Proceedings of the Twenty-Fourth International Joint Conference on Artificial Intelligence (IJCAI 2015)

Exploiting the Structure of Unsatisfiable Cores in MaxSAT[*]

Carlos Ansótegui
DIEI, Univ. of Lleida, Spain
carlos@diei.udl.es

Frederic Didier
Google Paris, France
fdid@google.com

Joel Gabàs
DIEI, Univ. of Lleida, Spain
jgabas@diei.udl.es

Abstract

We propose a new approach that exploits the good properties of core-guided and model-guided MaxSAT solvers. In particular, we show how to effectively exploit the structure of unsatisfiable cores in MaxSAT instances. Experimental results on industrial instances show that the proposed approach outperforms both complete and incomplete state-of-the-art MaxSAT solvers at the last international MaxSAT Evaluation in terms of robustness and total number of solved instances.

1 Introduction

Maximum Satisfiability (MaxSAT) is an optimization version of the well-known Satisfiability (SAT) problem. Recently, we have seen the successful application of MaxSAT techniques to solve several industrial or real combinatorial optimization problems: software package upgrade, debugging of hardware designs, fault localization in C code, bioinformatics, course timetabling, planing, scheduling, routing, electronic markets, combinatorial auctions, etc. See [Ansótegui *et al.*, 2013b; Morgado *et al.*, 2013] for citations.

Solving exactly combinatorial optimization problems, i.e., finding and certifying the best possible assignment, can be NP-hard from a computational point of view. However, many industrial problems are slightly beyond the reach of state-of-the-art techniques, and for many real domains we are often interested on improving in a reasonable time the best current assignment. Notice that even a small gain in the quality of the assignment can lead to important practical consequences for real domains.

This seems to suggest we should focus on developing incomplete algorithms for industrial problems. The experience achieved from the international SAT and MaxSAT competitions, shows us that the right research avenue is to focus first on improving complete or exact algorithms. Then, with the proper modifications, we can get an incomplete algorithm. Besides, we can always incorporate our complete algorithm into incomplete approaches such as Large Neighborhood Search [Shaw, 1998]. In LNS, we heuristically dive into promising regions of the search space (neighborhoods) that are explored with complete solvers.

The aim of this paper is to improve and combine effectively and efficiently techniques from complete state-of-the-art solvers MaxSAT solvers. In particular, we pay special attention to how to exploit the structure of the unsatisfiable cores in a MaxSAT instance. Finally, develop a complete algorithm that can be used in an incomplete approach.

In the MaxSAT community, we find two main classes of complete algorithms: branch and bound [Heras *et al.*, 2007; Li *et al.*, 2009; Kügel, 2010] and SAT-based [Ansótegui *et al.*, 2013b; Morgado *et al.*, 2013]. SAT-based approaches clearly dominate on industrial instances. SAT-based MaxSAT algorithms reformulate a MaxSAT instance into a sequence of SAT instances. By solving these SAT instances the MaxSAT problem can be solved. Intuitively, the SAT instances encode whether it is possible to find an assignment to the MaxSAT instances with a cost less than or equal to a certain k. The sequence is built in increasing order of k and it can be split into two parts. The instances in the first part are all unsatisfiable while the instances in the second one are all satisfiable. The value of k where the first satisfiable instance is located gives us the optimum of the MaxSAT instance. This transition from unsatisfiable to satisfiable SAT instances is usually associated with an easy-hard-easy pattern in terms of the hardness of the SAT instances [Shen and Zhang, 2003].

By solving the unsatisfiable SAT instances MaxSAT solvers refine the lower bound, while by solving satisfiable instances they refine the upper bound. Within SAT-based MaxSAT algorithms we find two main classes: (i) those that refine the lower bound, and guide the search with the unsatisfiable cores obtained from unsatisfiable SAT instances (core-guided algorithms) and, (ii) those that refine the upper bound, and guide the search with the satisfiable assignments obtained from satisfiable SAT instances (model-guided algorithms). Both approaches have strengths and weaknesses and there have been already some hybrid approaches [Morgado *et al.*, 2012; Ansótegui and Gabàs, 2013].

SAT-based MaxSAT solvers use Pseudo-Boolean (PB) constraints to create the SAT instances in the sequence. It is known that efficiency of SAT-based MaxSAT solvers heavily depends on how complex are these PB constraints, and how efficiently we manage them. Respect complexity, our current approach, as in [Andres *et al.*, 2012; Mor-

[*]Research partially supported by the Ministerio de Economía y Competividad research project TASSAT2: TIN2013-48031-C4-4-P and Google Faculty Research Award program.

gado *et al.*, 2014], only adds cardinality (Card) constraints (PB constraints with coefficients equal to 1) even if the MaxSAT instance is weighted. Respect management, unlike [Morgado *et al.*, 2014; Narodytska and Bacchus, 2014; Martins *et al.*, 2014], where Card constraints are incrementally constructed, our best approach does not use yet these incremental strategies for Card constraints. It exploits the structure of the unsatisfiable cores to produce a more efficient encoding for the Card constraints.

As we have mentioned, the algorithm we present is able to produce upper bounds during the search process. We also use these upper bounds to extend a very effective technique used in SAT solvers called *phase saving* [Pipatsrisawat and Darwiche, 2007] to MaxSAT.

Finally, we show an extensive experimental investigation on industrial instances. From the results, we can conclude that our approach outperforms clearly both in terms of robustness and number of solved instances the winners of the international MaxSAT Evaluation 2014 (MSE14) at the complete and incomplete tracks.

This paper proceeds as follows. Section 2 introduces some preliminary concepts. Section 3 presents the complete MaxSAT algorithm. Section 4 discusses how to encode efficiently in SAT the Card constraints generated by the MaxSAT algorithm. Section 5 explains how to exploit the upper bounds generated by the MaxSAT algorithm. Section 6 shows the experimental evaluation. Finally, Section 7 concludes.

2 Preliminaries

Definition 1 *A* literal *l is either a Boolean variable x or its negation $\overline{x}$. A* clause *c is a disjunction of literals. A* SAT formula *is a set of clauses that represents a Boolean formula in Conjunctive Normal Form (CNF), i.e. a conjunction of clauses.*

Definition 2 *A* weighted clause *is an ordered pair $\langle c, w\rangle$, where c is a clause and w is a natural number or infinity (indicating the cost of falsifying c, see Definitions 4 and 5). If w is infinite the clause is* hard, *otherwise it is* soft.

Definition 3 *A* Weighted Partial MaxSAT (WPMS) formula *is an ordered multiset of weighted clauses:*

$$\varphi = \langle\langle c_1, w_1\rangle, \ldots, \langle c_s, w_s\rangle, \langle c_{s+1}, \infty\rangle, \ldots, \langle c_{s+h}, \infty\rangle\rangle$$

where the first s clauses are soft and the last h clauses are hard. The presence of soft clauses with different weights makes the formula Weighted and the presence of hard clauses makes it Partial. The ordered multiset of weights of the soft clauses in the formula is noted as $w(\varphi)$. The top weight of the formula is noted as $W(\varphi)$, and defined as $W(\varphi) = \sum w(\varphi) + 1$. By φ_S we refer the set of soft clauses and by φ_H to the set of hard clauses. Finally, the set of variables occurring in the formula is noted as $var(\varphi)$.

Example 1 *Given the following WPMS formula: $\varphi = \langle\langle x_1, 5\rangle, \langle x_2, 3\rangle, \langle x_3, 3\rangle, \langle \overline{x}_1 \vee \overline{x}_2, \infty\rangle, \langle \overline{x}_1 \vee \overline{x}_3, \infty\rangle, \langle \overline{x}_2 \vee \overline{x}_3, \infty\rangle\rangle$, we have that $w(\varphi) = \langle 5, 3, 3\rangle$, $W(\varphi) = 12$, $\varphi_S = \langle\langle x_1, 5\rangle, \langle x_2, 3\rangle, \langle x_3, 3\rangle\rangle$, $\varphi_H = \langle\langle \overline{x}_1 \vee \overline{x}_2, \infty\rangle, \langle \overline{x}_1 \vee \overline{x}_3, \infty\rangle, \langle \overline{x}_2 \vee \overline{x}_3, \infty\rangle\rangle$ and $var(\varphi) = \{x_1, x_2, x_3\}$.*

Definition 4 *An* assignment *for a set of Boolean variables X is a function $\mathcal{I} : X \to \{0, 1\}$, that can be extended to literals, (weighted) clauses, SAT formulas and WPMS formulas as follows:*

$$\begin{aligned}
&\mathcal{I}(\overline{x}) = 1 - \mathcal{I}(x)\\
&\mathcal{I}(l_1 \vee \ldots \vee l_m) = \max\{\mathcal{I}(l_1), \ldots, \mathcal{I}(l_m)\}\\
&\mathcal{I}(\{c_1, \ldots, c_n\}) = \min\{\mathcal{I}(c_1), \ldots, \mathcal{I}(c_n)\}\\
&\mathcal{I}(\langle c, w\rangle) = w\,(1 - \mathcal{I}(c))\\
&\mathcal{I}(\langle\langle c_1, w_1\rangle, \ldots, \langle c_{s+h}, w_{s+h}\rangle\rangle) = \textstyle\sum_{i=1}^{s+h} \mathcal{I}(\langle c_i, w_i\rangle)
\end{aligned}$$

We will refer to the value returned by an assignment $\mathcal{I}$ on a weighted clause or a WPMS formula as the cost of $\mathcal{I}$.

Definition 5 *We say that an assignment $\mathcal{I}$ satisfies a clause or a SAT formula if the value returned by $\mathcal{I}$ is equal to 1. In the case of SAT formulas, we will refer also to this assignment as a satisfying assignment or solution. Otherwise, if the value returned by $\mathcal{I}$ is equal to 0, we say that $\mathcal{I}$ falsifies the clause or the SAT formula.*

Definition 6 *The* SAT problem *for a SAT formula φ is the problem of finding a solution for φ. If a solution exists the formula is satisfiable, otherwise it is unsatisfiable.*

Definition 7 *Given an unsatisfiable SAT formula φ, an* unsatisfiable core *φ_C is a subset of clauses $\varphi_C \subseteq \varphi$ that is also unsatisfiable.*

Definition 8 *A SAT algorithm for the SAT problem, takes as input a SAT formula φ and returns an assignment $\mathcal{I}$ such that $\mathcal{I}(\varphi) = 1$ if the formula is satisfiable. Otherwise, it returns an unsatisfiable core φ_C.*

Given unlimited resources of time and memory, we say that a SAT algorithm is complete *if it terminates for any SAT formula. Otherwise, it is* incomplete.

Definition 9 *The* optimal cost (or optimum) *of a WPMS formula φ is* $\text{cost}(\varphi) = \min\{\mathcal{I}(\varphi) \mid \mathcal{I} : var(\varphi) \to \{0, 1\}\}$ *and an* optimal assignment *is an assignment $\mathcal{I}$ such that $\mathcal{I}(\varphi) = \text{cost}(\varphi)$. We will refer to this assignment as a solution for φ if $\mathcal{I}(\varphi) \neq \infty$. Any cost above (below) cost(φ) is called an* upper (lower) bound *for φ.*

Example 2 *Given the WPMS formula φ of Example 1, we have that $\text{cost}(\varphi) = \min\{6, 8, 11, \infty\} = 6$ and the optimal assignment $\mathcal{I}$ maps $\langle x_1, x_2, x_3\rangle$ to $\langle 1, 0, 0\rangle$.*

Definition 10 *The* Weighted Partial MaxSAT problem *for a WPMS formula φ is the problem of finding a solution for φ. If a solution does not exist the formula is unsatisfiable.*

Definition 11 *A WPMS algorithm for the WPMS problem, takes as input a WPMS formula φ and returns an assignment $\mathcal{I}$, such that, $\mathcal{I}(\varphi) \geq cost(\varphi)$.*

Given unlimited resources of time and memory, we say that a WPMS algorithm is complete or exact *if for any input WPMS formula φ and returned $\mathcal{I}$, $\mathcal{I}(\varphi) = cost(\varphi)$. Otherwise, we say it is* incomplete.

Definition 12 *An* integer linear Pseudo-Boolean (PB) constraint *is an inequality of the form $w_1x_1 + \cdots + w_nx_n$ op k, where $op \in \{\leq, \geq, =, >, <\}$, k and w_i are integer coefficients, and x_i are Boolean variables. A PB* at-most constraint *is a PB constraint where op is $\leq$. A* cardinality (Card) constraint *is a PB constraint where the coefficients w_i are equal to* 1.

3 WPM3 MaxSAT Algorithm

Algorithm 1: WPM3

```
Input: $\varphi = \langle\langle c_1, w_1\rangle, ..., \langle c_s, w_s\rangle, \langle c_{s+1}, \infty\rangle, ..., \langle c_{s+h}, \infty\rangle\rangle$
1  $\langle AM, w_{strat}\rangle := \langle\langle\langle\langle 1\rangle, 0, w_1\rangle, \ldots, \langle\langle s\rangle, 0, w_s\rangle\rangle, \infty\rangle$
2  while $true$ do
3      $\langle st, C, \mathcal{I}\rangle := sat(\varphi, AM, w_{strat})$
4      if $st =$ SAT then
5          if $w_{strat} = \min_{w_j \in w(AM), w_j \neq 0}(\{w_j\})$ then return $\langle \mathcal{I}, \mathcal{I}(\varphi)\rangle$
6          $w_{strat} = decrease(AM, w_{strat})$
7      else
8          if $w_{strat} = \infty$ then return $\langle\emptyset, \infty\rangle$
9          $w_{min} := min(w(AM_C))$
10         $AM := AM[\langle A_j, k_j, w_j\rangle_{j\in C} / \langle A_j, k_j, w_j - w_{min}\rangle_{j\in C}]$
11         $\langle A, k_A\rangle := optimize(\varphi, AM_C)$
12         $AM := AM + \langle\langle A, k_A, w_{min}\rangle\rangle$
```

Function sat(φ, AM, w_{strat})

```
1  $\varphi^k := \varphi_H \cup \{c_i \vee b_i\}_{\langle c_i, w_i\rangle \in \varphi_S} \cup \{\, CNF(A_j, k_j) \vee a_j, \overline{a}_j \,\}_{\langle A_j, k_j, w_j\rangle \in AM, w_j \geq w_{strat}}$
2  $\langle st, \varphi^k_C, \mathcal{I}\rangle := satsolver(\varphi^k)$
3  $C := \{j \mid \{\overline{a}_j\} \cap \varphi^k_C \neq \emptyset\}$
4  return $\langle st, C, \mathcal{I}\rangle$
```

Function optimize(φ, AM)

```
1  $A := +_{\langle A_j, K_j, w_j\rangle \in AM} A_j$
2  $ub := \sum_{\langle A_j, k_j, w_j\rangle \in AM} \mid A_j \mid$
3  while $true$ do
4      $k := ub - 1$
5      $\langle st, \_, \mathcal{I}\rangle := sat(\varphi, \langle\langle A, k, \_\rangle\rangle, \_)$
6      if $st =$ SAT then $ub := \mathcal{I}(\langle\langle c_i, 1\rangle \mid \langle c_i, w_i\rangle \in \varphi\rangle_{i\in A})$
7      else return $\langle A, ub\rangle$
```

In this section, we present the WPM3 complete algorithm for the MaxSAT problem. Given an input WPMS formula φ, WPM3 solves the formula by testing the satisfiability of a sequence of SAT instances φ^k where $0 \leq k \leq W(\varphi)$. Each SAT instance φ^k encodes whether there is an assignment to φ with a cost $\leq k$. Notice that SAT instances with $k < cost(\varphi)$ are unsatisfiable, otherwise they are satisfiable. The optimum corresponds to the k of the first satisfiable SAT instance.

Roughly speaking, from every unsatisfiable SAT instance the algorithm finds and keeps an unsatisfiable core. WPM3 is designed to be aware of the global structure of theses cores. This is used both for producing more efficient cardinality (Card) constraint encodings (see Section 4) and focus the search on subproblems of the input MaxSAT instance.

The algorithm maintains a set of soft at-most Card constraints AM. We note these constraints as $\langle A, k, w\rangle$, where A is an ordered multiset of indexes of the original soft clauses, k indicates at most how many clauses from A can be falsified, and w is the cost for falsifying this soft constraint. The at-most constraints are used to do not accept those solutions whose cost exceeds the current k, where $k = \sum_{\langle A_j, k_j, w_j\rangle \in AM} k_j \cdot w_j$. Moreover, the algorithm guarantees that $k \leq cost(\varphi)$. The idea of maintaining multiple at-most Card constraints instead of a single one was originally introduced in [Ansótegui *et al.*, 2009a] for PM2 algorithm. Notice that from the AM set the global core structure can be obtained.

We start (line 1) by adding to AM a soft at-most constraint for each original soft clause. Then, the algorithm will iterate (line 2) till it is able to determine $cost(\varphi)$. This will happen if it detects that the set of hard clauses is unsatisfiable (line 8, $cost(\varphi) = \infty$) or when it is able to generate the first satisfiable instance (line 5, $cost(\varphi) = k = \mathcal{I}(\varphi)$). We obviate for the moment the role of w_{stat} and we consider it is ∞ for the first iteration and $\min_{w_j \in w(AM), w_j \neq 0}(\{w_j\})$ for the rest.

Function *sat* (line 3) builds the SAT instance φ^k at the current iteration and sends it to the SAT solver. φ^k is constructed through the union of the following sets expressed as SAT clauses: (i) the set of hard clauses, (ii) the reification to variables b_i of soft clauses, (iii) the reification to variables a_j of the translation into CNF of the at-most constraints in AM, and finally (iv) the unit clauses $\overline{a}_j$. The new b_i variables are true if the respective original soft clause becomes false. They are used to encode the at-most constraints which restrict the number of falsified soft clauses. The new a_j variables are true if the respective at-most constraint becomes false. The unit clauses $\overline{a}_j$ encode that we would like to satisfy all the at-most constraints. If this is not possible, some of them will appear into the unsatisfiable core φ^k_C.

Example 3 *Given* $\varphi = \langle\langle x_1, 1\rangle, \langle x_2, 1\rangle, \langle x_3, 1\rangle, \langle \overline{x}_1 \vee \overline{x}_2, \infty\rangle, \langle \overline{x}_1 \vee \overline{x}_3, \infty\rangle, \langle \overline{x}_2 \vee \overline{x}_3, \infty\rangle\rangle$. *At an intermediate step we have the set* $AM = \langle\langle\langle 1, 2\rangle, 1, 1\rangle, \langle\langle 3\rangle, 0, 1\rangle\rangle$. *Then,*

$$\varphi^k = \{\overline{x}_1 \vee \overline{x}_2,\ \overline{x}_1 \vee \overline{x}_3,\ \overline{x}_2 \vee \overline{x}_3\} \cup \{x_1 \vee b_1,\ x_2 \vee b_2,\ x_3 \vee b_3\}$$
$$\cup \{CNF(b_1 + b_2 \leq 1) \vee a_1, \overline{a}_1, CNF(b_3 \leq 0) \vee a_2, \overline{a}_2\}$$

To our best knowledge, the idea of introducing Card constraints as soft constraints in a core-guided algorithm was initially proposed in [Andres *et al.*, 2012] for Answer Set Programming (ASP) optimization problems, and recently adapted to MaxSAT problems in [Morgado *et al.*, 2014]. The approach in [Narodytska and Bacchus, 2014] (best solver for industrial instances at MSE14) also uses soft Card constraints. These approaches work locally and are not aware of the global core structure and therefore they can not exploit it.

Going back to WPM3 algorithm, if function *sat* returns satisfiable ($st = SAT$) (line 4), then we return the optimal assignment $\mathcal{I}$ and its cost (line 5). Otherwise (line 7), C is the set of indexes of at-most constraints involved into the last unsatisfiable core. If the core only involves original hard clauses, we can return unsatisfiable (line 8). If there are at-most constraints involved in the core, then, we need to relax some of them since they only allow assignments or solutions with a cost strictly less than $cost(\varphi)$.

At this stage (lines 9-12), we need to relax the set of AM constraints, but guaranteeing we do not exceed $cost(\varphi)$. Basically, we need to replace the set of at-most constraints AM_C involved in the last core with a new set of at-most constraints which allows assignments with a higher cost.

Since we will only use Card constraints we first apply the idea described in the WPM1/WBO [Ansótegui *et al.*, 2009b; Manquinho *et al.*, 2009] MaxSAT algorithms to deal with Weighted instances (lines 9-10). It basically prevents the algorithm to introduce PB constraints instead of Card constraints when the at-most constraints involved in the core have different weights. In this case, we compute the minimum weight w_{min} of the constraints in AM_C (line 9), and replace every soft constraint $\langle A_j, k_j, w_j\rangle$ by two copies with weights $w_j - w_{min}$ and w_{min}. The first set of copies will remain in AM (line 10) while the second will be replaced by the new at-most constraint $\langle A, k_A, w_{min}\rangle$ (line 12). Notice we guarantee that $\sum_{\langle A_j,k_j,w_j\rangle\in AM} \mid A_j \mid \cdot w_j = \sum w(\varphi)$.

Function *optimize* (line 11) allows to determine which is the new φ_k we will test. This function, basically, solves exactly the subproblem that involves the new at-most constraint we will generate on the original soft clauses, and the hard clauses. The result is the set of indexes of original soft clauses A of the new at-most constraint (notice that and index can be repeated) and the number of clauses k_A that we will at most allow to be falsified. To our best knowledge, the idea of solving a subproblem of the original optimization instance φ was originally applied for MaxSAT in WPM2 algorithm [Ansotegui *et al.*, 2010]. In [Davies and Bacchus, 2011] a similar approach is applied calling a MIP solver to solve the subproblem. Recently, in [Ansótegui *et al.*, 2013a], this process is extended and named as *cover optimization*. The best strategy reported consists on refining iteratively the upper bound on the subproblem using the model-guided MaxSAT algorithm in [Berre, 2006]. We apply it within function *optimize*, although depending on the particular family of instances other strategies or algorithms can have better performance. At this point we have increased k by $(k_A - \sum_{\langle A_j,k_j,w_j\rangle\in AM_C} k_j)\cdot w_{min}$ towards $cost(\varphi)$. Otherwise, without the *optimize* step, we can only increase k by w_{min}.

Treating explicitly the new at-most constraint (line 12) and its relation to the constraints it substitutes is fundamental, not only for function *optimize*, but also to encode more efficient Card constraints (see Section 4). In contrast, this information, the global core structure, is not explicitly present in recent approaches as [Morgado *et al.*, 2014; Narodytska and Bacchus, 2014] and therefore harder to be exploited efficiently.

During this description, we have obviated the role of w_{strat} (lines 5 and 6). It corresponds to the application of the stratified approach introduced in [Ansotegui *et al.*, 2009c]. The stratified approach consists in sending to the SAT solver only a subset of the soft clauses, i.e., those with a weight $\geq w_{strat}$. Function *decrease* updates conveniently w_{strat}. This can help to reduce the size of the unsatisfiable cores, produce simpler at-most constraints and progress faster to the optimum. We also apply *hardening* techniques like the ones described in [Ansótegui *et al.*, 2012; Morgado *et al.*, 2012; Ansótegui *et al.*, 2013a].

4 Efficient Card Constraints for MaxSAT

In the last decade, we have seen many contributions on how to encode efficiently PB and Card constraints into SAT [Bailleux and Boufkhad, 2003; Sinz, 2005; Eén and Sörensson, 2006; Bailleux *et al.*, 2009; Asín *et al.*, 2011]. The goal is to achieve an arc-consistent encoding (i.e., with good propagation properties) as small as possible.

Since WPM3 only uses Card constraints, let us consider the Card constraint: $b_1 + \cdots + b_n \leq k$. From the sake of clarity, the encoder is split into two black boxes: the *sum* and the operator *op* (in our case representing $\leq$). The *sum* takes as input a list of n variables $[b_1, \ldots, b_n]$ and returns a set of SAT clauses S and a list of m variables $[o_1, \ldots, o_m]$. The operator takes as input the o variables and integer k and returns a set of SAT clauses OP. The encoding of the Card constraint into SAT corresponds to the union $S \cup OP$.

$$\langle S, [o_1, \ldots, o_m]\rangle := sum([b_1, \ldots, b_n])$$
$$OP := op(k, [o_1, \ldots, o_m])$$

In our case, we use the Cardinality Networks encoding in [Asín *et al.*, 2011]. There, $m = k + 1$ and the sum builds a SAT formula such that if i of the input b variables are set to true then the first i of the output o variables are set to true and the rest to false. Therefore, *op* returns the unit clause $\overline{o}_{k+1}$.

Our first observation is that it is crucial for the efficiency of the MaxSAT solver in which order the b variables are fed into the sum. In previous MaxSAT solvers, the order of the b variables was not taken into account. They were just added in the same order of their respective soft clauses.

However, the b variables should be added taking into account the structure of the unsatisfiable cores. In particular, variables belonging to the same core should be as close as possible. In our algorithm the set A in an at-most constraint $\langle A, k, w\rangle$ is ordered. In line 1 of function *optimize* (see Section 3) when we generate the set A of the new at most constraint, we concatenate the sets of b variables of the at-most constraints that it replaces. Respect to latest advances in MaxSAT [Morgado *et al.*, 2014; Narodytska and Bacchus, 2014] a deeper explanation of their efficiency could be that these algorithms implicitly preserve the order.

Our second observation has to do again with the structure of the unsatisfiable cores we have detected so far. As we have just commented, the new at-most constraint $\langle A, k_A, w_{min}\rangle$ (line 12 of WPM3) replaces/merges other at-most constraints. In the end, we can consider that there is a tree structure that represents how we have merged the unsatisfiable cores and where the root node is the new at-most constraint. Instead of creating a Cardinality Network for the new constraint we can reproduce this tree structure. We basically reproduce the totalizer encoding proposed originally in [Bailleux and Boufkhad, 2003]. The leaf nodes join the at-most constraints related with a single soft clause of the input formula (i.e., with $k = 0$) that appeared in the same core. The leaf nodes are encoded with Cardinality Networks.

Example 4 *Imagine at-most constraint* $\langle A_3, 5, 1\rangle$ *(root node) replaces at-most constraints* $\langle A_1, 3, 1\rangle$ *and* $\langle A_2, 1, 1\rangle$ *(child 1 and child 2). Let us see how we start constructing the tree. Children are processed recursively in the same way.*

$$\overset{\substack{|A_3|=10 \\ sum\leq 5}}{\langle S', [o^3_1, \ldots, o^3_6]\rangle} := \mathit{totalizer}(\overset{\substack{|A_1|=6 \\ sum\leq 5}}{[o^1_1, ..., o^1_6]}, \overset{\substack{|A_2|=4 \\ sum\leq 4}}{[o^2_1, ..., o^2_5]})$$

$$S^3 := S' \cup S^1 \cup S^2 \quad OP^3 := \{\overline{o}^3_6\} \cup OP^1 \cup OP^2$$

The advantage of preserving this structure, in contrast to having a single Card constraint, is that we can again exploit it to derive smaller encodings. In particular, we can restrict the sums of the nodes using the lower bounds of their siblings. The upper bound for a non root node is set to the difference between the upper bound of its parent and the sum of the lower bounds of its siblings. We apply this in a top-down update process.

Example 5 *At example 4, the upper bound for the root node is* 5*. Child 1 (child 2) already contributes with a lower bound of* 3 *(*1*), therefore the new upper bound for child 1 (child 2) is* $5 - 1 = 4$ *(*$5 - 3 = 2$*).*

$$\overset{\substack{|A_3|=10 \\ sum\leq 5}}{\langle S', [o^3_1, \ldots, o^3_6]\rangle} := \mathit{totalizer}(\underset{lb=3}{\overset{\substack{|A_1|=6 \\ sum\leq 4}}{[o^1_1, ..., o^1_5]}}, \underset{lb=1}{\overset{\substack{|A_2|=4 \\ sum\leq 2}}{[o^2_1, ..., o^2_3]}})$$

$$S^3 := S' \cup S^1 \cup S^2 \quad OP^3 := \{\overline{o}^3_6\} \cup OP^1 \cup OP^2$$

Function *optimize* can provide assignments I_{ub} which are upper bounds (see Section 5) for the whole formula φ. Using this upper bound, we can set the upper bound for the root node which corresponds to the cost of the assignment I_{ub} on the set of soft clauses related to it.

Finally, notice that for a given at-most constraint we build from scratch its SAT encoding when we feed it into the SAT solver. Recently, in [Martins *et al.*, 2014] it has been proposed to build Card constraints incrementally. We will explore extending our approach in this sense. As we will see in Section 6, even without the incremental extension, our approach outperforms the rest of the solvers.

5 Upper Bounds and Phase Saving

In Section 3, we have described a complete core-guided algorithm that keeps increasing a lower bound towards the optimum. However, its design and, in particular, function *optimize*, which implements a model-guided MaxSAT algorithm, allows to produce upper bounds for φ during its execution. Whenever function *optimize* finds a new solution to the subproblem, we can just extend this solution to the rest of the formula and check its cost. We do this by considering that, those clauses in a undefined state under the solution to the subproblem, are falsified [1]. Function *optimize* keeps track of the assignment to the subproblem whose extension to the rest of the formula had the lowest cost. The cost of this assignment is an upper bound for the whole problem. We will refer to it as the best global assignment found by function *optimize*.

We can exploit further the best global assignment we get from function *optimize*. State-of-the-art SAT solvers apply a technique called *phase saving* introduced in [Pipatsrisawat and Darwiche, 2007]. In SAT solvers, when non-chronological backtracking is applied due to a conflict, lots of variable assignments get lost and have to be revealed again during search. The idea is to avoid redoing this work by storing the phase of the variables when we find a conflict. Then, we assign to the next decision variables the stored polarity till we find a new conflict and update the polarities again. This technique has been shown to be quite effective. We can extend this idea to MaxSAT in the following way. Within function *optimize* we let the SAT solver apply phase saving in the regular way (line 5). Then, if the best global assignment found so far has the same cost for the subproblem as the solution found by function *optimize*, we store the polarity of the variables in the best global assignment. We use these polarities to guide the search of the SAT solver in line 3 of WPM3, disabling the regular phase saving that would be applied. In particular, we only store the polarities of the original variables in φ.

6 Experimental Results

We have evaluated our approach on the industrial instances from the MaxSAT Evaluation 2014 (MSE14) [Argelich *et al.*, 2006 2014]. We run our experiments on a cluster featured with 2.6GHz processors and a memory limit of 3.5 GB. The instance set of MSE14 is divided into three categories depending on the variant of the MaxSAT problem: MaxSAT (MS), Partial MaxSAT (PMS) or Weighted Partial MaxSAT (WPMS). In each category, instances are grouped by families: 2 for MS, 22 for PMS and 8 for WPMS. Since families have different number of instances, we considered more fair to present the solvers ranked by mean family ratio of solved instances.

We provide results for the new $wpm3$ MaxSAT solver and the best solvers of the MSE14. We have excluded the MaxSAT solver $ISAC+$ [Ansótegui *et al.*, 2014], since it is a portfolio and our intention here is to compare ground solvers. The ground solvers with the best overall performance at MSE14 for industrial instances were: $eva500a$ [Narodytska and Bacchus, 2014], $mscg$ [Morgado *et al.*, 2014] and *open-wbo* [Martins *et al.*, 2014]. We also present results for an initial version we implemented within the *or-tools* package [Google, 2009] which is only core-guided.

Table 1 shows our first experiment, where we evaluate the impact of each improvement on WPM3 (with a timeout of 1800 seconds). All the variations on WPM3 are implemented on top of the glucose SAT solver (version 3.0) [Audemard and Simon, 2009] . The different variations and corresponding implementations are named $wpm3$ with different subindexes. Subindex $_o$ stands for *cover optimization* (see Section 3). Regarding how Card constraints are encoded, $_t$ stands for core based tree, $_{tk}$ stands for core based tree with refinement of the

[1]This can be improved looking at the structure of the undefined clauses.

	MS Ind. 100% 55	PMS Ind. 100% 568	WPMS Ind. 100% 410	Total Ind. 100% 1033
$wpm3_{tkop}$	**88,5% 43**	**84,0% 499**	**74,0% 367**	**81,7% 909**
$wpm3_{tko}$	87,5% 42	83,4% 494	73,9% 366	81,3% 902
$wpm3_{tk}$	87,5% 42	82,5% 483	73,0% 359	80,4% 884
$wpm3_{to}$	87,5% 42	83,4% 494	72,9% 358	81,0% 894
$wpm3_{t}$	87,5% 42	81,9% 480	72,9% 358	80,0% 880
$wpm3_{o}$	87,5% 42	82,9% 489	71,3% 349	80,3% 880
$wpm3$	87,5% 42	80,7% 470	70,4% 346	78,6% 858
$pm2_{o}$	84,0% 39	78,9% 458	-	-
$pm2$	84,0% 39	77,5% 445	-	-
$wpm1/wbo$	80,0% 36	61,8% 349	67,4% 348	64,4% 733

Table 1: WPM3 and improvements.

k upper bound in sub-sums (see Section 4). Finally, $_p$ stands for phase saving extended to MaxSAT (see Section 5).

At the table, we present for each category and solver the mean ratio of solved instances per family and the total number of instances. We introduce results for $wpm1/wbo$ [Ansótegui *et al.*, 2009b; Manquinho *et al.*, 2009] and $pm2$ algorithms [2]. $wpm3$ can be considered as a hybridization of these two algorithms (see Section 3). As we can see, $wpm3$ clearly outperforms $pm2$. This is because, as described in Section 4, we build Card constraints taking into account the order imposed by the unsatisfiable cores. If we add the *cover optimization* technique, see $wpm3_o$, then we get a version that would have ranked the first at MSE14 for industrial instances in terms of the total mean family ratio. The next two versions, $wpm3_t$ and $wpm3_{tk}$, that further exploit the structure of the cores and improve the construction of the Card constraint, provide a total of 13 additional solved instances for PMS and 13 for WPMS. As we can see, the *cover optimization* technique always improves, in particular for Weighted instances at version $wpm3_{tko}$. Finally, the extension of phase saving for MaxSAT improves the average running time, and it helps to solve 7 additional instances within the timeout.

	MS Ind. 100% 55	PMS Ind. 100% 568	WPMS Ind. 100% 410	Total Ind. 100% 1033
$wpm3_{tkop}$	**88.5% 43**	**84.0% 499**	**74.0%** 367	**81.7% 909**
$or\text{-}tools$	81.7% 36	81.9% 482	73.9% **369**	79.8% 887
$eva500a$	86.5% 41	80.0% 476	72.8% 368	78.7% 885
$mscg$	86.5% 41	80.2% 468	68.5% 361	77.7% 870
$open\text{-}wbo$	87.5% 42	81.0% 472	64.9% 335	77.3% 849

Table 2: WPM3 and $or\text{-}tools$ compared to best MSE14 solvers

Table 2 compares our best version $wpm3_{tkop}$ with the best performing complete solvers at MSE14 for industrial instances. Clearly, $wpm3_{tkop}$ dominates both on mean family ratio and total number of solved instances. A deeper analysis reveals that $wpm3_{tkop}$ has best ratio on 30 out of the 32 families that compose the categories, while $eva500a$ (the second one) has best ratio on 20.

Our last experiment is presented in Table 3. Since $wpm3_{tkop}$ is able to produce upper bounds we also compared

[2]$pm2$ algorithm is only designed for Partial MaxSAT instances

	$wpm3_{tkop}$	$open\text{-}wbo$	qms	$wpm2014$	$optimax$
$wpm3_{tkop}$		**50** **556 394**	**50** 502 **379**	**49** **547 386**	**53** **524 390**
$open\text{-}wbo$	45 466 344		**45** 458 **361**	44 **477** 341	**45** **466 378**
qms	15 **511** 281	19 **522** 286		17 **534** 279	18 **530 302**
$wpm2014$	48 427 369	**48** **477 387**	**48** 402 **379**		**51** 420 **367**
$optimax$	39 368 259	43 387 270	**47** 337 284	39 **423** 260	

Table 3: WPM3 compared to MSE14 best incomplete solvers.

it with the best performing solvers $wpm2014$ and $optimax$ [3] for industrial instances at the incomplete track of the MSE14. We also compared with other MaxSAT solvers that did not take part in the incomplete track but they are able to produce upper bounds: $open\text{-}wbo$ [Martins *et al.*, 2014] and qms [Koshimura *et al.*, 2012]. We do not include $eva500a$, $mscg$ or $or\text{-}tools$ since they can not produce upper bounds.

The timeout for the incomplete track at MSE is set to 300 seconds. For a given instance, the winners are the solvers that produce the best upper bound. The best solver is the one that won on more instances. Since these results give us a partial order, it can be misleading to report an overall winner. In table 3, we present the dominance relation between pairs of solvers on the three categories. For example, $wpm3_{tkop}$ ($open\text{-}wbo$) is able to obtain a better or equal upper bound than $open\text{-}wbo$ ($wpm3_{tkop}$) on 50 (45) ms instances, 556 (466) PMS instances and 394 (344) WPMS instances. As we can see, $wpm3_{tkop}$ practically dominates the rest of the solvers. The exception is qms on PMS where qms outperforms by 9 instances $wpm3_{tkop}$. For this case, we extended the timeout to 1800 seconds. We found that $wpm3_{tkop}$ outperformed qms by 5 instances. This is somehow expected since $wpm3_{tkop}$, within this timeout, solves to optimality 40 instances more than qms.

7 Conclusions

We have proposed a complete algorithm for MaxSAT that can be also used in an incomplete approach. We show how to combine several techniques. We have shown how that the design of the algorithm allows to exploit the structure of unsatisfiable cores in MaxSAT instances to build more efficient Card constraints. This opens a window to understand better the performance of the latest solvers and probably further improve them. Finally, we have shown that an extended notion of phase saving for MaxSAT is effective. Our resulting MaxSAT solver outperforms the winners for the complete and incomplete track at the last MaxSAT Evaluation.

References

[Andres *et al.*, 2012] Benjamin Andres, Benjamin Kaufmann, Oliver Matheis, and Torsten Schaub. Unsatisfiability-based optimization in clasp. In *ICLP (Technical Communications)*, pages 211–221, 2012.

[3]$optimax$ builds on top on the bincd2 algorithm [Morgado *et al.*, 2012]

[Ansótegui and Gabàs, 2013] Carlos Ansótegui and Joel Gabàs. Solving (weighted) partial maxsat with ilp. In *CPAIOR 13*, pages 403–409, 2013.

[Ansótegui *et al.*, 2009a] Carlos Ansótegui, Maria Luisa Bonet, and Jordi Levy. On solving MaxSAT through SAT. In *Proc. of CCIA'09*, pages 284–292, 2009.

[Ansótegui *et al.*, 2009b] Carlos Ansótegui, Maria Luisa Bonet, and Jordi Levy. Solving (weighted) partial maxsat through satisfiability testing. In *Proc. of SAT'09*, pages 427–440, 2009.

[Ansotegui *et al.*, 2009c] Carlos Ansotegui, Maria Luisa Bonet, and Jordi Levy. Solving (weighted) partial maxsat through satisfiability testing. In *Proc. of SAT'09*, pages 427–440, 2009.

[Ansotegui *et al.*, 2010] Carlos Ansotegui, Maria Luisa Bonet, and Jordi Levy. A new algorithm for weighted partial maxsat. In *Proc. of AAAI'10*, pages 867–872, 2010.

[Ansótegui *et al.*, 2012] Carlos Ansótegui, Maria Luisa Bonet, Joel Gabàs, and Jordi Levy. Improving sat-based weighted maxsat solvers. In *Proc. of CP'12*, pages 86–101, 2012.

[Ansótegui *et al.*, 2013a] Carlos Ansótegui, Maria Luisa Bonet, Joel Gabàs, and Jordi Levy. Improving wpm2 for (weighted) partial maxsat. In *Proc. of CP'13*, pages 117–132, 2013.

[Ansótegui *et al.*, 2013b] Carlos Ansótegui, Maria Luisa Bonet, and Jordi Levy. Sat-based maxsat algorithms. *Artif. Intell.*, 196:77–105, 2013.

[Ansótegui *et al.*, 2014] Carlos Ansótegui, Yuri Malitsky, and Meinolf Sellmann. Maxsat by improved instance-specific algorithm configuration. In *Proc. of AAAI'14*, pages 2594–2600, 2014.

[Argelich *et al.*, 2006 2014] Josep Argelich, Chu Min Li, Felip Manyà, and Jordi Planes. Maxsat evaluation, 2006-2014. http://www.maxsat.udl.cat.

[Asín *et al.*, 2011] Roberto Asín, Robert Nieuwenhuis, Albert Oliveras, and Enric Rodríguez-Carbonell. Cardinality networks: a theoretical and empirical study. *Constraints*, 16(2):195–221, 2011.

[Audemard and Simon, 2009] Gilles Audemard and Laurent Simon. Predicting learnt clauses quality in modern SAT solvers. In *Proc. of IJCAI'09*, pages 399–404, 2009.

[Bailleux and Boufkhad, 2003] Olivier Bailleux and Yacine Boufkhad. Efficient CNF encoding of boolean cardinality constraints. In *Proc. of CP'03*, pages 108–122, 2003.

[Bailleux *et al.*, 2009] Olivier Bailleux, Yacine Boufkhad, and Olivier Roussel. New encodings of pseudo-boolean constraints into cnf. In *Proc. of SAT'09*, pages 181–194, 2009.

[Berre, 2006] Daniel Le Berre. Sat4j, a satisfiability library for java, 2006. www.sat4j.org.

[Davies and Bacchus, 2011] Jessica Davies and Fahiem Bacchus. Solving maxsat by solving a sequence of simpler sat instances. In *Proc. of CP'11*, pages 225–239, 2011.

[Eén and Sörensson, 2006] Niklas Eén and Niklas Sörensson. Translating pseudo-boolean constraints into SAT. *JSAT*, 2(1-4):1–26, 2006.

[Google, 2009] Google. Or-tools: The google operations research suite, 2009. https://developers.google.com/optimization/.

[Heras *et al.*, 2007] Federico Heras, Javier Larrosa, and Albert Oliveras. MiniMaxSat: A new weighted Max-SAT solver. In *Proc. of SAT'07*, pages 41–55, 2007.

[Koshimura *et al.*, 2012] Miyuki Koshimura, Tong Zhang, Hiroshi Fujita, and Ryuzo Hasegawa. Qmaxsat: A partial max-sat solver. *JSAT*, 8(1/2):95–100, 2012.

[Kügel, 2010] Adrian Kügel. Improved exact solver for the weighted MAX-SAT problem. In *POS-10. Pragmatics of SAT, Edinburgh, UK, July 10, 2010*, pages 15–27, 2010.

[Li *et al.*, 2009] Chu Min Li, Felip Manyà, Nouredine Ould Mohamedou, and Jordi Planes. Exploiting cycle structures in Max-SAT. In *Proc. of SAT'09*, pages 467–480, 2009.

[Manquinho *et al.*, 2009] Vasco Manquinho, João Marques-Silva, and Jordi Planes. Algorithms for weighted boolean optimization. In *Proc. of SAT'09*, pages 495–508, 2009.

[Martins *et al.*, 2014] Ruben Martins, Saurabh Joshi, Vasco M. Manquinho, and Inês Lynce. Incremental cardinality constraints for maxsat. In *Proc. of CP'14*, pages 531–548, 2014.

[Morgado *et al.*, 2012] António Morgado, Federico Heras, and João Marques-Silva. Improvements to core-guided binary search for maxsat. In *Proc. of SAT'12*, pages 284–297, 2012.

[Morgado *et al.*, 2013] António Morgado, Federico Heras, Mark H. Liffiton, Jordi Planes, and Joao Marques-Silva. Iterative and core-guided maxsat solving: A survey and assessment. *Constraints*, 18(4):478–534, 2013.

[Morgado *et al.*, 2014] António Morgado, Carmine Dodaro, and Joao Marques-Silva. Core-guided maxsat with soft cardinality constraints. In *Proc. of CP'14*, pages 564–573, 2014.

[Narodytska and Bacchus, 2014] Nina Narodytska and Fahiem Bacchus. Maximum satisfiability using core-guided maxsat resolution. In *Proc. of AAAI'14*, pages 2717–2723, 2014.

[Pipatsrisawat and Darwiche, 2007] Knot Pipatsrisawat and Adnan Darwiche. A lightweight component caching scheme for satisfiability solvers. In *Proc. of SAT'07*, pages 294–299, 2007.

[Shaw, 1998] Paul Shaw. Using constraint programming and local search methods to solve vehicle routing problems. In *Proc. of CP'98*, pages 417–431, 1998.

[Shen and Zhang, 2003] Haiou Shen and Hantao Zhang. An empirical study of MAX-2-SAT phase transitions. *Electronic Notes in Discrete Mathematics*, 16:80–92, 2003.

[Sinz, 2005] Carsten Sinz. Towards an optimal CNF encoding of boolean cardinality constraints. In *Proc. of CP'05*, pages 827–831, 2005.