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 PRIMALDUAL.MAXSAT solver, shows robust performance and outperforms the best WPMS algorithms on industrial instances by a noticeable margin.

Background

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

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 weighted clauses, \( \phi = \{(C_1, w_1), \ldots, (C_m, w_m)\} \). A special case of WPMS is Partial MAXSAT (PMS), where the problem weights are in the set \( \{1, \infty\} \). We denote the set of soft clauses in \( \phi \) by \( \text{softCls}(\phi) \) and the set of hard clauses by \( \text{hardCls}(\phi) \). W.l.o.g., we can assume that soft clauses \((C, w)\) in \( \phi \) are unit clauses \((b, w)\) as \( \phi \) can be converted to this form by introducing replication variables: \( \phi = [\phi \setminus \{(C, w)\}] \cup \{(C \leftrightarrow b), (b, w)\} \). We also assume standard clausification, e.g., \( C \leftrightarrow b \) denotes \( \{(b \lor C)\} \cup \bigcup_{l \in C} \{l \lor b\} \).
An assignment $I$ is **feasible** for $\phi$ iff $I$ satisfies all hard clauses of $\phi$. The cost of $I$ is the sum of weights of clauses that it falsifies: $\text{cost}(I, \phi) = \sum_{(b, w) \in \phi \setminus I} w$. $I$ is an **optimal assignment** for $\phi$ iff it is a feasible assignment of minimal cost. We denote this minimal cost by $\text{cost}(\phi) = \min_{I \text{ feasible}} \text{cost}(I, \phi)$. The maximal finite cost is $\max_{I \text{ feasible}} \text{cost}(I, \phi) = \sum_{(b, w) \in \phi \setminus I} w$.

To simplify notation, we omit weights for hard clauses and we assume they are satisfiable. A **correction set** $I$ is a satisfiable formula. A **core** $\pi$ is a minimal formula simultaneously by performing a sequence of SAT searches (Algorithm 1, lines 7–10), we have a candidate core $\phi' = \{ b \mid I(b) = 1 \}$ and a candidate correction set $\pi' = \{ b \mid I(b) = 0 \}$. We call the SAT solver in the context of $\text{hardCls}(\phi')$, and partial assignment $I$ in line 13 to determine satisfiability of $\text{hardCls}(\phi') \mid I$. If $\text{hardCls}(\phi') \mid I$ is UNSAT then it returns a subset of $\phi'$ as a valid core; otherwise $\pi'$ is a valid correction set.

The partition phase has several useful properties. First, if the solver is satisfiable using a candidate set $\pi$, 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 $\{ b \mid I(b) = 0 \}$ is a core of $\phi$. In this case, if checking whether $\pi$ is a core 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.

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.

**PRIMALDUAL MAXSAT 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 $\phi'$ and $\pi'$ at the $i$th 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 $\phi'$ a candidate core and $\pi'$ a candidate correction set. Note that by splitting soft clauses, we guarantee that either $\phi'$ is a valid core or $\pi'$ 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 $\phi' = \{ b \mid I(b) = 1 \}$ and a candidate correction set $\pi' = \{ b \mid I(b) = 0 \}$. The second class of solvers perform so called UNSAT-SAT search (LINESAT), 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.

**Properties of the Transformation.** The transformation phase takes as input a core or a correction set. If $\text{hardCls}(\phi') \mid I$ is satisfiable and $\pi'$ is empty then we can satisfy all soft clauses in $\phi'$ and the algorithm terminates (line 16). A dual case is when $\text{hardCls}(\phi') \mid I$ is unsatisfiable and $\phi'$ is empty (line 18). In this case, $\text{hardCls}(\phi')$ 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 $\phi'$ and $\phi' \pi'$ as input. It introduces new variables, adds new clauses and deletes some soft clauses in $\phi$. Let $SI(\pi)$ be the set of solutions that assign all clauses in $\pi$ to false.

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

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

**Cost Update** $\text{cost}(\phi') = \text{cost}(\phi) - 1$.

**Termination** $\text{maxcost}(\phi') < \text{maxcost}(\phi)$.

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

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

247
Consider TransformCore and TransformCSet. Note that a standard way to transform a formula given the correction set \( S' \) is to add a PseudoBoolean constraint \( \sum_{i=1}^{m} x_i \leq \text{cost}(S') \). 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 \( \pi \) we transform \( \phi \) by adding clauses only over variables in clauses of \( \pi \) 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 \( \phi \).

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

\[
\phi' = (\phi \setminus \pi) \cup \{ \nu_{i=1}^{t} (b_i, 1) \} \cup \bigcup_{j=2}^{t} \{ b_j'^{t} \leftrightarrow (b_j \land (b_1 \lor \cdots \lor b_{j-1})) \},
\]

where \( b_{j'}' = 2, \ldots, t \) are new variables.
To establish that Algorithm 2 satisfies Property 2 we show:

Lemma 1 Consider \( \phi' = \text{TransformCSet}(\phi, \pi) \), where \( \pi = \{(b_1, 1), \ldots, (b_1, 1)\} \). There is a cost-preserving bijection between solutions of \( \phi \land (\lor_{i=1}^{m} b_i) \) and solutions of \( \phi' \).

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

For \( j > t \) the variables \( b_i \) are unchanged. Suppose \( j \leq t \). Consider two cases. We need to keep in mind that satisfaction of a soft clause \( (b_i', 1) \) in \( \phi' \) functionally depends on assignment of \( b_j \) as \( b_j' \leftrightarrow b_j \lor \cdots \lor b_{j-1} \). Suppose \( I(b_i) = 1 \). We show that for \( j = 2, \ldots, t \) under the assignment \( I \) the following holds: \( I(b_j') = I(b_j') \).

Consider the \( j \)th 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 \( I(b_j) = 1 \). 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 \( \phi \) and \( \phi' \).

Consider the \( j \)th 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 \( I(b_p) = 1 \), \( I(b_1) \lor \cdots \lor I(b_{j-1}) \) is true. Hence, \( I(b_j') = I(b_j) \).

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

Consider the reverse direction. Note that \( \phi' \) and \( \phi \land (\lor_{i=1}^{m} b_i) \) 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 \( \pi \) is a correction set in the proof of Lemma 1. Therefore, the following result holds.

Corollary 1 Consider \( \phi' = \text{TransformCSet}(\phi, \delta) \), where \( \delta \equiv \text{softCls}(\phi) \). Lemma 1 holds for \( \phi, \delta \) and \( \phi' \).

Corollary 2 Consider \( \phi' = \text{TransformCSet}(\phi, \pi) \), where \( \pi = \{(b_1, 1), \ldots, (b_1, 1)\} \). Then \( 1 / |\text{softCls}(\phi)| > |\text{softCls}(\phi')| \), \( \text{cost}(\phi') \geq \text{cost}(\phi) \) and \( \text{cost}(\phi') = \text{cost}(\phi') \). Hence, \( \text{cost}(\phi'') > \text{cost}(\phi) \).

Theorem 2 Consider \( \phi' = \text{TransformCSet}(\phi, \pi) \), where \( \pi = \{(b_1, 1), \ldots, (b_1, 1)\} \). Then \( \text{cost}(\phi) = \min\{\text{cost}(\phi'), \text{maxcost}(\pi)\} \).

Proof: Consider \( \pi = \) a minimum correction set of \( \phi \). By Corollary 2, \( \text{cost}(\phi') \geq \text{cost}(\phi) \), so we get \( \text{cost}(\phi) = \min\{\text{cost}(\phi'), \text{maxcost}(\pi)\} \). Suppose \( \pi \) is not a minimum correction set of \( \phi \). Consider a minimum correction set \( \pi^* \) of \( \phi \). As \( \pi \notin \pi^*, \pi^* \) is a minimum correction set of \( \phi \land (\lor_{i=1}^{m} b_i) \). By Corollary 2, \( \text{cost}(\phi') = \text{cost}(\phi \land (\lor_{i=1}^{m} b_i)) \). As \( \text{maxcost}(\pi^*) > \text{maxcost}(\pi) \) 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 [Morgan et al., 2013; Ansótegui et al., 2013]. However, there is one complication to take into account. Consider a correction set \( \pi = \{(b_1, w_1), \ldots, (b_t, w_t)\} \), \( w_1 \leq w_t, i = 2, \ldots, t \). We split a clause \( (b_i, w_i) \) into two clones: \( (b_i, w_i) \) and \( (b_t, w_t - w_i) \). Note that \( (b_i, w_i), \ldots, (b_t, w_t) \) might not be a correction set after splitting. We perform \( \phi' = \text{TransformCSet}(\phi, \{(b_1, w_1), \ldots, (b_t, w_t)\}) \). Corollary 1 tells us that \( \phi' \) contains all solutions of \( \phi \) that violate \( (\lor_{i=1}^{m} b_i) \). For Theorem 2 to hold, we need to prove that all correction sets that violate \( (\lor_{i=1}^{m} b_i) \) are supersets of \( \pi \). This condition holds as any feasible solution \( I \) of \( \phi \) that violates \( (\lor_{i=1}^{m} b_i) \) must falsify all clauses in \( \{(b_1, w_1), \ldots, (b_t, w_t)\} \) and their clones. Such a solution corresponds to a correction set that is a superset of \( \pi \).

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 \( \phi \) we convert it into a formula \( \phi' \) using TransformCore, such that \( \text{cost}(\phi') = \text{cost}(\phi) - 1 \). We here only provide the definition of TransformCore.

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

\[
\phi' = (\phi \land \varphi) \lor \lor_{i=1}^{m} b_i \lor_{j=2}^{t} \{(b_j, 1)\} \cup (2)
\]

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 $\phi$, the algorithm considers each clause in the core $\varphi$ 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 $\varphi' \equiv \sum_{i=1}^{m} w_i \gamma_i$ 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 $\phi$ 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 $\phi$. 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 $\phi$ such that $(b_i, w_i) \in softClie(\phi)$, $w_i = 2^i$, $i = 1, \ldots, m$. Suppose, there is a single correction set $\pi$. Both algorithms terminate after the first iteration. PRIMALDUALMaxSAT adds $O(m)$ clauses and variables to $\phi$. UBLINEAR adds $O(m \log(\sum_{i=1}^{m} w_i)) = O(m^2)$ clauses and variables to $\phi$ as it has to encode $\sum_{i=1}^{m} w_i b_i < \maxcost(\pi)$ [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 $\phi$ 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 $\phi', i = 2$, 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 $\phi$ 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. Fortly, 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
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 ($PD^m$) and PRIMALDUALMIN ($PD^n$), with state-of-the-art ground weighted partial MAXSat solvers: eva$^c$ [Narodytska and Bacchus, 2014], the ‘msu-oll-wo’ solver(msuoll) [Morgado et al., 2014], wpm$^c$ [Ansôtegui 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 $\pi^t$. The timeout was gradually decreased as search proceeded. We recall that if $I(b) = 0$ then $\{b\} \cup \{b' | I(b') = 1\}$ 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 our WPMS solvers that are published in the literature: eva$^c$ [Narodytska and Bacchus, 2014] and the ’msu-oll-wo’ solver [Morgado et al., 2014]. We used stratification and hardening techniques [Ansotidegui 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 benchmarks from the 2014 Evaluation of Max-SAT Solvers.

References


[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,
Table 3: PMS Industrial instances

<table>
<thead>
<tr>
<th>circuit-deb</th>
<th>scan-safapour</th>
<th>(PD_{avg})</th>
<th>(PD)</th>
<th>(ev)</th>
<th>maxsat</th>
<th>maxunc</th>
<th>wpunc</th>
<th>(wp)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td># avg t</td>
<td># avg t</td>
<td># avg t</td>
<td># avg t</td>
<td># avg t</td>
<td># avg t</td>
<td># avg t</td>
</tr>
<tr>
<td></td>
<td></td>
<td>3.177</td>
<td>3.33</td>
<td>3.167</td>
<td>2</td>
<td>49.9</td>
<td>35.0</td>
<td>2</td>
</tr>
<tr>
<td></td>
<td></td>
<td>34241.6</td>
<td>36223.7</td>
<td>27348.6</td>
<td>23</td>
<td>206.5</td>
<td>19</td>
<td>105.4</td>
</tr>
<tr>
<td>Total</td>
<td></td>
<td>37236.4</td>
<td>39209.1</td>
<td>40323.7</td>
<td>25</td>
<td>193.9</td>
<td>21</td>
<td>98.7</td>
</tr>
</tbody>
</table>

Table 4: MaxSat Industrial instances


