Anytime Weighted MaxSAT with Improved Polarity Selection and Bit-Vector Optimization

Alexander Nadel
Intel Corporation, P.O. Box 1659, Haifa 31015 Israel
Email: alexander.nadel@intel.com

***Abstract*—This paper introduces a new anytime algorithm for Weighted MaxSAT consisting of two main algorithmic components. First, we propose a new efficient polarity selection heuristic and an enhancement to the variable decision heuristic for SAT-based anytime Weighted MaxSAT solving (and, more generally, for solving any optimization problem with a SAT-based anytime algorithm). Second, we enhance an existing Bit-vector Optimization-based algorithm for solving Unweighted MaxSAT and generalize it to Weighted MaxSAT. Our resulting Weighted MaxSAT solver outscores the state-of-the-art solvers in the settings of both the 60-second and 300-second weighted incomplete tracks of MaxSAT Evaluation 2018. In addition, we describe a new application of incremental anytime Weighted MaxSAT solving: placement at the physical design stage of Computer-aided Design (CAD).**

I. Introduction

Weighted (Partial) MaxSAT (known as simply *MaxSAT*) is a well-studied optimization problem. Given a set of *hard* propositional clauses H and weighted *soft* clauses S, a MaxSAT solver is expected to find a model that minimizes the overall weight of the unsatisfied soft clauses. When the weight of all the soft clauses is 1, the problem is referred to as *Unweighted (Partial) MaxSAT*.

MaxSAT has a plethora of applications, including a variety of applications in Computer-aided Design (CAD). For example, [44] applies MaxSAT for solving three different problems in software analysis: automated verification, interactive verification, and static bug detection. MaxSAT is applied for post-silicon fault localization in [47]. Furthermore, MaxSAT is used for improving the performance and accuracy of error localization in [47] and [27] as well as for finding concurrency bugs and recommending repairs in [27]. [20] uses MaxSAT to detect malware in Android apps. A variety of other MaxSAT applications are listed in [35], [44].

Our own interest in MaxSAT stems from the following application at the physical design stage [43] of the CAD flow at Intel.

Assume that after a placement of standard cells has already been generated, a new set of design constraints, introduced late in the process, has to be taken into account by the placement flow. Re-running the placer from scratch with the new set of constraints would not satisfy backward compatibility, stability, and run-time requirements, hence a post-processing fixer tool is required. The goal of the fixer is to fix as many as possible of the violations resulting from applying the additional design constraints. Our previous work [40] describes this problem in detail. We considered a particular flavor of the problem, where the violations are arranged in a strict lexicographical order, in which case the problem can be efficiently reduced to Bit-Vector Optimization (OBV). Recently we encountered another flavor of the placement problem with the following three new major requirements.

First, instead of the lexicographical order, each violation can now be associated with an *integer weight*, reflecting the actual cost in dollars (induced by various physical properties of the circuit, such as resistance and capacitance) of leaving that violation un-fixed. This requirement makes the problem directly reducible to Weighted MaxSAT rather than to OBV.

Second, the tool is expected to be *incremental* to support the following two features: *a*) The violations might be divided into different lexicographically ordered classes, where in each class the violations might have different weights. This problem is directly reducible to Bounded Multilevel Optimization (BMO) [6], which itself can be easily reduced to incremental MaxSAT [6], [39]; *b*) Incrementality is required to enable What-if analysis capabilities, that is, to enable the possibility of undoing the changes made by the placer (e.g., to try out different weight modelings).

Third, the tool must be *anytime*. Recall that an anytime algorithm is expected to return a valid solution to a problem even if interrupted. An anytime algorithm for a SAT-based optimization problem (that is, a discrete optimization problem, reducible to SAT, such as MaxSAT) is expected to find an *improving* set of models (that is, satisfying assignments) $\{\mu_1, \mu_2, \ldots, \mu_n\}$ over time, that is, each μ_i is expected to be better than μ_{i-1} in terms of the problem's optimization goal. The anytime property is essential for our application, since it allows the user to get an approximate solution even for very difficult instances. Furthermore, experimental results will demonstrate that the performance of two leading non-anytime solvers does not suit the needs of our application.

In order to meet the three requirements above, we decided to enhance our `Mrs. Beaver` algorithm [39], designed for solving Unweighted MaxSAT, to Weighted MaxSAT. `Mrs. Beaver` is already anytime and incremental, which makes it an excellent candidate for such an enhancement. In fact, `Mrs. Beaver` is the only incremental MaxSAT algorithm we are aware of. The resulting algorithm – `Weighted Mrs. Beaver` (`WMB`) – is able to meet all the above-mentioned requirements and has been productized at Intel.

The MaxSAT community has been evaluating anytime MaxSAT algorithms since 2011, when the so-called *Incomplete* MaxSAT Evaluation tracks, where the instances are hard, the time-out is short, and optimality is not required, emerged.[1]

We were curious to compare `WMB` in non-incremental mode against the best performing solvers of the two (non-incremental) Weighted Incomplete tracks of the latest MaxSAT Evaluation 2018 – the 60-second-track and the 300-second-track – on the diverse set of instances used in the Evaluation.

Remarkably, `WMB` was able to considerably outscore the state-of-the art solvers in the settings of both tracks. Further analysis has shown that this result was enabled by two novel independent algorithmic components. This paper will mostly be dedicated to introducing and studying them:

- We introduce a new efficient polarity selection heuristic `Target-Optimum-Rest-Conservative` *(*`TORC`*)* for anytime SAT-based optimization (and, in particular, Weighted MaxSAT). We also propose a variable decision heuristic modification for anytime SAT-based optimization, called `Target-Score-Bump` *(*`TSB`*)*. We will see that `TORC` is highly beneficial not only in our new `WMB` algorithm, but also in another leading anytime Weighted MaxSAT algorithm, implemented in the winner of the Weighted-Incomplete-60-Second track of MaxSAT Evaluation 2018–`Open-WBO-Inc-BMO` [25].
- We improve the OBV-based `Mrs. Beaver` algorithm [39] and generalize it from Unweighted MaxSAT to Weighted MaxSAT. As we have mentioned, we call the resulting algorithm `Weighted Mrs. Beaver` *(*`WMB`*)*.

The rest of this paper is organized as follows. Sect. II provides the necessary background. Sect. III introduces our polarity selection heuristic `Target-Optimum-Rest-Conservative` (`TORC`) and the modification to the variable decision heuristic `Target-Score-Bump` (`TSB`). Sect. IV introduces `Weighted Mrs. Beaver` (`WMB`). Sect. V demonstrates that `WMB` with `TORC` and `TSB` outperforms the state-of-the-art solvers in the settings of both the weighted incomplete tracks of MaxSAT Evaluation 2018. We also show that `TORC` is highly beneficial within both `WMB` and our implementation of `Open-WBO-Inc-BMO`'s Weighted MaxSAT algorithm. Furthermore, we study the performance of several algorithms on a number of non-incremental instances from our placement application. Sect. VI concludes our work.

II. Background

Let us start with some basic terminology. A *literal* l is a Boolean variable v or its negation $\neg v$. A *clause* is a disjunction of literals. A formula F in *Conjunctive Normal Form (CNF)* is a conjunction (set) of clauses.

A Weighted MaxSAT instance comprises a set of *hard* satisfiable clauses H and a set of weighted *soft* constraints $T = \{t_{n-1}, t_{n-2}, \dots, t_0\}$, where each constraint t_i is associated with a strictly positive integer weight w_i. The *weight of a variable assignment* μ is *unsWt*$(T, \mu) = \sum_{i=0}^{n-1} \neg\mu(t_i) \times w_i$, that is, the overall weight of T's bits, falsified by μ. Given a Weighted MaxSAT instance, a Weighted MaxSAT solver is expected to return a model having the minimum possible weight. For the rest of the paper, for convenience and without restricting generality, it is assumed that every soft constraint is a unit clause (that is, a clause containing one literal).[2] Thus, T can be thought of as a bit-vector, where t_0 is its Least Significant Bit (LSB) and t_{n-1} is its Most Significant Bit (MSB). T is called the *target bit-vector*, or, simply, the *target* and every $t_i \in T$ is called a *target bit*. A (possibly partial) variable assignment μ is *ideal* if $\mu(t_i) = 1$ for every target bit $t_i \in T$.

A. Anytime MaxSAT Solving

We should mention first that anytime algorithms have been explored in the context of solving SAT-based optimization problems other than MaxSAT. For example, `PREF-DLL` [18] is an anytime algorithm for the problem of SAT with preferences. `PREF-DLL`'s authors highlight its "anytime" property. Other more recent examples of anytime algorithms for SAT-based optimization problems include branch-and-bound for sports league scheduling within `Barcelogic` [42] and Bit-Vector Optimization (OBV) [40].

Arguably the most well-known anytime MaxSAT algorithm is the so-called Linear Search SAT-UNSAT (`LSU`) [9]. `LSU` starts by invoking a SAT solver over the hard clauses to find the first approximation μ_1. Assume the overall weight of the unsatisfied soft clauses in μ_1 is w. Then, `LSU` blocks all the solutions of weight $>= w$ using Pseudo-Boolean (PB) or cardinality constraints and invokes the SAT solver again to find a better approximation. The process continues until the problem becomes unsatisfiable. The last encountered model is guaranteed to be optimal.

Plain `LSU` was still one of the best-performing anytime MaxSAT algorithms as of 2017, when the Unweighted-Incomplete-60-Second track of MaxSAT Evaluation 2017 [4] was won by an `LSU`-based solver, `Open-WBO-LSU` [32]. Local search-based [12], [13], [30] and Minimal Correction Subset (MCS) enumeration-based [31], [33] approaches to anytime MaxSAT solving have also been explored. Moreover, implicit hitting set-based MaxSAT solvers [14] can also be used for anytime solving.

The latest MaxSAT Evaluation 2018 is evidence that anytime MaxSAT development is becoming a hot research topic, as a number of new solvers that efficiently implemented existing and new ideas stood out:

[1]MaxSAT Evaluations have been held since 2006, but did not have the "anytime" Incomplete category till 2011.

[2]An arbitrary soft constraint t_i, reducible to a set of clauses F, can be transformed to a unit clause s', where s' is a fresh variable, by adding the clause $\neg s' \vee c$ to H for each clause $c \in F$.

1) `SATLike` and `SATLike-c` [29]: both solvers apply a recently introduced local search-based algorithm, where `SATLike-c` also uses `Open-WBO-LSU` in case of a failure of its main engine.
2) `LinSBPS` [16] is based on `Open-WBO-LSU`, upgraded with conservative polarity selection (see Sect. III for more detail) and weight descaling.
3) `Open-WBO-Inc-OBV` [24] (unweighted only) applies the OBV-based `Mrs. Beaver` algorithm [39] (see Sect. IV for more detail).
4) `Open-WBO-Inc-MSC` [24] (unweighted only) uses MCS enumeration.
5) `Open-WBO-Inc-BMO` [24] (weighted only) approximates Weighted MaxSAT with Bounded Multilevel Optimization (BMO) [6].
6) `Open-WBO-Inc-Cluster` [24] (weighted only) applies weight clustering [25].

B. Polarity Selection for SAT-based Optimization

Recall that modern SAT solvers apply phase saving [41] as their polarity selection heuristic. In phase saving, once a variable is picked by the variable decision heuristic, the literal is chosen according to its latest value, where the values are normally initialized with 0.

It turned out that overriding phase saving in the context of anytime SAT-based optimization algorithms, which generate an improving set of models $\{\mu_1, \mu_2, \ldots, \mu_n\}$ over time, is advantageous. One can classify the functionality of existing polarity selection algorithms, after a new model μ_i is found, according to the following criteria:

- `optimistic` or `conservative`: Set the polarity to an ideal assignment or to the previous best one, respectively. The optimistic approach sets the polarity of the target bits to 1; it works well when the actual solution is close to the ideal one [9], [40]. The conservative approach was shown to simulate local search solvers, where small incremental perturbations are performed on the best known solution [15], [16].
- `set-once` or `fix`: Set the polarity once and let the solver change it afterward or fix it, respectively. Fixing the polarity of the variable v to $b \in \{0, 1\}$ means that wherever v is chosen as a decision, its polarity is set to b. The technique of fixing all the polarities to a certain assignment M in order to look for a solution near M was first applied in [1], [38] in the context of diverse solution generation. Setting the polarity once means applying phase saving whenever v is picked later.
- `all-variables`, `original-variables` or `target-variables`: Set the polarity, respectively, for either: *a*) all the variables in the problem, or *b*) all the original variables (that is, exclude any variable, introduced to encode Pseudo-Boolean or cardinality constraints), or *c*) only the variables representing the target bits (that is, the soft constraints).

In addition, one may also set-once or fix the target bits to 1 prior to the initial SAT invocation.

The following approaches have been proposed to modify the default polarity selection heuristic of the SAT solver in the context of anytime SAT-based optimization:

- `Barcelogic` [42] (and, later, WPM3 [5]): After a new model μ_i is found, `Barcelogic` fixes the polarity of all the original variables to their values in μ_i. This approach can be classified as (`conservative,fix,original-variables`).
- `LinSBPS` [16]: Similar to `Barcelogic`, but the variables are not restricted to the original ones, hence this approach can be classified as (`conservative,fix,all-variables`). Note that `LinSBPS` won the Weighted-Incomplete-300-Second track of MaxSAT Evaluation 2018. `LinSBPS`'s polarity selection heuristic is mentioned as the key to `LinSBPS`'s successful performance [16].
- `Sat4j` [9]: The polarity of the target bits is set-once to 1 after a new model is encountered and also before the initial SAT invocation. This heuristic can be classified as (`optimistic,set-once,target-variables`). It is also used in the `OBV-BS` algorithm for solving the OBV problem [40].
- `SAT&PREF` [17]: A stronger version of `Sat4j`'s heuristic: whenever a target bit variable is selected by the variable decision heuristic, use 1 as its polarity. The heuristic can be classified as (`optimistic,fix,target-variables`).

C. Bit-Vector Optimization (OBV)

The problem of Bit-Vector Optimization (OBV), also known as Optimization Modulo Bit-Vectors (OMT(BV)), is a SAT-based optimization problem where, given a set of hard clauses H and the target bit-vector T, the algorithm has to maximize the *value* of T (where T is interpreted as an unsigned integer). Note that in OBV, unlike in Unweighted MaxSAT, the order of the target bits matters: informally, satisfying t_i is more important than satisfying *all* of the bits t_j for $j < i$.

OBV was first explicitly analyzed in [40], but it is closely related to other SAT-based optimization problems. Specifically, OBV can be seen as a special case of the problem of SAT with preferences, first discussed a decade earlier [17], [21]. In addition, Lexicographic SAT (LEXSAT) [28] and OBV are essentially the same problem. Furthermore, OBV can easily be reduced to Weighted MaxSAT [10], though solving OBV using such a reduction does not work well in practice [40].

Now we describe the `OBV-BS` OBV algorithm, shown in Alg. 1. `OBV-BS` was concurrently proposed in [40] and [28]. It serves as the basic building block for both the Unweighted MaxSAT `Mrs. Beaver` algorithm [39] and our new Weighted MaxSAT `WMB` algorithm. Alg. 1 leaves out `OBV-BS`'s polarity selection heuristic [40], discussed above in Sect. II-B.

Alg. 1 maintains the currently best-known model μ, initialized by an arbitrary model provided by the user. Assume for now that the user provides the algorithm a model encountered by a SAT solver invoked over the hard clauses. Alg. 1 also maintains a partial assignment α, which contains the target bits, whose values in any optimal model are already known. The main loop of the algorithm (starting at line 3) goes over all the target bits starting from the MSB t_{n-1} down to t_0, where each iteration extends α with either t_i or $\neg t_i$ and updates μ, if possible. Consider an arbitrary iteration i. If t_i is 1 in μ, α is simply extended by t_i; otherwise, the algorithm checks whether there exists a model with $t_i = 1$ by invoking a SAT solver under the assumptions $\alpha \cup \{t_i\}$. If such a model exists, α is updated with t_i and μ is updated with the new model; otherwise, α is updated with $\neg t_i$. The algorithm returns μ at the end.

We also proposed in [39] to use a *conflict threshold* to limit each SAT invocation. The idea is to prevent the algorithm from spending too much time on a single bit (by giving up guaranteeing the optimal result). If the threshold is exceeded on a certain bit, the algorithm continues as if the invocation were unsatisfiable.

In addition, [39] presented a modification of OBV-BS, denoted by UBS (UMS-OBV-BS in [39]). UBS modifies the order of T's bits by pushing any satisfied bits towards the MSB after a new model is found. UBS does not solve OBV anymore and instead is designed to be used for approximating Unweighted MaxSAT.

Algorithm 1 OBV-BS

```
1: function SOLVE(CNF Formula $F$, Target $T = \{t_{n-1}, t_{n-2}, \ldots, t_0\}$, Model $\mu$ to $F$)
Require: $\mu$ satisfies $F$
2:     $\alpha := \{\}$
3:     for $i \leftarrow n-1$ downto 0 step 1 do
4:         if $t_i \in \mu$ then                ▷ $t_i \in \mu \equiv t_i = 1$ in $\mu$
5:             $\alpha := \alpha \cup \{t_i\}$
6:         else
7:             $\tau :=$ SATUNDERASSUMPTIONS($\alpha \cup \{t_i\}$)
8:             if satisfiable then $\alpha := \alpha \cup \{t_i\}$; $\mu := \tau$ else $\alpha := \alpha \cup \{\neg t_i\}$
9:     return $\mu$
```

D. Mrs. Beaver *Algorithm*

Here we review the Mrs. Beaver algorithm for solving Unweighted MaxSAT [39].

We start with a definition. Given a bit-vector $S = \{s_{n-1}, s_{n-2}, \ldots, s_0\}$, the *totalizer* [8] is a binary tree whose top-most node, *tot*(S), is a bit-vector representing the sum of S's bits in unary representation. Intuitively, the totalizer can be thought of as an implementation of a full-adder summing all the bits of S, where the result–*tot*(S)–is in unary representation.

We denote by *tot*(S, b) the top-most node of the totalizer, which additionally encodes the cardinality constraint $\sum_{i=0}^{n-1} s_i \leq b$ (by restricting all the nodes to $b+1$ bits and asserting that the most significant bit is 0 for all the nodes having $b+1$ bits). The smaller b is, the more efficient the encoding of the totalizer is [11].

Alg. 2 presents the OBV-BS-based Mrs. Beaver algorithm for Unweighted MaxSAT. Consider its incomplete part (lines 3 to 6), intended to approximate Unweighted MaxSAT with OBV. Each iteration (where the number of iterations k is user-given) invokes either OBV-BS or UBS over the target T and gets a model μ'. The best model μ is updated with T whenever OBV-BS or UBS (the choice being based on a user-given parameter) is able to improve the model w.r.t MaxSAT's optimization goal. Although not shown in Alg. 2, it is important to mention that it is assumed that OBV-BS or UBS returns the best model encountered *according to the optimization goal of MaxSAT* (rather than OBV). At the end of each iteration, T's bits are either reversed or shuffled (based on a user-given parameter) in hope of getting a better approximation at the next iteration.

The complete part of Alg. 2 (lines 7, 8) proceeds as follows. It creates the totalizer $T' :=$ *tot*$(\neg T, \leq$ *unsWt*$(T, \mu))$, which sums up the unsatisfied bits of T using *unsWt*(T, μ) as the upper bound (where F is assumed to be updated with the totalizer's clauses). The algorithm then invokes OBV-BS over $\neg T'$ to minimize the sum of the unsatisfied bits in T and returns the resulting model. It is argued in [39] that the complete part of the algorithm is nothing but an LSU implementation. It is guaranteed to find an optimal solution.

[39] demonstrates that Mrs. Beaver is an incremental algorithm: it can be re-used across multiple invocations with different hard assumptions and target bit-vectors.

Algorithm 2 Mrs. Beaver

```
1: function SOLVE(CNF Formula $F$, Target $T = \{t_{n-1}, t_{n-2}, \ldots, t_0\}$)
Require: $F$ is satisfiable
2:     $\mu :=$ SAT()
3:     for $i \leftarrow 1$ to $k$ step 1 do
4:         $\mu' :=$ OBV-BS($F, T, \mu$) or UBS($F, T, \mu$)
5:         if unsWt(T,$\mu'$) < unsWt(T,$\mu$) then $\mu := \mu'$
6:         $T :=$ reverse(T) or shuffle(T)
7:     $T' :=$ tot($\neg T, \leq$ unsWt($T, \mu$))        ▷ $F$ is updated
8:     return OBV-BS($F, \neg T', \mu$)
```

III. Polarity and Variable Selection for Anytime SAT-based Optimization

This section introduces our new polarity selection heuristic and a modification to the variable decision heuristic for anytime SAT-based optimization.

A. *Target-Optimum-Rest-Conservative (*TORC*)*

We propose a new polarity selection heuristic, which we call *Target-Optimum-Rest-Conservative (*TORC*)*.

Before the initial SAT invocation, TORC *fixes* the polarity of all the *target* variables to the ideal value 1. Then, after each new improving model μ_i is encountered, the polarity of *all* the *non-target* variables are fixed to their values in μ_i.

In other words, whenever the variable decision heuristic chooses:

1) A target variable: TORC sets its polarity to 1 (to be optimistic).
2) A non-target variable: TORC sets its polarity to its value in the best model so far (to be conservative; only after the first SAT invocation is completed)

Note that, after the initial SAT call, TORC sets the polarity every time a new decision variable is picked, thus completely overriding phase saving.

TORC has been designed to leverage the best of both the conservative and the optimistic worlds. On one hand, we are interested in taking advantage of the conservative heuristic, which is known to find the next improved model more quickly than the default heuristic by looking near the previous model [15], [16]. At the same time, however, we would like to encourage the values of the target variables to be as close to the ideal as possible in order to move more quickly towards the optimum [17], [40].

Obviously, TORC can be applied for anytime MaxSAT solving. More generally, it can be used in any anytime SAT-based optimization algorithm (such as, the OBV-BS algorithm for solving OBV or the PREF-DLL algorithm for the problem of SAT with preferences).

B. Target-Score-Bump *(*TSB*)*

We aim to experiment with tuning the SAT solver's variable selection heuristic for anytime SAT-based optimization.

Modern SAT solvers use variants of the VSIDS variable decision heuristic [37] as part of their decision strategy. VSIDS associates a score with every variable and picks as the next decision the variable with the greatest score. Furthermore, many solvers inherited from Minisat [19] the API for bumping up the score of a given variable, thus increasing its chances of being selected soon.

Our proposed heuristic–Target-Score-Bump *(*TSB*)*– bumps up the variable scores of the *target bit* variables, so as to improve their chances of being picked early. Specifically, for each target bit variable, we bump up its score as if it participated in a conflict clause. We apply TSB only once prior to the initial SAT invocation. Our intuition is that TSB, in conjunction with TORC (or any other optimistic polarity selection heuristic), will improve the likelihood of the initial model being close to the ideal one.

IV. Weighted Mrs. Beaver (WMB) Weighted MaxSAT Algorithm

This section introduces our Weighted Mrs. Beaver *(*WMB*)* algorithm, which improves Mrs. Beaver and generalizes it to Weighted MaxSAT solving. WMB is compliant with any of the polarity and variable selection heuristics presented in Sect. II-B and Sect. III. In addition, WMB inherits the property of being incremental from Mrs. Beaver.

We need an additional definition. The *Generalized Totalizer (GT)* [26] is a generalization of the totalizer encoding for the weighted case. GT is identical to the totalizer when all the weights are 1. For the general case, given a bit-vector $S = \{s_{n-1}, s_{n-2}, \dots, s_0\}$ and a weight $w_i > 0$ for each $s_i \in S$, the GT is a binary tree whose top-most node $\mathit{tot}(S)$ is a bit-vector whose bits comprise all the partial sums of the weights associated with S's bits. $\mathit{tot}(S)_i$ holds iff the sum of the weights of the satisfied input bits is no less than the partial sum associated with $\mathit{tot}(S)_i$. $\mathit{tot}(S)$'s bits are sorted so that the most significant bit is associated with the greatest weight. We denote by $\mathit{tot}(S, b)$ the top-most node of the GT, which additionally encodes the PB constraint $\sum_{i=0}^{n-1} s_i \times w_i \leq b$ (by restricting the top-most node to bits with weights less than or equal to $\sum_{i=0}^{n-1} s_i \times w_i$, except for the most significant bit, which is asserted to 0).

1) Enhancing Mrs. Beaver*:* First we will introduce several enhancements to the basic Mrs. Beaver algorithm, which will be inherited by WMB:

(a) *Global stopping condition for* OBV-BS: during the incomplete stage, Mrs. Beaver always maintains μ–the best model so far w.r.t minimizing $\mathit{unsWt}(T, \mu)$. OBV-BS, on the other hand, maintains μ'–the best model so far w.r.t to maximizing the *value* of its target. During OBV-BS's execution, it may happen that $\mathit{unsWt}(T, \mu')$ is already as large as $\mathit{unsWt}(T, \mu)$. In this case, no further progress can be made by OBV-BS, hence we propose stopping OBV-BS's invocation at this point. Our stopping condition is also relevant for UBS.

(b) Size-based Switching to Complete Part *(*SSCP*)*: Mrs. Beaver has an intrinsic dilemma as to when (and whether) to switch to the complete part (or, otherwise, to continue the incomplete iterations). In [39]'s experiments, the complete part only marginally improved performance. We propose therefore to switch to the complete part after *gtI* iterations, where *gtI* is a user-given parameter, but only if the number of clauses expected to be generated by the totalizer encoding is not greater than the user-given threshold *gtThr* (otherwise, continue with the incomplete iterations until the time-out).

2) Generalizing Mrs. Beaver *to Weighted MaxSAT:* Now we show how to generalize Mrs. Beaver to Weighted MaxSAT solving. Our intuition is that OBV-BS, which takes the order of target bits into consideration, will serve as a *better* approximation for Weighted MaxSAT than for Unweighted MaxSAT. This is because one can order the target bits to reflect their weights, thus causing OBV-BS to start with the bits that weigh more and to progress towards the bits that weight less.

Alg. 3 presents our WMB algorithm. Syntactically, it is very similar to Alg. 2. Let us go over the syntactic and semantic differences:

(a) *Using the GT*: We use the GT instead of the totalizer for the complete part of the algorithm (line 9; change of semantics only). Note that the GT inherits the following desired property from the totalizer: running OBV-BS over the bits in the GT's top-most node is nothing but an LSU invocation.

(b) *Sorting T*: After the initial SAT invocation, WMB sorts the target bits according to their weights (line 3).

(c) Size-based Switching to Complete Part

(SSCP) *implementation*: The incomplete part of Mrs. Beaver loops for a user-given number of iterations k. WMB, on the other hand, stops the loop if the GT encoding is expected to be compact enough (that is, if the number of clauses expected to be generated by the encoding is less than a user-given threshold *gtThr*), but only after *gtI* iterations (line 8).

(d) *Dropping* UBS: UBS reorders the bits of OBV-BS's target without regard to their weights. Such an optimization is not expected to be useful in weighted settings (line 5).

(e) *Shuffle generalization*: Instead of shuffling the bits randomly, we need to take their weights into account. Hence we apply *weighted random shuffling* (line 7; change of semantics only).

(f) *Reverse strategy update*: Instead of reversing all the bits in T (which might ruin our Weighted MaxSAT approximation), we only reverse the bits in any block of adjacent bits in T which have the same weight. In addition, in order to impose some target-bit order perturbations even if blocks having the same weight are rare or nonexistent, we go over T from the MSB towards the LSB and, if both the target bits t_{i+1} and t_i, for any i, do not share weights with any of their neighbors, we reverse the order of t_i and t_{i+1} (line 7; change of semantics only).

In our implementation, WMB reverses the target at iterations $1, 3, 5, \dots$ and shuffles it at iterations $2, 4, 6, \dots$ (we start iteration numbering with 1). In addition, we use 10,000 as the conflict threshold per bit for OBV-BS when it is invoked over the original target during the incomplete part of WMB (line 5).

Recall from Sect. II-B that the optimistic (or mixed) polarity selection heuristics SAT&PREF, OBV-BS and TORC set the bits of the target to 1. For the sake of these heuristics, the complete stage of WMB uses the bits of the *original* target (rather than the top-most node of the GT).

Algorithm 3 Weighted Mrs. Beaver (WMB)

```
1:  function SOLVE(CNF Formula $F$, Target $T = \{t_{n-1}, t_{n-2}, \dots, t_0\}$, Weight $w_i > 0$ for each $t_i \in T$)
Require: $F$ is satisfiable
2:      $\mu :=$ SAT()
3:      Sort $T$ by weights (to have $w_i \geq w_j$ for any $i > j$)
4:      loop
5:          $\mu' :=$ OBV-BS$(F, T, \mu)$
6:          if unsWt(T,$\mu'$) < unsWt(T,$\mu$) then $\mu := \mu'$
7:          $T :=$ reverse(T) or shuffle(T)
8:          if gtI iterations passed and $|tot(\neg T, \leq unsWt(T, \mu))| <$ gtThr then break
9:      $T' := tot(\neg T, \leq unsWt(T, \mu))$          ▷ $F$ is updated
10:     return OBV-BS$(F, \neg T', \mu)$
```

V. Experimental Results

Subsection V-A below evaluates different algorithms on the MaxSAT Evaluation instances, while Subsection V-B is concerned with industrial instances from our placement application in the physical design stage of CAD.

A. MaxSAT Evaluation

The goal of this subsection is twofold.

First, we examine the performance of our new WMB algorithm and compare it to the performance of the three leading anytime Weighted MaxSAT solvers– Open-WBO-Inc-BMO, LinSBPS and maxroster [45]–in settings similar to the Weighted-Incomplete-60-Second and Weighted-Incomplete-300-Second tracks of MaxSAT Evaluation 2018. In addition, we study the overall impact of Size-based Switching to Complete Part (SSCP) with different combinations of SSCP's two parameters *gtI* and *gtThr* (recall Sect. IV-1).

Second, we isolate and study the impact of the TORC polarity selection heuristic and TSB variable score scheme within two algorithms: WMB and BMO, where BMO stands for our implementation of the BMO-based algorithm implemented in Open-WBO-Inc-BMO, the winner of the Weighted-Incomplete-60-Second track of MaxSAT Evaluation 2018. We also considered implementing the algorithm of LinSBPS– the winner of the Weighted-Incomplete-300-Second track–but unfortunately it has not been described in enough detail (in particular, the details of its weight descaling technique have not been revealed).

Let WMB (P,TSB,*gtI*,*gtThr*) be an invocation of WMB with the polarity selection heuristic P (where P = - means the default polarity selection heuristic is applied), with or without TSB (for TSB = $\top$ or $\bot$, respectively) and using the parameters *gtI* and *gtThr* (where we use - for both *gtI* and *gtThr* if the algorithm always switches to the complete part after 10,000 iterations, similarly to Mrs. Beaver). Let BMO (P,TSB $\in \{\top, \bot\}$) be an invocation of BMO with the polarity selection heuristic P and with or without TSB.

All in all, we launched the following experiments (on machines with 32Gb of memory running Intel® Xeon® processors with 3Ghz CPU frequency; the log files are available at https://tinyurl.com/y5a4eqom) for 60 and 300 second timeouts (in the algorithm below, BL, Lin and PREF stand for Barcelogic, LinSBPS and SAT&PREF, respectively):

```
1:  Run Open-WBO-Inc-BMO
2:  Run LinSBPS
3:  Run maxroster
4:  for all P $\in \{-, BL, Lin, PREF, Sat4j, TORC\}$ do
5:      for all TSB $\in \{\top, \bot\}$ do
6:          Run BMO (P,TSB)
7:          Run WMB (P,TSB,-,-)
8:          for all gtI $\in \{0, 1\}$ do
9:              for all gtThr $\in \{10^6, 10^7\}$ do
10:                 Run WMB (P,TSB,gtI,gtThr)
```

The main criteria for comparing anytime MaxSAT solvers during the MaxSAT Evaluation 2018 was their *score*, defined as follows for a particular solver S and i instances: $\sum_i (1 +$ the minimal weight of the unsatisfied target bits found by any participating solver) / (1 + the weight of the unsatisfied target bits found by S). The drawback impeding the usage of this criteria outside of the Evaluation is that the score changes,

TABLE I: Overall Performance Comparison

60-Second Timeout		300-Second Timeout	
Solver	Score	Solver	Score
WMB (TORC,$\bot$,1,10^7)	0.8700	WMB (TORC,$\top$,0,10^6)	0.9041
WMB (TORC,$\top$,0,10^6)	0.8616	LinSBPS	0.8916
LinSBPS	0.8303	WMB (TORC,$\bot$,1,10^7)	0.8903
BMO (TORC,$\top$)	0.8235	BMO (TORC,$\top$)	0.8657
Open-WBO-Inc-BMO	0.8202	Open-WBO-Inc-BMO	0.8391
maxroster	0.7885	maxroster	0.8169

TABLE II: Polarity Selection within WMB. The baseline WMB configuration is WMB (TORC,$\bot$,0,10^6).

60-Second Timeout		300-Second Timeout	
Polarity	Score	Polarity	Score
TORC	0.8616	TORC	0.9041
SAT&PREF	0.8240	LinSBPS	0.8700
LinSBPS	0.8226	Barcelogic	0.8683
Sat4j	0.8220	SAT&PREF	0.8499
Barcelogic	0.8184	Sat4j	0.8479
-	0.7734	-	0.8166

depending on the participating solvers.

To overcome this drawback, we calculated an absolute score against the following static criteria. We run the three MaxSAT Evaluation winners in the Weighted-Complete track (rc2b, rc2a [23], [34], [36], maxino [2], [3]) and the three MaxSAT Evaluation winners in the Weighted-Incomplete-300-Second track (LinSBPS, Open-WBO-Inc-BMO, maxroster) for 24 hours over all the instances (results available at https://tinyurl.com/y5a4eqom). Let the *optimal weight* for a particular instance be the minimal weight found in the above experiment. We define the score as follows: $\sum_i$(1 + the optimal weight) / (1 + the weight of the unsatisfied target bits found by S). None of the solvers was able to find any solution for 5 of the 172 instances, so we excluded those instances from the main experiments.

B. Overall Performance Comparison

Table I shows the absolute scores of the following solvers for both the 60-second and 300-second timeouts, where the following solvers are sorted by their scores: *a*) The three best-performing solvers in the Weighted-Incomplete-60-Second and Weighted-Incomplete-300-Second tracks of MaxSAT Evaluation; *b*) The two best configurations of WMB: one per each time-out; *c*) The best configuration of BMO (a single BMO configuration performed the best for both time-outs).

WMB emerges as the winner for both time-outs, where the gap against LinSBPS is larger for the 60-second time-out. Based on these results, for the rest of the paper WMB refers to WMB (TORC,$\top$,0,10^6), and BMO refers to BMO (TORC,$\top$). The results on the 60-second time-out suggest that, in the future, it might be worth combining LinSBPS and WMB, e.g., by applying WMB *instead* of LSU as LinSBPS's back-end.

C. Polarity Selection

Tables II and III show the impact of varying the polarity selection within WMB and BMO, respectively. TORC emerges as the best heuristic for both algorithms by far.

TABLE III: Polarity Selection within BMO. The baseline BMO configuration is BMO (TORC,$\top$).

60-Second Timeout		300-Second Timeout	
Polarity	Score	Polarity	Score
TORC	0.8235	TORC	0.8657
SAT&PREF	0.7903	SAT&PREF	0.8094
Sat4j	0.7790	LinSBPS	0.8046
LinSBPS	0.7556	Sat4j	0.800
Barcelogic	0.7522	Barcelogic	0.7951
-	0.7122	-	0.7392

TABLE IV: TSB and SSCP Impact on WMB (with TORC)

60-Second Timeout				300-Second Timeout			
TSB	*gtI*	*gtThr*	Score	TSB	*gtI*	*gtThr*	Score
$\bot$	1	10^7	0.8700	$\top$	0	10^6	0.9041
$\bot$	1	10^6	0.8689	$\top$	1	10^7	0.9019
$\bot$	0	10^6	0.8684	$\top$	1	10^6	0.9008
$\bot$	-	-	0.8638	$\top$	0	10^7	0.8972
$\top$	0	10^6	0.8616	$\top$	-	-	0.8971
$\top$	1	10^7	0.8609	$\bot$	1	10^7	0.8903
$\top$	1	10^6	0.8605	$\bot$	0	10^6	0.8901
$\bot$	0	10^7	0.8595	$\bot$	1	10^6	0.8898
$\top$	0	10^7	0.8565	$\bot$	0	10^7	0.8859
$\top$	-	-	0.8560	$\bot$	-	-	0.8854

D. Target Score Bump (TSB) and Size-based Switching to Complete Part (SSCP)

In our experiments, we found that TSB impacts BMO positively, though not by a large margin; when TSB is turned on, the score goes up from 0.8178 to 0.8235 and from 0.8527 to 0.8657 for the 60-second and 300-second time-outs, respectively.

The situation is more complex when it comes to the impact of TSB and SSCP on WMB. Table IV presents the results of running all the combinations of different TSB and SSCP strategies within WMB. The following observations can be made:

1) Applying SSCP is beneficial, but the tuning of its parameters depends on the time-out.
2) TSB's impact depends on the time-out: TSB is useful for the 300-second time-out, but not for the 60-second time-out.

E. Relative Scores

So far, we have presented and analyzed the results, based on the absolute scores. We offer another perspective on the performance by comparing the results of all the solvers shown in Tables I, II, III using the MaxSAT Evaluation criterion, that is, comparing all these solvers one against the other. The results are shown in Table V and Table VI for the 60-second and 300-second timeout, respectively.

Interestingly, unlike in the comparison of the absolute scores, a single WMB configuration–WMB (TORC,$\top$,0,10^6)–comes out as a winner for both time-outs. The gap between WMB and LinSBPS is still more significant for the 60-second timeout. WMB, TORC and TSB are all essential for enabling our solver to outscore LinSBPS, while both TORC and TSB are beneficial also for BMO.

TABLE V: Head-to-Head Comparison: 60-Second Timeout

Polarity	Score
WMB (TORC,$\top$,0,10^6)	0.913061517
WMB (TORC,$\bot$,1,10^7)	0.908857186
BMO (TORC,$\top$)	0.874093999
LinSBPS	0.868803544
WMB (SAT&PREF,$\top$,0,10^6)	0.858081393
Open-WBO-Inc-BMO	0.857702699
WMB (Sat4j,$\top$,0,10^6)	0.852056198
WMB (LinSBPS,$\top$,0,10^6)	0.851883838
WMB (Barcelogic,$\top$,0,10^6)	0.847634354
BMO (SAT&PREF,$\top$)	0.827579748
BMO (Sat4j,$\top$)	0.811722622
maxroster	0.811692104
BMO (LinSBPS,$\top$)	0.791335557
WMB (-,$\top$,0,10^6)	0.790396535
BMO (Barcelogic,$\top$)	0.785587982
BMO (-,$\top$)	0.730440952

TABLE VI: Head-to-Head Comparison: 300-Second Timeout

Polarity	Score
WMB (TORC,$\top$,0,10^6)	0.922435560
LinSBPS	0.909276301
WMB (TORC,$\bot$,1,10^7)	0.904152238
BMO (TORC,$\top$)	0.891196369
WMB (LinSBPS,$\top$,0,10^6)	0.885370780
WMB (Barcelogic,$\top$,0,10^6)	0.883654070
WMB (SAT&PREF,$\top$,0,10^6)	0.864222104
Open-WBO-Inc-BMO	0.860424891
WMB (Sat4j,$\top$,0,10^6)	0.860151932
WMB (-,$\top$,0,10^6)	0.825258820
BMO (SAT&PREF,$\top$)	0.824456819
maxroster	0.824387757
BMO (LinSBPS,$\top$)	0.822867873
BMO (Sat4j,$\top$)	0.813110679
BMO (Barcelogic,$\top$)	0.809867421
BMO (-,$\top$)	0.748293704

F. Industrial Application

This section analyzes the performance of several solvers on a number of instances from our industrial application, described in Sect. I.

For our experiments, we used the two winners of the Complete (non-anytime) category of MaxSAT Evaluation 2018 – rc2b [23], [34], [36] and maxino [2], [3] – in order to test the performance of leading non-anytime solvers on our instances. In addition, we ran our WMB algorithm and the best-performing anytime solver – LinSBPS (it outperformed other anytime solvers on our instances in preliminary experiments).

Note that we could not test the performance of MaxSAT solvers, other than WMB, inside our incremental flow, since none of the other algorithms is incremental. Instead, we isolated a number of non-incremental instances of our problem. In order to evaluate the solution quality of the different solvers over time, we ran the solvers for 9 different timeouts: 30, 60, 300, 900, 1800, 3600, 7200, 18000 and 36000 seconds.

Table VII shows the results of our experiment. Several observations can be made:

1) The leading complete solvers do not perform suitably in our context.
2) LinSBPS and WMB are roughly comparable, with a slight advantage for LinSBPS. Note that LinSBPS was able to reach a better solution than WMB for instances 4 and 6, while WMB reached a better solution for instance 7.

TABLE VII: Comparing LinSBPS (Lin), WMB, rc2b (rcb) and maxino (mxn) on 7 Industrial Placement Instances for 9 Different Time-outs. The overall unsatisfied weight is shown for each Instance&Solver&Timeout combination (∞ means that the problem was not solved within the timeout). Cells containing the best known result for each particular instance are highlighted. The instances are sorted from the easiest towards the most difficult one.

Sol-	Timeout								
ver	30	60	300	900	1800	3600	7200	18000	36000
Industrial Instance 1									
Lin	21	21	21	21	21	21	21	21	21
WMB	21	21	21	21	21	21	21	21	21
rcb	21	21	21	21	21	21	21	21	21
mxn	21	21	21	21	21	21	21	21	21
Industrial Instance 2									
Lin	58	58	58	58	58	58	58	58	58
WMB	58	58	58	58	58	58	58	58	58
rcb	∞	∞	58	58	58	58	58	58	58
mxn	∞	58	58	58	58	58	58	58	58
Industrial Instance 3									
Lin	58	58	58	58	58	58	58	58	58
WMB	58	58	58	58	58	58	58	58	58
rcb	∞	∞	∞	58	58	58	58	58	58
mxn	∞	∞	58	58	58	58	58	58	58
Industrial Instance 4									
Lin	66	66	66	66	66	66	66	66	66
WMB	94	94	67	67	67	67	67	67	67
rcb	∞	∞	∞	∞	∞	∞	∞	∞	∞
mxn	105	105	105	105	105	105	105	105	105
Industrial Instance 5									
Lin	102	101	101	101	101	101	101	101	101
WMB	102	102	101	101	101	101	101	101	101
rcb	∞	∞	∞	∞	∞	∞	∞	∞	∞
mxn	∞	∞	∞	∞	∞	∞	∞	∞	∞
Industrial Instance 6									
Svr	30	60	300	900	1800	3600	7200	18000	36000
Lin	97	84	72	72	72	72	72	72	72
WMB	∞	116	116	80	77	73	73	73	73
rcb	∞	∞	∞	∞	∞	∞	∞	∞	∞
mxn	92	92	92	92	92	92	92	92	92
Industrial Instance 7									
Lin	88	85	85	85	85	85	85	85	85
WMB	92	91	90	89	82	82	82	82	82
rcb	∞	∞	∞	∞	∞	∞	∞	∞	∞
mxn	∞	∞	∞	∞	∞	∞	∞	∞	∞

This result serves as another indication that combining LinSBPS and WMB is expected to be beneficial.

VI. Conclusion

This paper offers several contributions. First, we introduced a new efficient polarity selection heuristic for SAT-based anytime optimization algorithms, called Target-Optimum-Rest-Conservative (TORC), and demonstrated its superiority to existing heuristics within two leading anytime Weighted MaxSAT algorithms. In addition, we introduced a variable decision heuristic enhancement, Target-Score-Bump (TSB), whose impact is also positive. Second, we improved the OBV-based Unweighted MaxSAT algorithm Mrs. Beaver, and generalized it to Weighted MaxSAT. Our resulting Weighted MaxSAT solver, Weighted Mrs. Beaver (WMB), outscored the state-of-the-art solvers in the settings of both weighted incomplete tracks of MaxSAT Evaluation 2018. Third, we described a new application of incremental anytime Weighted MaxSAT for placement in the physical design stage of CAD.

References

[1] S. Agbaria, D. Carmi, O. Cohen, D. Korchemny, M. Lifshits, and A. Nadel. Sat-based semiformal verification of hardware. In R. Bloem and N. Sharygina, editors, *Proceedings of 10th International Conference on Formal Methods in Computer-Aided Design, FMCAD 2010, Lugano, Switzerland, October 20-23*, pages 25–32. IEEE, 2010.

[2] M. Alviano. Maxino. In Bacchus et al. [7], page 13.

[3] M. Alviano, C. Dodaro, and F. Ricca. A MaxSAT algorithm using cardinality constraints of bounded size. In Yang and Wooldridge [46], pages 2677–2683.

[4] C. Ansotegui, F. Bacchus, M. Järvisalo, and R. Martins, editors. *MaxSAT Evaluation 2017: Solver and Benchmark Descriptions*, volume B-2017-2 of *Department of Computer Science Series of Publications B*. University of Helsinki, 2017.

[5] C. Ansótegui and J. Gabàs. WPM3: an (in)complete algorithm for weighted partial MaxSAT. *Artif. Intell.*, 250:37–57, 2017.

[6] J. Argelich, I. Lynce, and J. P. M. Silva. On solving Boolean multilevel optimization problems. In C. Boutilier, editor, *IJCAI 2009*, pages 393–398, 2009.

[7] F. Bacchus, M. Järvisalo, M. Juhani, and R. Martins, editors. *MaxSAT Evaluation 2018: Solver and Benchmark Descriptions*, volume B-2018-2 of *Department of Computer Science Series of Publications B*. University of Helsinki, 2018.

[8] O. Bailleux and Y. Boufkhad. Efficient CNF encoding of boolean cardinality constraints. In F. Rossi, editor, *CP 2003*, volume 2833 of *Lecture Notes in Computer Science*, pages 108–122. Springer, 2003.

[9] D. L. Berre and A. Parrain. The sat4j library, release 2.2. *JSAT*, 7(2-3):59–6, 2010.

[10] N. Bjørner, A. Phan, and L. Fleckenstein. νz - an optimizing SMT solver. In C. Baier and C. Tinelli, editors, *TACAS 2015*, volume 9035 of *Lecture Notes in Computer Science*, pages 194–199. Springer, 2015.

[11] M. Büttner and J. Rintanen. Satisfiability planning with constraints on the number of actions. In S. Biundo, K. L. Myers, and K. Rajan, editors, *ICAPS 2005*, pages 292–299. AAAI, 2005.

[12] S. Cai, C. Luo, J. Thornton, and K. Su. Tailoring local search for partial MaxSAT. In C. E. Brodley and P. Stone, editors, *AAAI 2014*, pages 2623–2629. AAAI Press, 2014.

[13] S. Cai, C. Luo, and H. Zhang. From decimation to local search and back: A new approach to MaxSAT. In C. Sierra, editor, *IJCAI 2017*, pages 571–577. ijcai.org, 2017.

[14] J. Davies and F. Bacchus. Solving MAXSAT by solving a sequence of simpler SAT instances. In J. H. Lee, editor, *Principles and Practice of Constraint Programming - CP 2011 - 17th International Conference, CP 2011, Perugia, Italy, September 12-16, 2011. Proceedings*, volume 6876 of *Lecture Notes in Computer Science*, pages 225–239. Springer, 2011.

[15] E. Demirovíc, G. Chu, and P. J. Stuckey. Solution-based phase saving for CP: A value-selection heuristic to simulate local search behavior in complete solvers. In Hooker [22], pages 99–108.

[16] E. Demirovíc and P. J. Stuckey. Local-style search in the linear MaxSAT algorithm: A computational study of solution-based phase saving. In *Pragmatics of SAT Workshop*, 2018.

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

[18] E. Di Rosa, E. Giunchiglia, and M. Maratea. A new approach for solving satisfiability problems with qualitative preferences. In M. Ghallab, C. D. Spyropoulos, N. Fakotakis, and N. M. Avouris, editors, *ECAI 2008*, volume 178 of *Frontiers in Artificial Intelligence and Applications*, pages 510–514. IOS Press, 2008.

[19] N. Eén and N. Sörensson. An extensible SAT-solver. In E. Giunchiglia and A. Tacchella, editors, *SAT 2003*, volume 2919 of *Lecture Notes in Computer Science*, pages 502–518. Springer, 2003.

[20] Y. Feng, O. Bastani, R. Martins, I. Dillig, and S. Anand. Automated synthesis of semantic malware signatures using maximum satisfiability. In *24th Annual Network and Distributed System Security Symposium, NDSS 2017, San Diego, California, USA, February 26 - March 1, 2017*. The Internet Society, 2017.

[21] E. Giunchiglia and M. Maratea. Solving optimization problems with DLL. In G. Brewka, S. Coradeschi, A. Perini, and P. Traverso, editors, *ECAI 2006*, volume 141 of *Frontiers in Artificial Intelligence and Applications*, pages 377–381. IOS Press, 2006.

[22] J. N. Hooker, editor. *Principles and Practice of Constraint Programming - 24th International Conference, CP 2018, Lille, France, August 27-31, 2018, Proceedings*, volume 11008 of *Lecture Notes in Computer Science*. Springer, 2018.

[23] A. Ignatiev, A. Morgado, and J. Marques-Silva. RC2: a python-based MaxSAT solver. In Bacchus et al. [7], page 22.

[24] S. Joshi, P. Kumar, V. Manquinho, R. Martins, A. Nadel, and S. Rao. Open-WBO-Inc in MaxSAT evaluation 2018. In Bacchus et al. [7], page 16.

[25] S. Joshi, P. Kumar, R. Martins, and S. Rao. Approximation strategies for incomplete MaxSAT. In Hooker [22], pages 219–228.

[26] S. Joshi, R. Martins, and V. M. Manquinho. Generalized totalizer encoding for pseudo-boolean constraints. In G. Pesant, editor, *CP 2015*, volume 9255 of *Lecture Notes in Computer Science*, pages 200–209. Springer, 2015.

[27] S. Khoshnood, M. Kusano, and C. Wang. Concbugassist: constraint solving for diagnosis and repair of concurrency bugs. In M. Young and T. Xie, editors, *Proceedings of the 2015 International Symposium on Software Testing and Analysis, ISSTA 2015, Baltimore, MD, USA, July 12-17, 2015*, pages 165–176. ACM, 2015.

[28] D. E. Knuth. *The Art of Computer Programming, Volume 4, Fascicle 6: Satisfiability*. Addison Wesley, December 2015.

[29] Z. Lei and S. Cai. Solving (weighted) partial MaxSAT by dynamic local search for SAT. In J. Lang, editor, *IJCAI 2018*, pages 1346–1352. ijcai.org, 2018.

[30] C. Luo, S. Cai, W. Wu, Z. Jie, and K. Su. CCLS: an efficient local search algorithm for weighted maximum satisfiability. *IEEE Trans. Computers*, 64(7):1830–1843, 2015.

[31] J. Marques-Silva, F. Heras, M. Janota, A. Previti, and A. Belov. On computing minimal correction subsets. In F. Rossi, editor, *IJCAI 2013*, pages 615–622. IJCAI/AAAI, 2013.

[32] R. Martins, V. M. Manquinho, and I. Lynce. Open-wbo: A modular MaxSAT solver,. In C. Sinz and U. Egly, editors, *SAT 2014*, volume 8561 of *Lecture Notes in Computer Science*, pages 438–445. Springer, 2014.

[33] C. Mencía, A. Previti, and J. Marques-Silva. Literal-based MCS extraction. In Yang and Wooldridge [46], pages 1973–1979.

[34] A. Morgado, C. Dodaro, and J. Marques-Silva. Core-guided MaxSAT with soft cardinality constraints. In B. O'Sullivan, editor, *Principles and Practice of Constraint Programming - 20th International Conference, CP 2014, Lyon, France, September 8-12, 2014. Proceedings*, volume 8656 of *Lecture Notes in Computer Science*, pages 564–573. Springer, 2014.

[35] A. Morgado, F. Heras, M. H. Liffiton, J. Planes, and J. Marques-Silva. Iterative and core-guided MaxSAT solving: A survey and assessment. *Constraints*, 18(4):478–534, 2013.

[36] A. Morgado, A. Ignatiev, and J. Marques-Silva. MSCG: robust core-guided MaxSAT solving. *JSAT*, 9:129–134, 2014.

[37] M. W. Moskewicz, C. F. Madigan, Y. Zhao, L. Zhang, and S. Malik. Chaff: Engineering an efficient SAT solver. In *DAC 2001*, pages 530–535. ACM, 2001.

[38] A. Nadel. Generating diverse solutions in SAT. In K. A. Sakallah and L. Simon, editors, *Theory and Applications of Satisfiability Testing - SAT 2011 - 14th International Conference, SAT 2011, Ann Arbor, MI, USA, June 19-22, 2011. Proceedings*, volume 6695 of *Lecture Notes in Computer Science*, pages 287–301. Springer, 2011.

[39] A. Nadel. Solving MaxSAT with bit-vector optimization. In O. Beyersdorff and C. M. Wintersteiger, editors, *SAT 2018*, volume 10929 of *Lecture Notes in Computer Science*, pages 54–72. Springer, 2018.

[40] A. Nadel and V. Ryvchin. Bit-vector optimization. In M. Chechik and J. Raskin, editors, *TACAS 2016*, volume 9636 of *Lecture Notes in Computer Science*, pages 851–867. Springer, 2016.

[41] K. Pipatsrisawat and A. Darwiche. A lightweight component caching scheme for satisfiability solvers. In *SAT*, pages 294–299, 2007.

[42] I. A. Roig. *Solving hard industrial combinatorial problems with SAT*. Dissertation, Polytechnic University of Catalonia, 2013.

[43] N. A. Sherwani. *Algorithms for VLSI physical design automation*. Kluwer, 3 edition, November 1998.

[44] X. Si, X. Zhang, R. Grigore, and M. Naik. Maximum satisfiability in software analysis: Applications and techniques. In R. Majumdar and V. Kuncak, editors, *Computer Aided Verification - 29th International Conference, CAV 2017, Heidelberg, Germany, July 24-28, 2017, Proceedings, Part I*, volume 10426 of *Lecture Notes in Computer Science*, pages 68–94. Springer, 2017.

[45] T. Sugawara. MaxRoster: Solver description. In Bacchus et al. [7], page 25.

[46] Q. Yang and M. J. Wooldridge, editors. *Proceedings of the Twenty-Fourth International Joint Conference on Artificial Intelligence, IJCAI 2015, Buenos Aires, Argentina, July 25-31, 2015*. AAAI Press, 2015.

[47] C. S. Zhu, G. Weissenbacher, and S. Malik. Post-silicon fault localisation using maximum satisfiability and backbones. In P. Bjesse and A. Slobodová, editors, *International Conference on Formal Methods in Computer-Aided Design, FMCAD '11, Austin, TX, USA, October 30 - November 02, 2011*, pages 63–66. FMCAD Inc., 2011.