Proceedings of the Twenty-Eighth AAAI Conference on Artificial Intelligence

Maximum Satisfiability Using Core-Guided MaxSat Resolution

Nina Narodytska
University of Toronto and UNSW
Toronto, Canada and Sydney, Australia
ninan@cs.toronto.edu

Fahiem Bacchus
University of Toronto
Toronto, Canada
fbacchus@cs.toronto.edu

Abstract

Core-guided approaches to solving MaxSat have proved to be effective on industrial problems. These approaches solve a MaxSat formula by building a sequence of Sat formulas, where in each formula a greater weight of soft clauses can be relaxed. The soft clauses are relaxed via the addition of blocking variables, and the total weight of soft clauses that can be relaxed is limited by placing constraints on the blocking variables. In this work we propose an alternative approach. Our approach also builds a sequence of new Sat formulas. However, these formulas are constructed using MaxSat resolution, a sound rule of inference for MaxSat. MaxSat resolution can in the worst case cause a quadratic blowup in the formula, so we propose a new compressed version of MaxSat resolution. Using compressed MaxSat resolution our new core-guided solver improves the state-of-the-art, solving significantly more problems than other state-of-the-art solvers on the industrial benchmarks used in the 2013 MaxSat Solver Evaluation.

Introduction

Maximum Satisfiability (MaxSat) is an optimisation version of satisfiability (Sat) that its most general form contains both hard clauses and weighted soft clauses. Solving such a weighted partial MaxSat (wpm) formula involves finding a truth assignment that satisfies the hard clauses along with a maximum weight of soft clauses. Many industrial applications can be naturally encoded in MaxSat, and successful application areas include bioinformatics (Strickland, Barnes, and Sokol 2005; Graça et al. 2012), planning (Cooper et al. 2006; Zhang and Bacchus 2012), and scheduling (Vasquez and Hao 2001).

The most successful approachs to solving larger industrial problems has been core-guided approaches (Morgado et al. 2013) where a sequence of unsatisfiable cores are extracted using a Sat solver. The idea is to solve MaxSat problems by solving a sequence of Sat decision problems. These Sat decision problems are obtained via relaxations of the original problem. A drawback of this approach is that the relaxation requires adding new cardinality constraints to the formula at each step, making the problem harder to solve at each iteration.

In this work we stay within the framework of the core-guided approach. However, we avoid using cardinality constraints and instead use cores to construct a derivation of the empty clause (a refutation) using MaxRes, a sound rule of inference for MaxSat. Every MaxRes refutation has the effect of relaxing the formula, i.e., making the formula satisfiable at a lower cost, without needing blocking variables or cardinality constraints. The MaxRes rule can, however, introduce a large number of additional clauses, called *compensation clauses*, and in the worst case each refutation can increase the size of the formula quadratically. To address this problem we propose a compressed MaxSat resolution rule that introduces auxiliary variables to avoid this blowup.

Our approach allows us to significantly improve on the performance of weighted MaxSat solvers on industrial instances from the 2013 Evaluation of Max-SAT Solvers (Argelich et al. 2013). Our approach is the first to efficiently employ the MaxRes rule for solving modern industrial wpm problems, and our solver outperforms all wpm solvers that participated in the 2013 Evaluation on the industrial instances, solving 126 more instances than the best competing wpm solver on these benchmarks.

Background

A satisfiability problem ϕ consists of a set of Boolean variables $\mathcal{X}$ and a set of clauses $\mathcal{C}$, $\mathcal{C} = \{C_1, \ldots, C_m\}$. A literal l is either a variable $x_i \in \mathcal{X}$ or its negation $\overline{x_i}$. A clause C is a disjunction of literals $(l_1 \vee \cdots \vee l_n)$. We denote the set of variables in the formula ϕ by $vars(\phi)$. An assignment I of the variables $vars(\phi)$ is a mapping $vars(\phi) \mapsto \{0, 1\}$. A clause C is satisfied by an assignment, $I(C) = 1$, iff $I(l) = 1$ for some $l \in C$, otherwise C is falsified by I and $I(C) = 0$. A set of clauses $\mathcal{C}$ is satisfied by an assignment, $I(\mathcal{C}) = 1$, iff all clauses in $\mathcal{C}$ are satisfied by I. We use $\square$ to denote an empty clause containing no literals; $I(\square) = 0$ for all I. A unit clause is a clause containing only one literal. A solution of ϕ is an assignment satisfying all clauses of ϕ.

A weighted clause is a pair (C, w), where C is a clause and w is the weight of C, $w \in \mathbb{N}$, that gives the cost of violating C. Clauses with $w = \infty$ are called *hard* clauses; otherwise, the clause is a *soft* clause. A wpm consists of a set of Boolean variables $\mathcal{X}$ and a set of weighted clauses, $\phi = \{(C_1, w_1), \ldots, (C_m, w_m)\}$. A special case of wpm is Partial MaxSat, pm, where the problem weights are in

the set $\{1, \infty\}$. We denote the set of soft clauses in ϕ by *softCls*(ϕ) and the set of hard clauses by *hardCls*(ϕ).

Example 1 We use a running example throughout the paper. In our example, we consider the AtMostOne($x_1, \ldots, x_5$) constraint that ensures that at most one variable among $x_1, \ldots, x_5$ is *true*. In addition, we have a set of soft unit constraints of the form $(x_i, 1)$, $i = 1, \ldots, 5$, each incurring a cost of one if x_i is *false*. We encode the AtMostOne constraint as a set of *hard* clauses: $\cup_{i,j=1,i\neq j}^{5}\{(\overline{x_i} \vee \overline{x_j}, \infty)\}$. The full formula, which we will refer to in subsequent examples as Ψ, is Ψ = *softCls*(ϕ) $\cup$ *hardCls*(ϕ) with *softCls*(ϕ) = $\cup_{i=1}^{5}\{(x_i, 1)\}$ and *hardCls*(ϕ) = $\cup_{i,j=1,i\neq j}^{5}\{(\overline{x_i} \vee \overline{x_2}, \infty)\}$.

An assignment I is **feasible** for ϕ iff I satisfies all hard clauses of ϕ. The cost of I is the sum of weights of clauses that it falsifies: $cost(I) = \sum_{w:(C,w)\in\phi \wedge I(C)=0} w$. I is an **optimal assignment** for ϕ iff it is a feasible assignment of minimal cost. The cost of ϕ is equal to the cost of an optimal assignment and is denoted by $cost(\phi)$.

Example 2 Consider I with $I(x_5) = 1$ and $I(x_i) = 0$ $(1 \leq i \leq 4)$ in our example Ψ. I is a feasible assignment for Ψ as it satisfies all hard clauses of Ψ. I violates four soft clauses of Ψ so $cost(I) = 4$. Note that any assignment that sets more than one variable to 1 violates a hard clause, so any feasible assignment has cost 4 or more. Hence I is an optimal assignment and $cost(\Psi) = 4$.

To simplify notation, we omit weights for hard clauses. We use $C \leftrightarrow \ell$ to denote the set of clauses $\{(\bar{\ell}, C)\} \cup \bigcup_{l\in C}\{(\bar{l}, \ell)\}$. These clauses make the literal ℓ equivalent to C. We also use $cnf(\sum_i x_i \leq b)$ to denote a CNF encoding of the cardinality constraint $\sum_i x_i \leq b$.

Definition 1 (MaxRes (Larrosa and Heras 2005)) *Consider two clauses $(x \vee A, w_1)$ and $(\overline{x} \vee B, w_2)$, where A and B are disjunctions of literals. Let $m = \min(w_1, w_2)$. The weighted* MaxSat *resolution rule is:*

$$\frac{(x \vee A, w_1) \qquad (\overline{x} \vee B, w_2)}{\begin{array}{c}(A \vee B, m)\\ (x \vee A, w_1 - m)\,(\overline{x} \vee B, w_2 - m)\\ (x \vee A \vee \overline{B}, m)\,(\overline{x} \vee \overline{A} \vee B, m)\end{array}}$$

where $(x \vee A, w_1)$ and $(\overline{x} \vee B, w_2)$ are the premises, the first inferred clause is a soft version of the ordinary resolvant. The remaining "clauses" are called compensation clauses.

In MaxRes the premises are *removed* from the formula and the resolvant and compensation clauses are added. Note that $\overline{A}$ and $\overline{B}$ are conjunctions of literals, so the compensation "clauses" might not be clauses. However, they can be converted to a set of clauses using a special conversion rule. If A is empty then A is regarded as a *false* literal and $\overline{A}$ is a *true* literal (similarly for B). False literals can be removed from an inferred clause, and the clause vanishes if its weight is zero, it is a tautology, it contains a *true* literal, or it is subsumed by a hard clause (ignoring the weights). The MaxSat rule was shown to be sound in (Larrosa and Heras 2005).

We also need the notion of MaxSat reducibility from (Ansótegui, Bonet, and Levy 2013). ϕ_1 is **MaxSat reducible** to ϕ_2 if, for any assignment $I : var(\phi_1) \to \{0, 1\}$, we have $cost(I(\phi_1)) = cost(I(\phi_2))$.

A MaxRes Core-based algorithm

Before we describe our new MaxRes based algorithm for solving PM problems we define a restricted version of the MaxRes rule when $w_1 = w_2$ or $w_1 = \infty$.

Definition 2 (Restricted MaxRes) *When $w_1 \in \{w_2, \infty\}$*

$$\frac{(x \vee A, w_1) \qquad (\overline{x}, w_2)}{(A, w_2), (x \vee A, w_1 - w_2), (\overline{x} \vee \overline{A}, w_2)}$$

If $w_1 = w_2$ the clause $(x \vee A, w_1 - w_2)$ vanishes, and if $w_1 = \infty$ it simply preserves the hard premiss. We call (A, w_2) the resolvant and $(\overline{x} \vee \overline{A}, w_2)$ the compensation clause.

The restricted form follows directly from applying MaxRes to the premises specified under the given restrictions (the other two compensation clauses either contain the true literal $\overline{B}$ or have weight zero); hence its soundness follows from the soundness of MaxRes.

Algorithm 1 gives our method for solving an unweighted PM problem ϕ. Later on we will generalize it to the weighted WPM case. The algorithm works by solving a sequence of Sat problems, similar to other SAT-based MaxSat solvers (Ansotegui, Bonet, and Levy 2009; Manquinho, Silva, and Planes 2009; Heras, Morgado, and Marques-Silva 2011; Davies and Bacchus 2013). At each iteration we reason about a formula ϕ^i starting with $\phi^0 = \phi$. If ϕ^i is satisfiable then the SAT solver returns a model which must be an optimal assignment for ϕ. Otherwise, it returns an unsatisfiable core, $\kappa^i \subseteq \phi^i$. Using κ^i we convert ϕ^i into a cost equivalent formula ϕ^{i+1} by modifying the clauses in κ^i and then applying MaxRes to derive an empty clause (steps 6-8). We then proceed to the next iteration.

Generating ϕ^{i+1} involves three steps. First every soft clause $(C_j, 1)$ in the new core κ^i is converted into a *unit* soft clause by making C_j equivalent to a brand new literal $\overline{b_j^i}$ (Algorithm 2). This allows us to replace $(C_j, 1)$ with the unit soft clause $(\overline{b_j^i}, 1)$. Second, using the returned set of new variables we add the hard clause that is their disjunction (step 6 of Algorithm 1). And third, we apply a series of restricted MaxRes inferences resolving the new unit soft clauses, $(\overline{b_j^i}, 1)$ against the new hard clause that is their disjunction (Algorithm 3).

Algorithm 1 PMRes

Input: $\phi = \{C_1, \ldots, C_m\}$
Output: $(I, cost(\phi))$, where I is an optimal assignment

1 $i = 0$, $\phi^0 = \phi$
2 **while** *true* **do**
3 $\quad(issat, \kappa^i, I)$ = SolveSAT($\phi^i \setminus \{\cup_{j=1}^{i}(\square, 1)\}$)
4 $\quad$**if** $issat$ **return** (I, i)
5 $\quad(\phi^*, B)$ = ReifyCore(ϕ^i, κ^i)
6 $\quad\phi^+ = \phi^* \cup \{(\vee_{b\in B} b)\}$
7 $\quad\phi^{i+1}$ = ApplyMaxRes(ϕ^+, B)
8 $\quad i = i + 1$

Algorithm 2 ReifyCore

Input: ϕ, κ
Output: ϕ, B

1 $B = \{\}$
2 **for** $(C_j, 1) \in \mathit{softCls}(\kappa)$ **do**
3 $\quad \phi = (\phi - \{(C_j, 1)\}) \cup (C_j \leftrightarrow \overline{b_j^i}) \cup \{(\overline{b_j^i}, 1)\}$
4 $\quad B = B \cup \{b_j^i\}$
5 **return** ϕ, B

Algorithm 3 ApplyMaxRes

Input: $\phi, \{b_1, b_2, \ldots, b_p\}$
Output: ϕ

1 **for** $i = 1, \ldots, p$ **do**
2 $\quad$ {Restricted MaxRes $[(b_i \vee \cdots \vee b_p, 1/\infty), (\overline{b_i}, 1)]$}
3 $\quad \phi = \phi - \{(b_i \vee b_{i+1} \vee \cdots \vee b_p, 1), (\overline{b_i}, 1)\}$
4 $\quad$ **if** $i < p$ **then**
5 $\quad\quad \phi = \phi \cup \{(b_{i+1} \vee \cdots \vee b_p, 1)\}$
6 $\quad\quad \phi = \phi \cup \{(d_i \leftrightarrow (b_{i+1} \vee \cdots \vee b_p))\} \cup \{(\overline{b_i} \vee \overline{d_i}, 1)\}$
7 $\quad$ **else**
8 $\quad\quad \phi = \phi \cup \{(\square, 1)\}$
9 **return** ϕ

Consider Algorithm 3. It resolves the hard clause $(b_1 \vee \cdots \vee b_p)$ with the unit soft clauses $(\overline{b_1}, 1), \ldots, (\overline{b_p}, 1)$ from the core deriving the empty clause $(\square, 1)$ using p applications of restricted MaxRes. The first step is to resolve $(b_1 \vee \cdots \vee b_p)$ and $(\overline{b_1}, 1)$ obtaining $(b_2 \vee \cdots \vee b_p, 1)$ and $(\overline{b_2} \vee \overline{b_3 \vee \cdots \vee b_p}, 1)$ and removing $(\overline{b_1}, 1)$[1]. In the second step we resolve $(b_2 \vee \cdots \vee b_p, 1)$ and $(\overline{b_2}, 1)$ obtaining $(b_3 \vee \cdots \vee b_p, 1)$ and $(\overline{b_3} \vee \overline{b_4 \vee \cdots \vee b_p}, 1)$ and removing both premises. Each subsequent step is similar. Finally, when $i = p$ we resolve $(b_p, 1)$ and $(\overline{b_p}, 1)$ to obtain $(\square, 1)$.

It should be noted that all the unit soft clauses $(\overline{b_i}, 1)$ as well as all the intermediate soft clauses of the form $(b_i \vee \cdots \vee b_p, 1)$ generated by the i–1^{th} application of MaxRes (added by step 5 of the algorithm) are subsequently removed: they are consumed by the next (i^{th}) application of MaxRes (removed by step 3 of the algorithm during its next iteration). The only clauses remaining after all MaxRes steps have been preformed are the initial hard clause $(b_1 \vee \cdots \vee b_p)$, the soft empty clause $(\square, 1)$, and the compensation clauses each of which has the form $(\overline{b_i} \vee \overline{b_{i+1} \vee \cdots \vee b_p}, 1)$, $2 \leq i < p$.

The compensation "clauses" are not in clausal form and (Larrosa and Heras 2005) propose a special rule for converting them into clausal form. However, their rule generates many additional clauses. Our approach is to introduce new variables d_i with $d_i \leftrightarrow (b_{i+1} \vee \cdots \vee b_p)$. With d_i we can then covert each compensation "clause" $(\overline{b_i} \vee \overline{b_{i+1} \vee \cdots \vee b_p}, 1)$ into a true clause $(\overline{b_i} \vee \overline{d_i}, 1)$. Step 6 of the algorithm preforms these two steps and adds the converted compensation clause to the formula.

In sum, the p resolution steps in Algorithm 3 removes all unit soft clauses $(\overline{b_i}, 1)$ and adds the following clauses to the formula: $\bigcup_{2 \leq i < p} \{(\overline{b_i} \vee \overline{d_i}, 1) \cup (d_i \leftrightarrow (b_{i+1} \vee \cdots \vee b_p))\}$ along with $(\square, 1)$.

[1] Since $(b_1 \vee \cdots \vee b_p)$ is hard it is preserved, but in subsequent steps both premises will be soft and will be removed by MaxRes.

Example 3 The first column in Table 1 shows an execution of Algorithm 1 on our example formula Ψ. We describe the first iteration of the algorithm. Suppose the first core returned by SAT solver is $\kappa^1 = \{(x_1, 1), (x_2, 1)\}$. Algorithm 2 removes the original soft clauses in the core $(x_1, 1)$, $(x_2, 1)$, adds reification clauses, $(x_1 \leftrightarrow \overline{b_1}), (x_2 \leftrightarrow \overline{b_2})$, new unit soft clauses $(\overline{b_1}, 1), (\overline{b_2}, 1)$ and the hard clause (b_1, b_2) specifying that at least one of the clauses in the core must be falsified. We show the new unit clauses $(\overline{b_1}, 1)$, $(\overline{b_2}, 1)$ in gray as they will be removed by Algorithm 3. Algorithm 3 then constructs a MaxRes refutation by resolving the unit soft clauses $(\overline{b_1}, 1)$ and $(\overline{b_2}, 1)$ against the new hard clause (b_1, b_2). The MaxRes refutation derives a new empty clause with weight 1, $(\square, 1)$ and reduces the cost of the remaining formula by 1. This completes the first iteration and we obtain a new cost equivalent formula by Theorems 1–3. This procedure is repeated three more times until we obtain ϕ^4 (after processing κ^4) which is satisfiable and so the algorithm terminates at this point. The first column of Table 1 shows these subsequent iterations.

We prove correctness of the algorithm by proving that each transformations ϕ satisfies the MaxSat reducibility condition. We start with Algorithm 2.

Theorem 1 *Let ϕ^i be the transformed formula at the i-th step and ϕ^* be the formula returned by Algorithm 2 at line 5. Then ϕ^i is* MaxSat *reducible to ϕ^*.*

Proof: Consider an atomic transformation that Algorithm 2 performs for a soft clause C. It removes C from the formula and introduces hard clauses $C \leftrightarrow \overline{b}$ and a soft unit clause $(\overline{b}, 1)$. Consider an assignment I for variables in ϕ^i. We know that $\mathit{cost}(I(\phi^i \setminus \{C\})) = \mathit{cost}(I(\phi^* \setminus \{(C \leftrightarrow \overline{b}), (\overline{b}, 1)\}))$ as these parts are identical. If $I(C) = 0$ then C is satisfied by I. Therefore, I implies $b = 0$ and the soft clause $(\overline{b}, 1)$ is satisfied. If $I(C) = 1$ then C is falsified by I. Therefore, I implies $b = 1$ and $(\overline{b}, 1)$ is also falsified. ◀

Since at least one of the soft clauses of κ^i must be falsified by any feasible assignment, it follows from Theorem 1 that $\cup_{b \in B}\{(\overline{b}, 1)\}$ is an unsatisfiable core of ϕ^*.

Theorem 2 *Let ϕ^+ be a formula obtained at line 6. Then ϕ^+ is* MaxSat *reducible to ϕ^*.*

Proof: As $\cup_{b \in B}(\overline{b}, 1)$ is a core then in any solution one of the clauses the in core must be set to false. Hence, $(\vee_{b \in B} b)$ is implied by ϕ^*. As this clause is the only difference between two formulas, ϕ^+ is MaxSat reducible to ϕ^*. ◀

Theorem 3 *Let ϕ^+ be a formula at line 6 and ϕ^{i+1} be a formula returned by Algorithm 3 at line 7 Then ϕ^{i+1} is* MaxSat *reducible to ϕ^+.*

Proof: Algorithm 3 performs two atomic operations. First, it applies a special case of the MaxRes rule which is sound. Second, it introduces reification variables for disjunctions of literals. This operation is cost preserving as the new variables are functions of the variables of ϕ^+. ◀

Theorem 4 *Algorithm 1 is sound and complete.*

PMRes	PM1
First core: $\kappa^1 = \{(x_1, 1), (x_2, 1)\}$ ReifyCore $((x_1) \leftrightarrow \overline{b_1}), ((x_2) \leftrightarrow \overline{b_2}), (\overline{b_1}, 1), (\overline{b_2}, 1), (b_1 \vee b_2)$; ~~$(x_1, 1), (x_2, 1)$~~ ApplyMaxRes $(\overline{b_1} \vee \overline{d_2}, 1), (d_2 \leftrightarrow b_2)$; $(\square, 1)$	$\kappa^1 = \{(x_1, 1), (x_2, 1)\}$ Relax ~~$\{(x_1, 1)(x_2, 1)\}$~~ $\{(x_1 \vee b_1^1, 1), (x_2 \vee b_2^1, 1)\}$ $b_1^1 + b_2^1 = 1$
Second core: $\kappa^2 = \{(\overline{b_1} \vee \overline{d_2}, 1), (x_3, 1)\}$ ReifyCore $((x_3) \leftrightarrow \overline{b_3}), ((\overline{b_1} \vee \overline{d_2}) \leftrightarrow \overline{b_6}), (\overline{b_3}, 1), (\overline{b_6}, 1), (b_3 \vee b_6)$; ~~$(\overline{b_1} \vee \overline{d_2}, 1), (x_3, 1)$~~ ApplyMaxRes $(\overline{b_3} \vee \overline{d_6}, 1), (d_6 \leftrightarrow b_6)$; $(\square, 1)$	$\kappa^2 = \{(x_1 \vee b_1^1, 1), (x_2 \vee b_2^1, 1), (x_3, 1)\}$ Relax ~~$\{(x_1 \vee b_1^1, 1), (x_2 \vee b_2^1, 1)$~~, ~~$(x_3, 1)\}$~~ $\{(x_1 \vee b_1^1 \vee b_1^2, 1), (x_2 \vee b_2^1 \vee b_2^2, 1), (x_3 \vee b_3^2, 1)\}$ $b_1^2 + b_2^2 + b_3^2 = 1$
Third core: $\kappa^3 = \{(\overline{b_3} \vee \overline{d_6}, 1), (x_4, 1)\}$ ReifyCore $((x_4) \leftrightarrow \overline{b_4}), ((\overline{b_3} \vee \overline{d_6}) \leftrightarrow \overline{b_7}), (\overline{b_4}, 1), (\overline{b_7}, 1), (b_4, b_7)$; ~~$(\overline{b_3} \vee \overline{d_6}, 1), (x_4, 1)$~~ ApplyMaxRes $(\overline{b_4} \vee \overline{d_7}, 1), (d_7 \leftrightarrow b_7)$; $(\square, 1)$	$\kappa^3 = \{(x_1 \vee b_1^1 \vee b_1^2, 1), (x_2 \vee b_2^1 \vee b_2^2, 1), (x_3 \vee b_3^2, 1), (x_4, 1)\}$ Relax ~~$\{(x_1 \vee b_1^1 \vee b_1^2, 1)$~~, ~~$(x_2 \vee b_2^1 \vee b_2^2, 1)$~~, ~~$(x_3 \vee b_3^2, 1)$~~, ~~$(x_4, 1)\}$~~ $\{(x_1 \vee b_1^1 \vee b_1^2 \vee b_1^3, 1), (x_2 \vee b_2^1 \vee b_2^2 \vee b_2^3, 1), (x_3 \vee b_3^2 \vee b_3^3, 1), (x_4 \vee b_4^4, 1)\}$ $b_1^3 + b_2^3 + b_3^3 + b_4^3 = 1$
Fourth core: $\kappa^4 = \{(\overline{b_4} \vee \overline{d_7}, 1), (x_5, 1)\}$ ReifyCore $((x_5) \leftrightarrow \overline{b_5}), ((\overline{b_4} \vee \overline{d_7}) \leftrightarrow \overline{b_8}), (\overline{b_5}, 1), (\overline{b_8}, 1), (b_5 \vee b_8)$; ~~$(\overline{b_4} \vee \overline{d_7}, 1), (x_5, 1)$~~ ApplyMaxRes $(\overline{b_5} \vee \overline{d_8}, 1), (d_8 \leftrightarrow b_8)$; $(\square, 1)$	$\kappa^4 = \{(x_1 \vee .. \vee b_1^3, 1), (x_2 \vee .. \vee b_2^3, 1), (x_3 \vee b_3^2 \vee b_3^3, 1), (x_4 \vee b_4^3, 1), (x_5, 1)\}$ Relax ~~$\{(x_1 \vee b_1^1 \vee .. \vee b_1^3, 1)$~~, ~~$(x_2 \vee b_2^1 \vee .. \vee b_2^3, 1)$~~, ~~$(x_3 \vee b_3^2 \vee b_3^3, 1)$~~, ~~$(x_4 \vee b_4^3, 1)\}$~~ $\{(x_1 \vee .. \vee b_1^4, 1), (x_2 \vee .. \vee b_2^4, 1), (x_3 \vee .. \vee b_3^4, 1), (x_4 \vee b_4^3 \vee b_4^4, 1), (x_5 \vee b_5^4, 1)\}$ $b_1^4 + b_2^4 + b_3^4 + b_4^4 + b_5^4 = 1$

Table 1: An execution of Algorithm 1 on the running example.

Proof: Suppose the algorithm returns solution I at step $i+1$. As the algorithm terminates at step $i+1$ then $\phi^i \setminus \{\cup_{j=1}^{i}(\square, 1)\}$ is a satisfiable formula. Hence, the satisfying assignment I for $\phi^i \setminus \{\cup_{j=1}^{i}(\square, 1)\}$ is also an assignment of ϕ^i of cost i. From Theorems 1–3 it follows that ϕ^i is MAXSAT reducible to ϕ. Hence, $cost(I(\phi^i)) = cost(I(\phi)) = i$. There does not exist an assignment of a smaller cost as we have derived i empty clauses. ◀

Theorem 5 *The number of soft clauses at the i-th step of Algorithm 1 is $|softCls(\phi)|$, including i empty clauses. The number of hard clauses at the i-th step of Algorithm 1 is $|hardCls(\phi)| + O(\sum_{k=1}^{i} |softCls(\kappa^i)|^2))$.*

Proof: Algorithm 3 does not change the number of soft clause $|softCls(\phi)|$ as it replaces $O(|softCls(\kappa^i)|)$ clauses in a core with $O(|softCls(\kappa^i)|) - 1$ compensation clauses. and one empty clause. Algorithm 3 produces $O(|softCls(\kappa^i)|)$ clauses of the form $(\overline{x} \vee \overline{A}, 1)$. For each of these clauses we introduce a variable d and $O(|softCls(\kappa^i)|)$ clauses to encode the equivalence $A \leftrightarrow d$. Hence, the total number of introduced hard clauses is $O(\sum_{k=1}^{i} |softCls(\kappa^i)|^2))$. ◀

Special cases. We can improve our approach by recognizing some cases were a new reification variable does not need to be introduced. This obviously occurs when we have a unit clause: $((\ell) \leftrightarrow b)$ would simply introduce a new name for ℓ. We apply this optimization at line 3 of Algorithm 2 (avoiding replacing $(C_j, 1)$ by $(\overline{b_j^i}, 1)$) and at line 6 of Algorithm 3 when $i = p-1$ (avoiding creating a new variable d_{p-1} and instead adding $(\overline{b_{p-1}} \vee \overline{b_p})$ to ϕ. We also optimize cores of size two. If $\kappa^i = \{(C_1, 1), (C_2, 1)\}$ it can be seen that our approach would introduce two reification variables $(C_i \leftrightarrow \overline{b_i})$, $i = \{1, 2\}$ and the one compensation clause $(\overline{b_1}, \overline{b_2}, 1)$. However, other than the clauses introduced by reification, this compensation clause is the only one that contains the reification variables. This clause is equivalently to the clause $(C_1, C_2, 1)$, and so for cores of size two we directly introduce it and omit reification. These simple optimizations allows us to solve our running example without introduction any auxiliary variables (see Table 2).

PMRes with optimizations
First core: $\kappa^1 = \{(x_1, 1), (x_2, 1)\}$ Call ReifyCore and ApplyMaxRes with optimizations: $(x_1 \vee x_2, 1), (\square, 1)$
Second core: $\kappa^2 = \{(x_1 \vee x_2, 1), (x_3, 1)\}$ Call ReifyCore and ApplyMaxRes with opts: $(x_1 \vee x_2 \vee x_3, 1), (\square, 1)$
Third core: $\kappa^3 = \{(x_1 \vee x_2 \vee x_3, 1), (x_4, 1)\}$ Call ReifyCore and ApplyMaxRes with opts: $(x_1 \vee x_2 \vee x_3 \vee x_4, 1), (\square, 1)$
Fourth core: $\kappa^4 = \{(x_1 \vee x_2 \vee x_3 \vee x_4, 1), (x_5, 1)\}$ Call ReifyCore and ApplyMaxRes with opts: $(x_1 \vee x_2 \vee x_3 \vee x_4 \vee x_5, 1), (\square, 1)$

Table 2: A run of Alg. 1 with optimizations on Example 3.

A compressed MAXRES algorithm

By Theorem 5, Algorithm 1 introduces a quadratic number of hard clauses at each step. Looking more closely at the sequence of restricted MAXRES rules in Algorithm 3 we can find a further optimization to reduce this to a linear number.

Example 4 Suppose we find the first core $\{(x_1, 1), (x_2, 1), (x_3, 1), (x_4, 1), (x_5, 1)\}$. We introduce reification variables, $b_1, \ldots, b_5$, for these clauses, respectively. We consider an execution of the first two steps of ApplyMaxRes routine on this core:

$$\text{(1)}\quad \frac{(b_1 \vee b_2 \vee b_3 \vee b_4 \vee b_5, \infty) \qquad (\overline{b_1}, 1)}{(b_2 \vee b_3 \vee b_4 \vee b_5, 1), (\overline{b_1} \vee \overline{b_2 \vee b_3 \vee b_4 \vee b_5}, 1)}$$

$$\text{(2)}\quad \frac{(b_2 \vee b_3 \vee b_4 \vee b_5, 1) \qquad (\overline{b_2}, 1)}{(b_3 \vee b_4 \vee b_5, 1), (\overline{b_2} \vee \overline{b_3 \vee b_4 \vee b_5}, 1)}$$

It is easy to see that compensation clauses have many overlapping literals and hence can be compressed.

Theorem 6 *Let $(\overline{b_1} \vee \overline{d_1}), \ldots, (\overline{b_k} \vee \overline{d_k})$ be the sequence of the compensation clauses produced by Algorithm 3. Then, $d_i = (b_{i+1} \vee d_{i+1})$.*

Proof: Suppose, we have a core $(\overline{b_1}, 1) \vee \cdots \vee (\overline{b_k}, 1)$ and an implied clauses $(b_1 \vee \cdots \vee b_k)$. We prove the theorem by induction. Base case, $i = 1, 2$. We start by resolving $\overline{b_1}$ and $(b_1 \vee \cdots \vee b_k, \infty)$. We obtain two clauses: $(b_2 \vee \cdots \vee b_k, 1)$ and the first compensation clause $(\overline{b_1} \vee \overline{d_1}, 1) = (\overline{b_1} \vee \overline{b_2 \vee \cdots \vee b_k}, 1)$, $d_1 \leftrightarrow (b_2 \vee \cdots \vee b_k)$. Then we resolve $\overline{b_2}$ and $(b_2 \vee \cdots \vee b_k, 1)$. We obtain two clauses: $(b_3 \vee \cdots \vee b_k, 1)$ and the second compensation clause $(\overline{b_2} \vee \overline{d_2}, 1) = (\overline{b_2} \vee \overline{b_3 \vee \cdots \vee b_k}, 1)$, $d_2 \leftrightarrow (b_3 \vee \cdots \vee b_k)$. Hence, $d_1 = (b_2 \vee \cdots \vee b_k) = (b_2 \vee (b_3 \vee \cdots \vee b_k)) = (b_2 \vee d_2)$. The induction step is similar to the base step. ◀

Using Theorem 6, we can more compactly encode the compensation clauses by replacing $\{(d_i \leftrightarrow (b_{i+1} \vee \cdots \vee b_p))\}$ on line 6 in Algorithm 3 with $\{(d_i \leftrightarrow (b_{i+1} \vee d_{i+1}))\}$.

Theorem 7 *The number of hard constraints at the i-th step of Algorithm 1 with compression is $|hardCls(\phi)| + O(\sum_{k=1}^{i} |softCls(\kappa^i)|)$.*

Proof: Using compression, we only need a constant number of clauses to encode each compensation clause. ◀

Connection to the PM1 algorithm

In this section, we discuss how our algorithm relates to the original core-based algorithm, PM1, by Fu and Malik (Fu and Malik 2006). For completeness we recap the PM1 algorithm and point out the main conceptual differences between PM1 and PMRes. Algorithm 4 gives the pseudo code for PM1. The main routine finds a core at each step. The subroutine Relax adds a new Boolean variable to each clause in the core and a cardinality constraint $\sum_{b_j^i \in B^i} b_j^i = 1$ allowing one of the clauses in the core to be relaxed.

Algorithm 4 PM1

Input: $\phi = \{C_1, \ldots, C_m\}$
Output: $(I, cost(\phi))$, where I is an optimal assignment

```
1 i = 0
2 while true do
3     (issat, κ^i, I) = SolveSAT(φ^i)
4     if issat return (I, i)
5     (φ^{i+1}) = Relax(φ^i, κ^i)
6     i = i + 1
```

Algorithm 5 Relax

Input: ϕ, κ
Output: ϕ

```
1 B = {}, κ_r = {}
2 for C_j ∈ softCls(κ) do
3     κ_r = κ_r ∪ {(C_j ∨ b_j^i)}
4     B = B ∪ {b_j^i}
5 φ = (φ − softCls(κ)) ∪ κ_r ∪ cnf(Σ_{b_j^i ∈ B} b_j^i = 1)
6 return φ
```

The main difference between our algorithm and PM1 is that we do not use new variables in the clauses and cardinality constraints to relax each core. Instead, we use MAXRES to shift the cost to an empty clause.

Example 5 Table 1 shows an execution of PM1 in the right column.

Theorem 8 *The number of soft constraints at the i-th step of Algorithm 4 is $|softCls(\phi)|$. The number of hard constraints at the i-th step of Algorithm 4 is $|hardCls(\phi)| + O(\sum_{k=1}^{i} |softCls(\kappa^i)|)$.*

Proof: The number of soft constraints does not change during search. The number of hard constraints depends on the CNF encoding of the cardinality constraints which requires a linear number of clauses using a standard sequential counters encoding. ◀

From Theorems 5 and 8, it follows that our new algorithm and PM1 introduce a similar number of new clauses.

In (Ansotegui et al. 2012), the authors point out that the PM1 algorithm introduces symmetries over the blocking variables b_i^j. These symmetries allow multiple ways of relaxing the same set of soft clauses. A set of symmetry breaking constraints was proposed to address this problem. On some benchmarks the number of symmetry breaking clauses is very large and exhausts memory while on others there were very effective. An advantage of our approach is that we do not have these types of symmetries. Further research is need to identify whether our approach also introduces symmetries over the reification variables.

Weighted PMRes

Algorithm 1 can be easily extended for the weighted case in a similar way as PM1 is extended to WPM1 (Ansotegui, Bonet, and Levy 2009; Manquinho, Silva, and Planes 2009). We recap the main idea behind this extension. Consider a core $\kappa = ((C_1, w_1), \ldots, (C_k, w_k))$. Let $m = \min_{(C_i, w_i)}\{w_i\}$. The idea is to split the clauses (C_i, w_i) that have weight greater than m into two clauses: (C_i, m) and $(C_i, w_i - m)$. Then, we keep only the clauses of weight m in the core so that $\kappa = ((C_1, m), \ldots, (C_k, m))$. As all weights are the same this reduces the problem to the unweighted case and we can apply Algorithm 1 to process κ. The only change is that we derive a weighted empty clause $(\square, m)$ rather than $(\square, 1)$.

Related work

The MAXRES rule that we use in the paper was proposed by (Larrosa and Heras 2005), and the authors proved it to be sound. The question of completeness remains open, and in (Bonet, Levy, and Manya 2007) an alternate a sound and complete MAXRES rule was proposed. In the context of iterative SAT solving, the closest work (Heras and Marques-Silva 2011) uses read-once resolution to avoid introducing cardinality constraints in some cases. In this work, the solver relies on the resolution proof of SAT solver to build a MAXRES proof. This works in cases when the resolution proof is read-once. Another line of work uses MAXRES and its variants in branch and bound solvers (Heras, Larrosa, and Oliveras 2008; Kugel 2010; Li et al. 2009). These solvers employ restricted amounts of MAXRES inference to compute lower bounds. Unfortunately, these solvers are not competitive on industrial instances and we do not consider them.

In contrast, we introduce reification variables in a way that allows us (a) to use a more restricted form of MAXRES that generates a tractable number of compensation clauses, and (b) to represent the non-clausal compensation "clauses" much more compactly.

Experimental evaluation

In this section, we preform an experimental evaluation of our proposed algorithm. We run our experiments on a cluster of AMD Opteron@2.2 GHz machines restricted to 2.5Gb or RAM and 1800 sec. of CPU time. We used all industrial instances from the 2013 Evaluation of Max-SAT Solvers competition. We compare our solver with those state-of-the-art weighted partial MAXSAT solvers that performed best on these instances during the latest evaluation (Argelich et al. 2013): wpm_1^{13} (Ansótegui, Bonet, and Levy 2013), wpm_2 (Ansótegui, Bonet, and Levy 2013), wpm_1 (Janota 2013) and $msunc$ (Morgado, Heras, and Marques-Silva 2012). All solvers were provided in the binary or source code by the authors.[2] We replaced the base SAT solver, minisat (Een and Sorensson 2003), in wpm_1 by glucose (Audemard, Lagniez, and Simon 2013) to improve its performance. Note that our hardware is considerably slower than machines used in the competition. Therefore, the evaluated solvers solved less instances when compared to their results in the competition (Argelich et al. 2013).

We implemented our algorithm using the wpm_1 implementation by Mikola Janota as our base. Our solver is called eva. We use glucose as the SAT solver and preprocess hard clauses using standard subsumption technique before we start the search. In the experiments eva refers to Algorithm 1 with optimizations for reducing the number of introduced reification variables described above. eva^c is obtained by adding compression of the compensation clauses to eva. We also implemented the stratification and hardening techniques from (Ansótegui, Bonet, and Levy 2013) in eva for solving weighted partial MAXSAT instance. These improvements are not relevant for the unweighted instances. Tables 3–5 show the number of solved instances and the average time for each family.

First, we observe that eva outperforms all state-of-the-art solvers in the aggregate number of solved instances. It solves 821 instances in total while the solvers wpm_2, $msunc$ and wpm_1^{13} solve 761, 733 and 728 instances, respectively. However, we observe that eva performs poorly on some families. For example, on the *sean-safarpour* family eva solves half as many instances as wpm_1. We looked into the execution of the algorithm and found that large cores containing thousands of clauses must be discovered to solve the problem. Algorithm 3 introduces a number of clauses quadratic in the size of the core, causing a memory problem. Enhancing eva with the idea of compression resolves this issue. eva^c is the best solver across all industrial instances by solving 877 instances in total. It is also faster than eva on many families of problems even if these algorithms solve about the same number of benchmarks, e.g. *su*, *close solutions* or *timetabling* families. Our results demonstrate that the idea of processing each core with MAXRES performs better than adding cardinality constraints in almost all of the benchmarks.

[2]We would like to thank Carlos Ansotegui, Maria Luisa Bonet, Joel Gabas, Jordi Levy, and Joao Marques-Silva, Antonio Morgado, Federico Heras for providing binaries and Mikolas Janota for providing the source code of their solvers.

Conclusions

In this work we made the following main contributions. First, we proposed a novel algorithm for solving weighted partial MAXSAT problems based on the MAXRES rule. This is the first efficient approach based on MAXRES that can compete with modern MAXSAT solvers on industrial benchmarks. Second, we proposed a compressed version of the MAXRES rule that can reduce the number of clauses needed to encode the compensation clauses that MAXRES produces by a factor of n, where n is the number of soft clauses. Finally, we implemented the proposed algorithm and showed its efficiency on the industrial instances from the 2013 Evaluation of Max-SAT Solvers. Our new solver outperforms previous state-of-the-art solvers by a large margin in the total number of problems it can solve and shows the best performance on 16 out of 27 families of benchmarks.

	eva^c		eva		wpm_1		wpm_1^{13}		$msunc$		wpm_2	
	#	avg t	#	avg t	#	avg t	#	avg t	#	avg t	#	avg t
haplotyping	**97**	53.2	**97**	**50.8**	81	59.9	91	115.8	52	336.8	94	139.2
packup-wpms	**95**	**46.7**	90	34.8	46	37.7	89	17.5	27	223.1	42	403.5
pref. planning	**29**	93.6	**29**	102.4	11	5.5	27	56.3	27	79.0	**29**	**34.6**
timetabling	**11**	**313.1**	10	410.6	7	113.6	7	456.6	8	290.9	7	450.1
upgradeability	**100**	54.4	**100**	53.9	**100**	56.6	**100**	**6.4**	**100**	137.2	95	681.3
wcsp.dir	**14**	**2.4**	**14**	2.9	7	167.1	**14**	130.9	11	37.1	13	5.7
wcsp.log	**14**	151.4	**14**	**18.1**	6	0.0	12	155.4	11	26.4	**14**	59.3
Total	**360**	**64.8**	354	58.8	258	55.3	340	62.2	236	179.7	294	339.5

Table 3: WPMS Industrial instances

	eva^c		eva		wpm_1		wpm_1^{13}		$msunc$		wpm_2	
	#	avg t	#	avg t	#	avg t	#	avg t	#	avg t	#	avg t
aes	**1**	**1.0**	**1**	15.0	0	0.0	0	0.0	**1**	573.0	**1**	1694.0
bcp-fir	**50**	**39.9**	45	22.8	46	50.7	49	28.4	49	19.3	49	41.9
simp	**16**	73.4	**16**	**58.9**	11	14.8	15	147.2	15	158.2	14	51.1
su	**31**	**58.8**	**31**	73.8	17	43.5	26	142.5	29	135.4	30	53.1
bcp-msp	14	354.0	13	441.0	3	10.0	5	50.8	**28**	**247.0**	20	296.5
bcp-mtg	**40**	**0.8**	**40**	1.1	38	132.3	**40**	9.3	**40**	1.6	29	150.8
bcp-syn	**29**	**57.6**	28	34.6	16	27.8	26	75.7	27	108.0	23	226.1
circuit-trace	**4**	222.0	**4**	331.5	1	1157.0	**4**	284.8	**4**	235.5	**4**	**171.5**
close solutions	**44**	**123.3**	39	153.2	31	90.3	22	160.0	24	315.1	10	413.1
des	**36**	**346.1**	16	470.5	27	436.6	19	175.6	27	446.0	21	188.2
haplotype	**5**	**5.4**	**5**	5.6	**5**	5.6	**5**	**5.4**	4	97.3	**5**	107.2
packup-pms	**40**	53.8	**40**	53.5	37	58.8	**40**	**10.4**	37	460.9	31	519.2
nencdr	35	149.3	35	172.7	21	610.0	30	302.6	47	415.8	**48**	**406.5**
nlogencdr	43	226.5	40	157.7	25	160.5	30	100.7	**50**	**156.6**	**50**	284.8
pbo-routing	**15**	**0.1**	**15**	0.1	**15**	0.4	**15**	9.1	**15**	1.3	**15**	10.9
protein ins	4	405.3	4	551.0	1	2.0	2	172.5	3	551.0	**11**	**563.9**
Multiple path	32	177.7	31	155.0	0	0.0	15	39.6	27	191.7	**47**	**272.1**
One path	49	84.3	48	79.8	0	0.0	27	80.9	**50**	90.7	**50**	**15.9**
Total	**488**	**120.9**	451	113.5	294	148.0	370	91.1	477	198.0	458	219.8

Table 4: PMS Industrial instances

	eva^c		eva		wpm_1		wpm_1^{13}		$msunc$		wpm_2	
	#	avg t	#	avg t	#	avg t	#	avg t	#	avg t	#	avg t
circuit-deb.	**3**	**14.3**	2	17.0	**3**	**14.3**	2	677.0	2	36.0	1	1440.0
sean-saf.	**36**	**285.2**	14	75.0	35	313.5	16	215.4	18	116.9	8	349.4
Total	**39**	**264.4**	16	67.8	38	289.9	18	266.7	20	108.8	9	470.6

Table 5: MaxSat Industrial instances

References

Ansotegui, C.; Bonet, M. L.; Gabas, J.; and Levy, J. 2012. Improving SAT-Based Weighted MaxSAT Solvers. In Milano, M., ed., *Proceedings of the 18th International Conference on Principles and Practice of Constraint Programming (CP'12)*, volume 7514 of *Lecture Notes in Computer Science*, 86–101. Springer.

Ansotegui, C.; Bonet, M. L.; and Levy, J. 2009. Solving (Weighted) Partial MaxSAT through Satisfiability Testing. In Kullmann, O., ed., *Proceedings of the 12th International Conference on Theory and Applications of Satisfiability Testing (SAT'09)*, volume 5584 of *Lecture Notes in Computer Science*, 427–440. Springer.

Ansótegui, C.; Bonet, M. L.; and Levy, J. 2013. SAT-based MaxSAT algorithms. *Artificial Intelligence* 196:77–105.

Argelich, J.; Li, C. M.; Manya, F.; and Planes, J. 2013. Maxsat 2013, Eighth Max-SAT evaluation. http://maxsat.ia.udl.cat:81/13/results/index.html.

Audemard, G.; Lagniez, J.-M.; and Simon, L. 2013. Improving glucose for incremental SAT solving with assumptions: application to MUS extraction. In Jarvisalo, M., and Gelder, A. V., eds., *Proceedings of the 16th International Conference on Theory and Applications of Satisfiability Testing (SAT'13)*, volume 7962 of *Lecture Notes in Computer Science*, 309–317. Springer.

Bonet, M. L.; Levy, J.; and Manya, F. 2007. Resolution for Max-SAT. *Artificial Intelligence* 171(8-9):606–618.

Cooper, M. C.; Cussat-Blanc, S.; de Roquemaurel, M.; and Regnier, P. 2006. Soft Arc Consistency Applied to Optimal Planning. In Benhamou, F., ed., *Proceedings of the 12th International Conference on Principles and Practice of Constraint Programming (CP'06)*, volume 4204 of *Lecture Notes in Computer Science*, 680–684. Springer.

Davies, J., and Bacchus, F. 2013. Postponing optimization to speed up MaxSAT solving. In *Proceedings of the 19th International Conference on Principles and Practice of Constraint Programming (CP'13)*, volume 8124 of *Lecture Notes in Computer Science*, 247–262.

Een, N., and Sorensson, N. 2003. An extensible SAT-solver. In Giunchiglia, E., and Tacchella, A., eds., *Proceedings of the 6th International Conference on Theory and Applications of Satisfiability Testing (SAT'03)*, volume 2919 of *Lecture Notes in Computer Science*, 502–518. Springer.

Fu, Z., and Malik, S. 2006. On solving the partial MAX-SAT problem. In *Proceedings of the 9th International Conference on Theory and Applications of Satisfiability Testing (SAT'06)*, volume 4121 of *Lecture Notes in Computer Science*, 252–265. Springer.

Graça, A.; Lynce, I.; Marques-Silva, J. a.; and Oliveira, A. L. 2012. Efficient and accurate haplotype inference by combining parsimony and pedigree information. In *Proceedings of the 4th International Conference on Algebraic and Numeric Biology*, 38–56. Berlin, Heidelberg: Springer-Verlag.

Heras, F., and Marques-Silva, J. 2011. Read-once resolution for unsatisfiability-based Max-SAT algorithms. In Walsh, T., ed., *Proceedings of the 22th International Joint Conference on Artificial Intelligence (IJCAI'11)*, 572–577. IJCAI/AAAI.

Heras, F.; Larrosa, J.; and Oliveras, A. 2008. MiniMaxSAT: An efficient weighted Max-SAT solver. *J. Artif. Intell. Res. (JAIR)* 31:1–32.

Heras, F.; Morgado, A.; and Marques-Silva, J. 2011. Core-guided binary search algorithms for maximum satisfiability. In Burgard, W., and Roth, D., eds., *Proceedings of the 25th Conference on Artificial intelligence (AAAI'11)*. AAAI Press.

Janota, M. 2013. MiFuMaX, an unsat-based Max-SAT solver. http://sat.inesc-id.pt/~mikolas/sw/mifumax/.

Kugel, A. 2010. Improved exact solver for the weighted MAX-SAT problem. In Berre, D. L., ed., *POS@SAT*, volume 8 of *EPiC Series*, 15–27. EasyChair.

Larrosa, J., and Heras, F. 2005. Resolution in Max-SAT and its relation to local consistency in weighted CSPs. In Kaelbling, L. P., and Saffiotti, A., eds., *Proceedings of the 19th International Joint Conference on Artificial Intelligence (IJCAI'05)*, 193–198.

Li, C. M.; Manya, F.; Mohamedou, N. O.; and Planes, J. 2009. Exploiting cycle structures in Max-SAT. In Kullmann, O., ed., *Proceedings of the 12th International Conference on Theory and Applications of Satisfiability Testing (SAT'09)*, volume 5584 of *Lecture Notes in Computer Science*, 467–480. Springer.

Manquinho, V. M.; Silva, J. P. M.; and Planes, J. 2009. Algorithms for weighted Boolean optimization. In Kullmann, O., ed., *Proceedings of the 12th International Conference on Theory and Applications of Satisfiability Testing (SAT'09)*, volume 5584 of *Lecture Notes in Computer Science*, 495–508. Springer.

Morgado, A.; Heras, F.; Liffiton, M. H.; Planes, J.; and Marques-Silva, J. 2013. Iterative and core-guided MaxSAT solving: A survey and assessment. *Constraints* 18(4):478–534.

Morgado, A.; Heras, F.; and Marques-Silva, J. 2012. Improvements to core-guided binary search for MaxSAT. In Cimatti, A., and Sebastiani, R., eds., *Proceedings of the 15th International Conference on Theory and Applications of Satisfiability Testing (SAT'12)*, volume 7317 of *Lecture Notes in Computer Science*, 284–297. Springer.

Strickland, D. M.; Barnes, E.; and Sokol, J. S. 2005. Optimal protein structure alignment using maximum cliques. *Oper. Res.* 53(3):389–402.

Vasquez, M., and Hao, J.-K. 2001. A "logic-constrained" knapsack formulation and a tabu algorithm for the daily photograph scheduling of an earth observation satellite. *Comp. Opt. and Appl.* 20(2):137–157.

Zhang, L., and Bacchus, F. 2012. MAXSAT heuristics for cost optimal planning. In *Proceedings of the 26th Conference on Artificial intelligence (AAAI'12)*, 1846–1852.