Core-Guided Binary Search Algorithms for Maximum Satisfiability [*]

Federico Heras and **Antonio Morgado**
University College Dublin, Dublin, Ireland
{fheras,ajrm}@ucd.ie

Joao Marques-Silva
University College Dublin, Dublin, Ireland
IST/INESC-ID, Lisbon, Portugal
jpms@ucd.ie

Abstract

Several MaxSAT algorithms based on iterative SAT solving have been proposed in recent years. These algorithms are in general the most efficient for real-world applications. Existing data indicates that, among MaxSAT algorithms based on iterative SAT solving, the most efficient ones are *core-guided*, i.e. algorithms which guide the search by iteratively computing *unsatisfiable subformulas* (or *cores*). For weighted MaxSAT, core-guided algorithms exhibit a number of important drawbacks, including a possibly exponential number of iterations and the use of a large number of auxiliary variables. This paper develops two new algorithms for (weighted) MaxSAT that address these two drawbacks. The first MaxSAT algorithm implements *core-guided* iterative SAT solving with *binary search*. The second algorithm extends the first one by exploiting *disjoint cores*. The empirical evaluation shows that core-guided binary search is competitive with current MaxSAT solvers.

Introduction

Maximum Satisfiability (MaxSAT) is the well-known optimization variant of Boolean Satisfiability (SAT). MaxSAT finds a wide range of practical applications (Biere et al. 2009), it can be used for solving other optimization problems (Heras, Larrosa, and Oliveras 2008), and it has been extended to richer domains (e.g. (Biere et al. 2009)). Early MaxSAT algorithms were based on branch-and-bound search (Biere et al. 2009). These algorithms perform very well on random and crafted instances of MaxSAT, but are in general inefficient for industrial instances. An alternative solution to branch-and-bound search is based on iterative calls to a SAT solver (or *oracle*).

Two main approaches can be considered for solving MaxSAT with iterative calls to a SAT oracle: one based on *linear search* and one based on *binary search*. The most widely used approach consists of relaxing the *soft* clauses and then iteratively refining upper bounds on the optimum solution (e.g. (Berre and Parrain 2010)). Recent work proposed to guide the search with *unsatisfiable subformulas* (or *cores*) and is most often based on refining lower bounds (e.g. (Fu and Malik 2006; Marques-Silva and Planes 2008; Ansótegui, Bonet, and Levy 2009; Manquinho, Marques-Silva, and Planes 2009; Ansótegui, Bonet, and Levy 2010)).

One essential characterization of MaxSAT algorithms based on iterative SAT solving is the worst-case number of calls as a function of the problem instance size (usually reflected in the sum of weights). Unsurprisingly, non-core-guided linear search approaches require an exponential number of calls to a SAT oracle in the worst case. For core-guided approaches, the worst-case number of calls is unknown, but it is generally believed to be exponential in the worst-case (e.g. (Davies, Cho, and Bacchus 2010)).

An alternative approach for MaxSAT algorithms based on iterative SAT solving is to implement binary search by refining both lower and upper bounds on the optimum solution (e.g. (Gottlob 1995; Fu and Malik 2006; Biere et al. 2009; Cimatti et al. 2010)). Although binary search is optimal in terms of the number of calls to a SAT oracle, it has seldom been used in practical MaxSAT solvers; In particular, given that all clauses are relaxed, cardinality constraints are fairly complex, and this impacts severely the ability of SAT solvers for proving unsatisfiability.

This paper has two main contributions. First, the paper shows that recent core-guided MaxSAT algorithms (Ansótegui, Bonet, and Levy 2010) can require an exponential number of calls to a SAT oracle. This represents the first theoretical analysis of the number of calls to a SAT oracle for most of existing MaxSAT algorithms based on iterative SAT solving. Second, and more importantly, the paper develops two new algorithms for MaxSAT. The first algorithm is core-guided and, in order to guarantee that the number of calls to the SAT oracle is polynomial in the worst-case, it implements binary search. Thus, it is referred to as *core-guided binary search* for MaxSAT. The second algorithm improves the first one by taking advantage of *disjoint cores*. Experimental results, obtained on non-random instances from recent MaxSAT Evaluations (Argelich et al.), demonstrate that core-guided binary search with disjoint cores for MaxSAT is one of the most robust approaches in terms of the number of solved instances.

Preliminaries

This section introduces the necessary definitions and notation related to the SAT and MaxSAT problems.

[*]This work was supported by SFI grant 09/PI/I2618.

SAT. The standard SAT definitions apply (e.g. (Biere et al. 2009)), some of which are briefly reviewed below. Let $X = \{x_1, x_2, \ldots, x_n\}$ be a set of Boolean variables. A *literal* is either a variable x_i or its negation $\bar{x}_i$. A *clause* C is a disjunction of literals. A formula in *conjunctive normal form* (CNF) φ is a set of clauses. An *assignment* is a set of literals $\mathcal{A} = \{l_1, l_2, \ldots, l_n\}$. Given an unsatisfiable SAT formula φ, a subset of clauses φ_C whose conjunction is still unsatisfiable is called an *unsatisfiable core* of the original formula. Modern SAT solvers can be instructed to generate an unsatisfiable core (Zhang and Malik 2003).

MaxSAT. A *weighted* clause is a pair (C, w), where C is a clause and w is the cost of its falsification, also called its *weight*. Many real problems contain clauses that *must* be satisfied. Such clauses are called *mandatory* (or *hard*) and are associated with a special weight $\top$. Note that any weight $w \geq \top$ indicates that the associated clause must be necessarily satisfied. Thus, w can be replaced by $\top$ without changing the problem. Consequently, all weights take values in $\{0, \ldots, \top\}$. Non-mandatory clauses are also called *soft* clauses. A *weighted* formula in *conjunctive normal form* (WCNF) φ is a set of weighted clauses.

A *model* is a complete assignment $\mathcal{A}$ that satisfies all mandatory clauses. The *cost of a model* is the sum of weights of the soft clauses that it falsifies. Given a WCNF formula, the *Weighted Partial* MaxSAT is the problem of finding a model of minimum cost.

Notation for MaxSAT Algorithms. All algorithms considered in this paper are assumed to handle *weighted partial MaxSAT*. Hence, weighted formulas φ are of the form $\varphi = \varphi_S \cup \varphi_H$ where φ_S is a set of m soft clauses ($\varphi_S = \{(C_i, w_i) \mid w_i < \top \wedge\ 1 \leq i \leq m\}$), and φ_H is a set of hard clauses ($\varphi_H = \{(C_i, \top) \mid m+1 \leq i \leq m'\}$). The sum of the weights of the soft clauses ($\sum_{i=1}^{m} w_i$) is noted as $\mathcal{W}$. Also, $\mathcal{W} = m$ for unweighted instances.

The function $Cls(\varphi)$ returns a CNF formula, that is, it returns only the clauses (ignoring weights) for a weighted formula ($Cls(\varphi) = \{C \mid (C, w) \in \varphi\}$). Function $Soft(\varphi)$ gives the set of soft clauses in φ ($Soft(\varphi) = \{(C, w) \mid (C, w) \in \varphi \wedge w \neq \top\}$). The input of a MaxSAT algorithm is a weighted formula φ. This formula may be modified by adding *relaxation variables* and *cardinality / pseudo-Boolean constraints* (Hansen and Jaumard 1990; Eén and Sörensson 2006). All these modifications are performed on a *working formula* φ_W.

Throughout this paper, each soft clause can be associated with at most one *relaxation variable*. Relaxation variables are kept in a set R. Each relaxation variable r_i is always associated to the clause with weight w_i. Function $Relax(R, \varphi, \psi)$ adds relaxation variables to a subset of clauses ψ in the weighted formula φ and adds the relaxation variables to R.

For each call to a SAT solver, *cardinality / pseudo-Boolean constraints* (Fu and Malik 2006) may be added and translated to hard clauses. Such constraints usually state that the sum of the weights of the relaxed weighted clauses is lower or equal (*AtMostK* with $\sum_{i=1}^{m} w_i r_i \leq K$) or greater or equal (*AtLeastK* with $\sum_{i=1}^{m} w_i r_i \geq K$) than a specific value K.

Some algorithms maintain a *lower bound* λ and an *upper bound* μ on the optimum (minimum) model cost. When a satisfying complete assignment $\mathcal{A}$ is found, these algorithms need to know the sum of weights of soft clauses for which the relaxation variable is assigned to true. This is represented by $\sum_{i=1}^{m} w_i \times \mathcal{A}\langle r_i \rangle$. $last\mathcal{A}$ stores the assignment of the optimal solution returned by an algorithm, if it exists, otherwise it is empty.

Let $\mathcal{U} = \{U_1, \ldots, U_k\}$ be a set of cores and for each core U_i there is a set of relaxation variables R_i. A core $U_i \in \mathcal{U}$ is *disjoint* when $\forall_{U_j \in \mathcal{U}} (R_i \cap R_j = \emptyset \wedge i \neq j)$. Disjoint cores have been used to improve MaxSAT solver performance (Ansótegui, Bonet, and Levy 2010).

Iterative MaxSAT Algorithms

This section analyzes MaxSAT algorithms based on iterative calls to a SAT solver, in terms of the worst-case number of executed calls. Existing iterative algorithms either perform *linear search* or *binary search* on the sum of the weights of the soft clauses.

Algorithms based on linear search can be of one of two main variants: algorithms that refine an upper bound on the value of the optimum solution, and algorithms that refine a lower bound. Algorithms that iteratively refine upper bounds will be referred to as *linear search Sat-Unsat*, denoting that all SAT calls but the last will return true. Algorithms that iteratively refine lower bounds will be referred to as *linear search Unsat-Sat*, denoting that all SAT calls but the last will return false. Existing implementations of these algorithms, which do not exploit unsatisfiable cores, can take a number of iterations that grows linearly with $\mathcal{W}$, and so are exponential on the size of the problem instance. For example, cost refinement can take value 1 in each iteration, and so the number of iterations grows with $\mathcal{W}$, which in the worst-case is exponential on the instance size.

Proposition 1 *MaxSAT algorithms based on linear search Sat-Unsat or Unsat-Sat execute $\Theta(\mathcal{W})$ calls to a SAT oracle in the worst case.*

Recent linear search algorithms for (partial) (weighted) MaxSAT are core-guided (Fu and Malik 2006; Marques-Silva and Planes 2008; Ansótegui, Bonet, and Levy 2009; Manquinho, Marques-Silva, and Planes 2009; Ansótegui, Bonet, and Levy 2010). For these algorithms, there is no formal characterization of the worst case number of calls to a SAT oracle for the weighted MaxSAT case. However, it is believed that the number will also be exponential (Davies, Cho, and Bacchus 2010). The remainder of this section investigates this question, and starts by analyzing WPM2 (Ansótegui, Bonet, and Levy 2010) (see Algorithm 1).

Example 1 *Consider an instance $\varphi = \varphi_S \cup \varphi_H$, where $\varphi_S = \{(\bar{x}_1, 1), \ldots (\bar{x}_m, 2^{m-1})\}$ and $\varphi_H = \{(x_1 \vee x_2 \vee \ldots \vee x_m, \top), (x_2 \vee \ldots \vee x_m \vee y_1 \vee z_1, \top), \ldots, (x_m \vee y_1 \vee z_1, \top), (\bar{y}_1 \vee y_2, \top), (\bar{y}_1 \vee y_3, \top), (\bar{y}_2 \vee \bar{y}_3, \top), (y_2 \vee y_3, \top), (\bar{z}_1 \vee z_2, \top), (\bar{z}_1 \vee z_3, \top), (\bar{z}_2 \vee \bar{z}_3, \top), (z_2 \vee z_3, \top)\}$. The optimum solution for this instance is 2^{m-1}.*

WPM2 *starts by adding a relaxation variable r_i to each soft clause C_i and initializes the set of* covers $SC =$

Algorithm 1: WPM2 Algorithm

```
    Input: $\varphi$
1   function Newbound ($AL$, $B$)
2       $k \leftarrow$ Bound($AL$, $B$)// Bound()$= \sum\{k' \mid \sum_{i\in B'} w_i \times r_i \leq k' \in AM \wedge B' \subseteq B\}$
3       repeat
4           $k \leftarrow$ SubSetSum( $\{w_i \mid i \in B\}, k$)
5       until SAT( CNF( $AL \cup \{\sum_{i\in B} r_i = k\}$))
6       return $k$
7   end
8   $(R, \varphi_W) \leftarrow$ Relax($\emptyset, \varphi$,Cls(Soft($\varphi$)))
9   $SC \leftarrow \{\{i\} \mid C_i \in$ Soft($\varphi$)$\}$
10  $AL \leftarrow \emptyset$
11  $AM \leftarrow \{w_i \times r_i \leq 0 \mid (C_i \vee r_i, w_i) \in$ Soft($\varphi_W$) and $r_i \in R\}$
12  while true do
13      (st, $\varphi_C, \mathcal{A}$) $\leftarrow$ SAT(Cls($\varphi_W$)$\cup$CNF($AL \cup AM$))
14      if st $= true$ or $\varphi_C \cap$ Cls(Soft($\varphi_W$))$= \emptyset$ then
15          return Init($\mathcal{A}$)
16      end
17      $A \leftarrow \{i \mid (C_i \vee r_i) \in \varphi_C \cap$ Cls(Soft($\varphi_W$))$\}$
18      $RC \leftarrow \{B' \in SC \mid B' \cap A \neq \emptyset\}$
19      $B \leftarrow \bigcup_{B' \in RC} B'$
20      $k \leftarrow$ Newbound($AL, B$)
21      $SC \leftarrow SC \setminus RC \cup \{B\}$
22      $AL \leftarrow AL \cup \{\sum_{i\in B} w_i \times r_i \geq k\}$
23      $AM \leftarrow AM \setminus \{\sum_{i\in B'} w_i \times r_i \leq k' | B' \in RC\} \cup \{\sum_{i\in B} w_i \times r_i \leq k\}$
24  end
```

$\{\{1\},\ldots,\{m\}\}$, the set of AtLeast *constraints $AL = \emptyset$ and the set of* AtMost *constraints $AM = \{r_1 \leq 0, \ldots, 2^{m-1} \times r_m \leq 0\}$. The first call to the SAT solver returns unsatisfiable, and the unsatisfiable core contains all the soft clauses, all the AM constraints and clause $(x_1 \vee x_2 \vee \ldots \vee x_m)$. This core is discovered solely by applying unit propagation. Any other core requires the SAT solver to perform a decision assignment due to the y and z variables.* WPM2 *computes bound k with $Newbound()$. A new bound k is first computed with $Bound()$, which for the current AM is 0. Then, the algorithm calls $k = SubSetSum(\{1, 2, \ldots, 2^{m-1}\}, k)$ until $\{\sum_{i=1}^{m} 2^{i-1} r_i = k\} \cup AL$ becomes satisfied. Hence, $k = 1$. Sets SC, AL, AM are updated to $SC = \{\{1,\ldots,m\}\}$, $AL = \{\sum_{i=1}^{m} 2^{i-1} r_i \geq 1\}$, and $AM = \{\sum_{i=1}^{m} 2^{i-1} r_i \leq 1\}$.*

For the following iterations, any core discovered by the SAT solver will always intersect the set in SC, thus $RC = SC$, the $Bound()$ function returns the current k in AM and $SubSetSum()$ returns $k+1$. Also the call to the SAT solver in $Newbound()$ is always true and $Newbound()$ will return the previous k plus one. In each iteration, SC remains unchanged, AL is augmented with one new constraint and AM updates the right hand side of its only constraint. Thus, the main loop starts with $k = 1$ and iterates by increasing $k = k + 1$ until the optimum solution (2^{m-1}) is reached. Since both the main loop and $Newbound()$ make one call to the SAT solver per iteration, then the number of calls to the SAT solver is 2^m.

Theorem 1 WPM2 *(Ansótegui, Bonet, and Levy 2010) executes $\Theta(\mathcal{W})$ calls to a SAT oracle in the worst-case.*

Proof. For Example 1, WPM2 executes a number of calls to a SAT oracle that grows with the largest weight (so with the sum of the weights). □

For WPM1 / WMSU1 (Ansótegui, Bonet, and Levy 2009; Manquinho, Marques-Silva, and Planes 2009), the number of iterations can be related with the quality of computed unsatisfiable subformulas. Consider again Example 1. Suppose that the SAT solver always returns the complete CNF formula φ_W as the unsatisfiable core. Then, in each iteration, the lower bound is updated by 1. Since the MaxSAT solution is 2^{m-1}, then there will be an exponential number of calls to the SAT solver. In practice SAT solvers return reasonably accurate unsatisfiable cores. A detailed characterization of the worst-case number of iterations in this case is an open research topic. Another important drawback of WPM1 / WMSU1 is that clauses can have multiple relaxation variables (Fu and Malik 2006). This can reduce the ability of SAT solvers to exploit *unit propagation*.

Another class of iterative MaxSAT algorithms uses *binary search* (Fu and Malik 2006; Cimatti et al. 2010) Binary search maintains a lower bound λ and upper bound μ. Initially, all soft clauses are relaxed, and the lower and upper bounds are initialized, respectively, to -1 and to one plus the sum of the weights of the soft clauses. At each iteration, the middle value ν is computed, an AtMost constraint is added to the working formula, requiring that the sum of the weights of soft clauses should be no greater that ν, and the SAT solver is called on the resulting CNF formula. If the formula is satisfiable, then the optimum solution must be less than ν, and the upper bound is updated. If the formula is unsatisfiable, then the optimum solution must be larger than ν and the lower bound is updated. The algorithm terminates when $\lambda + 1 = \mu$.

Proposition 2 *Binary search for MaxSAT executes $\Theta(\log(\mathcal{W}))$ calls to a SAT oracle in the worst case.*

Core-Guided Binary Search Algorithms

This section develops *core-guided binary search* for MaxSAT (see Algorithm 2), a variant of binary search that guides the search with unsatisfiable cores. Similar to binary search, the new algorithm maintains two bounds, an upper bound μ and a lower bound λ. Unlike binary search, core-guided binary search does not add relaxation variables to the soft clauses before starting the main loop. The algorithm proceeds by iteratively calling a SAT solver with the current formula and with an AtMost constraint considering only the relaxation variables added so far (initially none). If the formula is unsatisfiable, it checks whether all soft clauses in the core have been relaxed. If it is the case, λ is updated, otherwise non-relaxed clauses in the core are relaxed. If the SAT solver returns satisfiable, μ is updated.

Lemma 1 *Let opt be of the optimum solution of a MaxSAT instance. During the execution of Algorithm 2, the invariant $\lambda <$ opt $\leq \mu$ holds.*

Proof. Both bounds are initialized with values that are guaranteed to satisfy the conditions of the Lemma.

Algorithm 2: Core-Guided Binary Search (BIN-C)

```
Input: $\varphi$
1  $(R, \varphi_W, \varphi_S) \leftarrow (\emptyset, \varphi, \texttt{Cls(Soft(}\varphi\texttt{))})$
2  $(\lambda, \mu, last\mathcal{A}) \leftarrow (-1, \sum_{i=1}^{m} w_i + 1, \emptyset)$
3  while $\lambda + 1 < \mu$ do
4      $\nu \leftarrow \lfloor \frac{\mu+\lambda}{2} \rfloor$
5      $\varphi^{cnf} \leftarrow \texttt{CNF(}\sum_{r_i \in R} w_i \times r_i \leq \nu\texttt{)}$
6      $(\text{st}, \varphi_C, \mathcal{A}) \leftarrow \texttt{SAT(Cls(}\varphi_W\texttt{)} \cup \varphi^{cnf}\texttt{)}$
7      if st $= true$ then
8          $(last\mathcal{A}, \mu) \leftarrow (\mathcal{A}, \sum_{i=1}^{m} w_i \times \mathcal{A}\langle r_i \rangle))$
9      else
10         if $\varphi_C \cap \varphi_S = \emptyset$ then
11             $\lambda \leftarrow \nu$
12         else
13             $(R, \varphi_W) \leftarrow \texttt{Relax(}R, \varphi_W, \varphi_C \cap \varphi_S\texttt{)}$
14         end
15     end
16 end
17 return $\texttt{Init(}last\mathcal{A}\texttt{)}$
```

The lower bound λ is only updated on iterations for which the SAT solver returns false. Consider one such iteration, for which the core returned only contains already relaxed soft clauses (otherwise λ is not updated). If *opt* $> \nu$, then λ is updated value to $\nu <$ *opt*. The case of *opt* $\leq \nu$ is impossible, because the assignment corresponding to *opt* must satisfy all the hard clauses, all the soft clauses in the computed core (since all have been relaxed) and the pseudo-Boolean constraint in line 5. Thus, the assignment for the optimal solution is able to satisfy all the clauses in the core. So the core would not be an unsatisfiable core; a contradiction.

Consider now a satisfiable iteration. Assume for the sake of contradiction that μ is updated such that $\mu <$ *opt*. Then we could consider the assignment returned by the SAT solver and extend it with assignments to new relaxation variables (one for each clause not yet relaxed). In particular, these variables can be assigned value false. Then, the sum of the weights of the relaxation variables assigned value true is less than *opt*. This is a contradiction since, by definition, the sum of weights of relaxed clauses is an upper bound on the MaxSAT solution. □

Theorem 2 *Algorithm 2 terminates in* $\Theta(m+\log(\mathcal{W}))$ *calls to a SAT oracle and returns the optimum.*

Proof. We prove that the algorithm always stops and that when it stops, $\lambda + 1 \geq \mu$. Then, by Lemma 1, when the algorithm stops, $\lambda + 1 = \mu$ and μ has the optimum solution.

The algorithm performs binary search on the range of values $\{\lambda + 1, \ldots, \mu\}$, except for unsatisfiable iterations in which the core contains soft clauses not yet relaxed. The number of soft clauses is limited to m, therefore the maximum number of iterations adding new relaxation variables is at most m. Since by the previous lemma, the bounds are guaranteed never to go over or under the optimum (depending on the bound), then binary search guarantees that the number of iterations is at most $log(\mathcal{W})$. Hence the algorithm terminates in $\Theta(m + log(\mathcal{W}))$ iterations, and due to the termination condition of line 3, $\lambda + 1 \geq \mu$. □

Algorithm 3: Core-Guided Binary Search with Disjoint Cores (BIN-C-D)

```
Input: $\varphi$
1  $(\varphi_W, \varphi_S, \mathcal{C}, last\mathcal{A}) \leftarrow (\varphi, \texttt{Cls(Soft(}\varphi\texttt{))}, \emptyset, \emptyset)$
2  repeat
3      $\forall_{C_i \in \mathcal{C}}, \quad \nu_i \leftarrow (\lambda_i + 1 = \mu_i)?\ \mu_i\ :\ \lfloor \frac{\mu_i+\lambda_i}{2} \rfloor$
4      $\varphi^{cnf} \leftarrow \bigcup_{C_i \in \mathcal{C}} \texttt{CNF(}\sum_{r_j \in R_i} w_j \times r_j \leq \nu_i\texttt{)}$
5      $(\text{st}, \varphi_C, \mathcal{A}) \leftarrow \texttt{SAT(Cls(}\varphi_W\texttt{)} \cup \varphi^{cnf}\texttt{)}$
6      if st $= \mathit{SAT}$ then
7          $last\mathcal{A} \leftarrow \mathcal{A}$
8          $\forall_{C_i \in \mathcal{C}}, \quad \mu_i \leftarrow \sum_{r_j \in R_i} w_j \times \mathcal{A}\langle r_j \rangle$
9      else
10         $sub\mathcal{C} \leftarrow \texttt{IntersectingCores(}\varphi_C, \mathcal{C}\texttt{)}$
11         if $\varphi_C \cap \varphi_S = \emptyset$ and $|sub\mathcal{C}| = 1$ then
12             $\lambda \leftarrow \nu$ // $sub\mathcal{C} = \{< R, \lambda, \nu, \mu >\}$
13         else
14             $(R, \varphi_W) \leftarrow \texttt{Relax(}\emptyset, \varphi_W, \varphi_C \cap \varphi_S\texttt{)}$
15             $(\lambda, \mu) \leftarrow (0, \sum_{r_j \in R} w_j + 1)$
16             $\forall_{C_i \in sub\mathcal{C}}, \quad R \leftarrow R \cup R_i$
17             $\forall_{C_i \in sub\mathcal{C}}, \quad (\lambda, \mu) \leftarrow (\lambda + \lambda_i, \mu + \mu_i)$
18             $\mathcal{C} \leftarrow \mathcal{C} \setminus sub\mathcal{C} \cup \{< R, \lambda, 0, \mu >\}$
19         end
20     end
21 until $\forall_{C_i \in \mathcal{C}}\ \lambda_i + 1 \geq \mu_i$
22 return $\texttt{Init(}last\mathcal{A}\texttt{)}$
```

Following (Ansótegui, Bonet, and Levy 2010), it is possible to exploit *disjoint cores*. This observation motivates a new algorithm, referred to as *core-guided binary search with disjoint cores* (See Algorithm 3). The goal is to maintain smaller lower and upper bounds for each disjoint core rather than just one global lower bound and one global upper bound. As a result, the algorithm will add several smaller AtMost constraints rather than just one global AtMost constraint. The algorithm maintains information about the previous cores in a set $\mathcal{C}$ (initially empty). Whenever a new core i is found a new entry in $\mathcal{C}$ is created, that contains the set of relaxation variables R_i in the core (after relaxing required soft clauses), a lower bound λ_i, an upper bound μ_i, and the current middle value ν_i, i.e. $C_i =< R_i, \lambda_i, \nu_i, \mu_i >$. The algorithm iterates while there exists a C_i for which $\lambda_i + 1 < \mu_i$. Before calling the SAT solver, for each $C_i \in \mathcal{C}$, the middle value ν_i is computed with the current bounds and an AtMost constraint is added to the working formula. If the SAT solver returns false, then $sub\mathcal{C}$ is computed with $IntersectingCores()$, and contains every C_i in $\mathcal{C}$ that intersects the current core (i.e. $sub\mathcal{C} \subseteq \mathcal{C}$). If no soft clause needs to be relaxed and $|sub\mathcal{C}| = 1$, then $sub\mathcal{C} = \{< R, \lambda, \nu, \mu >\}$ and λ is updated to ν. Otherwise, all the required soft clauses are relaxed and an entry for the new core is added to $\mathcal{C}$, which aggregates the information of the previous cores in $sub\mathcal{C}$. Also, each $C_i \in sub\mathcal{C}$ is removed from $\mathcal{C}$. If the SAT solver returns true, the algorithm iterates over each $C_i \in \mathcal{C}$ and its upper bound μ_i is updated according to the satisfying assignment $\mathcal{A}$. The proof of correctness of Algorithm 3 is omitted due to space limitations. Core-guided binary search algorithms implement two heuristics to improve the initial λ and μ.

Improving the upper bound μ. All soft clauses are relaxed and the resulting formula is given to a SAT solver which gives a satisfying assignment $\mathcal{A}$. Then, μ can be updated to $\sum_{i=1}^{m} w_i \times \mathcal{A}\langle r_i \rangle$. Afterwards, all relaxation variables are removed. This technique is adapted from (Giunchiglia and Maratea 2006).

Improving the lower bound λ. In a preprocessing step, a SAT solver is iteratively called while the formula is unsatisfiable. For each computed unsatisfiable core, the soft clauses in the core are relaxed, the minimum weight among the clauses is recorded, and the soft clauses are temporary removed from the formula. Upon termination, λ is assigned the sum of the recorded weights, and the removed clauses are added back to the formula. Also, already relaxed soft clauses are kept.

Experimental Evaluation

This section evaluates the algorithms developed in the previous section. Three weighted (partial) MaxSAT solvers have been implemented in C++: BIN denotes the original binary search algorithm (Fu and Malik 2006), BIN-C denotes core-guided binary search, and BIN-C-D denotes core-guided binary search with disjoint cores. These algorithms use PICOSAT (Biere 2008) as the underlying SAT solver. *Cardinality networks* (Asín et al. 2011) are used for encoding cardinality constraints, and *BDDs* (Eén and Sörensson 2006) are used for encoding pseudo-Boolean constraints. All unweighted MaxSAT industrial and unweighted partial MaxSAT industrial instances from the 2009 MaxSAT Evaluation were considered. Also, all unweighted and weighted partial MaxSAT crafted instances and weighted partial MaxSAT industrial instances from the 2010 MaxSAT Evaluation were considered. Experiments were conducted on a HPC cluster with 50 nodes, each node is a CPU Xeon E5450 3GHz, 32GB RAM and Linux. For each run, the time limit was 1200 seconds and the memory limit was 4GB.

The performance of the new algorithms is compared with a representative set of state-of-the-art MaxSAT solvers. With one exception, all these solvers are based on iterative SAT solving including WMSU1 (Manquinho, Marques-Silva, and Planes 2009), WPM1 (Ansótegui, Bonet, and Levy 2009), PM2 (Ansótegui, Bonet, and Levy 2009), WPM2 (Ansótegui, Bonet, and Levy 2010), and SAT4J (Berre and Parrain 2010). The exception is MINIM (Heras, Larrosa, and Oliveras 2008) which is an efficient *branch and bound* solver.

Figure 1 (Top) shows a cactus plot of the unweighted MaxSAT instances solved by each solver within the time limit. BIN-C-D outperforms all other solvers, whereas PM2 and BIN-C perform similarly. Figure 1 (Bottom) shows a cactus plot of the weighted MaxSAT instances solved by each solver within the time limit. Again, BIN-C-D solves the largest number of instances, followed by MINIM and SAT4J. Moreover, plain *binary search* performs slightly worse than WPM1 and better than WPM2, but all are clearly outperformed by the best performing solvers. SAT4J and MINIM perform reasonably well on the weighted instances. MINIM is especially good on 3 of 7 crafted sets

Sets	$\#Ins.$	WMSU1	WPM1	BIN	MINIM	SAT4J	(W)PM2	QMAXS	BIN-C	BIN-C-D
ms-ind	121	**104**	102	7	0	34	98	-	82	81
pms-ind	965	568	602	780	666	749	797	836	832	**864**
pms-cft	385	62	72	202	**304**	243	235	287	243	250
wpms-cft	357	57	77	130	**236**	180	55	-	131	138
wpms-ind	132	**112**	62	6	0	31	63	-	43	110
Total	1960	903	915	1125	1206	1237	1248	1123	1331	**1443**

Table 1: Number of solved instances

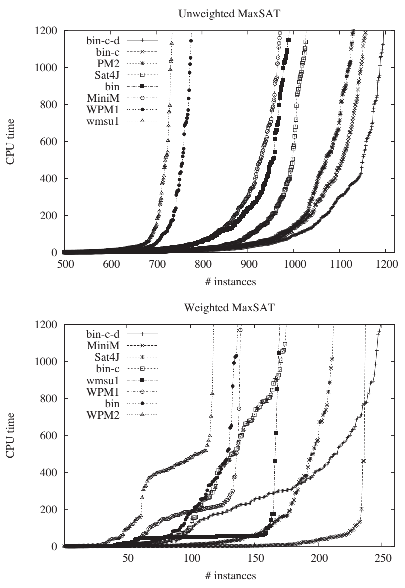

Figure 1: Run time distributions

considered, but it is unable to solve any instance of the 2 industrial sets considered.

Table 1 shows the total number of instances solved by each solver, and includes a breakdown for unweighted and weighted, as well as for crafted and industrial benchmarks. The first column indicates the set of benchmarks, where *ms* denotes MaxSAT, *w* denotes *weighted* and *p* denotes *partial*, and the category can either be *crafted* (*cft*) or industrial (*ind*). The second column shows the total number of instances for each set. The remaining columns show the number of instances solved by each solver within the time and memory limits. Observe that QMAXSAT (QMAXS) (Koshimura and Zhang) has also been added to this comparison. QMAXSAT was the best solver in the 2010 MaxSAT Evaluation in the unweighted partial industrial track.

BIN-C-D solves the largest number of instances, and is followed by BIN-C and by SAT4J and PM2 + WPM2 (noted (W)PM2). For unweighted partial industrial instances, BIN-C-D solves again the largest number of instances. For weighted partial industrial instances, BIN-C-D ranks second in terms of solved instances. For the crafted categories, the branch and bound solver MINIM solves the largest number of instances whereas BIN-C-D ranks third. For unweighted plain industrial instances WPM1 and WMSU1 solve the largest number of instances, whereas BIN-C-D performs quite well compared to the other solvers. The results confirm that BIN-C-D is one of the most robust MaxSAT solvers based on iterative SAT calls, and it is competitive with branch and bound solvers on crafted instances.

Conclusions

This paper introduces *core-guided binary search* algorithms for MaxSAT, that extend the standard *binary search* algorithm (Fu and Malik 2006; Cimatti et al. 2010) by guiding the search process with information provided by unsatisfiable cores. An optimization to the basic core-guided binary search algorithm is also proposed, which exploits disjoint cores. The experimental results show that core-guided binary search (with disjoint cores) is one of the most robust MaxSAT algorithms.

References

Ansótegui, C.; Bonet, M. L.; and Levy, J. 2009. Solving (weighted) partial MaxSAT through satisfiability testing. In *SAT*, 427–440.

Ansótegui, C.; Bonet, M. L.; and Levy, J. 2010. A new algorithm for weighted partial MaxSAT. In *AAAI*.

Argelich, J.; Li, C. M.; Manyà, F.; and Planes, J. MaxSAT evaluations. http://www.maxsat.udl.cat/.

Asín, R.; Nieuwenhuis, R.; Oliveras, A.; and R.-Carbonell, E. 2011. Cardinality networks: a theoretical and empirical study. *Constraints* 16(2):195–221.

Berre, D. L., and Parrain, A. 2010. The Sat4j library, release 2.2. *JSAT* 7:59–64.

Biere, A.; Heule, M.; van Maaren, H.; and Walsh, T., eds. 2009. *Handbook of Satisfiability*.

Biere, A. 2008. Picosat essentials. *JSAT* 4(2-4):75–97.

Cimatti, A.; Franzén, A.; Griggio, A.; Sebastiani, R.; and Stenico, C. 2010. Satisfiability modulo the theory of costs: Foundations and applications. In *TACAS*, 99–113.

Davies, J.; Cho, J.; and Bacchus, F. 2010. Using learnt clauses in MAXSAT. In *CP*, 176–190.

Eén, N., and Sörensson, N. 2006. Translating pseudo-boolean constraints into sat. *JSAT* 2:1–26.

Fu, Z., and Malik, S. 2006. On solving the partial MAX-SAT problem. In *SAT*, 252–265.

Giunchiglia, E., and Maratea, M. 2006. OPTSAT: A tool for solving SAT related optimization problems. In *JELIA*, 485–489.

Gottlob, G. 1995. NP trees and Carnap's modal logic. *J. ACM* 42(2):421–457.

Hansen, P., and Jaumard, B. 1990. Algorithms for the maximum satisfiability problem. *Computing* 44(4):279–303.

Heras, F.; Larrosa, J.; and Oliveras, A. 2008. MiniMaxSat: An efficient weighted Max-SAT solver. *JAIR* 31:1–32.

Koshimura, M., and Zhang, T. QMaxSat solver description http://www.maxsat.udl.cat/10/solvers/QMaxSat.pdf.

Manquinho, V.; Marques-Silva, J.; and Planes, J. 2009. Algorithms for weighted Boolean optimization. In *SAT*, 495–508.

Marques-Silva, J., and Planes, J. 2008. Algorithms for maximum satisfiability using unsatisfiable cores. In *DATE*, 408–413.

Zhang, L., and Malik, S. 2003. Validating sat solvers using an independent resolution-based checker: Practical implementations and other applications. In *DATE*, 10880–10885.