Journal on Satisfiability, Boolean Modeling, and Computation 11 (2019) 53-64

RC2: an Efficient MaxSAT Solver

Alexey Ignatiev aignatiev@ciencias.ulisboa.pt
Antonio Morgado ajmorgado@ciencias.ulisboa.pt
Joao Marques-Silva jpms@ciencias.ulisboa.pt
Faculty of Sciences
University of Lisbon
Lisbon, Portugal

Abstract

Recent work proposed a toolkit PySAT aiming at fast and easy prototyping with propositional satisfiability (SAT) oracles in Python, which enabled one to exploit the power of the original implementations of the state-of-the-art SAT solvers in Python. Maximum satisfiability (MaxSAT) is a well-known optimization version of SAT, which can be solved with a series of calls to a SAT oracle. Based on this fact and motivated by the ideas underlying the PySAT toolkit, this paper describes and evaluates RC2 (stands for *relaxable cardinality constraints*), a new core-guided MaxSAT solver written in Python, which won both *unweighted* and *weighted* categories of the main track of MaxSAT Evaluation 2018.

Keywords: *Maximum Satisfiability, Relaxable/Soft Cardinality Constraints, Python*

Submitted November 2018; revised March 2019; published September 2019

1. Introduction

Maximum Satisfiability (MaxSAT) represents the optimization problem for propositional formulas in conjunctive normal form (CNF). The MaxSAT problem can be formulated in a general setting as follows. Clauses can have associated weights, and clauses can be *hard* (corresponding to an infinite positive weight). The goal is to find the assignments that maximize the cost of the satisfied clauses. (The hard clauses are assumed to be satisfiable; otherwise the cost would be infinite.) MaxSAT finds a growing range of applications that include program analysis, fault localization, and model-based diagnosis in general.

This paper describes RC2[1], an open-source MaxSAT solver, written in Python and based on the PySAT framework[2] [11]. Despite being prototyped in Python, RC2 was ranked first in the two complete categories of the MaxSAT Evaluation 2018. This paper describes not only the design decisions supporting RC2, including the algorithm originally proposed in [19], but it also outlines the implementation details that enable a top-performing MaxSAT solver to be implemented on top of the PySAT framework.

The paper is organized as follows. Section 2 introduces the necessary definitions and the notation used throughout the paper. Section 3 describes the organization and the implementation of the RC2 solver and the heuristics used. Section 4 presents the experimental

1. RC2 is the acronym for *Relaxable Cardinality Constraints* and extends a MaxSAT algorithm based on relaxable cardinality constraints first proposed in [19].
2. `http://pysathq.github.io`

©2019 IOS Press, SAT Association and the authors.

results assessing the performance of various configurations of RC2. Availability of RC2 is discussed in Section 5, which is followed by the conclusions made in Section 6.

2. Preliminaries

This section introduces the notation and definitions used in this paper. Standard propositional logic definitions apply (e.g. [7]). Propositional formulas are defined over a set of Boolean variables. A propositional formula $\mathcal{F}$ is said to be in *conjunctive normal form* (CNF) if it is represented as a conjunction of clauses, also interpreted as a set of clauses. A clause is a disjunction of literals, also interpreted as a set of literals. A literal is a propositional variable or its *complement*. Hereinafter, CNF formulas are dealt with using SAT oracles. Given a CNF formula $\mathcal{F}$, a SAT oracle decides whether $\mathcal{F}$ is satisfiable, in which case it returns a satisfying assignment. Given an unsatisfiable CNF formula $\mathcal{F}$, a SAT oracle can also return an *unsatisfiable core* $\mathcal{U} \subseteq \mathcal{F}$, which is a subset of the clauses of $\mathcal{F}$ that is unsatisfiable on its own. A SAT oracle is assumed to be a *conflict-driven clause learning* (CDCL) SAT solver. CDCL SAT solvers are summarized in [7].

In the context of the *maximum satisfiability* (MaxSAT) problem, (*weighted*) *partial* CNF formulas are typically considered. Clauses in a partial CNF formula are characterized as *hard*, meaning that these must be satisfied, or *soft*, meaning that these are to be satisfied, if at all possible. An integer weight can be associated with each soft clause, and the goal of maximum satisfiability (MaxSAT) is to find an assignment to the propositional variables such that the hard clauses are satisfied, and the sum of the weights of the satisfied soft clauses is maximized. Whenever convenient, soft clauses will be represented as pairs (c, w), with c being a disjunction of literals and w its weight. Also, hard clauses in the same context will be marked by weight $\top$, i.e. $(c, \top)$. A number of algorithms for MaxSAT solving exist, including iterative and core-guided algorithms [22, 4, 20], as well as the algorithms based on implicit hitting set enumeration [6].

3. Solver Description

This section describes the organization of the solver, the MaxSAT algorithm used, as well as the heuristics integrated. Additionally, the section identifies a number of differences between the two configurations submitted to the MaxSAT Evaluation 2018, namely RC2-A and RC2-B.

In contrast to our previous MaxSAT solver MSCG [21] implementing a large set of numerous MaxSAT algorithms [20], e.g. including *progression-based* [10] and OLL [19] algorithms, RC2 focuses exclusively on a variant of the latter algorithm following the implementation of [21]. The algorithm is detailed in the following section. RC2 supports a variety of SAT solvers provided by PySAT, through an assumption-based Minisat-like [8] *incremental* interface. By default, RC2 uses Glucose 3.0 [5] as the underlying SAT oracle.

3.1 RC2 MaxSAT Algorithm

RC2 stands for Relaxable Cardinality Constraints. This section describes the MaxSAT algorithm implemented in RC2. Originally, the OLL algorithm was created for solving ASP optimization problems [1]. Then in 2014, the algorithm was adapted for solving the

MaxSAT problem [19] but reusing cardinality constraints as they are discovered. RC2 improves the OLL algorithm adapted to MaxSAT [19] with additional features explained in following sections. Before describing the algorithm, we will present an illustrative example.

In the example we will abuse the notation and refer to constraints as literals or clauses. Consider $\mathcal{F} = \mathcal{S} \cup \mathcal{H}$ a partial MaxSAT formula, where the set of soft clauses is $\mathcal{S} = \{(x_1, 1), (x_2, 1), (x_3, 1), (x_4, 1), (x_5, 1)\}$, and the set of hard clauses is $\mathcal{H} = \{(\neg x_1 \vee \neg x_2 \vee \neg x_3 \vee \neg x_4), (\neg x_1 \vee \neg x_2), (\neg x_3 \vee \neg x_4), (\neg x_1 \vee \neg x_5), (\neg x_2 \vee \neg x_5), (\neg x_3 \vee \neg x_5), (\neg x_4 \vee \neg x_5)\}$.

Initially, $\mathcal{F}$ is sent to the SAT solver, which reports that $\mathcal{F}$ is unsatisfiable. Let us assume that the unsatisfiable core returned is made of the soft clauses $(x_1, 1)$, $(x_2, 1)$, $(x_3, 1)$, $(x_4, 1)$ together with the hard clause $(\neg x_1 \vee \neg x_2 \vee \neg x_3 \vee \neg x_4)$. At least one of the soft clauses of the core will have to be disregarded in order to *fix* the core. Similar to other MaxSAT algorithms, the algorithm proceeds by relaxing each soft clause in the core (i.e. augmenting the clause with a fresh variable called *relaxation variable*) and constrains the sum of the relaxation variables to be at most one (by adding a cardinality constraint). New relaxation variables r_1, r_2, r_3, r_4 are created, and the soft clauses in the core are replaced in $\mathcal{F}$ by the hard clauses $(x_1 \vee r_1)$, $(x_2 \vee r_2)$, $(x_3 \vee r_3)$, $(x_4 \vee r_4)$. The algorithm differs from other known MaxSAT algorithms in that instead of turning the relaxation variables into new soft clauses, it makes the new cardinality constraint a *soft constraint*. The new soft constraint $((r_1 + r_2 + r_3 + r_4 \leq 1), 1)$ is added to $\mathcal{F}$, thus $\mathcal{F}$ is updated to:

$$\begin{aligned}
&\mathcal{S} \leftarrow \{(x_5, 1), (r_1 + r_2 + r_3 + r_4 \leq 1, 1)\}\\
&\mathcal{H} \leftarrow \mathcal{H} \cup \{(x_1 \vee r_1), (x_2 \vee r_2), (x_3 \vee r_3), (x_4 \vee r_4)\}
\end{aligned}$$

The total MaxSAT cost so far is set to 1.

At this point, a new iteration is started with the updated formula being sent to the SAT solver. The solver reports the formula to be unsatisfiable, with a new unsatisfiable core containing (besides the hard clauses) the soft constraint $((r_1 + r_2 + r_3 + r_4 \leq 1), 1)$. Since there are no original soft clauses in the core (only this soft constraint), no clause is relaxed. The soft constraint increases its *right-hand side* (*RHS*) to 2, i.e. $\mathcal{S}$ is updated to:

$$\mathcal{S} \leftarrow \{(x_5, 1), (r_1 + r_2 + r_3 + r_4 \leq 2, 1)\}$$

The MaxSAT cost is updated to 2.

A third iteration sends the updated formula to the SAT solver, which concludes again that the formula is unsatisfiable. This time the unsatisfiable core contains the soft clause $(x_5, 1)$ and the soft constraint $(r_1 + r_2 + r_3 + r_4 \leq 2, 1)$, together with some of the hard clauses. Since $(x_5, 1)$ is an original soft clause of the formula, the clause is relaxed with a new variable r_5. At this point, the algorithm removes both the soft clause and the soft constraint involved in the core, and adds two new soft constraints $(r_5 + \neg(r1 + r_2 + r_3 + r_4 \leq 2) \leq 1, 1)$ and $(r1 + r_2 + r_3 + r_4 \leq 3, 1)$. The first soft constraint added relates the relaxation variable with the previous soft constraint. The constraint is violated if r_5 is set to *true* and the previous soft constraint is violated (that is $r1 + r_2 + r_3 + r_4 \geq 3$). The second soft constraint increases the bound of the previous soft constraint. Together, both added constraints allow either r_5 to be set to *true* (satisfying the previous soft constraint), or to have one more of the previous relaxation variables set to *true*. The updated formula is:

$$\begin{aligned}
&\mathcal{S} \leftarrow \{(r_5 + \neg(r1 + r_2 + r_3 + r_4 \leq 2) \leq 1, 1), (r_1 + r_2 + r_3 + r_4 \leq 3, 1)\}\\
&\mathcal{H} \leftarrow \mathcal{H} \cup \{(x_5 \vee r_5)\}
\end{aligned}$$

Algorithm 1: RC2 MaxSAT Algorithm

```
Input: A partial MaxSAT formula $\mathcal{F}$
Function RelaxAndHarden(L, $\mathcal{F}$, $(l_1 \vee \ldots \vee l_m, 1)$)
    $\mathcal{F} \leftarrow \mathcal{F} \setminus \{(l_1 \vee \ldots \vee l_m, 1)\}$
    $\mathsf{L} \leftarrow \mathsf{L} \cup \{r\}$                    # $r$ is a fresh boolean (relaxation) variable
    $\mathcal{F} \leftarrow \mathcal{F} \cup \{(l_1 \vee \ldots \vee l_m \vee r)\}$
map $\leftarrow \{\}$                                        # map[lit] = (sumOutputs, rhs)
cost $\leftarrow 0$
while True :
    (st, $\mathcal{C}$, $\mathcal{A}$) $\leftarrow$ SATSolver($\mathcal{F}$)
    if st == True:
        return (cost, $\mathcal{A}$)
    else:
        cost $\leftarrow$ cost $+ 1$
        L $\leftarrow \emptyset$
        for $c \in$ Soft($\mathcal{C}$) :
            if $c$ is not a soft cardinality constraint:
                RelaxAndHarden(L, $\mathcal{F}$, $c$)
            else:                              # $c$ is a soft cardinality constraint
                $\mathcal{F} \leftarrow \mathcal{F} \setminus \{c\}$
                $(\neg s, 1) \leftarrow c$               # $c$ was enforced by the unit clause $(\neg s, 1)$
                L $\leftarrow$ L $\cup \{s\}$
                (so, rhs) $\leftarrow$ map[$\neg s$]
                if rhs $+ 1 <$ |so| :
                    $\mathcal{F} \leftarrow \mathcal{F} \cup \{(\neg$so[rhs $+ 1$]$, 1)\}$
                    map[$\neg$so[rhs $+ 1$]] $\leftarrow$ (so, rhs $+ 1$)
        (so, sc) $\leftarrow$ EncodeSum(L)              # EncodeSum(L) = (sumOutputs, sumCls)
        $\mathcal{F} \leftarrow \mathcal{F} \cup \{(\neg$so$[1], 1)\} \cup$ sc
        map[$\neg$so[1]] $\leftarrow$ (so, 1)
```

The MaxSAT cost is set to 3.

Finally, in the last iteration of the algorithm, the new formula is sent to the SAT solver, which reports it to be satisfiable, and so the final MaxSAT cost is concluded to be 3.

As the previous example illustrates, the idea of the algorithm is to go through unsatisfiable iterations until a satisfiable formula is obtained. Whenever a new unsatisfiable core is identified, the formula is updated such that either all the previous soft constraints in the core are satisfied and at most one of the new relaxation variables is allowed to be set to *true*, or one of the soft constraints is allowed to increase its bound by 1.

The example also illustrates that cardinality constraints are used as soft constraints. In RC2, the soft cardinality constraint $(\sum_i^m x_i \leq k, 1)$ is divided into the *left-hand side* (*LHS*) sum $(\sum_i^m x_i)$, and the right-hand side (RHS) $(\leq k)$. The LHS sum is encoded as a set of hard clauses (with the Iterative Totalizer encoding [17]) using auxiliary variables $o_1, \ldots, o_m$, representing the unary number $o_m \ldots o_1$. The RHS is enforced by adding the soft unit clause $(\neg o_{k+1}, 1)$.

The pseudo-code of the RC2 MaxSAT algorithm is presented in Algorithm 1. The algorithm takes as input a partial MaxSAT formula $\mathcal{F}$. Initially (line 5), a dictionary map is created which associates literals with soft cardinality constraints. For a given literal l, $\mathsf{map}[l]$ returns a pair $(\mathsf{sumOutputs}, \mathsf{rhs})$, where $\mathsf{sumOutputs}$ is a vector of auxiliary variables encoding the LHS sum of the soft cardinality constraint (as a unary number), and rhs is the current RHS of the soft cardinality constraint. Then in line 6 the initial cost is set to 0.

The main iterations of the algorithm occur in lines 7 to 27. The SAT solver is called on the current formula $\mathcal{F}$ (line 8). If the formula is satisfiable, then st is set to *true* and the algorithm terminates returning a pair with the optimum value cost and the satisfying assignment $\mathcal{A}$ (reported by the SAT solver) in line 10.

If the formula is unsatisfiable, then a core $\mathcal{C}$ is obtained, the cost is increased by 1 (line 12), and a new set L is created to hold the input variables of the LHS sum of a new soft cardinality constraint to be created (line 13). The algorithm proceeds by checking each soft constraint in the core (line 14). If the soft constraint is an original soft clause of $\mathcal{F}$, then a new relaxation variable is added to the clause, the clause is made hard in $\mathcal{F}$, and the relaxation variable is added to L. This process is done through function $\mathsf{RelaxAndHarden}(\mathsf{L}, \mathcal{F}, c)$ (defined in lines 1-4) called in line 16. Otherwise, the constraint is a soft cardinality constraint, and it is removed from $\mathcal{F}$ (line 18). The corresponding unit literal (s) enforcing the cardinality constraint is added to L (line 20). If it is possible to increase the RHS[3.] of the cardinality constraint (line 22), then a new soft cardinality constraint is enforced in $\mathcal{F}$ using the same LHS sum with increased RHS (lines 23 and 24).

When all soft constraints of the core have been processed, the LHS sum of the new soft cardinality constraint is encoded using function $\mathsf{EncodeSum}(\mathsf{L})$, which returns a pair containing the output variables so[4.] representing the sum and the set of hard clauses sc encoding the sum (line 25). The hard clauses, and a soft unit clause enforcing the RHS (≤ 1) of the new soft cardinality constraint are added to $\mathcal{F}$ (line 26). The information regarding the new soft cardinality constraint is added to map (line 27).

The algorithm described above deals with unweighted partial MaxSAT formulas. In the weighted case, RC2 splits the soft constraints according to the minimum weight of the soft constraints in the core (similar to other MaxSAT algorithms). Every time a new core is found, the minimum weight min of the soft constraints in the core is computed. Then each soft constraint (c_i, w_i) with a weight greater than the minimum is replaced by two soft constraints: (c_i, min) and $(c_i, w_i - min)$. Here, RC2 proceeds as in the partial case with the core containing the soft constraints of weight min, and also updates the cost by min.

3.2 Heuristics

The RC2 MaxSAT algorithm implemented in the solver is augmented with a number of additional heuristics aiming at improving its performance. All the heuristics integrated can be enabled or disabled with the use of the corresponding command-line options.

3. $\sum_i^m x_i \leq m$ is trivially satisfied, thus it is not encoded.
4. Note that so is a 0-indexed array where $\mathsf{so}[0] = o_1, \ldots, \mathsf{so}[m-1] = o_m$.

3.2.1 SAT Solver Interface

RC2 makes use of the standard MiniSat-like incremental interface [8] to a SAT oracle, i.e. multiple calls to the oracle are done by specifying a list of *assumption literals*, which may change from one oracle call to another. This is done in the following way. Given an unsatisfiable partial formula $\mathcal{H} \wedge \mathcal{S}$, the solver augments every soft clause $c \in \mathcal{S}$ with a fresh selector literal $\neg s$, i.e. a modified set of soft clauses is considered: $\mathcal{S}' = \{c \vee \neg s \,|\, c \in \mathcal{S}\}$. Now, by setting selector literals to *true* (resp. *false*), one can *activate* (resp. *deactivate*) the corresponding clauses, and these preferences can be specified as a list of *assumptions* given to the oracle. (Moreover, one may consider making the clauses of $\mathcal{S}'$ hard and use the corresponding selectors as *unit size* soft clauses.) The rationale behind this is that the incremental use of a SAT oracle makes it possible to reuse the oracle, once it is created. This enables the solver to keep all the learnt clauses from prior SAT calls.

3.2.2 Dealing With Weighted Formulas

One may opt to apply *Boolean lexicographic optimization* (BLO) [14] and *stratification* [2]. Both techniques prove helpful when dealing with weighted formulas. BLO and stratification are known to be crucial for MaxSAT algorithms that split soft clauses of the formula involved in an unsatisfiable core, e.g. in FM [9] and its improvements [16, 15, 3, 12], as well as OLL and RC2 [19] (also detailed above). Both heuristics aim at imposing a preference on the soft clauses based on their weights and feeding the MaxSAT algorithm with more soft clauses as soon as the problem gets solved for the clauses of higher weights.

3.2.3 Core Exhaustion

An important heuristic used in RC2 is *unsatisfiable core exhaustion* (originally referred to as *cover optimization*) [2]. The heuristic does the following. Recall that each newly identified unsatisfiable core gets relaxed and a new bound on the number of falsified clauses in the cores is set to 1. Assume every clause of unsatisfiable core $\mathcal{C}$ is relaxed, i.e. transformed into a set of relaxed clauses $\mathcal{C}_r$ and the corresponding set of relaxations variables is R. Note that $\mathcal{H} \wedge \mathcal{C}_r \wedge (\sum_{r \in R} r \leq 1)$ may still be unsatisfiable. The idea behind the core exhaustion heuristic is to quickly increase the bound from 1 to some value $k \leq |R|$ s.t. $\mathcal{H} \wedge \mathcal{C}_r \wedge (\sum_{r \in R} r \leq k)$ is still unsatisfiable while increasing k further makes the formula satisfiable. The rationale here is that calling an oracle incrementally on a a series of slightly modified formulas focusing only on the recently computed unsatisfiable core and disregarding the rest of the formula may be practically effective.

3.2.4 Intrinsic AtMost1 constraints

The solver can be instructed to detect *intrinsic* AtMost1 constraints on some of the soft clauses of the formula and deal with them *before* executing the main MaxSAT algorithm. This heuristic works as follows. For the sake of simplicity, consider a set $\mathcal{S}$ of *unweighted* soft clauses (the technique can be easily extended to the case of *weighted* soft clauses). Assume there is a subset $\mathcal{S}' \subseteq \mathcal{S}$ s.t. $\sum_{c \in \mathcal{S}'} c \leq 1$ holds, i.e. at most one of these clauses can be satisfied in the presence of the formula's hard clauses. In this situation, one can immediately conclude that the formula has cost at least $|\mathcal{S}'| - 1$. Furthermore, one can replace the original soft clauses of $\mathcal{S}'$ with a single unit-weight soft clause $\bigvee_{c\in\mathcal{S}'} c$. If a MaxSAT solution for the formula falsifies this new clause, all the original soft clauses of $\mathcal{S}'$ are falsified as well. Recall that each soft clause of the formula in RC2 is augmented with a fresh selector literal, i.e. the solver deals with clauses $c' = c \vee \neg s$ for each original soft clause $c \in \mathcal{S}$. Here, c' is made hard and its selector s is treated as a unit size soft clause. As a result, the implementation of intrinsic AtMost1 constraint heuristic in RC2 replaces the clauses of $\mathcal{S}'$ with a single disjunction of their selectors.

Note that a number of pairwise disjoint subsets of $\mathcal{S}$ can be detected, each being processed the way described above. Each such subset $\mathcal{S}'$ contributes to the total MaxSAT cost of the formula. Observe that *no* SAT oracle calls are *involved* in this process. The fact that at most one of the two soft clauses $(c_1 \vee \neg s_1)$ and $(c_2 \vee \neg s_2)$ can be satisfied, can be checked by setting one of their selectors, e.g. s_1, to *true* and applying unit propagation followed by a check whether or not the other selector, e.g. s_2, is implied to be *false*.[5.] As shown in the experimental results below (see Section 4), intrinsic AtMost1 constraints play a crucial role, tremendously improving the performance of RC2.

3.3 MSE18 Versions

Two variants of the solver were submitted to the MaxSAT Evaluation 2018 including RC2-A and RC2-B. Both versions use Glucose 3.0 [5] as an underlying SAT engine, apply BLO, stratification, core exhaustion, and detect intrinsic AtMost1 constraints. The only difference between the solver variants is the policy for unsatisfiable core reduction.

3.3.1 Core Reduction

In contrast to RC2-A, RC2-B applies heuristic unsatisfiable core minimization done with a simple deletion-based *minimal unsatisfiable subset* (*MUS*) extraction algorithm [13]. The implementation follows Algorithm 2. The idea is to try to deactivate soft clauses of the unsatisfiable core one by one while checking if the remaining soft clauses together with the hard part of the formula are unsatisfiable. Clauses that are necessary for preserving unsatisfiability comprise an MUS of the input formula (it is contained in the given unsatisfiable core) and are reported as a result of the procedure. During the core minimization phase in RC2-B, all SAT calls are dropped after obtaining 1000 conflicts. Note that core minimization in RC2-B is disabled for large *plain* MaxSAT formulas, i.e. those having no hard clauses but more than 100000 soft clauses. The reason is that having this many soft clauses (and, thus, as many assumption literals) and no hard clauses is deemed to make SAT calls too expensive.

Although core minimization is disabled in RC2-A, reducing the size of unsatisfiable cores can be still helpful for weighted instances due to the nature of the RC2 algorithm, i.e. because of the clause splitting technique applied to the clauses of an unsatisfiable core depending on their weight. Therefore, when dealing with weighted instances, RC2-A *trims* unsatisfiable cores at most 5 times (e.g. see [21] for details on *unsatisfiable core trimming*) aiming at getting rid of unnecessary clauses. Note that unsatisfiable core trimming is disabled in RC2-A for unweighted MaxSAT instances and it is not used in RC2-B at all.

5. Note that in general unit propagation does not suffice to check whether or not $\mathcal{H} \wedge s_1 \vDash \neg s_2$.

Algorithm 2: Deletion-based MUS extraction.

```
input  : hard clauses $\mathcal{H}$ and a core $\mathcal{C} \subseteq \mathcal{S}$ s.t. $\mathcal{H} \cup \mathcal{C} \vDash \bot$
output: MUS $\mathcal{M}$
1 $\mathcal{M} \leftarrow \mathcal{C}$
2 for $c \in \mathcal{M}$:
3     if not SATLimited($\mathcal{H} \cup \mathcal{M} \setminus \{c\}$):    # do until 1000 conflicts are obtained
4         $\mathcal{M} \leftarrow \mathcal{M} \setminus \{c\}$
5 return $\mathcal{M}$
```

3.4 Incremental MaxSAT Solving

Besides the features described above, RC2 also supports *incremental MaxSAT solving*. This means that one can apply RC2 not only to compute a satisfying assignment corresponding to a *MaxSAT solution*, i.e. corresponding to the optimum MaxSAT cost, but also to iteratively enumerate either a given number of assignments, or all of them exhaustively, with no need to restart the solver. Note that the enumeration mode can be used to compute *top-k* MaxSAT solutions in a sorted fashion, i.e. best cost solutions are computed first. Incremental MaxSAT solving in RC2 is associated with RC2's ability to add hard and soft clauses to the working formula *on demand*, i.e. whenever necessary (for details, see method `add_clause()` in the source code of RC2).

4. Experimental Results

This section is devoted to the evaluation of the basic version of RC2 and its competition versions, i.e. RC2-a and RC2-b. Besides that, it also assesses the contribution of the two heuristics described above, namely core exhaustion (see Section 3.2.3) and intrinsic AtMost1 constraints (as described in Section 3.2.4), in the overall performance of the solver. As a result, in the following, RC2 denotes the basic version of the solver (including the use of BLO and stratification for weighted formulas) while RC2* augments the basic version with unsatisfiable core exhaustion and RC2** *additionally* performs intrinsic AtMost1 constraints detection and adaptation.

For the comparison, we chose all (i.e. unweighted and weighted) benchmarks from the MaxSAT Evaluation 2018 (MSE18) [18]. The benchmarks suite contains 600 unweighted and 600 weighted MaxSAT instances. The experiments were performed in Ubuntu Linux on an Intel Xeon E5-2630 2.60GHz processor with 64GByte of memory. The time limit was set to 1800s and the memory limit to 10GByte for each individual process to run. For the purpose of comparison, two solvers best performing in MSE18 (besides RC2-a and RC2-b) were chosen for each of the categories of benchmarks. These include Maxino and MaxHS for the unweighted instances[6.] and also MaxHS and Pacose for the weighted instances[7.].

Figure 1 and Figure 2 show cactus plots depicting the performance of all the chosen competitors. The following conclusions can be made with respect to the experimental data.

6. https://maxsat-evaluations.github.io/2018/results/complete/unweighted/summary.html
7. https://maxsat-evaluations.github.io/2018/results/complete/weighted/summary.html

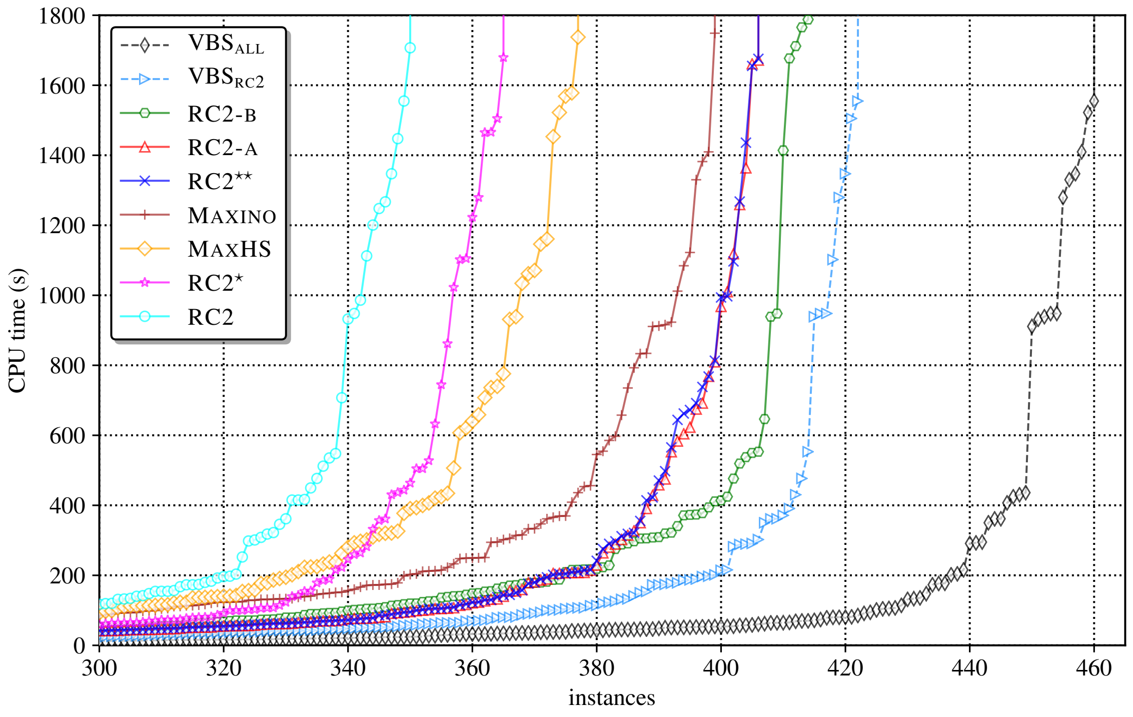

Figure 1. Unweighted instances of MSE18.

First, as one can observe, RC2-A and RC2-B are confirmed to be best performing MaxSAT solvers. They solve 406 and 414 unweighted instances and 394 and 403 weighted benchmarks, respectively. (Observe that the numbers of solved instances differ from those in the evaluation, which can be explained by different time and memory limits used, as well as different machine configurations.) Also, the basic version of RC2 comes the last in both cases with 350 unweighted and 314 weighted instances solved. The use of unsatisfiable core exhaustion, i.e. see RC2* in the plots, adds up 15 and 22 more instances solved in the unweighted and weighted categories, thus, resulting in 365 unweighted and 336 weighted formulas solved. The additional application of intrinsic AtMost1 constraints adds up more 41 unweighted and 50 weighted instances solved, i.e. RC2** solves 406 unweighted and 386 weighted instances in total. (Note that RC2-A and RC2** behave the same way in the unweighted category because they represent the same configuration of the solver for unweighted formulas, which is not the case in the presence of weighted clauses.) Also, as one can observe, unsatisfiable core reduction seems important when dealing with weighted MaxSAT formulas (see in Figure 2 how the competition versions of RC2 outperform RC2**).

Figure 1 and Figure 2 additionally depict the performance of the two configurations of a *virtual best solver* (VBS): VBS$_{\text{ALL}}$ and VBS$_{\text{RC2}}$. While the latter VBS includes all the configurations of RC2, the former one integrates all the solvers tested. Note that VBS$_{\text{RC2}}$ solves 422 unweighted and 418 weighted benchmarks. This is 8 unweighted and 15 weighted benchmarks more than the number of instances solved solely by the winning configuration RC2-B. Hence, tweaking the heuristics and the parameters used in RC2 may result in a

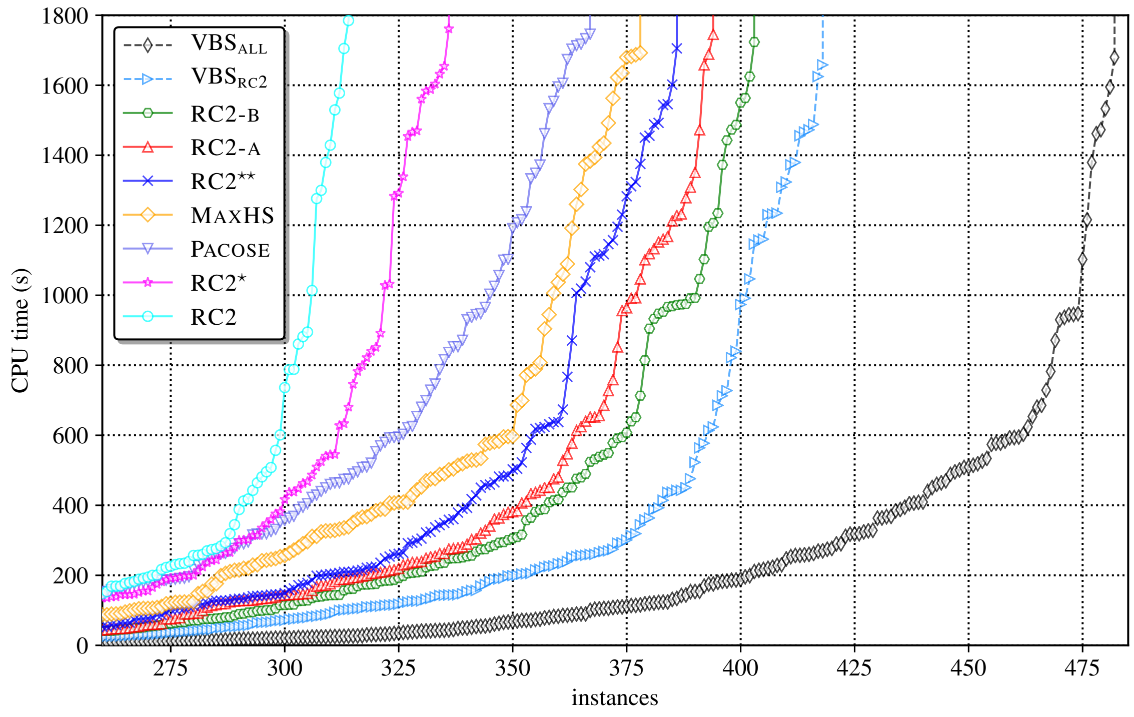

Figure 2. Weighted instances of MSE18.

better overall performance of the solver. Also note that VBS$_{ALL}$ solves 460 unweighted and 482 weighted benchmarks. This result comprises an impressive performance gap between VBS$_{ALL}$ and the best performing standalone solver, which suggests that using a portfolio of top MaxSAT solvers in practical problem solving with MaxSAT could be strategic.

To conclude, the experimental results shown not only confirm the advantage of RC2-A and RC2-B over the state the art in unweighted and weighted MaxSAT solving (represented by Maxino, MaxHS, and Pacose) but also suggests that the proposed heuristics are crucial for the overall performance of RC2. This especially holds for the detection of intrinsic AtMost1 constraints. Note that this heuristic can be applied to any MaxSAT solver, which may lead to significant performance improvements. Moreover, one of the lines of our future work on RC2 is to investigate whether or not the technique can be efficiently generalized to the case of AtMost$k$ constraints.

5. Availability

RC2 is distributed as a part of the PySAT framework, which is available under an MIT license at https://pysathq.github.io. It can also be installed as a Python package from PyPI:

```
pip install python-sat
```

The RC2 solver can be used as a standalone executable `rc2.py` and can also integrated into a complex Python-based problem solving tool, e.g. using the standard *import* interface of Python:

```
from pysat.examples import rc2
```

6. Conclusions

This paper describes the organization of the RC2 propotype MaxSAT solver, which ranked first in the two complete categories of the MaxSAT evaluation 2018 [18]. RC2 is based on the PySAT framework, and it is publicly available (see Section 5). Future work will seek to extend RC2 with additional MaxSAT algorithms, with the purpose of providing reference implementations.

Acknowledgments

This work was supported by FCT grants ABSOLV (PTDC/CCI-COM/28986/2017), Fault-Locker (PTDC/CCICOM/29300/2017), SAFETY (SFRH/BPD/120315/2016), and SAMPLE (CEECIND/04549/2017).

References

[1] Benjamin Andres, Benjamin Kaufmann, Oliver Matheis, and Torsten Schaub. Unsatisfiability-based optimization in clasp. In *ICLP*, pages 211–221, 2012.

[2] Carlos Ansótegui, Maria Luisa Bonet, Joel Gabàs, and Jordi Levy. Improving WPM2 for (weighted) partial MaxSAT. In *CP*, pages 117–132, 2013.

[3] Carlos Ansótegui, Maria Luisa Bonet, and Jordi Levy. Solving (weighted) partial MaxSAT through satisfiability testing. In *SAT*, pages 427–440, 2009.

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

[5] Gilles Audemard, Jean-Marie Lagniez, and Laurent Simon. Improving Glucose for incremental SAT solving with assumptions: Application to MUS extraction. In *SAT*, pages 309–317, 2013.

[6] Fahiem Bacchus, Antti Hyttinen, Matti Järvisalo, and Paul Saikko. Reduced cost fixing in MaxSAT. In *CP*, pages 641–651, 2017.

[7] Armin Biere, Marijn Heule, Hans van Maaren, and Toby Walsh, editors. *Handbook of Satisfiability*, **185** of *Frontiers in Artificial Intelligence and Applications*. IOS Press, 2009.

[8] Niklas Eén and Niklas Sörensson. Temporal induction by incremental SAT solving. *Electr. Notes Theor. Comput. Sci.*, **89**(4):543–560, 2003.

[9] Zhaohui Fu and Sharad Malik. On solving the partial MAX-SAT problem. In *SAT*, pages 252–265, 2006.

[10] Alexey Ignatiev, Antonio Morgado, Vasco M. Manquinho, Inês Lynce, and Joao Marques-Silva. Progression in maximum satisfiability. In *ECAI*, pages 453–458, 2014.

[11] Alexey Ignatiev, Antonio Morgado, and Joao Marques-Silva. PySAT: A Python toolkit for prototyping with SAT oracles. In *SAT*, pages 428–437, 2018.

[12] Vasco M. Manquinho, Joao Marques-Silva, and Jordi Planes. Algorithms for weighted boolean optimization. In *SAT*, pages 495–508, 2009.

[13] Joao Marques-Silva. Minimal unsatisfiability: Models, algorithms and applications (invited paper). In *ISMVL*, pages 9–14, 2010.

[14] Joao Marques-Silva, Josep Argelich, Ana Graca, and Inês Lynce. Boolean lexicographic optimization: algorithms & applications. *Annals of Mathematics and Artificial Intelligence (AMAI)*, **62**(3-4):317–343, 2011.

[15] Joao Marques-Silva and Vasco M. Manquinho. Towards more effective unsatisfiability-based maximum satisfiability algorithms. In *SAT*, pages 225–230, 2008.

[16] Joao Marques-Silva and Jordi Planes. On using unsatisfiability for solving maximum satisfiability. *CoRR*, **abs/0712.1097**, 2007.

[17] Ruben Martins, Saurabh Joshi, Vasco M. Manquinho, and Inês Lynce. Incremental cardinality constraints for MaxSAT. In *CP*, pages 531–548, 2014.

[18] MaxSAT Evaluation 2018.
https://maxsat-evaluations.github.io/2018/.

[19] Antonio Morgado, Carmine Dodaro, and Joao Marques-Silva. Core-guided MaxSAT with soft cardinality constraints. In *CP*, pages 564–573, 2014.

[20] Antonio Morgado, Federico Heras, Mark H. Liffiton, Jordi Planes, and Joao Marques-Silva. Iterative and core-guided MaxSAT solving: A survey and assessment. *Constraints*, **18**(4):478–534, 2013.

[21] Antonio Morgado, Alexey Ignatiev, and Joao Marques-Silva. MSCG: Robust core-guided MaxSAT solving. *JSAT*, **9**:129–134, 2015.

[22] Olivier Roussel and Vasco M. Manquinho. Pseudo-boolean and cardinality constraints. In *Handbook of Satisfiability*, pages 695–733. 2009.