Maximum Satisfiability Using Cores and Correction Sets

Nikolaj Bjorner
Microsoft Research
Redmond, USA
nbjorner@microsoft.com

Nina Narodytska
Carnegie Mellon University
Pittsburgh, USA
ninan@cs.cmu.edu

Abstract

Core-guided MAXSAT algorithms dominate other methods in solving industrial MAXSAT problems. In this work, we propose a new efficient algorithm that is guided by *correction sets* and *cores*. At every iteration, the algorithm obtains a correction set or a core, which is then used to rewrite the formula using incremental and succinct transformations. We theoretically show that correction sets and cores have complementary strengths and empirically demonstrate that their combination leads to an efficient MAXSAT solver that outperforms state-of-the-art WPMS solvers on the 2014 Evaluation on industrial instances.

Introduction

Maximum Satisfiability (MAXSAT) is an optimisation version of satisfiability (SAT) that contains both hard clauses and weighted soft clauses. A solution of MAXSAT is an assignment that satisfies all hard clauses and violates a minimum weight subset of soft clauses. Many industrial applications can be naturally encoded in MAXSAT. Successful application areas include bioinformatics [Strickland *et al.*, 2005; Graça *et al.*, 2012], planning and scheduling [Cooper *et al.*, 2006; Vasquez and Hao, 2001; Juma *et al.*, 2012].

There exist a number of different approaches to solving MAXSAT. The most successful approach for industrial problems performs iterative solving using a SAT solver as an oracle on each iteration [Morgado *et al.*, 2013; Ansótegui *et al.*, 2013]. Existing solvers differ in how they perform optimal cost estimation, e.g. linear or binary search, and the kind of information they use to estimate the cost, e.g. cores, the structure of cores, or satisfying assignments. While there are many algorithms that perform search on either upper bound or lower bound of the optimal cost [Ansótegui *et al.*, 2013; Morgado *et al.*, 2013; Narodytska and Bacchus, 2014; Morgado *et al.*, 2014], there are very few algorithms that efficiently perform combined search over two bounds, e.g. [Heras *et al.*, 2011; Ignatiev *et al.*, 2014]. The current state-of-the-art approach for combined search performs binary-like search over the range of possible cost values, which can be exponentially more efficient than linear search. However, one of the drawbacks of binary search-based algorithms is that they modify the formula using expensive PseudoBoolean (PB) constraints, while, for example, the lower bound-based linear search does not introduce PB constraints. PB constraints increase the size of the formula and, potentially, slow down the solver.

We present an efficient and robust algorithm that takes advantage of lower and upper bounds estimates. Our approach is in the spirit of linear search [Marques-Silva and Planes, 2008], as we create a sequence of SAT problems. A key strength of our algorithm is that it uses succinct formula transformations that do not use PB constraints and can be applied incrementally. Our algorithm is based on a fundamental duality between cores and correction sets. In each incremental step it arbitrates between cores and correction sets making low overhead progress towards optimality. Experimental results demonstrate the potential of this approach: our PRIMALDUALMAXSAT solver, shows robust performance and *outperforms* the best WPMS algorithms on industrial instances by a noticable margin.

Background

A satisfiability problem consists of a set of clauses $\phi = \{C_1, \ldots, C_m\}$ over a set of Boolean variables $\mathcal{X}$. 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 $(l_1 \vee \cdots \vee l_n)$. A unit clause is a clause containing only one literal. We denote the set of variables in the formula ϕ by vars(ϕ). An assignment I of the variables vars(ϕ) is a mapping vars$(\phi) \mapsto \{0, 1\}$, extended to literals by $I(\overline{x}) = 1 - I(x)$, clauses $I(C) = \max\{I(l) \mid l \in C\}$, and sets of clauses ϕ, by $I(\phi) = \min\{I(C) \mid C \in \phi\}$. If $I(\phi) = 1$, then I is a solution to the satisfiability problem ϕ.

A weighted clause is a pair (C, w), where C is a clause and w is the weight of C, $w \in \mathbb{N} \cup \{\infty\}$, that gives the cost of violating C. Clauses with $w = \infty$ are called *hard* clauses; otherwise, the clause is a *soft* clause. A Weighted Partial MAXSAT (WPMS) Problem consists of a set of a set of weighted clauses, $\phi = \{(C_1, w_1), \ldots, (C_m, w_m)\}$. A special case of WPM is Partial MAXSAT (PMS), 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). W.l.o.g., we can assume that soft clauses (C, w) in ϕ are unit clauses (b, w) as ϕ can be converted to this form by introducing reification variables: $\phi = [\phi \setminus \{(C, w)\}] \cup \{(C \leftrightarrow b), (b, w)\}$. We also assume standard clausification, e.g., $C \leftrightarrow b$ denotes $\{(\overline{b} \vee C)\} \cup \bigcup_{l \in C}\{(\overline{l} \vee b)\}$.

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: $cost(I, \phi) = \sum_{(b,w)\in\phi \wedge I(b)=0} w$. I is an **optimal assignment** for ϕ iff it is a feasible assignment of minimal cost. We denote this minimal cost by $cost(\phi) = \min_{I \text{ is feasible}} cost(I, \phi)$. The maximal finite cost is $maxcost(\phi) = \sum_{(b,w)\in\phi \wedge w<\infty} w$.

To simplify notation, we omit weights for hard clauses and we assume they are satisfiable. A $\varphi \subseteq \phi$ is a *core* of ϕ iff φ is unsatisfiable. A core φ is *minimal* iff $\forall (b, w) \in \varphi$, $\varphi \setminus \{(b, w)\}$ is a satisfiable formula. A $\pi \subseteq \phi$ is a *correction set* iff $\phi \setminus \pi$ is satisfiable. A correction set π is *minimal* iff $\forall (b, w) \in \pi$ $\phi \setminus (\pi \setminus \{(b, w)\})$ is unsatisfiable.The cost of a correction set π is $maxcost(\pi)$. A smallest minimal correction set π is a *minimum* correction set. We denote $\mathcal{I}(\phi)$ the set of solutions of ϕ. A solution I corresponds to a correction set π iff $I(b_i) = 0$ for $(b_i) \in \pi$ and $I(b_i) = 1$ otherwise.

Related work

There are two main classes of SAT-based algorithms for solving MAXSAT [Morgado *et al.*, 2013]. Algorithms in the first class perform linear search on the value of the cost. It can be partitioned into two subclasses. The first subclass of solvers perform so called UNSAT-SAT search (LBLINEAR), e.g. [Ansótegui *et al.*, 2013; Manquinho *et al.*, 2009; Narodytska and Bacchus, 2014; Morgado *et al.*, 2014]. They start with an unsatisfiable formula and gradually increase the lower bound on the optimal cost by performing a sequence of relaxations of the formula. The second subclass of solvers perform so called SAT-UNSAT search (UBLINEAR). They start with a satisfiable formula and perform a sequence of strengthenings on it [Parrain, 2010; Martins *et al.*, 2014]. The second class combines upper and lower bound searches by performing binary or linear search on the cost value [Cimatti *et al.*, 2010; Heras *et al.*, 2011; Morgado *et al.*, 2012; Marques-Silva and Planes, 2008; Ignatiev *et al.*, 2014].

The work [Marques-Silva and Planes, 2008] is the most relevant to us as it performs lower and upper bound search. The main distinction of our approach is in how we transform the formula based on using a core or a correction set. In particular, we do not use PB constraints in our transformations.

PRIMALDUALMAXSAT algorithm

We first present our algorithm for unweighted partial MAXSAT, so that all soft clauses have weight one. The weighted case, solved using cloning, is discussed later.

The algorithm searches for cores and correction sets of the formula simultaneously by performing a sequence of SAT calls. At each step the algorithm detects either a core or a correction set. However, we do not know in advance which of two will be found and let our search procedure decide on this. The algorithm terminates in at most m steps, which is the same worst case guarantee as linear search algorithms. The algorithm uses two phases: a *partition* and a *transformation* phase. Algorithm 1 shows the pseudocode. Core minimization (line 23) is optional. It does not affect correctness. Let us detail the two phases.

Partition. The algorithm partitions all soft clauses into two sets φ^i and π^i at the ith round. The first set of soft clauses must be satisfied along with the hard clauses and the second set of clauses can be violated at this round. We call φ^i a *candidate* core and π^i a *candidate* correction set. Note that by splitting soft clauses, we guarantee that either φ^i is a valid core or π^i a valid correction set. The main question here is how to perform partitioning of the clauses. Our idea is inspired by relaxation search [Rosa and Giunchiglia, 2013; Bacchus *et al.*, 2014]. Relaxation search branches on variables from soft clauses positively. Using unit propagation, these decisions may force other variables to be falsified. After all variables from soft clauses are instantiated (Algorithm 1, lines 7–10), we have a candidate core $\varphi^i = \{b \mid I(b) = 1\}$ and a candidate correction set $\pi^i = \{b \mid I(b) = 0\}$. We call the SAT solver in the context of $hardCls(\phi^i)$, and partial assignment I in line 13 to determine satisfiability of $hardCls(\phi^i)\,|_I$. If $hardCls(\phi^i)\,|_I$ is UNSAT then it returns a subset of φ^i as a valid core; otherwise π^i is a valid correction set.

The partition phase has several useful properties. First, if the solver is satisfiable using a candidate set π, then it is minimal (Proposition 2 in [Bacchus *et al.*, 2014]). So there is no need to reduce it. Second, suppose $I(b) = 0$ is implied during the partition phase. Then $\{b\} \cup \{b' \mid I(b') = 1\}$ is a core of ϕ. In this case, if checking whether π is a correction set is computationally expensive, we can apply the primal transformation to this core. Third, suppose I is a partial assignment such that $|\{b \mid I(b) = 0\}| > ub$. Then I is not going to be extended to a minimal solution. In this case, we can block this assignment and resume the search. We omit these extensions in our pseudocode.

Transformation. The transformation phase takes as input a core or a correction set. If $hardCls(\phi^i)\,|_I$ is satisfiable and π^i is empty then we can satisfy all soft clauses in ϕ^i and the algorithm terminates (line 16). A dual case is when $hardCls(\phi^i)\,|_I$ is unsatisfiable and φ^i is empty (line 18). In this case, $hardCls(\phi^i)$ is unsatisfiable and the algorithm terminates. If neither of these cases hold, we perform transformations. To ensure overall correctness we identify three main properties of the transformations of Algorithm 1. The properties ensure that costs are computed correctly, the algorithm terminates and that the overall space overhead is linear in each round. Each transformation takes ϕ and φ/π as input. It introduces new variables, adds new clauses and deletes some soft clauses in ϕ. Let $\mathcal{SI}(\pi)$ be the set of solutions that assign all clauses in π to false.

Property 1 (TransformCore) *Let φ be a core of ϕ and suppose $\phi' = TransformCore(\phi, \varphi)$. Then we require:*

Solutions *There is a bijection Δ between $\mathcal{I}(\phi)$ and $\mathcal{I}(\phi')$ such that for $I \in \mathcal{I}(\phi)$ and $I' = \Delta(I)$: $cost(I) = cost(I') + 1$.*

Cost Update $cost(\phi') = cost(\phi) - 1$,

Termination $maxcost(\phi') < maxcost(\phi)$.

Property 2 (TransformCSet) *Let π be a correction set of ϕ and suppose $\phi' = TransformCSet(\phi, \pi)$. Then we require:*

Solutions *There is a bijection Γ between $\mathcal{I}(\phi')$ and $\mathcal{I}(\phi) \setminus \mathcal{SI}(\pi)$ such that for $I \in \mathcal{I}(\phi) \setminus \mathcal{SI}(\pi)$ and $I' = \Gamma(I)$: $cost(I) = cost(I')$.*

Cost Update $cost(\phi) = \min\{cost(\phi'), maxcost(\pi)\}$,

Termination $maxcost(\phi') < maxcost(\phi)$.

Algorithm 1 PRIMALDUALMAXSAT

Input: $\phi = \{(b_1, 1), \ldots, (b_m, 1), C_{m+1}, \ldots, C_n\}$
Output: $(I^*, cost(\phi))$

```
1  i = 0, φ^0 = φ, ub = m, lb = 0
2  I* = {b_1 = 0, .., b_m = 0} {The current best solution}
3  solver.add(hardCls(φ^0)) {Add hard clauses to a SAT solver}
4  while lb < ub do
5     Undo all branches, reset I
6     {The 'partition' phase}
7     for (b, 1) ∈ φ^i do
8        if I(b) is undefined then
9           solver.branch(b = 1) { updates I(b) = 1}
10          solver.propagate() { updates I with implied assignments }
11    φ^i = {(b, 1) ∈ φ^i | I(b) = 1} {A candidate core}
12    π^i = {(b, 1) ∈ φ^i | I(b) = 0} {A candidate min correction set}
13    (issat, φ^i) = solver.solve(I)
14    {The 'transformation' phase}
15    if (issat ∧ π^i = ∅) then
16       return (I, lb)
17    if (¬issat ∧ φ^i = ∅) then
18       return (I*, ub)
19    if (issat ∧ π^i ≠ ∅) then
20       φ^{i+1} = TransformCSet(φ^i, π^i)
21       if maxcost(π^i) < ub − lb then I* = I, ub = maxcost(π^i) + lb
22    if (¬issat ∧ φ^i ≠ ∅) then
23       [minimize(φ^i)] {Optional core minimisation}
24       φ^{i+1} = TransformCore(φ^i, φ^i)
25       lb = lb + 1
26    i = i + 1
27    solver.add(hardCls(φ^{i+1}) \ hardCls(φ^i)) {Add new hard clauses}
28 return (I*, ub)
```

Next we prove correctness of Algorithm 1 assuming that the transformation procedure satisfies Properties 1–2.

Theorem 1 *Consider TransformCore and TransformCSet that satisfy Properties 1–2. Then Algorithm 1 is correct and terminates in at most m steps, where $m = |softCls(\phi)|$.*

Proof: First, we prove that the algorithm terminates. Note that at each step we have $maxcost(\phi^{i+1}) < maxcost(\phi^i)$. Hence, as $maxcost(\phi^0) = m$, the algorithm terminates in at most m steps. Second, we sketch a proof of correctness. The algorithm implicitly partitions solutions of ϕ into two sets: preserved and eliminated solutions. At the ith step, a preserved solution I of ϕ is a solution that is mapped to a solution I' in ϕ^i by a sequence of transformations. Other solutions are eliminated. We will show that the algorithm keeps track of an optimum solution in each partition.

Suppose φ^i is a core. By Property 1, TransformCore guarantees that for $I \in \mathcal{I}(\phi^i)$ $cost(I) = cost(\Delta(I)) + 1$. Suppose π^i is a correction set. By Property 2, we know that TransformCSet *eliminates* solutions, $\mathcal{SI}(\pi^i)$, that assign all clauses in π^i to false. Moreover, it guarantees that other solutions are preserved: for $I \in \mathcal{I}(\phi^i) \setminus \mathcal{SI}(\pi^i)$ $cost(I) = cost(\Gamma(I))$. By induction, we can prove that at the ith step there is a bijection Θ_i between solutions $\mathcal{I}(\phi^i)$ and a subset of solutions of the original formula $\mathcal{I}(\phi) \setminus \mathcal{RI}_i$, where $\mathcal{RI}_i = \{\cup_{\pi^j \text{is a correction set}, j<i} \Theta_j^{-1}(\mathcal{SI}(\pi^j))\}$, such that (a) for $I \in \mathcal{I}(\phi) \setminus \mathcal{RI}_i$ and $I' = \Theta_i(I)$ we have $cost(I) = cost(I') + lb$, where lb is the number of cores the algorithm found before round i, and (b) $\forall I \in \mathcal{RI}_i$, $cost(I) \geq ub \geq cost(\phi)$.

Suppose the algorithm terminates at round i. We assume that ϕ^i is satisfiable and I is a corresponding solution, so that $cost(I) = 0$ (line 16). By the reasoning above, there exists a preimage $I^* = \Theta_i^{-1}(I)$ such that $cost(I^*) = cost(I) + lb = lb$. Moreover, for all $I \in \mathcal{RI}_i$, $cost(I) \geq ub \geq cost(\phi)$, and $lb < ub$. Hence, the algorithm returns the correct result.

Suppose, ϕ^i is unsatisfiable (line 18). In this case, $S' = \mathcal{I}(\phi^i) = \varnothing$. Hence, the optimal solution was eliminated in the previous steps and $\mathcal{RI}_i$ contains it. Let π^j be the minimum size correction set that the algorithm encountered and I^j be the corresponding solutions. Let $I^* = \Theta_j^{-1}(I^j)$. We know that $cost(I^*) = cost(I^j) + lb$, where lb is the lower bound value at the jth step. We set $ub = cost(I^j) + lb$ in line 21. As $|\pi^j|$ is minimum, $\forall I' \in \mathcal{RI}_i$, $cost(I') \geq cost(I^*)$. Hence, the algorithm returns the correct value ub and the corresponding solution. ◀

Note that our transformations change the set of soft clauses so the variables used in the partition phase changes. Next we present our transformations and discuss their properties.

Correction Set Transformation

In this section we describe a transformation for a correction set. Note that a standard way to transform a formula given the correction set π is to add a PseudoBoolean constraint $[\sum_{(b_j, w_j) \in softCls(\phi)} w_j \overline{b_j} < maxcost(\pi)]$. However, this constraint can be very large. For example, the number of soft clauses in the majority of industrial MAXSAT benchmarks is more than one million. Hence, even if the size of the correction set is small, we will have to add a constraint over one million variables, which is impractical. In the weighted case, such transformations are also very costly. It has to be noted that if the number of soft clauses is not very large and there are no weights, the problem of expensive cardinality constraints was mitigated in [Martins *et al.*, 2014] by smart reuse of cardinality constraints in an incremental manner.

In this section, we propose an alternative approach that avoids adding expensive cardinality/PseudoBoolean constraints. Our idea is that on discovering a correction set π we transform ϕ by adding clauses only over variables in clauses of π and some fresh variables. Unfortunately, it does not ensure that we can improve the upper bound at each step when a correction set is found. Nevertheless, our transformation procedure satisfies Property 2, which is sufficient to show that PRIMALDUALMAXSAT terminates in at most m rounds, where m is the number of soft clauses in ϕ.

Definition 1 (TransformCSet) *Consider a* MAXSAT *formula ϕ with an optimum cost w. Let $\pi = \{(b_1, 1), \ldots, (b_t, 1)\}$ be a correction set of ϕ. The transformation from ϕ to ϕ', $\phi' = TransformCSet(\phi, \pi)$, is defined as follows:*

$$\phi' = (\phi \setminus \pi) \cup \{(\vee_{i=1,\ldots,t} b_i)\} \cup_{j=2}^{t} \{(b'_j, 1)\} \cup \cup_{j=2}^{t} \{b'_j \leftrightarrow (b_j \wedge (b_1 \vee \cdots \vee b_{j-1}))\}, \quad (1)$$

where b'_j, $j = 2, \ldots, t$ are new variables.

The intuition behind TransformCSet is that we eliminate all correction sets π', $\pi \subseteq \pi'$, from ϕ. Algorithm 2 shows pseudocode that implements this transformation. It tracks common sub-expressions [Narodytska and Bacchus, 2014] to ensure that the transformation has linear overhead.

Algorithm 2 TransformCSet

Input: $\phi, \pi = \{(b_1, 1) \ldots, (b_t, 1)\}$
Output: ϕ
1 $d_1 = false$
2 $\phi = (\phi \setminus \pi) \cup \{(\vee_{j=1}^{t} b_j)\}$
3 **for** $j = 2, \ldots, t$ **do**
4 $\quad \phi = \phi \cup \{d_j \leftrightarrow (b_{j-1} \vee d_{j-1})\}$, where d_j is a new variable
5 $\quad \phi = \phi \cup \{b'_j \leftrightarrow (b_j \wedge d_j)\}$, where b'_j is a new variable
6 $\quad \phi = \phi \cup \{(b'_j, 1)\}$
7 **return** ϕ

To establish that Algorithm 2 satisfies Property 2 we show:

Lemma 1 *Consider ϕ' = TransformCSet(ϕ, π), where $\pi = \{(b_1, 1), \ldots, (b_t, 1)\}$. There is a cost-preserving bijection between solutions of $\phi \wedge (\vee_{i=1,\ldots,t} b_i)$ and solutions of ϕ'.*

Proof: Consider a solution I of $\phi \wedge (\vee_{i=1,\ldots,t} b_i)$. We perform a functional extension of I to variables b'_j, $j = 2, \ldots, t$ as follows: $I(b'_j) = I(b_j) \wedge (I(b_1) \vee \cdots \vee I(b_{j-1}))$. Now I is a well-defined complete assignment for both ϕ and ϕ'. We show that I has the same cost in ϕ and ϕ'.

For $j > t$ the variables b_j are unchanged. Suppose $j \leq t$. Consider two cases. We need to keep in mind that satisfaction of a soft clause $(b'_j, 1)$ in ϕ' functionally depends on assignment of b_j as $b'_j \leftrightarrow b_j \wedge (b_1 \vee \cdots \vee b_{j-1})$.

Suppose $I(b_1) = 1$**.** We show that for $j = 2, \ldots, t$ under the assignment I the following holds: $I(b_j) = I(b'_j)$.

Consider the jth pair of clauses $(b_j, 1)$ and $(b'_j, 1)$. If $I(b_j) = 0$ then both clauses are violated. If $I(b_j) = 1$ then both clauses are satisfied as $I(b_j) = 1$ and $I(b_1) = 1$. The last clause to consider is $(b_1, 1)$. It is also satisfied as $I(b_1) = 1$. Therefore, we proved that I has the same cost in ϕ and ϕ'.

Suppose $I(b_1) = 0$**.** There exists a least $p \in [2, t]$ such that $I(b_p) = 1$ as I satisfies the formula $(\vee_{i=1,\ldots,t} b_i)$ by the assumption. We will prove that for $j = [2, \ldots, t] \setminus \{p\}$ under the assignment I we have $I(b_j) = I(b'_j)$.

Consider the jth pair of clauses $(b_j, 1)$ and $(b'_j, 1)$.

- *Case $j < p$.* As $I(b_j) = 0$, both soft clauses are violated.
- *Case $p < j \leq t$.* Note that as $I(b_p) = 1$, $(I(b_1) \vee \ldots \vee I(b_p) \vee \ldots \vee I(b_{j-1}))$ is true. Hence, $I(b'_j) = I(b_j)$.

Three clauses are left to consider: $\{(b_1, 1), (b_p, 1)\} \in \phi$ and $(b'_p, 1) \in \phi'$. As $I(b_1) = 0$ and $I(b_p) = 1$, we have $cost(I, (b_1, 1)) = 1$, $cost(I, (b_p, 1)) = 0$ and $cost(I, (b'_p, 1)) = 1$. Therefore, we proved that I has the same cost in ϕ and ϕ'.

Consider the reverse direction. Note that ϕ' and $\phi \wedge (\vee_{i=1,\ldots,t} b_i)$ have the same set of hard constraints. Hence, they have the same set of solutions. The cost preserving argument is analogous to the forward case.

◀

Note that we never used the fact that π is a correction set in the proof of Lemma 1. Therefore, the following result holds.

Corollary 1 *Consider ϕ' = TransformCSet(ϕ, δ), where $\delta \subseteq$ softCls(ϕ). Lemma 1 holds for ϕ, δ and ϕ'.*

Corollary 2 *Consider ϕ' = TransformCSet(ϕ, π), where $\pi = \{(b_1, 1), \ldots, (b_t, 1)\}$. Then (1) $|$softCls$(\phi)|$ > $|$softCls$(\phi')|$, $cost(\phi') \geq cost(\phi)$ and (2) $cost(\phi') = cost(\phi \wedge (\vee_{i=1,\ldots,t} b_i))$.*

Theorem 2 *Consider ϕ' = TransformCSet(ϕ, π), where $\pi = \{(b_1, 1), \ldots, (b_t, 1)\}$. Then $cost(\phi) = \min\{cost(\phi'), maxcost(\pi)\}$.*

Proof: Suppose π is a minimum correction set of ϕ. By Corollary 2, $cost(\phi') \geq cost(\phi)$, so we get $cost(\phi) = \min\{cost(\phi'), maxcost(\pi)\}$. Suppose π is not a minimum correction set of ϕ. Consider a minimum correction π^* of ϕ. As $\pi \notin \pi^*$, π^* is a minimum correction set of $\phi \wedge (\vee_{i=1}^{t} b_i)$. By Corollary 2, $cost(\phi') = cost(\phi \wedge (\vee_{i=1}^{t} b_i))$. As $maxcost(\pi) > maxcost(\pi^*)$ we proved the result. ◀.

Hence, from Lemma 1, Theorem 2, Corollaries 1–2 it follows that TransformCSet satisfies Property 2.

An extension of Algorithm 2 to the weighted case can be obtained using a standard cloning technique [Morgado *et al.*, 2013; Ansótegui *et al.*, 2013]. However, there is one complication to take into account. Consider a correction set $\pi = \{(b_1, w_1), \ldots, (b_t, w_t)\}$, $w_1 \leq w_i$, $i = 2, \ldots, t$. We split a clause (b_i, w_i) into two clones: (b_i, w_1) and $(b_i, w_i - w_1)$. Note that $\{(b_1, w_1), \ldots, (b_t, w_1)\}$ might not be a correction set after splitting. We perform ϕ' = TransformCSet$(\phi, \{(b_1, w_1), \ldots, (b_t, w_1)\})$. Corollary 1 tells us that ϕ' contains all solutions of ϕ that violate $(\vee_{i=1,\ldots,t} b_i)$. For Theorem 2 to hold, we need to prove that all correction sets that violate $(\vee_{i=1,\ldots,t} b_i)$ are supersets of π. This condition holds as any feasible solution I of ϕ that violates $(\vee_{i=1,\ldots,t} b_i)$ must falsify all clauses in $\{(b_1, w_1), \ldots, (b_t, w_1)\}$ and their clones. Such a solution corresponds to a correction set that is a superset of π.

Core Transformation

Our core transformer is dual to the correction set transformer. It is based on a recently proposed MaxRes-based algorithm [Narodytska and Bacchus, 2014]. Following MaxRes, given a core φ we convert ϕ into a formula ϕ' using TransformCore, such that $cost(\phi') = cost(\phi) - 1$. We here only provide the definition of TransformCore.

Definition 2 (TransformCore) *Consider a* MaxSat *formula ϕ with an optimum cost w. Let $\varphi = \{(b_1, 1), \ldots, (b_t, 1)\}$ be a core of ϕ. The transformation from ϕ to ϕ', ϕ' = TransformCore(ϕ, φ), is defined as follows:*

$$\phi' = (\phi \setminus \varphi) \cup (\vee_{i=1,\ldots,t} \overline{b_i}) \cup_{j=2}^{t} \{(b'_j, 1)\} \cup \cup_{j=2}^{t} \{b'_j \leftrightarrow (b_j \vee (b_1 \wedge \cdots \wedge b_{j-1}))\}, \quad (2)$$

where b'_j, $j = 2, \ldots, t$ are new variables.

The fact that TransformCore satisfies Property 1 can be easily derived from [Narodytska and Bacchus, 2014]; alternatively it can be proved by dualizing the proof of lemma 1. Note that the TransformCore and TransformCSet fulfill the following property: $|\phi'|$ is linear in $|\phi| + |\gamma|$, $\gamma \in \{|\pi|, |\varphi|\}$ which is helpful to keep the formula small in each iteration.

Core minimisation. Core minimisation is considered an expensive routine in MAXSAT solving. However, it was shown to be beneficial for example in verification domain [Belov *et al.*, 2013]. We found a useful lightweight core minimisation procedure. Our algorithm is based on destructive core minimisation proposed in [Belov *et al.*, 2012]. We also borrow the clause-set refinement technique from MUS extraction algorithms [Belov *et al.*, 2012]. We omit pseudocode here due to space restrictions. Given a core φ, the algorithm considers each clause in the core φ and checks whether the remaining clauses still form a core. If so, this clause is removed and we can reduce the size of the core using the new core φ' returned by the SAT solver. Otherwise, we move to the next clause. For efficiency, we only minimise cores of size at most 500. We also count the number of unsuccessful attempts to remove a clause from a core and terminate if it exceeds a constant threshold, which was 25 in our case. We call this variant of our algorithm PRIMALDUALMIN.

Theoretical analysis and discussion

Next, we carry out a theoretical analysis of our algorithm. It can produce many possible sequences of SAT and UNSAT problems. We analyse two corner executions of it, PRIMAL and DUAL, and show that they complement each other. We also compare PRIMALDUALMAXSAT and UBLINEAR to demonstrate the space-time tradeoff between the two algorithms. As the most expensive operation in these search procedures is a SAT call, we compare these algorithms in terms of the number of SAT calls they perform. The number of calls usually depends on core/correction sets that the SAT solver returns. Thus, for uniformity we only consider lower bounds on the number of SAT calls for each algorithm.

Proposition 1 *There is a family of formulas on which* UBLINEAR *performs* $O(1)$ SAT *calls and* PRIMALDUALMAXSAT *performs* $\Omega(\sqrt{m})$ SAT *calls.*

Proof: Consider a problem ϕ such that there are $\sqrt{m}$ disjoint correction sets of size $\sqrt{m}$ each and similarly $\sqrt{m}$ disjoint cores. As all correction sets are disjoint and all cores are disjoint, Algorithm 1 will make at least $\sqrt{m}$ SAT calls before it terminates. UBLINEAR solves the problem in two calls. It gets one correction set of size $\sqrt{m}$, and adds a strengthening cardinality constraint to ϕ. This results in an unsatisfiable formula and UBLINEAR terminates. ◀

Note that UBLINEAR solves any problem with two SAT calls in the best case. Hence, Proposition 1 shows that UBLINEAR dominates PRIMALDUALMAXSAT in terms of the number of calls if we consider the best possible execution. However, best possible executions are rare in practice. In fact, the SAT solver tends to return large correction sets as those are easy to find and UBLINEAR needs to post cardinality/PseudoBoolean constraint over a large set of variables.

Proposition 2 *There is a family of formulas on which* UBLINEAR *and* PRIMALDUALMAXSAT *performs* $O(1)$ SAT *calls. However,* UBLINEAR *increases the size of the formula by* $O(m^2)$ *and* PRIMALDUALMAXSAT *by* $O(m)$.

Proof: Consider a problem ϕ such that $(b_i, w_i) \in \mathit{softCls}(\phi), w_i = 2^i$, $i = 1, \ldots, m$. Suppose, there is a single correction set π. Both algorithms terminate after the first iteration. PRIMALDUALMAXSAT adds $O(m)$ clauses and variables to ϕ. UBLINEAR adds $O(m \log(\sum_{i=1}^{m} w_i)) = O(m^2)$ clauses and variables to ϕ as it has to encode $\sum_{i=1}^{m} w_i \overline{b_i} < \mathit{maxcost}(\pi)$ [Aavani *et al.*, 2013]. ◀

Next we investigate relations between two corner executions of PRIMALDUALMAXSAT, PRIMAL and DUAL.

Proposition 3 *There is a family of formulas where* DUAL *makes* $O(1)$ SAT *calls and* PRIMAL *makes* $\Omega(m)$ SAT *calls.*

Proof: Consider a problem ϕ such that there are m unit cores and a single correction set. In the best execution, PRIMAL takes m calls to a SAT solver. On the other hand, DUAL finds the single correction set at the first iteration. Hence, the formula on the 2nd step ϕ^i, $i = 2$, is unsatisfiable and algorithm terminates with two calls. ◀

Proposition 4 *There is a family of formulas where* PRIMAL *makes* $O(1)$ SAT *calls and* DUAL *makes* $\Omega(m)$ SAT *call.*

Proof: Consider a formula ϕ that has a single core of size m. In the best execution, PRIMAL takes 2 calls to the SAT solver. On the other hand, DUAL needs to call the solver m times in order to enumerate all correction sets. ◀

Next we discuss high-level properties of PRIMALDUALMAXSAT which, we believe, make it efficient in practice as we will show in our experiments. First, Propositions 3–4 demonstrate that corner executions of the solver, PRIMAL and DUAL, are complementary in their strengths. If we have a small number of minimal correction sets then DUAL execution might solve the problem. If we have a small number of cores then PRIMAL might occur to solve the problem efficiently. Second, as our partition phase is inherited from the relaxation search, we only obtain minimal correction sets that are potentially more useful in proving the optimality and, at the same time we do not need to perform costly correction set minimisation. Third, as mentioned before, our new procedure to process correction sets is very lightweight compared to the standard UBLINEAR procedure with PB constraints. Forthly, the structure of PRIMALDUALMAXSAT allows incremental implementations so that learnt clauses are carried across SAT calls regardless of whether we find a core or a correction set.

Experimental Evaluation

We ran our experiments on a cluster of AMD Opteron@2.1 GHz machines restricted to 3.5Gb or RAM and 3600 seconds of CPU time. We used all industrial instances

	PD^m	PD	eva^c	$msuoll$	$msunc$	wpm_1^{13}
PD^m	–	21	61	87	191	199
PD	5	–	46	78	174	184
eva^c	7	8	–	58	158	144
$msuoll$	3	10	28	–	117	134
$msunc$	3	2	24	13	–	110
wpm_1^{13}	2	3	1	21	101	–

Table 1: The number of instances solved by algorithm i but not by algorithm j is shown in row i, column j.

from the 2014 Evaluation of Max-SAT Solvers competition. We compare our solver, PRIMALDUALMAXSAT (PD) and PRIMALDUALMIN (PD^m), with state-of-the-art ground weighted partial MAXSAT solvers: eva^c [Narodytska and Bacchus, 2014], the 'msu-oll-sn-wo' solver($msuoll$) [Morgado *et al.*, 2014], wpm_1^{13} [Ansótegui *et al.*, 2013], and $msunc$ [Morgado *et al.*, 2012]. PRIMALDUALMAXSAT (PD) and PRIMALDUALMIN (PD^m) are implemented inside MiniSAT [Een and Sorensson, 2003] and takes advantage of properties mention in the 'PRIMALDUALMAXSAT algorithm' section. We observed that some candidate correction sets were expensive to check. We, therefore, apply a timeout of three seconds when checking π^i. The timeout was gradually decreased as search proceeded. We recall that if $I(b) = 0$ then $\{b\} \cup \{b' \mid I(b') = 1\}$ is a core. If the timeout is exceeded we apply the primal transformation to this core. The eva^c solver also uses MiniSAT as a SAT solver. Note that the top performing (non-portfolio) solvers in MaxSAT'14 evaluation, $Eva500a$ and $MSCG$ are hybrid solvers, incorporating different algorithms in problem categories. We compare against their WPMS solvers that are published in the literature: eva^c [Narodytska and Bacchus, 2014] and the 'msu-oll-sn-wo' solver [Morgado *et al.*, 2014]. We used stratification and hardening techniques [Ansótegui *et al.*, 2013] in PRIMALDUALMAXSAT for solving weighted partial MAXSAT instances.

Tables 2–4 show that PRIMALDUALMAXSAT and PRIMALDUALMIN significantly improve over other solvers in the majority of benchmarks, solving more instances and using less time. For example, on 'pref. plan.' and 'One path' benchmarks, PRIMALDUALMIN is about three times faster than eva^c, $msuoll$ and $msunc$. Our algorithms also solve significantly more instances in 'Multiple path' and 'msp' sets compared to other solvers. Note also that our use of minimization does not help in the MAXSAT category, but is useful in the two other categories. Table 1 gives a pairwise comparison between all pairs of the algorithms and Figure 1 shows a cactus plot over all industrial instances. Table 1 illustrates that PRIMALDUALMIN solves almost all problems also solved by other solvers, and other solvers do not offer any significant edge. For example, eva^c can only solve 7 instances not solved by PRIMALDUALMIN, while PRIMALDUALMIN solves 61 problems not solved by eva^c.

Conclusions

We developed a new iterative algorithm for MAXSAT that solves MAXSAT using a sequence of SAT and UNSAT formulas. We investigated theoretical properties of the algorithm and demonstrated its efficiency on a wide range of industrial

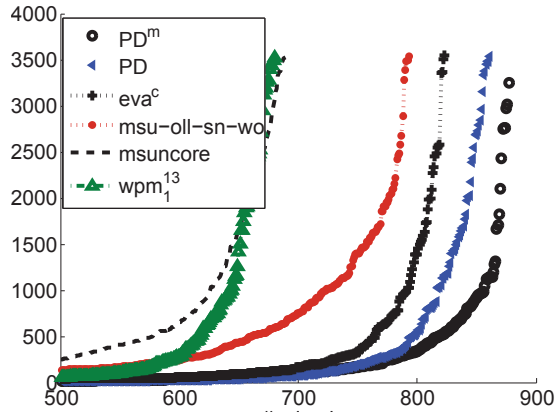

Figure 1: The cactus plot for all industrial instances from the 2014 Evaluation of MaxSAT Solvers competition.

benchmarks from the 2014 Evaluation of Max-SAT Solvers.

	PD^m		PD		eva^c		$msuoll$		$msunc$		wpm_1^{13}	
	#	avg t	#	avg t	#	avg t	#	avg t	#	avg t	#	avg t
haplotyping	**98**	**51.1**	97	19.0	95	32.8	97	370.6	60	589.8	92	130.6
hs-time.	1	176.0	1	342.0	**2**	**1202.0**	1	123.0	1	1130.0	0	0.0
packup	**97**	**16.5**	95	35.6	89	45.1	69	170.4	28	208.5	90	26.3
pref. plan.	**29**	51.1	**29**	**36.6**	**29**	125.6	27	230.0	28	175.3	27	58.1
timetabling	12	618.8	**13**	**554.7**	11	285.1	10	616.7	8	272.5	8	687.5
upgrad.	**100**	1.2	**100**	**0.9**	**100**	44.1	**100**	128.4	**100**	128.2	**100**	6.2
wcsp.dir	**14**	1.6	**14**	**1.2**	**14**	5.9	**14**	95.4	11	35.2	**14**	124.4
wcsp.log	**14**	**12.5**	13	378.3	**14**	90.3	**14**	37.6	12	316.8	**14**	527.6
Total	**365**	**43.9**	362	52.1	354	62.3	332	225.6	248	267.9	345	90.4

Table 2: WPMS Industrial instances

References

[Aavani *et al.*, 2013] Amir Aavani, David G. Mitchell, and Eugenia Ternovska. New encoding for translating pseudo-boolean constraints into sat. In Alan M. Frisch and Peter Gregory, editors, *SARA*. AAAI, 2013.

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

[Bacchus *et al.*, 2014] Fahiem Bacchus, Jessica Davies, Maria Tsimpoukelli, and George Katsirelos. Relaxation Search: A Simple Way of Managing Optional Clauses. In *Proceedings of the 28th Conference on Artificial intelligence (AAAI'14)*, pages 2717–2723, 2014.

[Belov *et al.*, 2012] Anton Belov, Ines Lynce, and Joao Marques-Silva. Towards efficient mus extraction. *AI Commun.*, 25(2):97–116, 2012.

[Belov *et al.*, 2013] Anton Belov, Huan Chen, Alan Mishchenko, and Joao Marques Silva. Core minimization in SAT-based abstraction. In *Design, Automation and Test in Europe (DATE)*, pages 1411–1416. ACM, March 2013.

[Cimatti *et al.*, 2010] Alessandro Cimatti, Anders Franzen, Alberto Griggio, Roberto Sebastiani, and Cristian Stenico. Satisfiability modulo the theory of costs: Foundations and applications. In Javier Esparza and Rupak Majumdar, editors,

	PD^m		PD		eva^c		$msuoll$		$msunc$		wpm_1^{13}	
	#	avg t	#	avg t	#	avg t	#	avg t	#	avg t	#	avg t
aes	**1**	1.0	**1**	1.0	**1**	**0.0**	**1**	24.0	**1**	658.0	0	0.0
mesat	**11**	584.7	**11**	**530.5**	**11**	661.8	8	973.1	10	1021.5	5	1574.8
sugar	**12**	**294.8**	**12**	346.3	**12**	342.8	11	729.5	11	732.7	8	1551.1
fir	29	234.1	29	205.6	**32**	**63.1**	31	18.4	31	28.8	31	44.6
simp	**9**	149.6	**9**	**66.2**	**9**	94.0	**9**	285.2	**9**	415.0	**9**	381.4
su	32	75.4	**33**	**178.1**	31	82.5	32	214.8	**33**	443.4	27	212.3
msp	**21**	**98.6**	19	164.5	8	1300.9	14	356.1	19	290.7	2	93.0
mtg	**30**	3.8	**30**	9.3	**30**	**1.0**	**30**	3.8	**30**	2.6	**30**	10.5
syn	**20**	**94.8**	19	230.4	16	19.8	15	94.1	14	137.5	15	264.0
circuit.	**4**	**114.8**	**4**	202.0	**4**	317.8	**4**	592.0	**4**	191.5	**4**	297.0
close sol.	49	281.6	**50**	**94.6**	49	288.9	38	540.9	26	413.8	25	415.2
des	**37**	**371.2**	32	355.4	30	345.6	35	576.4	31	657.8	19	177.9
hap.-assem.	**5**	15.8	**5**	12.0	**5**	**4.0**	**5**	17.6	4	86.8	**5**	5.6
hs-time.	**1**	149.0	**1**	120.0	**1**	146.0	**1**	63.0	**1**	**45.0**	0	0.0
mbd	42	82.3	39	23.0	41	84.9	**43**	**137.5**	37	484.6	36	180.7
packup	**40**	12.7	**40**	**8.9**	**40**	16.7	**40**	136.6	**40**	418.0	**40**	10.8
nencdr	**25**	**134.3**	**25**	486.2	23	415.0	21	763.0	**25**	461.3	15	271.7
nlogencdr	**25**	**25.4**	**25**	251.6	**25**	230.2	**25**	421.9	**25**	104.3	17	399.9
routing	**15**	4.5	**15**	0.2	**15**	**0.1**	**15**	1.3	**15**	1.3	**15**	8.3
protein ins	**6**	**1412.2**	3	291.0	2	50.5	3	822.3	3	512.3	2	164.5
Mult. path	**36**	**250.3**	33	1102.2	21	400.0	30	971.2	26	1249.3	3	121.7
One path	**25**	**15.0**	**25**	45.3	23	145.3	**25**	42.6	**25**	142.5	9	2759.2
Total	**475**	**165.7**	460	229.1	429	197.7	436	335.4	420	391.5	317	295.7

Table 3: PMS Industrial instances

	PD^m		PD		eva^c		$msuoll$		$msunc$		wpm_1^{13}	
	#	avg t	#	avg t	#	avg t	#	avg t	#	avg t	#	avg t
circuit-deb.	**3**	177.3	**3**	33.3	**3**	**16.7**	2	49.0	2	35.0	2	838.5
sean-safarpour	34	241.6	36	223.7	**37**	**348.6**	23	206.5	19	105.4	16	177.6
Total	37	236.4	39	209.1	**40**	**323.7**	25	193.9	21	98.7	18	251.1

Table 4: MaxSat Industrial instances

TACAS, volume 6015 of *Lecture Notes in Computer Science*, pages 99–113. Springer, 2010.

[Cooper *et al.*, 2006] Martin C. Cooper, Sylvain Cussat-Blanc, Marie de Roquemaurel, and Pierre Regnier. Soft Arc Consistency Applied to Optimal Planning. In Frederic Benhamou, editor, *Proceedings of the 12th International Conference on Principles and Practice of Constraint Programming (CP'06)*, volume 4204 of *Lecture Notes in Computer Science*, pages 680–684. Springer, 2006.

[Een and Sorensson, 2003] Niklas Een and Niklas Sorensson. An extensible SAT-solver. In Enrico Giunchiglia and Armando Tacchella, editors, *Proceedings of the 6th International Conference on Theory and Applications of Satisfiability Testing (SAT'03)*, volume 2919, pages 502–518. Springer, 2003.

[Graça *et al.*, 2012] Ana Graça, Inês Lynce, João Marques-Silva, and Arlindo L. Oliveira. Efficient and accurate haplotype inference by combining parsimony and pedigree information. In *Proceedings of the 4th International Conference on Algebraic and Numeric Biology*, pages 38–56, Berlin, Heidelberg, 2012. Springer-Verlag.

[Heras *et al.*, 2011] Federico Heras, António Morgado, and João Marques-Silva. Core-Guided Binary Search Algorithms for Maximum Satisfiability. In *AAAI*, 2011.

[Ignatiev *et al.*, 2014] Alexey Ignatiev, António Morgado, Vasco M. Manquinho, Inês Lynce, and João Marques-Silva. Progression in maximum satisfiability. In *Proceedings of the 21th European Conference on Artificial Intelligence(ECAI'14)*, pages 453–458, 2014.

[Juma *et al.*, 2012] Farah Juma, Eric I. Hsu, and Sheila A. McIlraith. Preference-based planning via maxsat. In *Proceedings of the Canadian Conference on Artificial Intelligence*, pages 109–120, 2012.

[Manquinho *et al.*, 2009] Vasco M. Manquinho, Joao P. Marques Silva, and Jordi Planes. Algorithms for weighted Boolean optimization. In Oliver Kullmann, editor, *Proceedings of the 12th International Conference on Theory and Applications of Satisfiability Testing (SAT'09)*, volume 5584, pages 495–508. Springer, 2009.

[Marques-Silva and Planes, 2008] Joao Marques-Silva and Jordi Planes. Algorithms for maximum satisfiability using unsatisfiable cores. In *DATE*, pages 408–413. IEEE, 2008.

[Martins *et al.*, 2014] Ruben Martins, Saurabh Joshi, Vasco Manquinho, and Inês Lynce. Incremental Cardinality Constraints for MaxSAT. In *International Conference on Principles and Practice of Constraint Programming*. LNCS, 2014.

[Morgado *et al.*, 2012] Antonio Morgado, Federico Heras, and Joao Marques-Silva. Improvements to core-guided binary search for MaxSAT. In Alessandro Cimatti and Roberto Sebastiani, editors, *Proceedings of the 15th International Conference on Theory and Applications of Satisfiability Testing (SAT'12)*, volume 7317, pages 284–297. Springer, 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 *Principles and Practice of Constraint Programming - 20th International Conference, CP 2014, Lyon, France, September 8-12, 2014. Proceedings*, pages 564–573, 2014.

[Narodytska and Bacchus, 2014] Nina Narodytska and Fahiem Bacchus. Maximum satisfiability using core-guided maxsat resolution. In *Proceedings of the 28th Conference on Artificial intelligence (AAAI'14)*, pages 2717–2723, 2014.

[Parrain, 2010] Daniel LeBerre Anne Parrain. The sat4j library, release 2.2. *JSAT*, 7(2-3):59–6, 2010.

[Rosa and Giunchiglia, 2013] Emanuele Di Rosa and Enrico Giunchiglia. Combining approaches for solving satisfiability problems with qualitative preferences. *AI Commun.*, 26(4):395–408, 2013.

[Strickland *et al.*, 2005] Dawn M. Strickland, Earl Barnes, and Joel S. Sokol. Optimal protein structure alignment using maximum cliques. *Oper. Res.*, 53(3):389–402, May 2005.

[Vasquez and Hao, 2001] Michel Vasquez and Jin-Kao Hao. 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, 2001.