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

# A MaxSAT Algorithm Using Cardinality Constraints of Bounded Size*

**Mario Alviano** and **Carmine Dodaro** and **Francesco Ricca**
Department of Mathematics and Computer Science, University of Calabria, Italy
{alviano, dodaro, ricca}@mat.unical.it

## Abstract

Core-guided algorithms proved to be effective on industrial instances of MaxSAT, the optimization variant of the satisfiability problem for propositional formulas. These algorithms work by iteratively checking satisfiability of a formula that is relaxed at each step by using the information provided by unsatisfiable cores. The paper introduces a new core-guided algorithm that adds cardinality constraints for each detected core, but also limits the number of literals in each constraint in order to control the number of refutations in subsequent satisfiability checks. The performance gain of the new algorithm is assessed on the industrial instances of the 2014 MaxSAT Evaluation.

## 1 Introduction

MaxSAT is the optimization variant of the satisfiability problem for propositional formulas. In its most general formulation, also known as *Weighted Partial MaxSAT* [Cha *et al.*, 1997], the input formula is partitioned into hard and soft clauses, where soft clauses are associated with a weight. Hard clauses must necessarily be satisfied, while soft clauses should be possibly satisfied. In fact, the goal is to find an assignment satisfying hard clauses and minimizing the total weight of unsatisfied soft clauses. The research in this field is active as many industrial instances, originated by real-world problems [Morgado *et al.*, 2013] such as routing [Xu *et al.*, 2003] and reasoning over biological networks [Guerra and Lynce, 2012], still represent a challenge for modern solvers [Argelich *et al.*, 2014]. Several MaxSAT algorithms were proposed, also in terms of broader frameworks [Bacchus *et al.*, 2014; Marques-Silva and Janota, 2014]. Nowadays, the most promising algorithms work by iteratively checking satisfiability of a formula that is properly modified during the search until a satisfying assignment is found, which eventually results in an optimum solution. In more detail, the check does not distinguish between hard and soft clauses, and each inconsistency is associated with a set of unsatisfiable clauses, referred to as *unsatisfiable core* in the literature. Soft clauses in an unsatisfiable core are thus relaxed by adding fresh variables. Clauses or cardinality constraints are also introduced in order to minimize the effect of the relaxation.

Several core-guided strategies were proposed for implementing such a relaxation step. Among them are approaches that introduce a cardinality constraint, possibly encoded in clauses [Eén and Sörensson, 2006; Roussel and Manquinho, 2009], and those that add clauses resulting from MaxSAT resolution [Larrosa and Heras, 2005]. Two prominent examples of algorithms based on these strategies are OLL [Andres *et al.*, 2012] and PMRES [Narodytska and Bacchus, 2014], implemented in solvers that had excellent results in the 2014 MaxSAT Evaluation [Argelich *et al.*, 2014]. In particular, OLL is used by CLASP and MSCG [Morgado *et al.*, 2014], and PMRES is implemented by EVA. However, besides pros there are also cons, as will be clarified soon.

OLL introduces a cardinality constraint for enforcing that at most one soft clause in the core can be unsatisfied in a solution. If such a constraint cannot be satisfied then a core containing the new constraint is detected. In this case, the algorithm introduces another constraint for enforcing that at most two soft clauses in the first core can be unsatisfied in a solution, and so on. Hence, a first drawback of OLL is that it may introduce several constraints for handling one core, possibly slowing down the propagation. Such a drawback was overcome by MSCG thanks to a smart translation into a sorting network that compactly represents all constraints possibly introduced by OLL for handling one core. However, even within this smart encoding, adding cardinality constraints may increase exponentially the number of refutations in subsequent checks [Bacchus and Narodytska, 2014].

PMRES introduces clauses of bounded size resulting from MaxSAT resolution. Essentially, two soft clauses are replaced by two new soft formulas being the disjunction and the conjunction of the original clauses. The conjunction is encoded in clauses, and the variable associated with its satisfaction is resolved with another soft clause in the core, so that the process is repeated hierarchically for all clauses in the core. However, even if on the one hand this process inhibits the exponential grow up on the number of refutations, on the other hand several variables introduced by PMRES for processing a

*This work was partially supported by MIUR within project "SILAB BA2KNOW – Business Analitycs to Know", and by Regione Calabria, POR Calabria FESR 2007-2013, within project "ITravel PLUS" and project "KnowRex". Mario Alviano was partly supported by the National Group for Scientific Computation (GNCS-INDAM), and by Finanziamento Giovani Ricercatori UNICAL.

core can jointly be part of a new core, and this event is influenced by the order in which soft clauses are processed.

A new algorithm for processing unsatisfiable cores, namely ONE, is introduced in this paper. ONE is inspired by OLL but has the advantage of introducing exactly one cardinality constraint for each detected core, which also means that it is simpler to implement. In fact, while new constraints must be added by OLL when a previously introduced constraint belongs to some unsatisfiable core, the constraint added by the new algorithm does not require special treatments in subsequent satisfiability checks. Actually, this is also the case for the variant of OLL using sorting networks, as implemented in MSCG, but the algorithm presented here does not interdict the use of native pseudo-Boolean solvers, which may be an advantage for sufficiently small constraints.

The generalization of ONE, called K, is motivated by the exponential grow up on the number of refutations due to the addition of cardinality constraint. In this second form, a parameter bounds the size of cardinality constraints introduced for handling cores. The processed core is indeed partitioned in sets of bounded size, and each partition is associated with a cardinality constraint. It is interesting to observe that K also generalizes PMRES, which is obtained for the smallest allowed value of the parameter.

ONE and K have been implemented in a new pseudo-Boolean solver extending GLUCOSE [Audemard and Simon, 2009], and their performance have been assessed on the industrial instances of the 2014 MaxSAT Evaluation. Several values of the parameter bounding the size of cardinality constraints were tested, many of which resulted in a performance comparable with state-of-the-art solvers such as EVA and MSCG. Actually, when constraints are bound to contain at most 48 literals, MAXINO solves 12 instances more than EVA and 17 more than MSCG. Also the average execution time benefits of the new algorithm on many tested instances. Prototype and formal proofs are available online at the following link: http://alviano.net/software/maxino/.

## 2 Background

Let $\mathcal{V}$ be a countable set of propositional variables. A cost function $w$ is a function $w : \mathcal{V} \to \mathbb{N}$ associating each variable $x$ with its cost $w(x)$. A literal is either a variable $x$, or its negation $\neg x$. The complement of a literal $\ell$ is denoted $\overline{\ell}$, i.e., $\overline{x} = \neg x$ and $\overline{\neg x} = x$ for all $x \in \mathcal{V}$. A pseudo-Boolean constraint, or simply *constraint*, is of the following form:

$$a_1\ell_1 + \cdots + a_n\ell_n \geq k \tag{1}$$

where $n \geq 1$, $k \geq 0$, $a_i \geq 1$ and $\ell_i$ is a literal, for all $i \in [1..n]$. If $a_1 = \cdots = a_n = 1$ then (1) is a *cardinality constraint*. If in addition $k = 1$ then (1) is also called *clause* and possibly denoted as a disjunction $\ell_1 \vee \cdots \vee \ell_n$, or as an implication $\overline{\ell_1} \wedge \cdots \wedge \overline{\ell_{n-1}} \to \ell_n$. A propositional theory $\phi$ is a set of constraints. If all constraints in $\phi$ are clauses, $\phi$ is also called propositional formula. The set of variables occurring in $\phi$ is denoted $vars(\phi)$.

An interpretation $I$ is a function $I : \mathcal{V} \to \{\mathbf{T}, \mathbf{F}\}$ associating each variable $x$ with a Boolean truth value $I(x)$. An interpretation $I$ is also denoted as the set $\{x \in \mathcal{V} \mid I(x) = \mathbf{T}\}$.

**Algorithm 1:** Core-guided MaxSAT algorithm

**Input** : A consistent formula $\phi$, a cost function $w(\cdot)$
**Output**: An optimum model $I$ of $\phi$, and its cost

```
1 begin
2     lower_bound := 0;
3     (sat, I, C) := Solve(φ ∪ {x ∈ vars(φ) | w(x) ≥ 1});
4     if sat then return (I, lower_bound);
5     lower_bound := lower_bound + min_{x∈C} w(x);
6     for x ∈ C do w(x) := w(x) − min_{x∈C} w(x);
7     φ = ProcessCore(φ, w, C);
8     goto 3;
```

Two sets $S, S'$ of interpretations are *equivalent* with respect to a context $V \subseteq \mathcal{V}$, denoted $S \equiv_V S'$, if $\{I \cap V \mid I \in S\} = \{I \cap V \mid I \in S'\}$. Relation $\models$ is inductively defined as follows: For a variable $x \in \mathcal{V}$, $I \models x$ if $x \in I$, and $I \models \neg x$ if $x \notin I$. For a constraint $c$ of the form (1), $I \models c$ if the following inequality is satisfied:

$$\sum_{i \in [1..n],\ I \models \ell_i} a_i \geq k. \tag{2}$$

For a theory $\phi$, $I \models \phi$ if $I \models c$ for all $c \in \phi$. If $I \models \phi$ then $I$ is a *model* of $\phi$. The cost of $I$ is defined as follows:

$$w(I) := \sum_{x \in \mathcal{V} \setminus I} w(x). \tag{3}$$

A theory $\phi$ is *consistent* if $\phi$ has at least one model, otherwise $\phi$ is *inconsistent* and each subset $C$ of $\phi$ being inconsistent is called an *unsatisfiable core* of $\phi$. A model $I$ of $\phi$ is *optimum* with respect to a cost function $w$ if there is no $J \subseteq \mathcal{V}$ such that $J \models \phi$ and $w(J) < w(I)$. The set of optimum models of $\phi$ with respect to $w$ is denoted $\mathcal{O}(\phi, w)$.

The *relaxation* of a clause $c$, denoted $relax(c)$, is the clause $c \vee \neg r_c$, where $r_c$ is a fresh variable, i.e., a variable not occurring elsewhere. The computational problem analyzed in this paper is *Weighted Partial MaxSAT* (WPMS): Given a formula $\phi^h \cup \phi^s$, and a function $w^s : \phi^s \to \mathbb{N}^+$, compute an optimum model of $\phi^h \cup \{relax(c) \mid c \in \phi^s\}$ with respect to a cost function $w$ such that $w(r_c) = w^s(c)$ for all $c \in \phi^s$, and $w(x) = 0$ for all other variables. If $w^s(c) = 1$ for all $c \in \phi^s$ then the computational problem is called *Partial MaxSAT* (PMS), and if in addition $\phi^h = \emptyset$ then it is referred to as *MaxSAT* (MS).

## 3 Core-guided Algorithm

Algorithms for computing solutions of the computational problems introduced in the previous section are presented here. To simplify the presentation, algorithms will be given for an input comprising a formula $\phi$ and a cost function $w : \mathcal{V} \to \mathbb{N}$, obtained as described in the previous section and with the additional assumption that $\phi$ is consistent.

### 3.1 The Basic Algorithm ONE

An abstract procedure for computing an optimum model of $\phi$ is reported in Algorithm 1. It uses an external function Solve for searching a model or an unsatisfiable core of a formula

**Function** ProcessCore($\phi, w, C$)

1 **begin**
2 Let $C = \{x_0, \ldots, x_n\}$, where $n \geq 0$;
3 Let $r_1, \ldots, r_n$ be fresh variables;
4 **for** $i \in [1..n]$ **do** $w(r_i) := \min_{x \in C} w(x)$;
5 $\phi := \phi \cup \{x_0 + \cdots + x_n + \neg r_1 + \cdots + \neg r_n \geq n\}$;
6 **for** $i \in [1..n-1]$ **do** $\phi := \phi \cup \{r_i \rightarrow r_{i+1}\}$;

$\phi'$ obtained by extending $\phi$ with unary clauses of the form $x$, where $x$ is a variable with positive cost (line 3). In more detail, the output of $Solve(\phi')$ is either $(\mathbf{T}, I, -)$ in case $I \models \phi'$, or $(\mathbf{F}, -, C)$ if there is an unsatisfiable core $\phi''$ of $\phi'$ such that $\phi'' \cap \{x \in vars(\phi) \mid w(x) \geq 1\} = C$.

In case function Solve returns $(\mathbf{F}, -, C)$, the minimum cost associated with variables in $C$ is computed, i.e., $\min_{x \in C} w(x)$. The core actually provides evidence that the cost of any model of $\phi$ must be greater or equal to this value. Hence, a lower bound is stored and updated during the execution of the algorithm. Initially, its value is 0 (line 2), and it is increased by $\min_{x \in C} w(x)$ whenever function Solve returns an unsatisfiable core $C$ (line 5). Quantity $\min_{x \in C} w(x)$ is then removed from the cost of all variables in $C$ (line 6), and a total cost of $(|C| - 1) \cdot \min_{x \in C} w(x)$ is equally distributed to $|C| - 1$ new variables $r_1, \ldots, r_n$ by function ProcessCore (line 4 of the function). Such a function can implement one of several strategies. The one reported here is new and uses a constraint to enforce the satisfaction of as many as possible variables in the set $C = \{x_0, \ldots, x_n\}$ (line 5). If $n$ of these variables can be assigned $\mathbf{T}$ in a model of $\phi$, then all new variables $r_1, \ldots, r_n$ can be assigned $\mathbf{T}$ as well. On the other hand, if only $n - 1$ variables in $\{x_0, \ldots, x_n\}$ can be assigned $\mathbf{T}$ in a model of $\phi$, then one variable among $r_1, \ldots, r_n$ must be assigned $\mathbf{F}$ in order to satisfy the constraint. Actually, clauses of the form $r_i \rightarrow r_{i+1}$ ensure that such a variable must be $r_1$ (line 6). By generalizing the previous argument, formalized later in Lemma 1, it is possible to conclude that if only $m \in [0..n-1]$ variables in $\{x_0, \ldots, x_n\}$ can be assigned $\mathbf{T}$ in a model of $\phi$, then variables $r_1, \ldots, r_{n-m}$ must be assigned $\mathbf{F}$ in order to satisfy the constraint.

The algorithm then continues to search for an optimum model, processing unsatisfiable cores as they are identified and incrementing the lower bound accordingly. Finally, when a model $I$ is returned by function Solve, the algorithm terminates, and outputs $I$ and the current lower bound.

**Example 1.** Let $\phi$ be a formula equivalent to the theory $\{\neg a + \neg b + \neg c + \neg d \geq 2\}$, and let $w$ be such that $w(a) = w(b) = 1$ and $w(c) = w(d) = 2$. Algorithm 1 searches for a model or an unsatisfiable core of $\phi \cup \{a, b, c, d\}$. Function Solve returns $(\mathbf{F}, -, \{a, c, d\})$, and set $\{a, c, d\}$ is processed: The lower bound is increased to 1, fresh variables $r_1, r_2$ are introduced, constraint $a + c + d + \neg r_1 + \neg r_2 \geq 2$ and clause $r_1 \rightarrow r_2$ are added to $\phi$, and $w$ becomes $w(a) = 0$, $w(b) = w(c) = w(d) = w(r_1) = w(r_2) = 1$. Function Solve is now invoked for $\phi \cup \{b, c, d, r_1, r_2\}$, and a new unsatisfiable core $\{b, c, d\}$ is found and processed: The lower bound is increased to 2, fresh variables $r_3, r_4$ are introduced, constraint $b + c + d + \neg r_3 + \neg r_4 \geq 2$ and clause $r_3 \rightarrow r_4$ are added to $\phi$, and $w$ becomes $w(a) = w(b) = w(c) = w(d) = 0$, $w(r_1) = w(r_2) = w(r_3) = w(r_4) = 1$. Theory $\phi \cup \{r_1, r_2, r_3, r_4\}$ is now satisfiable, and function Solve returns $(\mathbf{T}, \{c, d, r_1, r_2, r_3, r_4\}, -)$. It is an optimum model of the original formula, and its cost is 2. ■

**Function** $k$-ProcessCore($\phi, w, C$)

1 **begin** // use constraints of size $2 \cdot (k+1)$
2 Let $C = \{x_0, \ldots, x_{n \cdot k}\}$, where $n \geq 0$;
3 Let $r_1, \ldots, r_{n \cdot k}$ be fresh variables;
4 **for** $i \in [1..n \cdot k]$ **do** $w(r_i) := \min_{x \in C} w(x)$;
5 Let $c_0 = x_0$, and $c_1, \ldots, c_n$ be fresh variables;
6 **for** $i \in [1..n]$ **do**
7 $\phi := \phi \cup \{c_{i-1} + x_{(i-1)k+1} + \cdots + x_{i \cdot k} + \neg c_i + \neg r_{(i-1)k+1} + \cdots + \neg r_{i \cdot k} \geq k+1\}$;
8 $\phi := \phi \cup \{c_i \rightarrow r_{(i-1)k+1}\}$;
9 **for** $j \in [1..k-1]$ **do**
10 $\phi := \phi \cup \{r_{(i-1)k+j} \rightarrow r_{(i-1)k+j+1}\}$;

## 3.2 The Parametrized Algorithm K

A drawback of core-guided algorithms making use of cardinality constraints was reported in [Bacchus and Narodytska, 2014]. Actually, each $r_i$ introduced by function ProcessCore is associated with several assignments, which may become computationally expensive to refute in subsequent calls to function Solve when the cost of some $r_i$ is decreased to 0. In particular, note that in $x_0 + \cdots + x_n + \neg r_1 + \cdots + \neg r_n \geq n$, variable $r_1$ is associated with the satisfaction of $x_0 + \cdots + x_n = n$, which is the case for $n + 1$ different interpretations of variables $x_0, \ldots, x_n$. Similarly, $r_2$ is associated with the satisfaction of $x_0 + \cdots + x_n = n - 1$, i.e., with $\binom{n+1}{2}$ different interpretations. For a generic $i \in [1..n]$, $r_i$ is associated with $x_0 + \cdots + x_n = i$, i.e., $\binom{n+1}{i}$ interpretations, which is an exponential number in general.

A different strategy for processing unsatisfiable cores, referred to as PMRES [Narodytska and Bacchus, 2014], introduces clauses of two and three literals instead of cardinality constraints. In fact, for a core $\{x_0, \ldots, x_n\}$, PMRES produces, for all $i \in [1..n]$, the following clauses:

$$c_{i-1} \vee x_i \vee \neg r_i \quad c_{i-1} \wedge x_i \rightarrow c_i \quad c_i \rightarrow c_{i-1} \quad c_i \rightarrow x_i$$

where $c_0 = x_0$ and $c_1, \ldots, c_n$ are fresh variables with cost 0, and $r_1, \ldots, r_n$ are as in function ProcessCore. In a nutshell, the above clauses enforce the satisfaction of as many as possible of the following formulas: $x_0 \vee x_1$, $(x_0 \wedge x_1) \vee x_2$, $(x_0 \wedge x_1 \wedge x_2) \vee x_3$, and so on, where conjunctions are compactly represented by variables $c_1, \ldots, c_n$. Actually, variable $c_n$ is not really introduced, but it is left here to simplify the presentation. Moreover, correctness of PMRES can be proved also if clauses of the form $c_{i-1} \wedge x_i \rightarrow c_i$ are replaced by clauses of the form $c_i \rightarrow r_i$, as will be proved later.

By elaborating on the previous observations, a new characterization of PMRES in terms of cardinality constraints is provided below:

$$c_{i-1} + x_i + \neg c_i + \neg r_i \geq 2 \qquad c_i \rightarrow r_i$$

for all $i \in [1..n]$. In fact, the truth of $c_i$ implies the truth of $r_i$ because of $c_i \rightarrow r_i$, and therefore also of $c_{i-1}$ and $x_i$ because of the constraint. It is now evident that PMRES limits the exponential blow up on the number of refutations associated with new variables by limiting the size of constraints to a small, fixed constant. A natural question is thus whether the above formulation of PMRES can be combined somehow with the constraints and clauses introduced by function ProcessCore as presented in the previous section, in order to relax the limit on the size of constraints. A positive answer to this question is provided by function $k$-ProcessCore, where parameter $k$ limits the size of cardinality constraints to be at most $2 \cdot (k+1)$. To simplify the presentation, the function is presented for a set $C$ of exactly $n \cdot k + 1$ variables, where $n \geq 0$. Such an assumption does not limit anyhow the generality of the algorithm because any set of variables can be properly padded with fresh variables to reach the required cardinality, or more pragmatically the last added constraint may be properly modified for containing less literals. An example should better clarify how unsatisfiable cores are processed by the new function.

**Example 2.** Let $C = \{a, b, c, d, e\}$. Function ProcessCore would introduce a constraint of the form $a + b + c + d + e + \neg r_1 + \neg r_2 + \neg r_3 + \neg r_4 \geq 4$, and clauses $r_1 \rightarrow r_2$, $r_2 \rightarrow r_3$, $r_3 \rightarrow r_4$. On the other hand, function 2-ProcessCore would introduce two constraints, $a + b + c + \neg c_1 + \neg r_1 + \neg r_2 \geq 3$ and $c_1 + d + e + \neg c_2 + \neg r_3 + \neg r_4 \geq 3$, and clauses $c_1 \rightarrow r_1$, $r_1 \rightarrow r_2$, $c_2 \rightarrow r_3$ and $r_3 \rightarrow r_4$. If instead $C = \{a, b, c, d\}$, the last constraint would be $c_1 + d + \neg c_2 + \neg r_3 \geq 2$. ■

Note that PMRES is essentially function 1-ProcessCore, while function ProcessCore is obtained for unbounded $k$, or more precisely for $k > \sum_{x \in \mathcal{V}} w(x)$.

### 3.3 Correctness

In order to prove the correctness of Algorithm 1, the argument reported in Section 3.1 is first formalized.

**Lemma 1.** *Let $\phi$ be a propositional theory, $w$ be a cost function, and $\{x_0, \ldots, x_n\} \subseteq \mathcal{V}$ be such that $n \geq 0$ and $w(x_i) \geq 1$ for all $i \in [0..n]$. Let $m = \min_{i \in [0..n]} w(x_i)$, and $r_0, \ldots, r_n$ be fresh variables. Define $\phi'$ and $w'$ as follows:*

$$\begin{aligned} \phi' := \phi \cup \{x_0 + \cdots + x_n + \neg r_0 + \cdots + \neg r_n \geq n+1\} \\ \cup \{r_i \rightarrow r_{i+1} \mid i \in [0..n-1]\} \end{aligned} \tag{4}$$

$$w'(x) := \begin{cases} w(x) - m & \text{if } x \in \{x_0, \ldots, x_n\} \\ m & \text{if } x \in \{r_0, \ldots, r_n\} \\ w(x) & \text{otherwise.} \end{cases} \tag{5}$$

*The following relation holds: $\mathcal{O}(\phi, w) \equiv_{vars(\phi)} \mathcal{O}(\phi', w')$.*

Correctness of the algorithm using function ProcessCore easily follows by noting that if set $C = \{x_0, \ldots, x_n\}$ is returned by function Solve, then $x_0 + \cdots + x_n = n + 1$ cannot be satisfied, and therefore $r_0$ must be assigned **F** in all models of the theory $\phi$. Similarly, correctness when function $k$-ProcessCore is used can be established by iteratively applying Lemma 1 on chunks of set $C$. In more detail, if $C = \{x_0, \ldots, x_{n \cdot k}\}$, then the constraint and clauses added for $i = 1$ preserve optimum models. Moreover, variable $r_0$ in Lemma 1 is variable $c_1$ in the function, and it is essentially equivalent to the conjunction $x_0 \wedge \cdots \wedge x_n$. This variable is then used in the subsequent constraint, following the idea of PMRES, and Lemma 1 can be applied again to prove that optimum models are preserved for all $i \in [1..n]$. As for ProcessCore, the unsatisfiable core returned by function Solve guarantees that at least one variable among $x_0, \ldots, x_{n \cdot k}$, and therefore $c_n$, must be assigned **F** in any model of $\phi$.

**Theorem 1.** *Algorithm 1 is correct, that is, for any consistent formula $\phi$ and cost function $w$, it outputs a pair $(I, w(I))$, where $I \in \mathcal{O}(\phi, w)$. Correctness holds also if function $k$-ProcessCore is used, for all $k \geq 1$.*

Since 1-ProcessCore is essentially PMRES, the following is a corollary of the above theorem:

**Corollary 1.** PMRES *is correct also if clauses of the form $c_{i-1} \wedge x_i \rightarrow c_i$ are replaced by clauses of the form $c_i \rightarrow r_i$.*

## 4 Implementation and Evaluation

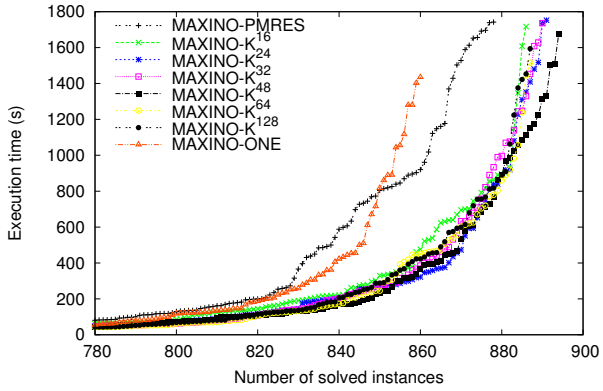
ONE and K have been implemented in MAXINO, a new pseudo-Boolean solver extending the single-thread version of GLUCOSE 4 [Audemard and Simon, 2009], and featuring additional evaluation techniques such as disjoint core analysis and, for weighted instances, stratification, hardening and Boolean multilevel optimization [Ansótegui *et al.*, 2009; 2013; Argelich *et al.*, 2009]. To ease a comparison with state-of-the-art solvers, PMRES has been implemented as well according to [Narodytska and Bacchus, 2014].

The performance of MAXINO was assessed on the industrial instances of the 2014 MaxSAT Evaluation [Argelich *et al.*, 2014], which includes three tracks, namely *MaxSAT* (MS; 55 instances), *Partial MaxSAT* (PMS; 568 instances) and *Weighted Partial MaxSAT* (WPMS; 410 instances). The experiment was run on an Intel Xeon CPU 2.4 GHz with 16 GB of RAM. CPU and memory usage were limited to 1,800 seconds and 8 GB, respectively. MAXINO was compared with the (non-portfolio) solvers that better performed on industrial instances of the 2014 MaxSAT Evaluation, namely EVA 500a [Narodytska and Bacchus, 2014], OPENWBO 1.3.0 [Martins *et al.*, 2014], MSCG [Ignatiev *et al.*, 2014; Morgado *et al.*, 2014] and CLASP 3.1.1 [Andres *et al.*, 2012] (with options --opt-strategy=usc,1 --opt-sat --configuration=trendy). The comparison with EVA, CLASP and MSCG is of particular interest. In fact, EVA implements PMRES, while OLL is used by CLASP, and also by MSCG on weighted instances.

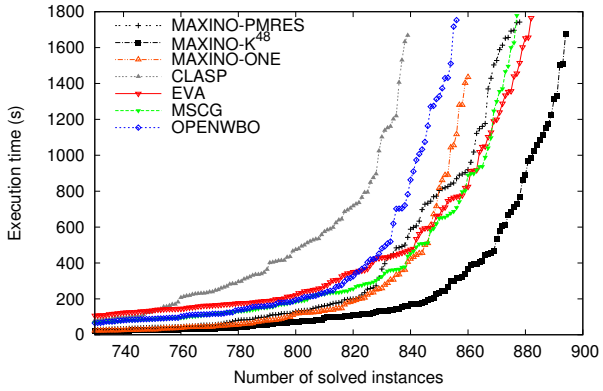
A summary of the result is reported in Table 1, where the best performance overall and in each track is emphasized in bold. In the table, PMRES is MAXINO running PMRES, ONE is MAXINO running ONE, and $\text{K}^n$ is MAXINO running K with constraints of size at most $n$. As a general comment, it can be observed that MAXINO does not reach performance of state-of-the-art solvers neither with PMRES nor by ONE, which respectively solve 4 and 22 instances less than EVA. On the other hand, MAXINO has a performance boost when K is used, and in fact the tested versions solve at least 4 instances more than EVA, with a peak of 12 for $\text{K}^{48}$. Comparing with the other solvers, $\text{K}^{48}$ solves 17, 38 and 55 more instances than MSCG, OPENWBO and CLASP, respectively. The same ranking is obtained if memory is limited to 3.5 GB as in the Evalu-

| | PMRES | | $K^{16}$ | | $K^{24}$ | | $K^{32}$ | | $K^{48}$ | | $K^{64}$ | | $K^{128}$ | | ONE | | CLASP | | EVA | | MSCG | | OPENWBO | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **Track** | **sol.** | **avg t** | **sol.** | **avg t** | **sol.** | **avg t** | **sol.** | **avg t** | **sol.** | **avg t** | **sol.** | **avg t** | **sol.** | **avg t** | **sol.** | **avg t** | **sol.** | **avg t** | **sol.** | **avg t** | **sol.** | **avg t** | **sol.** | **avg t** |
| **MS** | 41 | 148.1 | 46 | 148.5 | 45 | 119.0 | 44 | 107.1 | 45 | 122.8 | 44 | 95.7 | **46** | **128.0** | 40 | 185.9 | 41 | 95.6 | 40 | 159.2 | 44 | 98.9 | 45 | 70.5 |
| **PMS** | 469 | 99.1 | 474 | 62.8 | 478 | 65.8 | 480 | 70.4 | **481** | **68.9** | 476 | 58.8 | 473 | 55.2 | 453 | 43.2 | 458 | 73.4 | 474 | 121.4 | 470 | 102.8 | 476 | 81.4 |
| **WPMS** | 368 | 18.4 | 366 | 10.2 | 368 | 12.8 | 366 | 8.6 | **368** | **11.1** | 367 | 8.1 | 368 | 14.3 | 367 | 10.6 | 340 | 48.9 | 368 | 54.4 | 363 | 18.7 | 335 | 30.0 |
| **Total** | 878 | 67.5 | 886 | 45.5 | 891 | 46.6 | 890 | 46.8 | **894** | **47.8** | 887 | 39.7 | 887 | 42.0 | 860 | 35.9 | 839 | 64.6 | 882 | 95.2 | 877 | 67.8 | 856 | 60.7 |

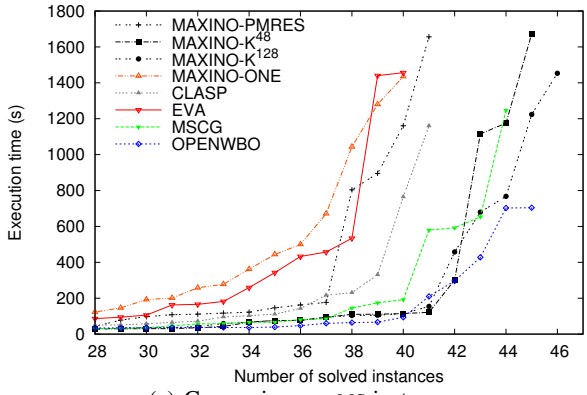
Table 1: Number of solved instances and average execution time in seconds of MAXINO (left) and state-of-the-art solvers (right).



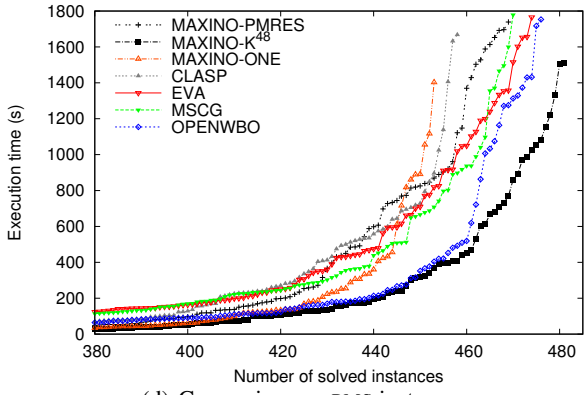

(a) Overall performance of the implemented solver.



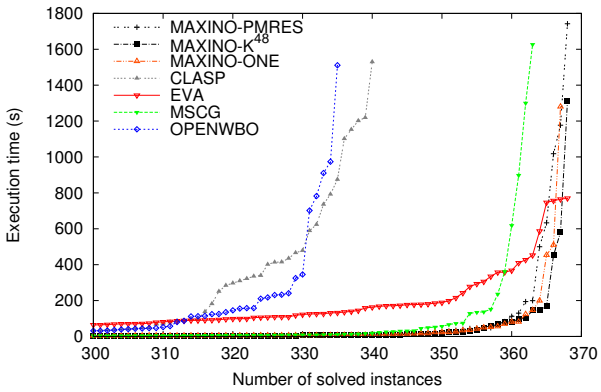

(b) Overall comparison with state-of-the-art solvers.



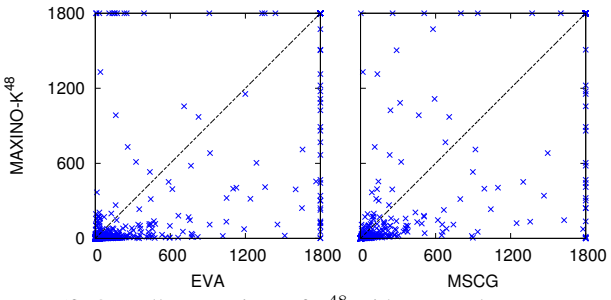

(c) Comparison on MS instances.




(d) Comparison on PMS instances.




(e) Comparison on WPMS instances.




(f) Overall comparison of $K^{48}$ with EVA and MSCG.

Figure 1: Cactus plots (a)–(e) and scatter plots (f) comparing the performance of MAXINO with state-of-the-art solvers.

ation, as the difference is of 4 instances for MSCG, 1 instance for CLASP and OPENWBO, and 2 instances for MAXINO.

Figure 1(a) is a cactus plot of the tested versions of MAXINO. Recall that a cactus plot reports for each solver the number of solved instances in a given time, producing an aggregated view of its overall performance. The performance of ONE is the first that deteriorate, followed by PMRES, while the several K$^n$ can solve more instances and always in less time. Even if approximately the same good performance is achieved for all tested parameters, the line of K$^{48}$ clearly indicate a leader in this group. It is also interesting to observe that the performance of MAXINO improves by 8 instances when PMRES is replaced with K$^{16}$, and even more instances are solved when the parameter is further increased, until a point in which the overall performance starts to deteriorate and is expected to align with ONE. Indeed, the benchmark was also run for K$^{256}$, K$^{512}$ and K$^{1,024}$, which are not reported in the table, and the expected trend was observed as 880, 875 and 870 instances were solved, respectively.

Further comparisons with state-of-the-art solvers are shown in Figure 1. In particular, Figure 1(b) highlights that the overall performance of PMRES, EVA and MSCG are comparable, and are all overcome by K$^{48}$. The remaining cactus plots report the results track by track. Concerning MS, reported in Figure 1(c), the performance of K$^{48}$ is overcome by OPENWBO, which solves the same number of instances but in less time. However, it is also true that K$^{128}$ solves 1 instance more than K$^{48}$ and OPENWBO, which can be explained by the fact that huge cores of up to 500,000 relaxing variables are detected for instances in this track. For such instances it seems to be preferable to introduce larger constraints, so that less constraints have to be added for processing a single core. Concerning PMS, K$^{48}$ solves 5, 7, 11 and 23 instances more than OPENWBO, EVA, MSCG and CLASP, respectively. Figure 1(d) also shows that K$^{48}$ is usually faster than EVA, MSCG and CLASP, while OPENWBO is more comparable in speed. Concerning WPMS, PMRES, K$^{24}$, K$^{48}$, K$^{128}$ and EVA have the best performance in terms of solved instances, but Figure 1(e) highlights that K$^{48}$ is around 5 time faster than EVA.

An instance by instance comparison of K$^{48}$ with EVA and MSCG, the state-of-the-art solvers that best performed, is reported in the scatter plots in Figure 1(f). In these plots a point $(x, y)$ is reported for each instance, where $x$ is the running time of EVA or MSCG, and $y$ is the running time of K$^{48}$. It can be observed that K$^{48}$ is generally much faster than EVA and MSCG, as the majority of points is located below the diagonal. In fact, K$^{48}$ is faster than EVA and MSCG in respectively 758 and 672 instances, and slower only 133 and 198 times.

## 5 Related Work

The first core-guided algorithm for MaxSAT was introduced in the seminal work of [Fu and Malik, 2006], which originated the MSU family of algorithms [Marques-Silva and Manquinho, 2008; Marques-Silva and Planes, 2008], eventually extended to solve weighted instances [Manquinho *et al.*, 2009; Ansótegui *et al.*, 2009]. The paper follows this line of research, and introduces two core-guided algorithms, namely ONE and K. ONE is inspired by OLL [Andres *et al.*, 2012]. Intuitively, for an unsatisfiable core $\{x_0, \ldots, x_n\}$ ($n \geq 0$), OLL introduces a cardinality constraint of the form $x_0 + \cdots + x_n + n \cdot \neg r_1 \geq n$, where $r_1$ is a fresh variable whose weight is set to $\min_{i \in [0..n]} w(x_i)$. The idea here is that at least one variable in $\{x_0, \ldots, x_n\}$ must be assigned $\mathbf{F}$, but there are still chances that $n$ of them can be assigned $\mathbf{T}$. If this is not the case and an unsatisfiable core containing variable $r_1$ is detected, a new constraint $x_0 + \cdots + x_n + n \cdot \neg r_2 \geq n - 1$ is introduced in addition to the one associated with the new core. Hence, the weighted variable $r_2$ is associated with the satisfaction of at least $n-1$ variables in $\{x_0, \ldots, x_n\}$. Therefore, in general OLL may introduce $n$ different constraints for a core of $n - 1$ variables. ONE instead always adds a single constraint for each detected core. A similar strategy was implemented in MSCG [Morgado *et al.*, 2014], where a sorting network with input $\{x_0, \ldots, x_n\}$ and output $\{r_1, \ldots, r_n\}$ smartly encodes all constraints introduced by OLL.

K, instead, is inspired by PMRES [Narodytska and Bacchus, 2014], an algorithm based on MaxSAT resolution [Larrosa and Heras, 2005] overcoming the drawback associated with the exponential number of refutations that current algorithms based on cardinality constraints must face. Indeed, as discussed in Section 3.2, PMRES handles cores by means of clauses of two and three literals. However, by restating PMRES in terms of cardinality constraints, a more general algorithm emerges, which eventually results in the parametrized K. Actually, K combines the strengths of both PMRES and OLL. In fact, by bounding the size of constraints as in PMRES, K is not affected by the exponential increase of the number of refutations. Moreover, it can be observed empirically that OLL and ONE usually discover smaller cores than PMRES, which is due to the fact that several variables introduced by PMRES for processing a core can possibly be part of a new core, while this cannot happen for OLL and ONE.

## 6 Conclusion

ONE and K extend and generalize two successful core-guided algorithms, namely OLL and PMRES. ONE and K are implemented in MAXINO, a pseudo-Boolean solver extending GLUCOSE. Even if the solver itself is new, and therefore it does not have the maturity of state-of-the-art systems, it performed well on the industrial instances of the 2014 MaxSAT Evaluation. In fact, while the unbounded ONE performed worse than CLASP running OLL on unweighted instances, and the implementation of PMRES solved a few instances less than EVA, sensible performance gains have been observed for the parametrized K. It is interesting to note that such performance gains are obtained for several values of the parameter, which is an insight that usually the performance of OLL and similar algorithms deteriorates because of a few, huge cores. This may suggest, as a future work, a dynamic implementation of K, where the parameter is not fixed a priori but set on a per-core basis, according to a function taking into account the size of the core. Configuration techniques such as those used by ISAC+ [Ansótegui *et al.*, 2014] may be also considered for this purpose. Finally, ONE and K will be extended to Answer Set Programming and implemented in the ASP solver WASP [Alviano *et al.*, 2013; 2014].

## References


[Alviano *et al.*, 2013] Mario Alviano, Carmine Dodaro, Wolfgang Faber, Nicola Leone, and Francesco Ricca. WASP: A native ASP solver based on constraint learning. In Pedro Cabalar and Tran Cao Son, editors, *Logic Programming and Nonmonotonic Reasoning, 12th International Conference, LPNMR 2013, Corunna, Spain, September 15-19, 2013. Proceedings*, volume 8148 of *Lecture Notes in Computer Science*, pages 54–66. Springer, 2013.

[Alviano *et al.*, 2014] Mario Alviano, Carmine Dodaro, and Francesco Ricca. Anytime computation of cautious consequences in answer set programming. *TPLP*, 14(4-5):755–770, 2014.

[Andres *et al.*, 2012] Benjamin Andres, Benjamin Kaufmann, Oliver Matheis, and Torsten Schaub. Unsatisfiability-based optimization in clasp. In *28th International Conference on Logic Programming*, pages 211–221, Budapest, Hungary, September 2012.

[Ansótegui *et al.*, 2009] Carlos Ansótegui, Maria Luisa Bonet, and Jordi Levy. Solving (weighted) partial maxsat through satisfiability testing. In *SAT 2009*, pages 427–440, Swansea, UK, June 2009. Springer.

[Ansótegui *et al.*, 2013] Carlos Ansótegui, Maria Luisa Bonet, and Jordi Levy. SAT-based MaxSAT algorithms. *Artificial Intelligence*, 196(0):77–105, March 2013.

[Ansótegui *et al.*, 2014] 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 Twenty-Eighth AAAI Conference on Artificial Intelligence, July 27 -31, 2014, Québec City, Québec, Canada.*, pages 2594–2600. AAAI Press, 2014.

[Argelich *et al.*, 2009] Josep Argelich, Inês Lynce, and João P. Marques Silva. On solving boolean multilevel optimization problems. In *21st International Joint Conference on Artificial Intelligence*, pages 393–398, Pasadena, California, July 2009. IJCAI Organization.

[Argelich *et al.*, 2014] Josep Argelich, Chu Min Li, Felip Manya, and Jordi Planes. The 2014 MaxSAT Evaluation, 2014. http://www.maxsat.udl.cat/14.

[Audemard and Simon, 2009] Gilles Audemard and Laurent Simon. Predicting learnt clauses quality in modern SAT solvers. In *21st International Joint Conference on Artificial Intelligence*, pages 399–404, Pasadena, California, July 2009. IJCAI Organization.

[Bacchus and Narodytska, 2014] Fahiem Bacchus and Nina Narodytska. Cores in core based maxsat algorithms: An analysis. In *SAT 2014*, pages 7–15, Vienna, Austria, July 2014. Springer.

[Bacchus *et al.*, 2014] Fahiem Bacchus, Jessica Davies, Maria Tsimpoukelli, and George Katsirelos. Relaxation search: A simple way of managing optional clauses. In Carla E. Brodley and Peter Stone, editors, *AAAI 2014*, pages 835–841. AAAI Press, 2014.

[Cha *et al.*, 1997] Byungki Cha, Kazuo Iwama, Yahiko Kambayashi, and Shuichi Miyazaki. Local search algorithms for partial MAXSAT. In *Fourteenth National Conference on Artificial Intelligence and Ninth Innovative Applications of Artificial Intelligence Conference*, pages 263–268, Providence, Rhode Island, July 1997. AAAI Press / The MIT Press.

[Eén and Sörensson, 2006] Niklas Eén and Niklas Sörensson. Translating pseudo-boolean constraints into SAT. *Journal on Satisfiability, Boolean Modeling and Computation*, 2(1-4):1–26, March 2006.

[Fu and Malik, 2006] Zhaohui Fu and Sharad Malik. On solving the partial MAX-SAT problem. In *SAT 2006*, pages 252–265, Seattle, Washington, August 2006. Springer.

[Guerra and Lynce, 2012] João Guerra and Inês Lynce. Reasoning over biological networks using maximum satisfiability. In *Principles and Practice of Constraint Programming - 18th International Conference*, pages 941–956, Québec City, Canada, October 2012. Springer.

[Ignatiev *et al.*, 2014] Alexey Ignatiev, António Morgado, Vasco M. Manquinho, Inês Lynce, and João Marques-Silva. Progression in maximum satisfiability. In *ECAI 2014*, pages 453–458, Prague, Czech Republic, August 2014. IOS Press.

[Larrosa and Heras, 2005] Javier Larrosa and Federico Heras. Resolution in Max-SAT and its relation to local consistency in weighted CSPs. In *19th International Joint Conference on Artificial Intelligence*, pages 193–198, Edinburgh, Scotland, August 2005.

[Manquinho *et al.*, 2009] Vasco M. Manquinho, João P. Marques Silva, and Jordi Planes. Algorithms for weighted boolean optimization. In *SAT 2009*, pages 495–508, Swansea, UK, June 2009. Springer.

[Marques-Silva and Janota, 2014] João Marques-Silva and Mikolás Janota. Computing minimal sets on propositional formulae I: problems & reductions. *CoRR*, abs/1402.3011, 2014.

[Marques-Silva and Manquinho, 2008] João Marques-Silva and Vasco M. Manquinho. Towards more effective unsatisfiability-based maximum satisfiability algorithms. In *SAT 2008*, pages 225–230, Guangzhou, China, May 2008. Springer.

[Marques-Silva and Planes, 2008] João Marques-Silva and Jordi Planes. Algorithms for maximum satisfiability using unsatisfiable cores. In *Design, Automation and Test in Europe, DATE 2008*, pages 408–413, Munich, Germany, March 2008. IEEE.

[Martins *et al.*, 2014] Ruben Martins, Vasco M. Manquinho, and Inês Lynce. Open-wbo: A modular MaxSAT solver. In *SAT 2014*, pages 438–445, Vienna, Austria, July 2014. Springer.

[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, October 2013.

[Morgado *et al.*, 2014] António Morgado, Carmine Dodaro, and Joao Marques-Silva. Core-guided MaxSAT with soft cardinality constraints. In *Principles and Practice of Constraint Programming - 20th International Conference*, pages 564–573, Lyon, France, September 2014. Springer.

[Narodytska and Bacchus, 2014] Nina Narodytska and Fahiem Bacchus. Maximum satisfiability using core-guided MaxSAT resolution. In *Twenty-Eighth AAAI Conference on Artificial Intelligence*, pages 2717–2723, Québec City, Canada, July 2014. AAAI Press.

[Roussel and Manquinho, 2009] Olivier Roussel and Vasco M. Manquinho. Pseudo-boolean and cardinality constraints. In *Handbook of Satisfiability*, volume 185, pages 695–733. IOS Press, 2009.

[Xu *et al.*, 2003] Hui Xu, Rob A. Rutenbar, and Karem A. Sakallah. sub-SAT: a formulation for relaxed boolean satisfiability with applications in routing. *IEEE Trans. on CAD of Integrated Circuits and Systems*, 22(6):814–820, June 2003.