56(44,7)**	55(46,7)	55(56,9)	55(29,8)	55(97,5)	55(143,4)	30(187,3)	24(164)	6(0,7)	9(119,8)	11(86,5)
		set-covering	10	0(-)	0(-)	0(-)	0(-)	0(-)	0(-)	0(-)	0(-)	**1(46,3)**	0(-)	0(-)
	random	highgirth	82	7(1166)	**7(1152,5)**	0(-)	1(1737,3)	0(-)	0(-)	0(-)	0(-)	0(-)	0(-)	0(-)
		max2sat	100	**100(89,5)**	100(257)	100(177,9)	100(124,4)	97(304,4)	8(1023,4)	0(-)	0(-)	0(-)	0(-)	0(-)
		max3sat	100	100(231,2)	100(308,9)	100(251,6)	**100(202,1)**	97(383,9)	56(536,7)	2(1609,7)	5(1077)	0(-)	0(-)	0(-)
		min2sat	96	**96(2,6)**	96(13,7)	96(9,8)	96(6,7)	77(186,5)	61(338,1)	80(116)	60(297,3)	17(481,4)	0(-)	0(-)
pms	crafted	frb	25	5(154,2)	5(383,7)	5(178,2)	5(467)	0(-)	5(265)	11(216,7)	12(295,2)	14(347,4)	**25(141,8)**	21(289,7)
		job-shop	3	0(-)	0(-)	0(-)	0(-)	0(-)	1(85,5)	0(-)	0(-)	**3(32,4)**	3(132,8)	3(32,6)
		maxclique	158	133(30,2)	133(35,8)	133(17,8)	133(41,8)	109(84,3)	131(12,5)	132(70,9)	131(111,6)	**135(54,3)**	99(209,7)	85(121,2)
		maxone	140	109(50,1)	111(64,4)	136(34,3)	95(78,1)	137(128,2)	140(6,9)	138(76,7)	139(37,8)	**140(3,7)**	138(106,5)	139(28,4)
		min-enc/kbtree	42	34(476,2)	14(728,8)	10(363,9)	1(269,9)	8(508,7)	14(556,5)	42(197,8)	**42(69,9)**	2(99,7)	5(214,4)	6(26,4)
		pseudo/miplib	4	2(< 0, 1)	2(< 0, 1)	3(580,8)	3(320,3)	2(0,1)	2(0,4)	4(22,1)	4(31,1)	**4(3,2)**	4(54,9)	4(10,2)
		reversi	44	8(129,2)	7(138,5)	5(2,8)	7(421,4)	7(145,1)	32(56,7)	13(332)	15(307,1)	31(15,1)	32(23,8)	**33(13,8)**
		scheduling	5	0(-)	0(-)	0(-)	0(-)	0(-)	0(-)	0(-)	0(-)	0(-)	1(1117,5)	**1(505,6)**
	random	min2sat	60	**58(186,8)**	44(518,2)	21(786,1)	44(454,6)	43(401,6)	1(414)	48(329,4)	13(747,1)	5(270)	0(-)	7(600,9)
		min3sat	60	**58(249,9)**	53(464)	49(596,8)	43(602,1)	45(553,9)	6(1034)	13(898,2)	12(1156,6)	0(-)	0(-)	0(-)
		pmax2sat	60	**60(3,2)**	60(4,6)	59(3,1)	60(11,4)	60(8,5)	56(210,3)	12(574,4)	8(652,6)	0(-)	0(-)	0(-)
		pmax3sat/hi	30	30(60,1)	30(81)	**30(42,9)**	30(82,3)	30(71)	29(362,2)	9(1028,1)	12(736,3)	0(-)	0(-)	3(515,6)
wpms	crafted	auction	40	40(115,5)	40(99)	40(257,1)	40(265,7)	34(217,6)	40(38,8)	40(0,1)	40(0,5)	**40(< 0, 1)**	38(5,7)	20(4,7)
		CSG	10	4(452,3)	4(447,9)	1(404)	3(1055,7)	2(967,2)	8(321,8)	4(133,5)	4(35,4)	**10(5,9)**	7(98,5)	5(3,4)
		frb	34	14(58,7)	14(142,3)	14(65,5)	14(157,8)	9(12,2)	14(143,7)	19(124,1)	21(168,4)	23(228,2)	19(63,6)	**30(172,4)**
		min-enc	74	64(108,8)	58(10)	47(53,7)	41(188,4)	51(102,7)	59(42,1)	73(73,9)	74(23,1)	**74(0,8)**	58(3,8)	56(1,3)
		pseudo/miplib	12	4(158,8)	4(290,8)	**5(322,1)**	2(2)	3(131,8)	3(47,3)	3(128,9)	3(378,7)	3(12,1)	5(330,4)	3(3,7)
		ramsey	15	4(44,8)	**5(344,8)**	4(55,2)	4(50)	4(93,2)	4(75,4)	3(187,5)	3(171,8)	1(0,9)	1(0,7)	1(0,2)
		random-net	32	3(572,5)	0(-)	0(-)	1(1122,8)	0(-)	0(-)	25(90,8)	1(1092,1)	**32(9,4)**	12(89,6)	12(8,9)
		set-covering	45	25(46,4)	25(100,9)	0(-)	0(-)	0(-)	19(476,5)	**35(39,1)**	35(51,5)	35(48,6)	9(274,4)	1(0,3)
		wmaxcut	48	**46(49,4)**	44(61,4)	44(108,3)	45(39,6)	42(114,7)	43(162,7)	25(280,9)	20(251,5)	2(20,5)	1(48,6)	0(-)
	random	wmax2sat	120	**120(45,5)**	119(196,9)	120(139,3)	120(50,2)	118(277,4)	21(814,8)	2(819,3)	0(-)	0(-)	0(-)	0(-)
		wmax3sat	40	40(99,5)	40(91,9)	40(136,7)	**40(60)**	40(177,7)	33(561,2)	1(1699,8)	1(915,8)	0(-)	0(-)	0(-)
		wpmax2sat	90	**90(5,5)**	90(6,4)	89(7,2)	90(27,7)	90(21,7)	83(255,8)	42(171,2)	43(294,7)	4(25,9)	0(-)	0(-)
		wpmax3sat/hi	30	30(84,1)	30(72,1)	**30(52,8)**	30(78,4)	30(79,6)	29(451,4)	11(653,4)	13(415,2)	0(-)	1(216,6)	0(-)
		Overall	1776	**1440(96,9)**	1389(164,7)	1335(136,3)	1303(119,6)	1289(197,3)	1038(249,4)	817(164,5)	735(176,7)	582(52,6)	467(110,2)	441(76,6)



(a) Random and crafted instances.



(b) Crafted instances.



(c) Random instances.

Figure 13: Comparison of the cumulative solving time of the solvers on the MSE 2014 benchmark.

the partial-crafted ones. The iterative SAT solvers, which show mixed results in this study, usually dominate the industrial instance categories which are not considered here.

If we consider only the BnB solvers, we can see (Table 2) that the two variants of ahmaxsat solve more instances than any of the other compared solvers. The next best performing solver is wmaxsatz 2013, which solves 105 instances less than ahmaxsat-all. Then come akmaxsat, wmaxsatz 2009 and MiniMaxSat which solve respectively 137, 151 and 402 less instances than ahmaxsat-all. If we consider the average solving time on the whole benchmark, ahmaxsat-none, wmaxsatz 2013, akmaxsat, wmaxsatz 2009 and MiniMaxSat are respectively 70%, 41%, 23%, 104% and 157% slower than ahmaxsat-all.

We can observe some specificities among these solvers. wmaxsatz 2009 is globally efficient, without clearly dominating the other solvers on any instance category. wmaxsatz 2013 is also well balanced. It dominates other solvers on random partial max-3sat instance and performs honorably on crafted ones. akmaxsat performs well on non-partial random instances, especially on the max3sat ones. But it seems less competitive on (weighted) partial random instances and on crafted ones. MiniMaxSat is efficient on structured instances and especially on partial ones, but it is not competitive on random instances.

5.2.2 Extended Benchmark

We have tested the BnB solvers on the generated instances described above. The results are presented in Table 3. Our solver ahmaxsat-all is the most efficient, with 2472 instances solved over 4350. It is followed by akmaxsat which solves 63 instances less. Then come maxsatz2013, ahmaxsat-none, wmaxsatz2009 and MiniMaxSat with respectively 603, 778, 907 and 2070 less instances solved than ahmaxsat-all. One can note that ahmaxsat-all is significantly more efficient than the other solvers on random/max2sat instances (both unweighted and weighted) as well as on the crafted/maxcut ones. However it solves less random/max3sat instances than akmaxsat.

Figure 14 shows the cumulative run-time of each of the tested solvers. It shows that our solver is closely followed by akmaxsat, while wmaxsatz 2013, ahmaxsat-none and wmaxsatz 2009 are significantly less efficient. It also confirms the lack of efficiency of

Table 3: Results of ahmaxsat-all, MiniMaxSat, wmaxsatz 2009, wmaxsatz 2013, akmaxsat and ahmaxsat-none on the generated instances. For each solver, the number of solved instances is given with in brackets the average solving time.

	Instance classes	#	ahmaxsat-all	akmaxsat	wmaxsatz 2013	ahmaxsat-none	wmaxsatz 2009	MiniMaxSat
ms	crafted	1350	**412(457,4)**	405(516,3)	236(670,2)	206(707,8)	248(649,1)	146(766,1)
ms	random/max2sat	1150	**679(412,7)**	644(477,5)	540(477,5)	461(524,4)	431(547,5)	11(949,1)
ms	random/max3sat	350	224(505,4)	**237(508,7)**	217(478,7)	203(520,7)	184(554,1)	88(712,3)
wms	random/max2sat	1150	**921(340,9)**	857(339,2)	673(422,9)	588(515,7)	528(502,5)	60(959,2)
wms	random/max3sat	350	236(476,1)	**266(411,8)**	203(425,7)	236(467,9)	174(397,0)	97(596,8)
	Overall	4350	**2472(402,4)**	2409(424,8)	1869(469,8)	1694(527,4)	1565(523,6)	402(724,8)

MINIMAXSAT on random instances. The scatter plots presented in Figure 15 compare instance per instance the solving time of AHMAXSAT-ALL with the ones of the other solvers. It shows that AHMAXSAT-ALL is more efficient than WMAXSATZ 2009 and MINIMAXSAT on almost every instances and than AKMAXSAT and WMAXSATZ 2013 on the majority of them.

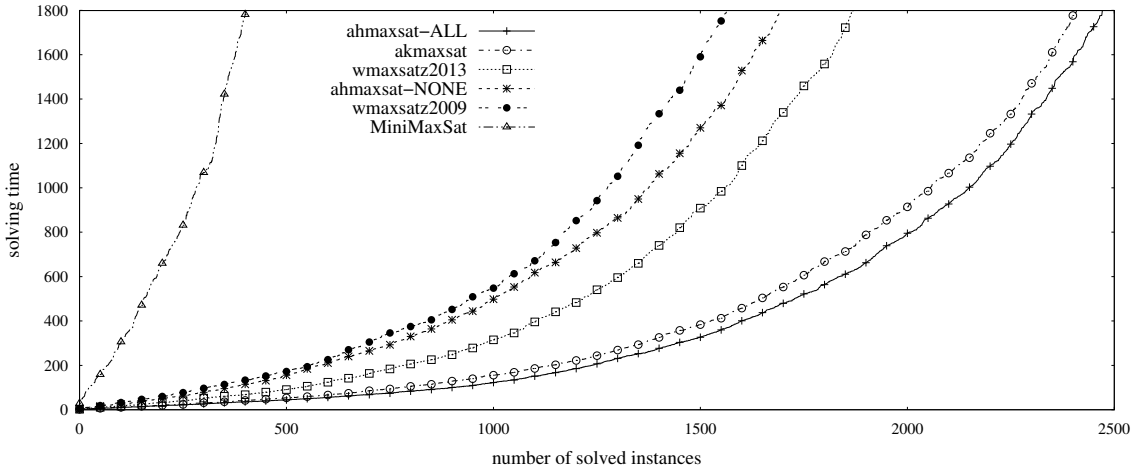


Figure 14: Comparison of the solver cumulative run times on the new generated instances.

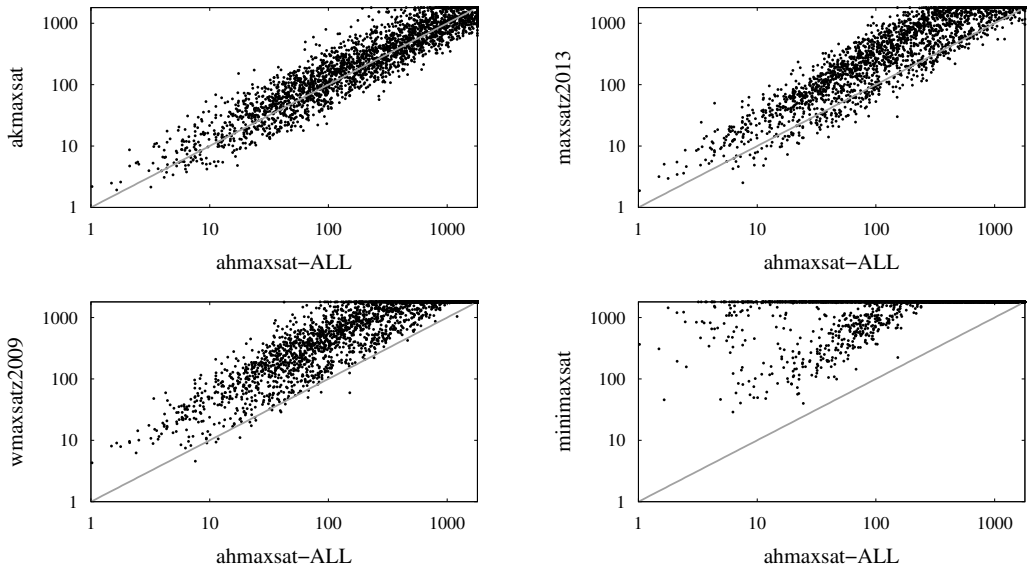


Figure 15: Comparison of the solving time of AHMAXSAT-ALL with the one of the other solvers on the new generated instances. All axis are in log scale.

6. Conclusion

We have given in this paper an overview of our solver AHMAXSAT which was ranked first in three of the nine categories of the Max-SAT Evaluation 2014. We have described the four main original components it includes and we have shown how they are used together in its LB estimation procedure. We have also given some other details such as the branching heuristic of AHMAXSAT or the way it implements the failed literals mechanism.

The results of the experimental evaluation we have conducted confirm the ones of the Max-SAT Evaluation 2014 and show that AHMAXSAT is more efficient than the existing BnB Max-SAT solvers on a large panel of the considered instances. Especially, our solver is very efficient on unweighted random max2sat and crafted instances (non-partial and partial).

Even if BnB solvers perform well on random and some crafted instances compared to the other kind of Max-SAT solvers, they are outperformed on structured (industrial and some crafted) instances. One possible way to improve BnB solvers on structured instances would be to increase the amount of learning by keeping some of the transformations performed during the LB computation in the upper nodes of the search tree. In order to do so, we have analyzed and characterized in a recent work the impact of the max-resolution rule on the unit propagation process efficiency [6]. We will continue in this direction in the future. We will also study the impact of the new learned information on the formula size and try to exploit it in designing new branching heuristics.

References

[1] André Abramé and Djamal Habet. Inference rules in local search for Max-SAT. In *Proceedings of the 24th IEEE International Conference on Tools with Artificial Intelligence (ICTAI 2012)*, pages 207–214. IEEE Computer Society, 2012.

[2] André Abramé and Djamal Habet. Efficient application of Max-SAT resolution on inconsistent subsets. In Barry O'Sullivan, editor, *Proceedings of the 20th International Conference on Principles and Practice of Constraint Programming (CP 2014)*, **8656** of *Lecture Notes in Computer Science*, pages 92–107. Springer International Publishing, 2014.

[3] André Abramé and Djamal Habet. Local max-resolution in branch and bound solvers for Max-SAT. In *Proceedings of the 26th IEEE International Conference on Tools with Artificial Intelligence (ICTAI 2014)*, pages 336–343. IEEE Computer Society, 2014.

[4] André Abramé and Djamal Habet. Maintaining and handling all unit propagation reasons in exact Max-SAT solvers. In Stefan Edelkamp and Roman Barták, editors, *Proceedings of the 7th Annual Symposium on Combinatorial Search (SOCS 2014)*, pages 2–9. AAAI Press, 2014.

[5] André Abramé and Djamal Habet. On the extension of learning for Max-SAT. In Ulle Endriss and João Leite, editors, *Proceedings of the 7th European Starting AI Researcher Symposium (STAIRS 2014)*, **241** of *Frontiers in Artificial Intelligence and Applications*, pages 1–10. IOS Press, 2014.

[6] André Abramé and Djamal Habet. On the resiliency of unit propagation to max-resolution. In Qiang Yang and Michael Wooldridge, editors, *Proceedings of the 24th International Joint Conference on Artificial Intelligence (IJCAI 2015)*, pages 268–274. AAAI Press, 2015.

[7] Tobias Achterberg. SCIP: solving constraint integer programs. *Mathematical Programming Computation*, **1**(1):1–41, 2009.

[8] Teresa Alsinet, Felip Manyà, and Jordi Planes. Improved branch and bound algorithms for Max-SAT. In Enrico Giunchiglia and Armando Tacchella, editors, *Proceedings of the 6th International Conference on the Theory and Applications of Satisfiability Testing (SAT 2003)*, **2919** of *Lecture Notes in Computer Science*, pages 502–518. Springer, 2003.

[9] Teresa Alsinet, Felip Manyà, and Jordi Planes. A Max-SAT solver with lazy data structures. In Christian Lemaître, Carlos Reyes, and Jesùs Gonzàlez, editors, *Proceedings of the 9th Ibero-American on Artificial Intelligence (IBERAMIA 2004)*, **3315** of *Lecture Notes in Computer Science*, pages 334–342. Springer Berlin / Heidelberg, 2004.

[10] Carlos Ansótegui and Joel Gabàs. Solving (weighted) Partial MaxSAT with ILP. In Carla P. Gomes and Meinolf Sellmann, editors, *Proceedings of the 10th International Conference on Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems (CPAIOR 2013)*, **7874** of *Lecture Notes in Computer Science*, pages 403–409. Springer, 2013.

[11] Carlos Ansótegui, Yuri Malitsky, and Meinolf Sellmann. MaxSAT by improved instance-specific algorithm configuration. In Carla E. Brodley and Peter Stone, editors, *Proceedings of the 28th AAAI Conference on Artificial Intelligence (AAAI 2014)*, pages 2594–2600. AAAI Press, 2014.

[12] Gilles Audemard, Lucas Bordeaux, Youssef Hamadi, Said Jabbour, and Lakhdar Sais. A generalized framework for conflict analysis. In Hans Kleine Bning and Xishun Zhao, editors, *Proceedings of the 11th International Conference on Theory and Applications of Satisfiability Testing (SAT 2008)*, **4996** of *Lecture Notes in Computer Science*, pages 21–27. Springer Berlin Heidelberg, 2008.

[13] María Luisa Bonet, Jordi Levy, and Felip Manyà. Resolution for Max-SAT. *Artificial Intelligence*, **171**(8-9):606–618, 2007.

[14] Shaowei Cai, Chuan Luo, John Thornton, and Kaile Su. Tailoring local search for partial MaxSAT. In Carla E. Brodley and Peter Stone, editors, *Proceedings of the 28th AAAI Conference on Artificial Intelligence (AAAI 2014)*, pages 2623–2629. AAAI Press, 2014.

[15] Shaowei Cai, Kaile Su, and Abdul Sattar. Local search with edge weighting and configuration checking heuristics for minimum vertex cover. *Artificial Intelligence*, **175**(910):1672–1696, 2011.

[16] Byungki Cha, Kazuo Iwama, Yahiko Kambayashi, and Shuichi Miyazaki. Local search algorithms for partial MAXSAT. In Benjamin Kuipers and Bonnie L. Webber, editors, *Proceedings of the 14th National Conference on Artificial Intelligence and 9th Innovative Applications of Artificial Intelligence Conference (AAAI/IAAI 97)*, pages 263–268. AAAI Press / The MIT Press, 1997.

[17] Jessica Davies and Fahiem Bacchus. Solving MAXSAT by solving a sequence of simpler SAT instances. In Jimmy Lee, editor, *Proceedings of the 17th International Conference on Principles and Practice of Constraint Programming (CP 2011)*, **6876** of *Lecture Notes in Computer Science*, pages 225–239. Springer Berlin Heidelberg, 2011.

[18] Jessica Davies and Fahiem Bacchus. Exploiting the power of MIP solvers in MaxSAT. In Matti Järvisalo and Allen Van Gelder, editors, *Proceedings of the 16th International Conference on Theory and Applications of Satisfiability Testing (SAT 2013)*, **7962** of *Lecture Notes in Computer Science*, pages 166–181. Springer, 2013.

[19] Jessica Davies and Fahiem Bacchus. Postponing optimization to speed up MAXSAT solving. In Christian Schulte, editor, *Proceedings of the 19th International Conference on Principles and Practice of Constraint Programming (CP 2013)*, **8124** of *Lecture Notes in Computer Science*, pages 247–262. Springer, 2013.

[20] Niklas Eén and Niklas Sörensson. An extensible sat-solver. In Enrico Giunchiglia and Armando Tacchella, editors, *Proceedings of the 6th International Conference on Theory and Applications of Satisfiability Testing (SAT 2003)*, **2919** of *Lecture Notes in Computer Science*, pages 502–518. Springer, 2003.

[21] Eugene C. Freuder and Richard J. Wallace. Partial constraint satisfaction. *Artificial Intelligence*, **58**(13):21–70, 1992.

[22] Zhaohui Fu and Sharad Malik. On solving the Partial MAX-SAT problem. In Armin Biere and Carla P. Gomes, editors, *Proceedings of the 9th International Conference on Theory and Applications of Satisfiability Testing (SAT 2006)*, **4121** of *Lecture Notes in Computer Science*, pages 252–265. Springer Berlin Heidelberg, 2006.

[23] Carla Gomes, Willem-Jan van Hoeve, and Lucian Leahu. The power of semidefinite programming relaxations for MAX-SAT. In J. Christopher Beck and Barbara Smith, editors, *Proceedings of the 3rd Interantional Conference on Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems (CPAIOR 2006)*, **3990** of *Lecture Notes in Computer Science*, pages 104–118. Springer Berlin / Heidelberg, 2006.

[24] Pierre Hansen and Brigitte Jaumard. Algorithms for the maximum satisfiability problem. *Computing*, **44**:279–303, 1990.

[25] Federico Heras and Javier Larrosa. New inference rules for efficient Max-SAT solving. In Anthony Cohn, editor, *Proceedings of the 21st National Conference on Artificial Intelligence (AAAI 2006)*, pages 68–73. AAAI Press, 2006.

[26] Federico Heras, Javier Larrosa, and Albert Oliveras. MiniMaxSat: A new weighted Max-SAT solver. In João Marques-Silva and Karem Sakallah, editors, *Proceedings of the 10th International Conference on Theory and Applications of Satisfiability Testing (SAT 2007)*, **4501** of *Lecture Notes in Computer Science*, pages 41–55. Springer Berlin / Heidelberg, 2007.

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

[28] Federico Heras and João Marques-Silva. Read-once resolution for unsatisfiability-based Max-SAT algorithms. In Toby Walsh, editor, *Proceedings of the 22nd International Joint Conference on Artificial Intelligence (IJCAI 2011)*, pages 572–577. AAAI Press, 2011.

[29] John N. Hooker and V. Vinay. Branching rules for satisfiability. *Journal of Automated Reasoning*, **15**(3):359–383, 1995.

[30] Robert G. Jeroslow and Jinchang Wang. Solving propositional satisfiability problems. *Annals of Mathematics and Artificial Intelligence*, **1**:167–187, 1990.

[31] Steve Joy, John Mitchell, and Brian Borchers. A branch and cut algorithm for Max-SAT and Weighted Max-SAT. In Dingzhu Du, Jun Gu, and Panos M. Pardalos, editors, *Satisfiability Problem: Theory and Applications, volume 35 of DIMACS series on Discrete Mathematics and Theoretical Computer Science*, pages 519–536. American Mathematical Society, 1997.

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

[33] Adrian Kügel. Improved exact solver for the Weighted MAX-SAT problem. In Daniel Le Berre, editor, *Proceedings of the 1st Pragmatics of SAT Workshop (POS 2010)*, **8** of *EasyChair Proceedings in Computing*, pages 15–27. EasyChair, 2010.

[34] Javier Larrosa and Federico Heras. Resolution in Max-SAT and its relation to local consistency in weighted CSPs. In Leslie Pack Kaelbling and Alessandro Saffiotti, editors, *Proceedings of the 19th International Joint Conference on Artificial Intelligence (IJCAI 2005)*, pages 193–198. Professional Book Center, 2005.

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

[36] Chu Min Li, Felip Manyà, Nouredine Mohamedou, and Jordi Planes. Exploiting cycle structures in Max-SAT. In Oliver Kullmann, editor, *Proceedings of the 12th International Conference on Theory and Applications of Satisfiability Testing (SAT 2009)*, **5584** of *Lecture Notes in Computer Science*, pages 467–480. Springer Berlin / Heidelberg, 2009.

[37] Chu Min Li, Felip Manyà, Nouredine Ould Mohamedou, and Jordi Planes. Resolution-based lower bounds in MaxSAT. *Constraints*, **15**(4):456–484, 2010.

[38] Chu Min Li, Felip Manyà, and Jordi Planes. Exploiting unit propagation to compute lower bounds in branch and bound Max-SAT solvers. In Peter van Beek, editor, *Proceedings of the 11th Interantional Conference on Principles and Practice of Constraint Programming (CP 2005)*, **3709** of *Lecture Notes in Computer Science*, pages 403–414. Springer Berlin / Heidelberg, 2005.

[39] Chu Min Li, Felip Manyà, and Jordi Planes. Detecting disjoint inconsistent subformulas for computing lower bounds for Max-SAT. In Anthony Cohn, editor, *Proceedings of the 21st National Conference on Artificial Intelligence (AAAI 2006)*, pages 86–91. AAAI Press, 2006.

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

[41] João Marques-Silva and Karem A. Sakallah. Grasp: A search algorithm for propositional satisfiability. *IEEE Transactions on Computers*, **48**(5):506–521, August 1999.

[42] Ruben Martins, Vasco Manquinho, and Inês Lynce. On partitioning for maximum satisfiability. In Luc De Raedt, Christian Bessière, Didier Dubois, Patrick Doherty, Paolo Frasconi, Fredrik Heintz, and Peter J. F. Lucas, editors, *Proceedings of the 20th European Conference on Artificial Intelligence (ECAI 2012)*, **242** of *Frontiers in Artificial Intelligence and Applications*, pages 913–914. IOS Press, 2012.

[43] Ruben Martins, Vasco Manquinho, and Inês Lynce. Community-based partitioning for MaxSAT solving. In Allen Van Gelder Matti Järvisalo, editor, *Proceedings of the 16th International Conference on Theory and Applications of Satisfiability Testing (SAT 2013)*, **7962** of *Lecture Notes in Computer Science*, pages 182–191. Springer Berlin / Heidelberg, 2013.

[44] Ruben Martins, Vasco Manquinho, and Inês Lynce. Open-WBO: A modular MaxSAT solver,. In Carsten Sinz and Uwe Egly, editors, *Proceedings of the 17th International Conference on Theory and Applications of Satisfiability Testing (SAT 2014)*, **8561** of *Lecture Notes in Computer Science*, pages 438–445. Springer International Publishing, 2014.

[45] António Morgado, Carmine Dodaro, and João Marques-Silva. Core-guided MaxSAT with soft cardinality constraints. In Barry O'Sullivan, editor, *Proceedings of the 20th International Conference on Principles and Practice of Constraint Programming (CP 2014)*, **8656** of *Lecture Notes in Computer Science*, pages 564–573. Springer, 2014.

[46] Matthew W. Moskewicz, Conor F. Madigan, Ying Zhao, Lintao Zhang, and Sharad Malik. Chaff: engineering an efficient sat solver. In *Proceedings of the 38th Design Automation Conference (DAC 2001)*, pages 530–535. ACM, 2001.

[47] Nina Narodytska and Fahiem Bacchus. Maximum satisfiability using core-guided MaxSAT resolution. In Carla E. Brodley and Peter Stone, editors, *Proceedings of the 28th AAAI Conference on Artificial Intelligence (AAA1 2014)*, pages 2717–2723. AAAI Press, 2014.

[48] Rolf Niedermeier and Peter Rossmanith. New upper bounds for maximum satisfiability. *Journal of Algorithms*, **36**(1):63 – 88, 2000.

[49] Christos H. Papadimitriou. *Computational complexity*. Addison-Wesley, 1994.

[50] Daniele. Pretolani. Efficiency and stability of hypergraph sat algorithms. In David S. Johnson and Michael A. Trick, editors, *Cliques, Coloring, and Satisfiability: Proceedings of the Second DIMACS Implementation Challenge*, **26** of *DIMACS Series in Discrete Mathematics and Theoretical Computer Science*, pages 479–498. American Mathematical Society, 1993.

[51] Haiou Shen and Hantao Zhang. Study of lower bound functions for max-2-sat. In Deborah L. McGuinness and George Ferguson, editors, *Proceedings of the 19th National Conference on Artificial Intelligence (AAAI 2004)*, pages 185–190. AAAI Press, 2004.

[52] Kevin Smyth, Holger Hoos, and Thomas Stützle. Iterated robust tabu search for max-sat. In Yang Xiang and Brahim Chaib-draa, editors, *Proceedings of the 16th Australian Conference on Artificial Intelligence (AI 2003)*, **2671** of *Lecture Notes in Computer Science*, pages 995–995. Springer Berlin / Heidelberg, 2003.

[53] Niklas Sörensson and Niklas Eén. Minisat 2.1 and minisat++ 1.0 — sat race 2008 editions. Technical report, 2008.

[54] Dawn M. Strickland, Earl Barnes, and Joel S. Sokol. Optimal protein structure alignment using maximum cliques. *Operations Research*, **53**(3):389–402, 2005.

[55] Dave Tompkins and Holger Hoos. UBCSAT: An implementation and experimentation environment for SLS algorithms for SAT and MAX-SAT. In Holger Hoos and David Mitchell, editors, *Proceedings of the 7th International Conference on Theory and Applications of Satisfiability Testing (SAT 2004)*, **3542** of *Lecture Notes in Computer Science*, pages 898–898. Springer Berlin / Heidelberg, 2004.

[56] Allen Van Gelder. Generalized conflict-clause strengthening for satisfiability solvers. In Karem Sakallah and Laurent Simon, editors, *Proceedings of the 14th International Conference on Theory and Applications of Satisfiability Testing (SAT 2011)*, **6695** of *Lecture Notes in Computer Science*, pages 329–342. Springer Berlin / Heidelberg, 2011.

[57] Hans van Maaren and Linda van Norden. Sums of squares, satisfiability and maximum satisfiability. In Fahiem Bacchus and Toby Walsh, editors, *Proceedings of the 8th International Conference on Theory and Applications of Satisfiability Testing*, **3569** of *Lecture Notes in Computer Science*, pages 294–308. Springer Berlin / Heidelberg, 2005.

[58] Richard Wallace and Eugene C. Freuder. Comparative studies of constraint satisfaction and davis-putnam algorithms for maximum satisfiability problems. In David S. Johnson and Michael A. Trick, editors, *Cliques, Coloring, and Satisfiability: Proceedings of the Second DIMACS Implementation Challenge*, **26** of *DIMACS Series in Discrete Mathematics and Theoretical Computer Science*, pages 587–615. American Mathematical Society, 1996.

[59] Zhao Xing and Weixiong Zhang. Maxsolver: An efficient exact algorithm for (weighted) maximum satisfiability. *Artificial Intelligence*, **164**(1-2):47–80, 2005.

[60] Hui Xu, Rob A. Rutenbar, and Karem Sakallah. sub-sat: a formulation for relaxed boolean satisfiability with applications in routing. In *Proceedings of the 2002 International Symposium on Physical Design (ISPD 2002)*, pages 182–187. ACM, 2002.

[61] Mutsunori Yagiura and Toshihide Ibaraki. Efficient 2 and 3-flip neighborhood search algorithms for the max sat. In Wen-Lian Hsu and Ming-Yang Kao, editors, *Proceedings of the 4th Annual International Conference on Computing and Combinatorics (COCOON 1998)*, **1449** of *Lecture Notes in Computer Science*, pages 105–116. Springer, 1998.

[62] Qiang Yang, Kangheng Wu, and Yunfei Jiang. Learning action models from plan examples using weighted MAX-SAT. *Artificial Intelligence*, **171**(2-3):107–143, 2007.