Improvements to Core-Guided Binary Search for MaxSAT

Antonio Morgado[1], Federico Heras[1], and Joao Marques-Silva[1,2]

[1] CASL, University College Dublin, Ireland
[2] IST/INESC-ID, Lisbon, Portugal

Abstract. *Maximum Satisfiability* (MaxSAT) and its weighted variants are well-known optimization formulations of *Boolean Satisfiability* (*SAT*). Motivated by practical applications, recent years have seen the development of *core-guided algorithms* for MaxSAT. Among these, *core-guided binary search with disjoint cores* (BCD) represents a recent robust solution. This paper identifies a number of inefficiencies in the original BCD algorithm, related with the computation of *lower* and *upper bounds* during the execution of the algorithm, and develops solutions for them. In addition, the paper proposes two additional novel techniques, which can be implemented on top of core-guided MaxSAT algorithms that maintain both lower and upper bounds. Experimental results, obtained on representative problem instances, indicate that the proposed optimizations yield significant performance gains, and allow solving more problem instances.

1 Introduction

Maximum Satisfiability (MaxSAT) and its variants, namely (*Weighted*) (*Partial*) MaxSAT, find a growing number of practical applications. Concrete recent examples include hardware design debugging [19] and fault localization in C code [9]. In addition, reference applications that use *Pseudo-Boolean Optimization* (*PBO*) can be cast as MaxSAT [7, 4]. Another major application of MaxSAT is in algorithms for *Minimal Unsatisfiable Subset* (*MUS*) enumeration [13]. Indeed, the most efficient MUS enumeration algorithms build on MaxSAT algorithms for computing all *Maximal Satisfiable Subsets* (*MSSes*) and, from these, MUSes can be enumerated using a standard *hitting set* approach [13, 18]. The variety of relevant applications of MUS enumeration (e.g. [13, 1]), further highlights the practical significance of efficient MaxSAT algorithms.

Motivated by the practical applications of MaxSAT, recent years have witnessed a large number of MaxSAT algorithms being proposed. MaxSAT approaches for solving practical problem instances differ significantly from early work on MaxSAT [12, 7]. These approaches are characterized by guiding the search with *unsatisfiable subformulas* [20] and are referred to as *core-guided* MaxSAT algorithms [6, 16, 14, 2, 3]. Recent work has proposed two core-guided versions of binary search for MaxSAT [8]. These include a basic version (BC) and a version that maintains a set of disjoint unsatisfiable

* This work is partially supported by SFI grant BEACON (09/IN.1/I2618), and by FCT grants ATTEST (CMU-PT/ELE/0009/2009) and POLARIS (PTDC/EIA-CCO/123051/2010).

cores (BCD). The BCD algorithm was shown to be one of the most efficient on a comprehensive set of problem instances from recent MaxSAT evaluations. Nevertheless, recent detailed analysis of BCD revealed a number of possible inefficiencies, that result from relaxed and conservative maintenance of lower and upper bounds.

This paper addresses the inefficiencies in the original BCD algorithm, and develops a number of key optimizations. These optimizations can be categorized as: (i) modifications to how the upper bound of each disjoint core is initialized, updated, and an associated maintenance of a *global upper bound*; (ii) modifications on how the lower bounds are updated when disjoint cores are *merged*; and (iii) techniques for refining the lower bound so that it reflects a feasible sum of weights. The previous optimizations are implemented in a new algorithm, BCD2, that often requires fewer SAT solver calls than BCD. The paper also proves the correctness of BCD2 and shows that BCD2 is significantly more efficient than BCD on a comprehensive set of benchmarks from recent MaxSAT Evaluations.

In addition, the paper proposes two novel techniques, that can be implemented on top of any core-guided MaxSAT algorithm that maintains both lower and upper bounds, namely the *hardening rule* and *biased search*. The *hardening rule*, which has been extensively used in *branch and bound* algorithms [5, 12, 11, 7], is adapted for core-guided binary search algorithms. As a result, many soft clauses can be declared *hard*. Binary search algorithms always compute the middle value between a lower bound and an upper bound. The *biased search* technique allows biasing the search with the outcomes of the previous iterations and compute a value between the lower and upper bounds, though *not necessarily* the middle one.

The remainder of the paper is organized as follows. Section 2 introduces the MaxSAT problem and core-guided binary search MaxSAT algorithms. Section 3 details the inefficiencies of BCD, and develops a new improved algorithm for core-guided binary search with disjoint cores (BCD2). Section 4 presents the hardening rule and biased search techniques for core-guided MaxSAT. Section 5 evaluates the performance of the algorithms with the proposed techniques. Section 6 presents some concluding remarks.

2 Preliminaries

Let $X = \{x_1, x_2, \ldots, x_n\}$ be a set of Boolean variables. A *literal* l is either a variable x_i or its negation $\bar{x}_i$. A *clause* c is a disjunction of literals. A clause may also be regarded as a set of literals. An *assignment* $\mathcal{A}$ is a mapping $\mathcal{A} : X \rightarrow \{0, 1\}$ which satisfies (unsatisfies) a Boolean variable x if $\mathcal{A}(x) = 1$ ($\mathcal{A}(x) = 0$). Assignments can be extended in a natural way for literals (l) and clauses (c):

$$\mathcal{A}(l) = \begin{cases} \mathcal{A}(x), & \text{if } l = x \\ 1 - \mathcal{A}(x), & \text{if } l = \neg x \end{cases} \qquad \mathcal{A}(c) = \max\{\mathcal{A}(l) \mid l \in c\}$$

Assignments can also be regarded as set of literals, in which case the assignment $\mathcal{A}$ satisfies (unsatisfies) a variable x if $x \in \mathcal{A}$ ($\bar{x} \in \mathcal{A}$). A *complete assignment* contains a literal for each variable, otherwise is a *partial assignment*. A CNF formula φ is a set of clauses. A *model* is a complete assignment that satisfies all the clauses in a CNF formula φ. The *Propositional Satisfiability Problem* (*SAT*) is the problem of deciding

whether there exists a model for a given formula. Given an unsatisfiable formula φ, a subset of clauses φ_C (i.e. $\varphi_C \subseteq \varphi$) 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 for unsatisfiable formulas [20]. 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 $\top$. Non-mandatory clauses are also called *soft* clauses. A *weighted* formula in *conjunctive normal form* (WCNF) φ is a set of weighted clauses. For MaxSAT, 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, *Weighted Partial MaxSAT* is the problem of finding a model of minimum cost.

Core-guided binary search algorithms for MaxSAT Several MaxSAT solvers in the literature are based on iteratively calling a SAT solver and refining a *lower bound*, an *upper bound* or both [6, 2, 3, 16, 14, 8, 10]. *Core-guided MaxSAT algorithms* are those that additionally take advantage of unsatisfiable cores computed at each unsatisfiable iteration to guide the search [6, 2, 3, 16, 14, 8], (some of which use binary search [8]).

Auxiliary notation is introduced to describe core-guided binary search MaxSAT algorithms. The remainder of the paper assumes a WCNF formula φ with m soft clauses. Core-guided algorithms use *relaxation variables*, which are fresh Boolean variables. The algorithms described add *at most* one relaxation variable to each soft clause. The process of adding a relaxation variable to a clause, is referred to as *relaxing* the clause. Relaxation variables are maintained in a set R, and it is assumed that relaxation variable r_i is associated to the soft clause c_i with weight w_i, $1 \leq i \leq m$. In order to add relaxation variables to soft clauses, the algorithms use the function $Relax(R, \varphi, \psi)$ which receives a set of existing relaxation variables R, a WCNF formula φ and a set of soft clauses ψ and returns the pair (R_o, φ_o). φ_o corresponds to φ whose soft clauses included in ψ have been augmented with fresh relaxation variables. R_o corresponds to R augmented with the relaxation variables added in φ_o. Given the set of relaxation variables in R, the algorithms add *cardinality / pseudo-Boolean constraints* [4] and translate them to hard clauses. Such constraints usually state that the sum of the weights of the relaxed clauses is less than or equal to a specific value K (`AtMostK` with $\sum_{i=1}^{m} w_i r_i \leq K$). The algorithms use the following functions:

- *Soft*(φ) returns the set of all *soft* clauses in φ.
- *SATSolver*(φ) makes a call to the SAT solver and returns a triple $(st, \varphi_C, \mathcal{A})$, where st is the status of the formula φ, that is whether φ is satisfiable (SAT or UNSAT). If st =UNSAT, then φ_C contains an unsatisfiable core of φ, and if st =SAT, then $\mathcal{A}$ corresponds to a complete satisfying assignment of φ. Throughout the paper, by abuse of notation, st is referred to as the outcome of the SAT solver.
- *CNF*(c) returns a set of clauses that encode the constraint c into CNF.

Core-guided binary search (`BC`) and its extension *with disjoint cores* (`BCD`) [8] compute both a lower bound and an upper bound and have been shown to be very robust approaches for MaxSAT solving (in terms of number of solved instances). In what follows, the most sophisticated version (`BCD`) is briefly overviewed.

Algorithm 1: BCD

```
   Input: φ
1  (φ_W, φ_S, C, last_A) ← (φ, Soft(φ), ∅, ∅)        // C - set of disj. core's information
2  repeat
3      ∀_{C_i∈C}, ν_i ← (λ_i + 1 = μ_i) ? μ_i : ⌊(μ_i+λ_i)/2⌋
4      (st, φ_C, A) ← SATSolver(φ_W ∪ ⋃_{C_i∈C} CNF(Σ_{r_j∈R_i} w_j · r_j ≤ ν_i))
5      if st = SAT then
6          last_A ← A
7          ∀_{C_i∈C}, μ_i ← Σ_{r_j∈R_i} w_j · A(r_j)
8      else
9          subC ← Intersect(φ_C, C) // subC - set of disj. cores that intersect φ_C
10         if φ_C ∩ φ_S = ∅ and |subC| = |{< R_s, λ_s, ν_s, μ_s >}| = 1 then
11             λ_s ← ν_s
12         else
13             (R_s, φ_W) ← Relax(∅, φ_W, φ_C ∩ φ_S)
14             (λ_s, μ_s) ← (0, Σ_{r_j∈R_s} w_j + 1)
15             ∀_{C_i∈subC}, (R_s, λ_s, μ_s) ← (R_s ∪ R_i, λ_s + λ_i, μ_s + μ_i)
16             C ← C \ subC ∪ {< R_s, λ_s, 0, μ_s >}
17         end
18     end
19 until ∀_{C_i∈C} λ_i + 1 ≥ μ_i
20 return last_A
```

The pseudo-code of BCD is shown in Algorithm 1. BCD maintains information about disjoint cores in a set $\mathcal{C}$ (initially empty). Whenever a new core is found, a new entry C_s in $\mathcal{C}$ is created, that contains the set of relaxation variables R_s in the core (after relaxing required soft clauses), a lower bound λ_s, an upper bound μ_s, and the current middle value ν_s, i.e. $C_s =< R_s, \lambda_s, \nu_s, \mu_s >$. The algorithm iterates while there exists a C_i for which $\lambda_i + 1 < \mu_i$ (line 19). 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 AtMostK constraint is added to the working formula (lines 3-4). If the SAT solver returns SAT, the algorithm iterates over each core $C_i \in \mathcal{C}$ and its upper bound μ_i is updated according to the satisfying assignment $\mathcal{A}$ (lines 6-7). If the SAT solver returns UNSAT, then the set $subC$ is computed which contains every C_i in $\mathcal{C}$ that intersects the current core (i.e. $subC \subseteq \mathcal{C}$, line 9). If no soft clause needs to be relaxed and $|subC| = 1$, then $subC = \{< R_s, \lambda_s, \nu_s, \mu_s >\}$ and λ_s is updated to ν_s (line 11). Otherwise, all the required soft clauses are relaxed, an entry for the new core C_s is added to $\mathcal{C}$, which aggregates the information of the previous cores in $subC$, and each $C_i \in subC$ is removed from $\mathcal{C}$ (lines 13-16).

A concept similar to disjoint cores (namely covers) is used by the core-guided (non binary search) algorithm WPM2 [3] coupled with the constraints to add in each iteration.

3 Improving BCD

Detailed analysis of BCD has revealed two key inefficiencies, both related with how the lower and upper bounds are computed and updated. The first observation is that BCD does *not* maintain a *global upper bound*. When the SAT solver outcome is satisfiable (SAT), each μ_i value is updated for each disjoint core $C_i \in \mathcal{C}$, with an overall sum

Algorithm 2: BCD2

Input: φ

1 $(\varphi_W, \varphi_S) \leftarrow (\varphi,$Soft($\varphi$))
2 $\forall_{c_j \in \varphi_S},\ \sigma_j \leftarrow w_j$
3 $(\mathcal{C}, \mathcal{A}_\mu, \mu) \leftarrow (\emptyset, \emptyset, 1 + \sum_{c_j \in \varphi_S} \sigma_j)$
4 **repeat**
5 $\quad \forall_{C_i \in \mathcal{C}},\ \nu_i \leftarrow \lfloor \frac{\lambda_i + \epsilon_i}{2} \rfloor$
6 $\quad$ (st, $\varphi_C, \mathcal{A}$) $\leftarrow$ SATSolver($\varphi_W \cup \bigcup_{C_i \in \mathcal{C}}$CNF($\sum_{r_j \in R_i} w_j \cdot r_j \leq \nu_i$))
7 $\quad$ **if** st $=$ *SAT* **then**
8 $\quad\quad \forall_{c_j \in \varphi_S},\ \sigma_j \leftarrow 0$
9 $\quad\quad \forall_{C_i \in \mathcal{C}} \forall_{r_j \in R_i},\ \sigma_j \leftarrow w_j \cdot (1 - \mathcal{A}(c_j \setminus \{r_j\}))$ // $c_j \in \varphi_W$ and $r_j \in c_j$
10 $\quad\quad \forall_{C_i \in \mathcal{C}},\ \epsilon_i \leftarrow \sum_{r_j \in R_i} \sigma_j$
11 $\quad\quad (\mu, \mathcal{A}_\mu) \leftarrow (\sum_{C_i \in \mathcal{C}} \sum_{r_j \in R_i} \sigma_j, \mathcal{A})$
12 $\quad$ **else**
13 $\quad\quad subC \leftarrow$ Intersect($\varphi_C, \mathcal{C}$)
14 $\quad\quad$ **if** $\varphi_C \cap \varphi_S = \emptyset$ **and** $|subC| = |\{< R_s, \lambda_s, \nu_s, \epsilon_s >\}| = 1$ **then**
15 $\quad\quad\quad \lambda_s \leftarrow$ Refine($\{w_j\}_{r_j \in R_S},\ \nu_s$)
16 $\quad\quad$ **else**
17 $\quad\quad\quad (R_s, \varphi_W) \leftarrow$ Relax($\bigcup_{C_i \in subC} R_i,\ \varphi_W,\ \varphi_C \cap \varphi_S$)
18 $\quad\quad\quad \Delta \leftarrow \min\{1 + \min\{\nu_i - \lambda_i \mid C_i \in subC\}, \min\{w_j \mid r_j$ is a new relax. var.$\}\}$
19 $\quad\quad\quad \lambda_s \leftarrow$Refine($\{w_j\}_{r_j \in R_s},\ \sum_{C_i \in subC} \lambda_i + \Delta - 1$)
20 $\quad\quad\quad \epsilon_s \leftarrow ((\mathcal{A}_\mu = \emptyset)\ ?\ 1 : 0) + \sum_{r_j \in R_s} \sigma_j$
21 $\quad\quad\quad \mathcal{C} \leftarrow \mathcal{C} \setminus subC \cup \{< R_s, \lambda_s, 0, \epsilon_s >\}$
22 $\quad\quad$ **end**
23 $\quad$ **end**
24 **until** $\sum_{C_i \in \mathcal{C}} \lambda_i = \sum_{C_i \in \mathcal{C}} \epsilon_i = \mu$
25 **return** $\mathcal{A}_\mu$

given by $K_1 = \sum_{C_i \in \mathcal{C}} \mu_i$. However, after merging disjoint cores, if the SAT solver outcome is again SAT, it can happen that $K_2 = \sum_{C_i \in \mathcal{C}} \mu_i > K_1$. Although this issue does not affect the correctness of the algorithm, it can result in a number of iterations higher than needed to compute the optimum. The second observation is that the lower bound updates for each disjoint core are conservative. A more careful analysis of how the algorithm works allows devising significantly more aggressive lower bound updates. Again, the main consequence of using conservative lower bounds is that this can result in a number of iterations higher than needed to compute the optimum.

This section presents the new algorithm BCD2. Although similar to BCD, BCD2 proposes key optimizations that address the inefficiencies described above. As the experimental results demonstrate, these optimizations lead to significant performance gains, that can be explained by a reduced number of iterations.

The pseudo-code of BCD2 is shown in Algorithm 2. The organization of BCD2 is similar to the organization of BCD but with important differences. The first difference between BCD and BCD2 is the way the algorithms use the information of the upper bounds. As stated before, BCD does not maintain a global upper bound, and as such, whenever an upper bound is needed, then the worst case scenario is used. Concretely in line 14 of BCD, the upper bound is updated with the weights of the new relaxed clauses.

On the other hand, BCD2 keeps a *global upper bound* μ and its corresponding assignment $\mathcal{A}_\mu$. More importantly it maintains the cost of each soft clause for the current global upper bound. In order to achieve this, BCD2 associates with each soft clause j

a variable σ_j that represents the contribution of the clause to the overall cost of the global upper bound. σ_j can take as value either 0 or w_j (the weight of the soft clause j) depending on whether $\mathcal{A}_\mu$ unsatisfies the clause or not. In contrast to BCD, the contribution of soft clauses is with respect to the original variables. As such in line 9 of BCD2, the update of σ_j considers the satisfiability of the clause c_j without the relaxation variable ($w_j \cdot (1 - \mathcal{A}(c_j \setminus \{r_j\}))$), rather than the satisfiability of the relaxation variable ($w_j \cdot \mathcal{A}(r_j)$) as in BCD (line 7). Considering the satisfiability of the original soft clause instead of the associated relaxation variable, has the benefit of tightening the upper bound on assignments that satisfy the clause without the relaxation variable but still satisfy the relaxation variable.

Unlike BCD, BCD2 does not maintain upper bounds in the disjoint cores. Instead, each disjoint core C_i maintains an *estimate* ϵ_i that represents the contribution of the disjoint core to the cost of the global upper bound. Each ϵ_i takes the role of the upper bounds μ_i in BCD, with updates that respect the last satisfying assignment. The difference is that in BCD2, the updates of the estimates, done in lines 10 and 20, include the contribution of the soft clauses to the global upper bound (stored in the σ_j variables).

The use of σ_j variables in the computation of estimates ϵ_i, allow BCD2 to use the information of the current upper bound assignment for a tighter bound, specifically, when merging cores with soft clauses not previously relaxed. The contribution of the newly relaxed clauses in the update of ϵ_i in line 20, is dependent on a previous discovery of a satisfying assignment. Before the first satisfying assignment is found, the contribution is the same as in BCD, that is the weight of the soft clause ($\sigma_j = w_j$, initialization of σ_j in line 2 of BCD2), whereas after the first satisfying assignment, newly relaxed clauses are satisfied by $\mathcal{A}_\mu$ (thus $\sigma_j = 0$ from line 8) and its contribution to ϵ_i is 0.

The reason why the ϵ_i variables are called estimates is that, unlike the upper bound μ_i of BCD, the ϵ_i variables are allowed to have a value lower than the cost of the optimum model restricted to the clauses associated to the disjoint core. In such situations ϵ_i is said to be *optimistic* and represents a local optimum of a MaxSAT model. BCD2 can shift ϵ_i away from the local optimum by merging with different cores as needed.

Example 1. Consider an execution of the algorithm with the current working formula $\varphi_W = \varphi^S \cup \varphi^H$, where $\varphi^S = \{(x_1 \vee r_1, 5), (x_2 \vee r_2, 10), (x_3 \vee r_3, 30), (x_4 \vee r_4, 10)\}$ and $\varphi^H = \{(\neg x_1 \vee \neg x_2), (\neg x_2 \vee \neg x_3), (\neg x_3 \vee \neg x_4)\}$. Consider the upper bound assignment $\mathcal{A}_\mu = \{x_1 = x_3 = r_2 = r_4 = 0, x_2 = x_4 = r_1 = r_3 = 1\}$ with a cost of 35, and two disjoint cores $C_1 =< R_1 = \{r_1, r_2\}, \lambda_1 = 5, \nu_1 = 5, \epsilon_1 = 5 >$, $C_2 =< \{r_3, r_4\}, 10, 20, 30 >$.

The optimum cost of φ is 20. Considering the optimum model, the contribution of the clauses associated to C_1 is 10 which is lower than ϵ_1, thus ϵ_1 is *optimistic*. The next core returned by the SAT solver merges C_1 and C_2 into a new disjoint core C_3 with $\epsilon_3 = 35$.

Another improvement in BCD2 is the way the lower bound is computed when merging cores. In this case, BCD2 proposes a stronger update in lines 18 and 19, which corresponds to the expression in Equation 1.

$$\sum_{C_i \in subC} \lambda_i \; + \; \min\left\{1 + \min\{\nu_i - \lambda_i | C_i \in subC\},\; \min\{w_j | r_j \text{ new relax. var.}\}\right\} \quad (1)$$

The update of the lower bound of the merged disjoint cores in Equation 1, is obtained by summing all the previous lower bounds, as is done by BCD in line 15, but also by adding an increment Δ (line 18 in BCD2). The rationale for the increment Δ comes as a justification for obtaining the current core. At this point of the algorithm, there are three possible reasons why the current core was obtained: (i) one or more of the newly relaxed soft clauses has a non-zero contribution to the cost of the final optimum model; (ii) one or more of the disjoint cores is unable to satisfy the corresponding constraint $\sum_{r_j \in R_i} w_j \cdot r_j \leq \nu_i$; (iii) a combination of the previous two.

Suppose that the reason for obtaining the current core is as stated in (i). Since the number of newly relaxed soft clauses with a non-zero contribution is unknown, then Δ corresponds to the weight of the relaxation variable with the lowest weight, that is, in this case $\Delta = \min\{w_j | r_j \text{ new relax. var.}\}$.

Consider now that the reason for obtaining the current core is as stated in (ii). Then at least one of the disjoint cores merged, requires its lower bound to be increased from λ_i to $\nu_i + 1$ (an increment of $1 + \nu_i - \lambda_i$). Since it is unknown which disjoint cores require to be increased, then in Δ is only considered the disjoint core with the lowest increment, that is $\Delta = 1 + \min\{\nu_i - \lambda_i | C_i \in subC\}$.

Finally, in the case of reason (iii), the increment Δ can be obtained by summing the increments corresponding to the previous reasons. Nevertheless, it is unknown exactly which of the three reasons explains the current core, then BCD2 uses as increment the minimum of the previous increments, thus obtaining the expression in Equation 1.

An additional difference between the algorithms is the use of the *Refine*() function to further improve the update of the lower bound in lines 15 and 19 of BCD2. The result of *Refine*$(\{w_j\}, \lambda)$ is the smallest integer greater than λ that can be obtained by summing a subset of the input weights $\{w_j\}$. In BCD2, *Refine*$(\{w_j\}, \lambda)$ starts by searching if all weights are equal, in which case the minimum sum of weights greater than λ is returned, otherwise, *subsetsum*$(\{w_j\}, \lambda)$ is computed as used by WPM2 [3].

Finally, the last difference between BCD and BCD2 is the stopping criteria. Given the new bounds, BCD2 stops when the sum the lower bounds of each disjoint core is the same as the global upper bound.

3.1 Proof of correctness

This subsection proves the correctness of the BCD2 algorithm. First, the correctness of the updates of the lower bound are proven, followed by a proof of the invariant of BCD2. The section ends with a proof of the correctness of BCD2.

Proposition 1. *Consider a disjoint core C_s in the conditions of the update of λ_s in line 15. There is no MaxSAT model for which the clauses associated to C_s contribute to the cost with a value smaller than $Refine(\{w_j\}_{r_j \in R_s}, \nu_s)$.*

Proof. Consider an iteration where the SAT solver returned a core which only contains clauses previously relaxed, and that these clauses belong to the same disjoint core C_s.

For the purpose of contradiction, assume there is a model for which the clauses of C_s contribute with a cost lower than $\nu_s + 1$. Then the assignment of the model can be augmented with assignments to the relaxation variables, such that, each relaxation

variable $r_i \in R_s$ is assigned true iff the assignment of the model does not satisfy the corresponding clause c_i. The augmented assignment is able to satisfy the constraint $\sum_{r_i \in R_s} w_i \cdot r_i \leq \nu_s$, all the hard clauses (because it is a MaxSAT model), and all the soft clauses (due to the assignments to the relaxation variables). Then the core returned by the SAT solver is not an unsatisfiable subformula, which is a contradiction, thus the update $\lambda_s \leftarrow \nu_s + 1$ is correct.

Since there is no model with $\lambda_s \leq \nu_s$, then the next value to consider for $\sum_{r_i \in R_s} w_i \cdot r_i$ is the minimum sum of subsets of $\{w_j\}_{r_j \in R_s}$ that is greater than ν_s. This corresponds to the value returned by $Refine(\{w_j\}_{r_j \in R_s}, \nu_s)$. Thus the update on line 15 is correct.

Proposition 2. *Consider the subset of disjoint cores $subC = \{C_1, \ldots, C_m\}$ and a new set of relaxation variables and Δ as in the conditions of line 19, then there is no MaxSAT model for which the clauses associated to the resulting disjoint core C_s contribute to the cost with a value smaller than $Refine(\{w_j\}_{r_j \in R_s}, \sum_{C_i \in subC} \lambda_i + \Delta - 1)$.*

Proof. There is no model with cost lower than $\sum_{C_i \in subC} \lambda_i$ because at this point of the algorithm, each disjoint core $C_i \in subC$ has been proved to have a lower bound of at least λ_i. Then the union of disjoint sets of the clauses of each C_i together with the clauses that just got relaxed have a cost of at least $\sum_{C_i} \lambda_i$ in any MaxSAT model.

Consider by contradiction, that there is a model, for which the clauses associated to the resulting disjoint core C_s, have a cost $costSol \in [\sum_{C_i \in subC} \lambda_i,\ \sum_{C_i \in subC} \lambda_i + \Delta[$. Two cases are considered.

1) In the first case, suppose that the model assigns to true at least one of the new relaxation variables (of the soft clauses that just got relaxed), and that the cost associated to that relaxation variable is w_{newRV}. Then,

$$w_{newRV} \geq \min\{w_j | r_j \text{ is a new relax. var.}\} \geq \Delta$$

Consider the contribution of all the clauses without the newly relaxed clause:

$$costSol - w_{newRV} \leq costSol - \Delta$$

but by contradiction $costSol < \sum_{C_i \in subC} \lambda_i + \Delta$ and then

$$costSol - w_{newRV} \leq costSol - \Delta < \sum_{C_i \in subC} \lambda_i$$

which means that the contribution of the remaining clauses is lower than $\sum_{C_i \in subC} \lambda_i$; but this is a contradiction (previously the cost of the union of clauses of $C_i \in subC$ was proven to be at least $\sum_{C_i \in subC} \lambda_i$).

2) In the second case suppose that the model assigns all newly relaxed clauses to false, then the contribution of the newly relaxed clauses is 0. Since by contradiction

$$costSol < \sum_{C_i \in subC} \lambda_i + \Delta \leq \sum_{C_i \in subC} \lambda_i + 1 + \min\{\nu_i - \lambda_i | C_i \in subC\}$$

then

$$costSol - \sum_{C_i \in subC \setminus \{C_1\}} \lambda_i \leq \lambda_1 + \min\{\nu_i - \lambda_i | C_i \in subC\} \leq \nu_1$$

Let $costSol\langle C_i\rangle$ be the contribution of the clauses of C_i to the cost of the model. Previously, was proven that $costSol\langle C_i\rangle \geq \lambda_i$, then

$$costSol\langle C_1\rangle = costSol - \sum_{C_i \in subC \setminus \{C_1\}} costSol\langle C_i\rangle \leq costSol - \sum_{C_i \in subC \setminus \{C_1\}} \lambda_i \leq \nu_1$$

By analogy, for each of the disjoint cores merged $C_i \in subC$, $costSol\langle C_i\rangle \leq \nu_i$ Then the model is able to satisfy all the new soft clauses and the constraints $\sum_{r_j \in R_i} w_j \cdot r_j \leq \nu_i$. Since the model is a MaxSAT model, then it is also able satisfy all the hard clauses, meaning that the model is able to satisfy all the clauses in the core; but this is again a contradiction.

Since there is no model with $\lambda_s < \sum_{C_i \in subC} \lambda_i + \Delta$, then the next value to consider for $\sum_{r_i \in R_s} w_i \cdot r_i$ is the minimum sum of subsets of $\{w_j\}_{r_j \in R_s}$ that is greater than $\sum_{C_i \in subC} \lambda_i + \Delta - 1$. This corresponds to the value returned by $Refine(\{w_j\}_{r_j \in R_s}, \sum_{C_i \in subC} \lambda_i + \Delta - 1)$. Thus the update on line 19 is correct.

Proposition 3 (Invariant of BCD2). *Let opt be cost of the optimum model of a MaxSAT instance. During the execution of BCD2, the invariant $\sum_{C_i \in \mathcal{C}} \lambda_i \leq opt \leq \mu$ holds.*

Proof. Initially $\mathcal{C}$ is empty, and $\sum_{C_i \in \mathcal{C}} \lambda_i$ is 0. On the other hand, μ is initialized to $\sum_{(c_j,w_j) \in Soft(\varphi)} w_j + 1$. Since $0 \leq opt \leq \sum_{(c_j,w_j) \in Soft(\varphi)} w_j$, then initially the invariant holds.

Each λ_i is only updated on unsatisfiable iterations in lines 15 and 19 and each update was proved to be correct in Propositions 1 and 2, respectively. Then after the updates we are guaranteed that $\sum_{C_i \in \mathcal{C}} \lambda_i \leq opt$.

Consider now a satisfiable iteration. Assume for the sake of contradiction that μ is updated such that $\mu < opt$. Then the assignment returned by the SAT solver can be extended 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 lower than *opt* which is a contradiction since, by definition, the sum of weights of relaxed clauses is an upper bound on the optimum MaxSAT model.

Proposition 4. *For any disjoint core C_s, the invariant $\lambda_s \leq \epsilon_s$ holds.*

Proof. The values of variables ϵ_i are only updated in lines 10 and 20 (see Algorithm 2). The updates are due to assignments that are models to the MaxSAT formula, and represent the cost of the model with respect to the clauses associated to the disjoint core C_i. Line 20 also considers the case where no model has been found yet, and updates ϵ_i to one plus the sum of all the weights of the soft clauses considered.

On the other hand, the values of variables λ_i are only updated in lines 15 and 19. In Propositions 1 and 2, was proven that there is no MaxSAT model with a cost smaller than the update of the lower bound in lines 15 and 19 (with respect to the clauses associated with the resulting core C_s). Hence, $\lambda_s \leq \epsilon_s$ for each disjoint core C_s.

Proposition 5. *BCD2 is correct and returns the optimum model for any WCNF formula.*

Proof. The algorithm performs binary search on the range of values $\{\sum_{C_i \in \mathcal{C}} \lambda_i, \ldots, \mu\}$. In each iteration the algorithm asks for a model with a cost at most $\sum_{C_i \in \mathcal{C}} \nu_i$. Due to the assignment of each ν_i in line 5 and Proposition 4, then $\sum_{C_i \in \mathcal{C}} \lambda_i \leq \sum_{C_i \in \mathcal{C}} \nu_i \leq \mu$. If the SAT solver returns with a satisfiable answer, then μ is updated to a lower value than the current upper bound (due to the added constraints). If the SAT solver returns with an unsatisfiable answer, then either $\sum_{C_i \in \mathcal{C}} \lambda_i$ increases or more than one of the disjoint cores are merged. Since the number of clauses to be relaxed is bounded by the number of soft clauses, then the maximum number of merges of disjoint cores is also bounded (disjoint cores only contain clauses that are relaxed). Thus the number of iterations where the algorithm does not increase the sum $\sum_{C_i \in \mathcal{C}} \lambda_i$, is bounded.

Finally, Proposition 3 proves that during the execution of the algorithm, there is always an optimum MaxSAT model between the bounds. Since the bounds are integer numbers, then the algorithm is guaranteed to stop with the optimum MaxSAT model.

4 Additional Techniques

This section introduces two additional techniques to improve the performance of core-guided binary search algorithms, namely, the hardening rule and biased search.

4.1 Hardening rule

The *hardening rule* is widely used in *branch and bound* (*BB*) algorithms for MaxSAT [12, 11, 7] which are based on a *systematic enumeration* of all possible assignments, where large subsets of *useless* assignments are discarded by computing *upper* and *lower bounds* on the cost of the optimum model. Whenever the weight of a soft clause plus the lower bound reaches the upper bound, the clause can be *made* hard. Indeed, the hardening rule was introduced in the most primitive BB algorithm for MaxSAT in the literature [5], but nowadays is still not used in core-guided MaxSAT algorithms. In what follows, a first integration of the hardening rule is proposed for core-guided MaxSAT algorithms that maintain both a lower bound and upper bound. To explain the idea, each soft clause (c, w) is extended with two weights (c, w, w') where w is the original weight and w' represents the weight of the clause after its *contributions* to the lower bound have been deducted. w' will be referred as the *deducted* weight. Let φ_d be a set of soft clauses involved in an increment d of the global lower bound. Then, the deducted weight of all the soft clauses in φ_d needs to be decreased by d. As a result, the hardening rule is applied taking into account the deducted weight rather than the original one. Hence, the hardening rule is shown in Equation 2

$$\text{if } w' + \lambda \geq \mu \text{ then } (c, w, w') \text{ can be replaced by } (c, \top, \top) \tag{2}$$

Let (c, w, w') be a soft clause that is made hard due to the hardening rule. There are two situations. If the soft clause has no relaxation variable, the weight of the clause is just replaced by $\top$. If the soft clause has a relaxation variable, the weight is updated to $\top$ and additionally, the relaxation variable is removed.

Example 2. Consider the formula $\{(x, 3, 3), (\bar{x}, 4, 4), (y, 3, 3), \ldots\}$. An initial upper bound $\mu = 5$ is obtained using any heuristic [8]. An initial lower bound $\lambda = 3$ can

be obtained due to an unsatisfiable core between the two first clauses. The minimum weight for the conflicting clauses $(x, 3, 3)$ and $(\bar{x}, 4, 4)$ is 3. The resulting formula is $\{(x, 3, 0), (\bar{x}, 4, 1), (y, 3, 3), \dots\}$ with $\lambda = 3$. Then, the hardening rule can be applied to the clause $(y, 3, 3)$ given that $3 + 3 \geq 5$. Hence, $(y, 3, 3)$ is replaced by $(y, \top, \top)$. The current formula is $\{(x, 3, 0), (\bar{x}, 4, 1), (y, \top, \top), \dots\}$ with $\lambda = 3$ and $\mu = 5$.

The integration of the hardening rule in BCD2 is as follows. Assume BCD2 maintains internally the deducted weight of each soft clause, then any of the initial lower bounds introduced in [8] can be used. Such lower bounds iteratively compute unsatisfiable cores until a satisfiable instance is reached. For each unsatisfiable core, the minimum weight is subtracted to the deducted weight of each soft clause in the core.

Assume any arbitrary iteration of the main loop of BCD2. Let $\lambda = \sum_{C_i \in \mathcal{C}} \lambda_i$ be the global lower bound and let $\mu = \sum_{C_i \in \mathcal{C}} \epsilon_i$ be the global upper bound, before the call to the SAT solver (line 6). After the call to the SAT solver, there are two possibilities:

- The SAT solver returns satisfiable (SAT). The global upper bound μ is updated and the hardening rule is checked with the new global upper bound.
- The SAT solver returns unsatisfiable (UNSAT) and the global lower bound is increased. Let $\lambda' = \sum_{C_i \in \mathcal{C}} \lambda_i$ be the new global lower bound in line 23. Let d be the difference between the previous and the current global lower bounds, i.e., $d = \lambda' - \lambda$. Such increment d is due to the disjoint core C_s in line 15 or in line 21. Hence, the deducted weight of each soft clause in the proper disjoint core C_s is decreased by d. Afterwards, the hardening rule is checked.

4.2 Biased Search

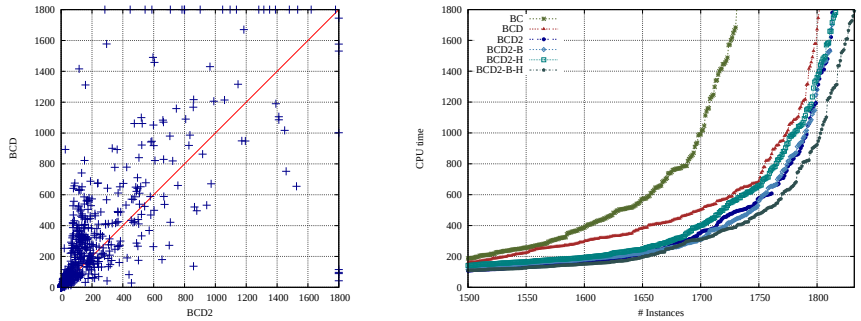
At each iteration, binary search algorithms compute a middle value ν between an upper bound μ and a lower bound λ (i.e. $\nu \leftarrow \lfloor \frac{\mu+\lambda}{2} \rfloor$). However, when the cost of the optimum model is close to one of the bounds, binary search can make several iterations before realizing that. In fact, QMAXSAT (0.4 version) solver [10] alternates iterations which compute the middle value between the bounds, and iterations which use the value of the upper bound. As such, QMAXSAT favors the discovery of models with a cost closer to the upper bound. Note that QMAXSAT was the best performing solver on recent MaxSAT Evaluations in the *partial MaxSAT industrial* category.

This paper proposes to compute a value between the lower bound and upper bound (i.e. $\nu \in [\lambda, \mu]$) based on the previous iterations. Two counters are maintained. A counter of the iterations that returned satisfiable (SAT) $nsat$, and a counter of the iterations that returned unsatisfiable (UNSAT) $nunsat$. Both counters are initialized to 1. At each iteration of the binary search algorithm the following percentage is computed:

$$p = nunsat/(nunsat + nsat)$$

The expression compares the number of unsatisfiable iterations against the total number of iterations, and gives a value closer to the bound with fewer outcomes in terms of a percentage. The value ν to be considered at each iteration is $\nu = \lambda + p \times (\mu - \lambda)$.

Note that the QMAXSAT approach is similar to always alternating the percentage p between 50% (middle value) and 100% (upper bound). The integration in BCD2 is straightforward. For each disjoint core C_i with estimate of the upper bound ϵ_i and lower bound λ_i, BCD2 computes the value ν_i as $\nu_i = \lambda_i + p \times (\epsilon_i - \lambda_i)$.



(a) (b)

Fig. 1. (a) Scatter plot of BCD vs BCD2, (b) Cactus plot of BC, BCD, BCD2 and BCD2 with additional techniques

5 Experimental Evaluation

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 set to 1800 seconds and the memory limit to 4GB. BCD2 and the additional techniques were implemented in the MSUNCORE [17] system, and compared against BC and BCD[3].

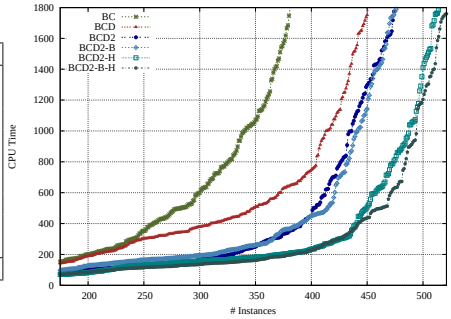
Figure 1 presents results on the performance of BCD2 (from Section 3) and the new techniques (from Section 4) in all of the *non-random* instances from 2009-2011 MaxSAT Evaluations (for a total of 2615 instances). The scatter plot (Figure 1.a) shows a comparison of the original BCD [8] with BCD2 (as described in Section 3). Note that BCD2 (1813) solves 12 more instances than BCD (1801). The scatter plot indicates that in general BCD requires larger run times than BCD2. A more detailed analysis indicates that, out of 1305 instances where the performance difference between BCD and BCD2 exceeds 20%, BCD2 outperforms BCD in 918, whereas BCD outperforms BCD2 in 387. Moreover, over the 1793 instances solved by both BCD and BCD2, the total number of SAT solver calls for BCD is 124907 and for BCD2 is 68690. This represents an average of 31.5 fewer SAT solver calls per instance for BCD2 (from 69.7 to 38.3), i.e. close to 50% fewer calls in BCD2 than in BCD on average. The difference is quite significant; it demonstrates the effectiveness of the new algorithm, but also indirectly suggests that some of the SAT solver calls, being closer to the optimum, may be harder for BCD2 than for BCD. Nevertheless, BCD2 consistently outperforms BCD overall.

The cactus plot (Figure 1.b) shows the run times for BCD, BCD2, BCD2 with hardening rule (BCD2-H), BCD2 with biased search (BCD2-B) and BCD2 with both techniques (BCD2-B-H). The original core-guided binary search algorithm [8] (BC) is also included. The performance difference between BCD and BCD2 is conclusive, and confirmed by the area below each plot. For the vast majority of instances, BCD2 outperforms BCD. The hardening rule (BCD2-H) allows solving 3 additional instances than

[3] Observe that in [8], BCD was shown to solve more instances than a representative sample of MaxSAT solvers.

Set	$\#I.$	BC	BCD	BCD2	BCD2-B	BCD-H	BCD-B-H
Upgrade	100	65	100	100	100	100	100
TimeT	32	11	12	12	13	13	13
Pedi-A	45	37	38	39	39	41	44
Pedi-B	45	45	44	44	44	45	45
Pedi-C	90	68	73	77	76	84	83
Pedi-D	50	44	44	43	43	45	45
Pedi-E	90	42	50	57	59	66	67
Pedi-F	90	49	59	63	62	73	74
Pedi-G	90	20	30	39	40	47	50
Total	632	381	450	474	476	514	521

(a)



(b)

Fig. 2. (a) Table number of solved instances per algorithm (b) Cactus plot with the different algorithms.

BCD2, whereas biased search (BCD2-B) allows solving one more instance. However, the integration of both techniques (BCD2-B-H) allows solving 1832 instances, i.e. 19 more instances than BCD2 and 31 more than the original BCD. As expected, BC is the worst performing algorithm (solves 1730 instances), and indirectly demonstrates that maintaining disjoint cores is essential to obtain a more robust algorithm.

The effect of the more accurate bounds maintained by BCD2 and the additional techniques is even more significant on weighted partial MaxSAT industrial instances. A second experiment, see Figure 2, shows the results for 100 *upgradeability* instances, 32 *timetabling* instances [3] and 500 *haplotyping with pedigrees* instances [15]. Observe that the haplotyping with pedigrees instances are divided in 7 sets (A, B, C, D, E, F, G). The results are summarized in the table of Figure 2.a. The first column shows the name of benchmark set. The second column shows the total number of instances in the set. The remaining columns show the total number of solved instances within the time and memory limits by BC, BCD and the different versions of BCD2. The same results are presented with a cactus plot in Figure 2.b to highlight the runtimes.

BC is again the worst performing algorithm, and is the only approach unable to solve the 100 upgradeability problems. BCD outperforms BC and solves 69 more instances. BCD2 is clearly better than BCD, being able to solve 26 more instances. Biased search (BCD2-B) has small effect and solves 2 more instances than BCD2. The hardening rule (BCD2-H) is quite helpful on these instances and solves 40 more instances than BCD2. Finally, the integration of the two new techniques (BCD2-B-H) allows solving 521 instances, i.e. 47 more instances than BCD2 and 71 more than the original BCD.

6 Conclusions

This paper proposes a number of improvements to a recently proposed MaxSAT algorithm [8] that implements core-guided binary search. The first improvement addresses the organization of the original algorithm, and modifies the algorithm to (i) maintain a global upper bound, that results in tighter local upper bounds for each disjoint core;

and (ii) use of more aggressive lower bounding techniques. The improvements to the upper and lower bound result in significant reduction in the number of SAT solver calls made by the algorithm. The second improvement consists of two techniques that can be implemented on top of any core-guided algorithm that uses lower and upper bounds. One of the techniques is referred to as the hardening rule and has been extensively used in branch-and-bound algorithms [5, 12, 11, 7], but not in core-guided algorithms. The second technique is referred to as *biased search*, and is shown to work effectively with the hardening rule. Experimental results, obtained on a comprehensive set of instances from past MaxSAT Evaluations, demonstrates that the new algorithm BCD2 significantly outperforms (an already quite robust) BCD.

References

1. Z. S. Andraus, M. H. Liffiton, and K. A. Sakallah. Reveal: A formal verification tool for verilog designs. In *LPAR*, pages 343–352, 2008.
2. C. Ansótegui, M. L. Bonet, and J. Levy. Solving (weighted) partial MaxSAT through satisfiability testing. In *SAT*, pages 427–440, 2009.
3. C. Ansótegui, M. L. Bonet, and J. Levy. A new algorithm for weighted partial maxsat. In *AAAI Conference on Artificial Intelligence*. AAAI, 2010.
4. A. Biere, M. Heule, H. van Maaren, and T. Walsh, editors. *Handbook of Satisfiability*, 2009.
5. Brian Borchers and Judith Furman. A two-phase exact algorithm for max-sat and weighted max-sat problems. *J. Comb. Optim.*, 2(4):299–306, 1998.
6. Z. Fu and S. Malik. On solving the partial MAX-SAT problem. In *SAT*, pages 252–265, August 2006.
7. F. Heras, J. Larrosa, and A. Oliveras. MiniMaxSat: An efficient weighted Max-SAT solver. *JAIR*, 31:1–32, Jan 2008.
8. F. Heras, A. Morgado, and J. Marques-Silva. Core-guided binary search algorithms for maximum satisfiability. In *AAAI*, 2011.
9. M. Jose and R. Majumdar. Cause clue clauses: error localization using maximum satisfiability. In *PLDI*, pages 437–446, 2011.
10. M. Koshimura, T. Zhang, H. Fujita, , and R. Hasegawa. QMaxSAT: A partial Max-SAT solver. *JSAT*, pages 95–100, 2012.
11. J. Larrosa, F. Heras, and S. de Givry. A logical approach to efficient Max-SAT solving. *Artificial Intelligence*, 172(2-3):204–233, 2008.
12. C. M. Li, F. Manyà, and J. Planes. New inference rules for Max-SAT. *Journal of Artificial Intelligence Research*, 30:321–359, October 2007.
13. Mark H. Liffiton and Karem A. Sakallah. Algorithms for computing minimal unsatisfiable subsets of constraints. *J. Autom. Reasoning*, 40(1):1–33, 2008.
14. V. Manquinho, J. Marques-Silva, and J. Planes. Algorithms for weighted Boolean optimization. In *SAT*, pages 495–508, 2009.
15. J. Marques-Silva, J. Argelich, A. Graca, and I. Lynce. Boolean lexicographic optimization: Algorithms and applications. *Annals of Mathematics and A. I.*, pages 1–27, 2011.
16. J. Marques-Silva and J. Planes. Algorithms for maximum satisfiability using unsatisfiable cores. In *DATE*, pages 408–413, 2008.
17. A. Morgado, F. Heras, and J. Marques-Silva. The MSUnCore MaxSAT solver. In *POS*, 2011.
18. R. Reiter. A theory of diagnosis from first principles. *Artif. Intell.*, 32(1):57–95, 1987.
19. S. Safarpour, H. Mangassarian, A. Veneris, M. H. Liffiton, and K. A. Sakallah. Improved design debugging using maximum satisfiability. In *FMCAD*, 2007.
20. L. Zhang and S. Malik. Validating sat solvers using an independent resolution-based checker: Practical implementations and other applications. In *DATE*, pages 10880–10885, 2003.