Solving MaxSAT with Bit-Vector Optimization

Alexander Nadel

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

Abstract. We explore the relationships between two closely related optimization problems: MaxSAT and Optimization Modulo Bit-Vectors (OBV). Given a bit-vector or a propositional formula F and a *target bit-vector* T, Unweighted Partial MaxSAT maximizes *the number of satisfied bits* in T, while OBV maximizes the *value* of T. We propose a new OBV-based Unweighted Partial MaxSAT algorithm. Our resulting solver–`Mrs. Beaver`–outscores the state-of-the-art solvers when run with the settings of the Incomplete-60-Second-Timeout Track of MaxSAT Evaluation 2017. `Mrs. Beaver` is the first MaxSAT algorithm designed to be incremental in the following sense: it can be re-used across multiple invocations with different hard assumptions and target bit-vectors. We provide experimental evidence showing that enabling incrementality in MaxSAT significantly improves the performance of a MaxSAT-based Boolean Multilevel Optimization (BMO) algorithm when solving a new, critical industrial BMO application: cleaning-up weak design rule violations during the Physical Design stage of Computer-Aided-Design.

1 Introduction

Modern SAT solvers [44, 30, 9] can be applied to solve various optimization problems in the domain of propositional and bit-vector logic. One such well-known problem is Weighted MaxSAT [24, 23][1]. A Weighted MaxSAT instance comprises a set of *hard* satisfiable propositional clauses H (H may also contain bit-vector constraints, reducible to propositional clauses) and a set of weighted *soft* constraints $T = \{t_{n-1}, t_{n-2}, \ldots, t_0\}$, where each constraint t_i is associated with a strictly positive integer weight w_i. To solve such an instance, the solver is required to return an assignment which satisfies H and maximizes the function $\sum_{i=0}^{n-1} t_i * w_i$, comprising the overall weight of the satisfied soft constraints. For the rest of the paper, for convenience and without restricting generality, we assume that every soft constraint is a unit clause.[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). We call T the *target bit-vector*, or, simply, the *target* and every $t_i \in T$ a *target bit*.

[1] For the rest of the paper, MaxSAT refers to Partial MaxSAT, where arbitrary hard constraints are allowed.

[2] An arbitrary soft constraint t_i, reducible to a set of propositional 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 $c \in F$.

Various optimization problems can be expressed as a sub-class of Weighted MaxSAT. Unweighted MaxSAT comprises a restriction of Weighted MaxSAT to problems where all the weights are equal to 1. Essentially, in Unweighted MaxSAT, one has to maximize *the number of satisfied target bits* or, in another words, minimize the number of unsatisfied target bits.

Optimization Modulo Bit-Vectors (OBV), also known as Lexicographic SAT (LEXSAT), is another optimization problem, recently studied in [10, 11, 35, 25, 41, 40]. In OBV, the *value* of T has to be maximized (where T is interpreted as an unsigned integer). To reduce OBV to Weighted MaxSAT, one can simply assign every target bit t_i the weight 2^i, thus ensuring that the weight of any bit $t_{i\neq 0}$ is greater than the overall weight of the bits less significant than i. The first OBV algorithm to be implemented, νZ [10, 11], solved OBV by applying this very reduction. However, it was shown in [35] that dedicated SAT-based OBV algorithms are substantially more efficient.

Can one then take the opposite route, that is, reduce MaxSAT to OBV? Our answer is affirmative.

We propose a new OBV-based Unweighted MaxSAT algorithm, called `Mrs. Beaver`. `Mrs. Beaver` is composed of two stages: the *incomplete* stage, followed by the *complete* stage. The basic version of the algorithm, applied at the incomplete stage, invokes an OBV algorithm with the *original target* to approximate an Unweighted MaxSAT solution. We propose several enhancements to the basic algorithm in order to find a better approximation faster. The basic version of the complete stage invokes an OBV algorithm whose target comprises the *sum of the bits of the original target* starting with the approximate solution, generated by the incomplete stage.

At its core, `Mrs. Beaver` is purely SAT-based. It re-uses a single incremental SAT instance across all the SAT invocations. Performance-wise, `Mrs. Beaver` is especially useful in the context of incomplete solving. It outperforms the state-of-the-art Unweighted MaxSAT solvers when run with the settings of the Incomplete-60-Second-Timeout Track of the MaxSAT Evaluation 2017.

Unlike the state-of-the-art Unweighted MaxSAT algorithms, `Mrs. Beaver` was designed to be incremental in the following sense: it can always be reused with different hard assumptions and targets. We demonstrate that incrementality in MaxSAT is useful in the context of the Boolean Multilevel Optimization problem (BMO) [26]. BMO can be thought of as the following generalization of Unweighted MaxSAT: instead of a target bit-vector T, there are multiple target bit-vectors $T_{m-1}, T_{m-2}, \ldots, T_0$. The goal is to maximize the number of satisfied bits in each of the targets, where satisfying one bit of T_i is preferred to satisfying all the bits in $T_{i-1}, T_{i-2}, \ldots, T_0$. Note that when $m = 1$, BMO is essentially identical to Unweighted MaxSAT, while if every target has only one bit, BMO is identical to OBV. BMO can be solved with iterative invocations of an Unweighted MaxSAT solver [26]. We show that, on benchmarks generated by a critical industrial problem we encountered at Intel and have described in the following paragraph, an incremental `Mrs. Beaver`-based solution is 6 times faster than a non-incremental `Mrs. Beaver`-based one, and 10 times faster than

a non-incremental Unweighted MaxSAT-based algorithm which applies the best state-of-the-art Unweighted MaxSAT solver. In addition, while our incremental `Mrs. Beaver`-based algorithm is 1.2 times slower than the best dedicated BMO solver, it uses 50 times less memory (2Gb vs. 100Gb on average).

As part of the Physical Design stage of Computer-Aided-Design (CAD), one has to solve the problem of placing and routing all the devices, while making sure that the resulting layout satisfies so-called hard design rules that originate in the manufacturing requirements. This problem can be solved by reducing it to bit-vector logic and applying some dedicated algorithms [32, 33]. In practice, however, there also exist *soft* design rules, whose satisfaction is not necessary but desirable. A failure to satisfy a soft rule increases the manufacturing cost. The soft design rules are divided into classes according to the actual cost of their violation, such that satisfying a design rule of a certain class i is more important that satisfying all the design rules of lower classes. The problem of satisfying the soft design rules after completing the process of place & route under the hard design rules is immediately reducible to BMO.

In the text that follows, Sect. 2 contains preliminaries. Sect. 3 discusses two desirable properties of SAT-based optimization algorithms: responsiveness and incrementality, while Sect. 4 reviews OBV algorithms in light of these two properties. Sect. 5 introduces our new Unweighted MaxSAT algorithm–`Mrs. Beaver`. Sect. 6 discusses how to apply `Mrs. Beaver` to solve BMO. Section 7 analyzes the experimental results, and Section 8 sums up our work and conclusions.

2 Preliminaries

2.1 SAT Solving

A SAT solver [44, 30, 9] receives a propositional formula F in Conjunctive Normal Form (CNF) and returns a satisfying assignment (also called a *model*) if one exists. In *incremental SAT solving under assumptions* [19, 34, 37], the user may invoke the SAT solver multiple times, each time with a different set of *assumptions*, where each assumption is a literal, and, possibly, additional clauses. The solver then checks the satisfiability of all the clauses provided so far while enforcing the values of the current set of assumptions only.

Modern SAT solvers apply phase saving [20, 46, 42] 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.

2.2 State-of-the-Art Unweighted MaxSAT Solvers

Unweighted MaxSAT is an active area of research as can be seen from the ever-improving results in the MaxSAT Evaluations held since 2006 [2]. We briefly summarize the state-of-the-art in Unweighted MaxSAT solving, based on the MaxSAT Evaluation 2017 results [1].

Since 2011, the MaxSAT Evaluations have had two types of categories: *complete* and *incomplete*. Complete solvers look for a solution that guarantees the absolute optimum, given a relatively generous time-out. Incomplete solvers on the other hand seek to find a good solution (that is, a solution in which there are as few as possible unsatisfied target bits), given a small time-out. Incomplete solving can be useful in applications where time resources are limited and good enough solutions are sufficient.

The winner in the complete category of MaxSAT Evaluation 2017 [1] is `Open-WBO-RES` [38], closely followed by `MaxHS` [14]. `Open-WBO-RES` is a strategy implemented within the framework of the SAT-based `Open-WBO` solver. `Open-WBO-RES` applies unsatisfiable core [17, 36, 5] analysis-based algorithms [21, 27–29] with resolution-based partitioning [38]. `MaxHS` combines SAT and Mixed Integer Programming (MIP) [16, 15, 14].

There were two incomplete categories in MaxSAT Evaluation 2017, based on time-outs of 60 and 300 seconds. An `Open-WBO` strategy–`Open-WBO-LSU` (based on the linear search SAT-UNSAT algorithm (LSU) [6])–won the 60-second category, followed by `MaxHS` and the `MaxRoster` [47] solver. `MaxRoster` won the 300-second category, followed by `Open-WBO-LSU` and `MaxHS`. `MaxRoster` is a SAT-based solver that switches dynamically between different MaxSAT strategies [47].

2.3 Totalizer Encoding

Given a target bit-vector $T = \{t_{n-1}, t_{n-2}, \ldots, t_0\}$ and a model μ, let $\mathit{unsBits}(T, \mu) = \sum_{i=0}^{n-1} \neg\mu(t_i)$ be the number of unsatisfied target bits in μ. We drop μ and use simply $\mathit{unsBits}(T)$, when allowed by the context.

Our algorithms need a way to *a*) efficiently create a bit-vector representing the number of unsatisfied target bits, while at the same time *b*) imposing an upper bound on the number of unsatisfied target bits, or, in other words, asserting the *cardinality constraint* $\mathit{unsBits}(T) \leq b$ for a given b.

To that end, we apply totalizer encoding [4]. The totalizer encodes the sum of the bits in a bit-vector in *unary representation*, which is known to be much more efficient than binary representation in terms of propagation power [4, 3].

Given a bit-vector $S = \{s_{n-1}, s_{n-2}, \ldots, s_0\}$, the totalizer is a binary tree whose top-most node–$\mathit{tot}(S)$–is a bit-vector of width n, representing the sum of S's bits in unary representation; that is, we have $\mathit{tot}(S)_i = 1$ iff $\sum_{j=0}^{n-1} s_j \geq i$. The totalizer encoding requires $O(n * log(n))$ variables and $O(n^2)$ clauses.

The totalizer encoding is substantially more efficient if a (low) upper bound–b–on the number of unsatisfied bits in S is known [13]. In that case, the order of the number of clauses goes down to $O(n * b)$. This is because the width of all the nodes longer than b (including the top node $\mathit{tot}(S)$) can be cut down to b. To impose the cardinality constraint $\sum_{i=0}^{n-1} s_i \leq b$, one has to add one additional bit $\mathit{tot}(S)_b$ to every bit-vector longer than b (in the original totalizer encoding), and set $\mathit{tot}(S)_b$ to 0.

We denote by $\mathit{tot}(S, \leq b)$ the bit-vector of width $b+1$, representing the totalizer's top node, which encodes the sum of all the bits in S in unary representation, where the cardinality constraint $\sum_{i=0}^{n-1} s_i \leq b$ is asserted.

3 Responsiveness and Incrementality

This section discusses two desirable properties of SAT-based optimization algorithms: responsiveness and incrementality. We also briefly review existing MaxSAT algorithms with respect to these properties.

3.1 Responsiveness

By *responsiveness*, we mean the ability of the solver to keep generating and outputting better and better solutions during the solving process. Such a property can be useful in various applications when the time resources are limited. Responsiveness is essential for incomplete MaxSAT solving.

3.2 Incrementality

By *incrementality*, we mean the ability of the solver to stay alive and handle many queries, as incremental SAT solving under assumptions does.

Incrementality in Current MaxSAT Algorithms Unfortunately, we are unaware of any state-of-the-art MaxSAT solver that does not become invalid after a single invocation. The only attempt at incremental MaxSAT solving was made in [43], where `Open-WBO` was modified so that one could re-use the solver and add hard and soft clauses between invocations. Unfortunately, the proposed algorithm has not been integrated into the main `Open-WBO` release. In any event, however, the ability merely to *add* clauses is not sufficient for implementing a BMO algorithm, based on incremental MaxSAT.

Incrementality under Soft Assumptions For our application, we need a more generic incremental API, where each invocation has its own target bit-vector. In other words, we want to be able to change the set of soft clauses completely between invocations. Our Unweighted MaxSAT solver–`Mrs. Beaver`–meets that requirement.

Incrementality under Hard and Soft Assumptions We believe that, in addition to the ability to change the target, MaxSAT users would benefit if the solvers could handle *hard assumptions*, which hold only for one particular invocation of the solver (like incremental SAT solving). Given such an API, one could de-activate irrelevant clauses and alternate between MaxSAT and pure SAT calls (where pure SAT calls would have an empty target). Our application does not require hard assumptions, but hard assumptions could make it possible to use MaxSAT across other applications. For example, one could then integrate Unweighted MaxSAT into the IC3 [12] (aka PDR [18]) algorithm for incremental SAT-based model checking for maximizing the number of state elements that are assigned don't care values in satisfiable queries (based on dual-rail encoding).

Hard assumptions do not appear in the pseudo-code of the algorithms proposed in this paper. However, adding hard assumptions β to our algorithms is trivial, since, at their core, our algorithms are SAT-based. One simply has to add the assumptions in β to the list of hard assumptions for every SAT invocation.

4 Optimization Modulo Bit-Vectors (OBV) Algorithms

This section reviews existing OBV algorithms in light of their performance, responsiveness, and incrementality. We needed to analyze these properties in order to choose the underlying OBV algorithm for `Mrs. Beaver`.

As we mentioned in Sect.1, the first OBV solver was νZ [10, 11]. νZ applied a straightforward reduction to Weighted MaxSAT. However, this approach does not scale [35]. Two dedicated OBV algorithms were proposed in [35]: `OBV-WA` and `OBV-BS`.

`OBV-WA` can be thought of as a linear search for the maximal model, starting with the highest possible value of the target and working towards 0. The algorithm stops at the first satisfying assignment. `OBV-WA` is an incremental algorithm implemented inside a SAT solver. `OBV-WA` can be quite efficient, but, unfortunately, it is not responsive as it finds only one (best) model. Thus it cannot serve as the underlying building block for `Mrs. Beaver`.

`OBV-BS` is depicted in Alg. 1. Essentially, the algorithm implements a binary search over the possible values of a target T. The algorithm receives a CNF formula F and the target T.[1] It maintains the current model μ, initialized with an arbitrary model to F at line 3, and a partial assignment α, which is empty at the beginning. The main loop of the algorithm (starting at line 6) goes over all the bits of target T starting from the MSB t_{n-1} down to t_0. Each iteration extends α with either t_i or $\neg t_i$, where t_i is preferred over $\neg t_i$ iff there exists a model where t_i is assigned 1 while bits higher than i have already been assigned in previous iterations. *Phase saving optimization*, shown in lines 2 and 10, sets the phase saving array for the target bits with 1's before every SAT invocation, thus encouraging the solver to prefer a higher value for T. Phase saving optimization improves the performance of the algorithm. `OBV-BS` is incremental, since it is based on incremental SAT solving. It is also quite responsive, since it keeps finding better models throughout its execution.

Independently, `OBV-BS`, without phase saving optimization, was also suggested in [25] in the context of solving the LEXSAT problem, which is, essentially, identical to OBV.

`BINARY` [40, 41] is another OBV algorithm. `BINARY` can be thought of as a partial integration of `OBV-WA` into `OBV-BS`. `BINARY` applies `OBV-BS`, where, for every SAT solver iteration inside the main loop, it adds the upper half of the bits, that is, $\{t_i, t_{i+1}, \ldots, t_{\lceil i/2 \rceil}\}$, as assumptions, rather than only the current bit $\{t_i\}$. If the invocation is satisfiable, the solver can update i to bit number $\lceil i/2 \rceil$. Otherwise, it halves the number of satisfied assumptions and stays at iteration

[1] If the original formula F is a bit-vector formula; it is preprocessed and translated to CNF first.

Algorithm 1 `OBV-BS`

```
1: function SOLVE(CNF Formula F, Target T = {t_{n-1}, t_{n-2}, ..., t_0})
2:     Set the phase saving values of {t_{n-1}, t_{n-2}, ..., t_0} to 1
3:     μ := SAT()
4:     if SAT solver returned UNSAT then return UNSAT
5:     α := {}
6:     for i ← n − 1 downto 0 step 1 do
7:         if t_i ∈ μ then                                  ▷ t_i ∈ μ ≡ t_i = 1 in μ
8:             α := α ∪ {t_i}
9:         else
10:            Set the phase saving values of {t_{n-1}, t_{n-2}, ..., t_0} to 1
11:            τ := SATUNDERASSUMPTIONS(α ∪ {t_i})
12:            if SAT solver returned SAT then μ := τ else α := α ∪ {¬t_i}
13:    return μ
```

i. In addition, `BINARY` skips the first SAT invocation. `BINARY` was reported to be faster than `OBV-BS` in [40, 41]. However, `BINARY` is less responsive than `OBV-BS`, since it apparently increases the number of unsatisfiable queries.

All things considered, we picked `OBV-BS` as the baseline algorithm for `Mrs. Beaver`, since it combines good performance, responsiveness, and incrementality. Note that `OBV-BS` can easily be updated to handle user-given hard assumptions β by adding β's literals to the assumption list for every SAT invocation.

5 `Mrs. Beaver`: An Unweighted MaxSAT Algorithm

This section introduces our new Unweighted MaxSAT algorithm `Mrs. Beaver`. The high-level algorithm is shown in Alg. 2. It receives a satisfiable CNF formula F, the target bit-vector T, the incrementality mode *incrMode* and the search mode *searchMode*. Alg. 2 outputs a model μ which minimizes $\mathit{unsBits}(T, \mu)$. *incrMode* lets the user decide whether the algorithm should be incremental, and how it should operate in incremental mode. *searchMode* determines the behavior of the algorithm at the complete stage, as will be explained later in Sect. 5.1.

Assume for now that *incrMode* = *none*, that is, that the algorithm is *not* incremental, and that *searchMode* = *SU*. First, for the *incomplete* stage of the algorithm, `Mrs. Beaver` invokes a preprocessor, `Mrs. Beaver-Inc` (described in Sect. 5.2), designed to quickly find a model μ with as low $\mathit{unsBits}(T, \mu)$ as possible. Then, during the *complete* stage, the algorithm invokes `OBV-BS` to minimize a new target $T' := \mathit{tot}(\neg T, \leq \mathit{unsBits}(T, \mu) - 1)$, comprising the sum of unsatisfied target bits starting with the value $\mathit{unsBits}(T, \mu) - 1$. If the latter invocation is satisfiable with the model μ', `Mrs. Beaver` returns μ'. Otherwise, there is no better model than μ, hence μ is returned.

It is imperative for performance to count the number of unsatisfied target bits towards 0, rather than the number of satisfied target bits towards n. This

is because creating a totalizer with an upper-bound on the sum is substantially more efficient than creating one with a lower-bound on the sum.

Our algorithm reuses the same SAT solver instance across all the calls, hence all learning is re-used. As we mentioned, when *incrMode* = *none*, the algorithm is *not* incremental. This is because the totalizer encoding asserts a cardinality constraint which is, apparently, not inferred by F. Sect. 5.4 describes `Mrs. Beaver`'s behavior in incremental modes.

5.1 `Mrs. Beaver` and Linear Search

`Mrs. Beaver` is closely related to the linear search SAT-UNSAT (LSU) and UNSAT-SAT (LUS) algorithms [6].

LSU starts by finding a solution μ using a SAT solver. It then enters the *SAT-UNSAT loop*, which adds a cardinality constraint ensuring that the next solution will have strictly fewer unsatisfied target bits than $\mathit{unsBits}(T, \mu)$ after which it invokes a SAT solver. The algorithm updates μ with any newly found solution and terminates when the SAT solver returns UNSAT. It is guaranteed that the latest solution is an optimal one.

LUS keeps a lower bound l (initialized to 0) for the number of unsatisfied target bits for which no solution exists. LUS operates in an *UNSAT-SAT loop* which runs a SAT solver assuming that $\mathit{unsBits}(T) = l$. If the solver returns UNSAT, LUS updates l to $l + 1$ and proceeds to the next iteration of the loop. If the solver finds a solution μ, LUS terminates, in which case μ is guaranteed to be an optimal solution.

Note that the *complete* stage of `Mrs. Beaver` can behave as either the SAT-UNSAT loop (when *searchMode* = *SU*) or the UNSAT-SAT loop (when *searchMode* = *US*). In the latter case, the solver reverses the bits of T, so as to start falsifying T from the LSB towards the MSB. Thus it is the usage of the incomplete preprocessor–`Mrs. Beaver-Inc`–that differentiates between the linear search algorithms and `Mrs. Beaver`. Specifically, the difference between LSU and `Mrs. Beaver` in *SU* mode is that LSU uses a single SAT invocation for the incomplete stage, while `Mrs. Beaver` applies `Mrs. Beaver-Inc`. The difference between `Mrs. Beaver` in *US* mode and the LUS algorithm is that the former finds an upper bound on the number of unsatisfied target bits using `Mrs. Beaver-Inc`, while the latter may use a single SAT invocation to find an upper bound (if incremental weakening [28] is applied).

5.2 `Mrs. Beaver-Inc`: The Incomplete Preprocessor

`Mrs. Beaver-Inc` is designed to quickly find improving models. Our basic idea is to run `OBV-BS` over the target T to gradually reduce the number of unsatisfied target bits for the *current order* of T's literals. We realized that, to find tighter lower bounds faster, the following two optimizations would be useful:

1. Change `OBV-BS` so as to satisfy more target bits, even if the resulting algorithm no longer solves the OBV problem. We present such an algorithm–`UMS-OBV-BS`–next.

Algorithm 2 `Mrs. Beaver`

1: **function** SOLVE(CNF Formula F, Target $T = \{t_{n-1}, t_{n-2}, \ldots, t_0\}$, *incrMode* $\in$ $\{none, full, maxPreserving\}$, *searchMode* $\in \{SU, US\}$)
Require: F is satisfiable
2: $\mu :=$ `Mrs. Beaver-Inc`(F,T)
3: **if** $unsBits(T, \mu) = 0$ **then return** μ
4: **if** $incrMode = none$ **then**
5: $T' := tot(\neg T, \leq unsBits(T, \mu) - 1)$
6: **else if** $incrMode = maxPreserving$ **then**
7: $T' := tot(\neg T, \leq unsBits(T, \mu))$
8: **else** ▷ $incrMode = full$
9: $T' := tot(\neg T, \leq unsBits(T, \mu) - 1)$ with a fresh selector; see text
10: **if** $searchMode = US$ **then** $T :=$ `reverse(T)`
11: $\mu' :=$ `OBV-BS`$(F, \neg T')$ ▷ Maximizing $\neg T' \equiv$ minimizing T'
12: **if** `OBV-BS` solver returned SAT **then return** μ' **else return** μ

2. Run *several* iterations of `UMS-OBV-BS` and/or `OBV-BS`, where the target bits are the same, but their order changes (by reversing or shuffling). Changing the order of the target bits increases the chances of encountering a MaxSAT-friendly order. Below we assume that any algorithm that changes the order of the bits in the target T recreates the original T just before it finishes.

From `OBV-BS` to `UMS-OBV-BS` We propose modifying `OBV-BS` to increase the chances of satisfying more target bits as follows: after a new model μ is encountered, the algorithm pushes all the target bits assigned 1 towards the most significant bit, so as to fix the value of such bits to 1 for the rest of the algorithm. Alg. 3 shows a function that transforms `OBV-BS` to `UMS-OBV-BS`. It is designed to be invoked immediately after Alg. 1 finds a new model for bit i at line 11. Note that `UMS-OBV-BS` no longer solves the OBV problem.

`UMS-OBV-BS` maintains an index k, initialized with the current index i minus 1. It visits every bit whose value has not been set and swaps any newly satisfied bits with t_k, where, when a satisfied bit is discovered, k is decreased by 1.

Algorithm 3 `UMS-OBV-BS`

1: **function** MODIFYING `OBV-BS` TO `UMS-OBV-BS`
Require: Invoke this function immediately after line 11 of Alg. 1
2: $k := i - 1$
3: **for** $j \leftarrow i - 1$ **downto** 0 **step** 1 **do**
4: **if** $\mu(t_j) = 1$ **then**
5: Swap the bits t_k and t_j
6: $k := k - 1$
7: **return** μ

The Preprocessor The generic scheme of our preprocessor `Mrs. Beaver-Inc` is shown in Alg. 4. It allows some freedom, the concrete heuristics being regulated

by several user-given parameters discussed below. `Mrs. Beaver-Inc` receives a CNF formula F and the target T. It operates in a loop which runs for a user-given number of iterations. Each iteration invokes either `UMS-OBV-BS` or `OBV-BS`. The returned model μ is stored after the initial iteration and updated whenever a better model is found. After each iteration, T is either reversed or shuffled. The algorithm is regulated by the following 3 user-given parameters:

1. `ALG`: the inner algorithm, applied at line 3. It can either be *a*) `UMS-OBV-BS` or *b*) `OBV-BS` or *c*) `Mixed-OBV`, which is an alternation between `UMS-OBV-BS` and `OBV-BS`. If `ALG` is either plain `OBV-BS` or plain `UMS-OBV-BS`, the target is reversed at line 5 after each odd iteration and randomly shuffled after each even iteration. If `ALG` is `Mixed-OBV`, then `UMS-OBV-BS` is applied at iterations $i : i\%4 \in \{0, 1\}$, while `OBV-BS` is applied at iterations $i : i\%4 \in \{2, 3\}$. The target is reversed after iterations $i : i\%4 \in \{1, 2, 3\}$ (note that reversing T after iteration $i : i\%4 = 3$ recreates the original order) and shuffled after iteration $i : i\%4 = 3$.
2. *itNum*: the number of iterations.
3. *obvConfThr*: a threshold on the number of conflicts for each invocation of SAT-under-assumptions to find the satisfiability status of a single bit inside `UMS-OBV-BS` and `OBV-BS` (line 11 in Alg. 1). Since `Mrs. Beaver-Inc` is incomplete, we found it useful to stop the solver when a threshold on the number of conflicts is reached in order not to get stuck with difficult bits. An unsolved target bit is assigned 0 by the algorithm.

Algorithm 4 `Mrs. Beaver-Inc`

1: **function** `Mrs. Beaver-Inc`(CNF Formula F, Target $T = \{t_{n-1}, t_{n-2}, \ldots, t_0\}$)
2: **for** $i \leftarrow 1$ **to** *itNum* **step** 1 **do** ▷ *itNum* is a user-given threshold
3: $\mu' :=$ `UMS-OBV-BS`(F, T) or `OBV-BS`(F, T)
4: **if** μ doesn't exist or *unsBits*(T,μ') < *unsBits*(T,μ) **then** $\mu := \mu'$
5: $T :=$ `reverse(T)` or `shuffle(T)`

5.3 Responsiveness

`Mrs. Beaver-Inc` is quite responsive. Not only can each invocation of `OBV-BS`/`UMS-OBV-BS` update the best model, the best model can also be updated by the inner iterations of `OBV-BS`/`UMS-OBV-BS`. Hence, the main algorithm `Mrs. Beaver` is responsive at the incomplete stage. At the complete stage `Mrs. Beaver` is responsive only in the *SU* mode.

5.4 Incrementality

Recall that `Mrs. Beaver` can operate in non-incremental mode, fully incremental mode, or maximization-preserving incremental mode (described below), depending on the user-given value $incrMode \in \{none, full, maxPreserving\}$.

Full Incrementality An algorithm is fully incremental if all the learned clauses are inferred by the input formula F. To make `Mrs. Beaver` fully incremental, we need to eliminate any clauses created by the totalizer. To that end, we simply add a fresh selector variable s to every clause generated by the totalizer and add $\neg s$ as a hard assumption to `OBV-BS`, applied at line 11 of Alg. 2. One can also add the unit clause s after `Mrs. Beaver` is completed to remove all the clauses, generated by the totalizer.

Maximization-preserving Incrementality An invocation of incremental Unweighted MaxSAT is *maximization-preserving* if it asserts that the number of unsatisfied bits in the current target T cannot be higher than the number $\mathit{unsBits}(T, \mu)$, found by the algorithm. As we shall see, a maximization-preserving incremental Unweighted MaxSAT solution is useful in the context of BMO solving.

Alg. 2, in the mode $\mathit{incrMode} = \mathit{none}$, is *almost* maximization-preserving, except than the totalizer, created at line 5, asserts that the number of unsatisfied bits is strictly lower than $\mathit{unsBits}(T, \mu)$. This might cause the formula to become unsatisfiable if the actual minimum happens to be $\mathit{unsBits}(T, \mu)$. To fix this problem for the maximization-preserving mode $\mathit{incrMode} = \mathit{maxPreserving}$, we simply provide the totalizer the number $\mathit{unsBits}(T, \mu)$ as the upper bound. Note that this might result in a certain performance degradation.

6 Applying `Mrs. Beaver` to Solve BMO

Recall that in BMO, instead of a target bit-vector T, there are multiple target bit-vectors $T_{m-1}, T_{m-2}, \ldots, T_0$. The goal is to maximize the number of satisfied bits in each of the targets, where satisfying one bit of T_i is preferred to satisfying all the bits in $T_{i-1}, T_{i-2}, \ldots, T_0$.

One way to solve BMO, proposed in [26], is by reducing the problem to Weighted MaxSAT by concatenating the bits of all the target bit-vectors into one target bit-vector, and assigning each bit $t_i^0 \in t_0$ the weight $w^0 = 1$, and each bit $t_i^l \in T_{l>0}$ the weight $w^l = 1 + \sum_{k=0}^{l-1} w^k * |T_k|$. However, as we shall see, such a solution does not scale.

Alg. 5 shows our BMO algorithm–`Oh Mrs. Beaver`–which adapts the iterative MaxSAT-based BMO algorithm from [26] to apply an *incremental* Unweighted MaxSAT solver underneath. `Oh Mrs. Beaver` takes full advantage of `Mrs. Beaver`'s functionality in maximization-preserving mode. `Oh Mrs. Beaver` simply goes over all the targets from the most important one towards the least important one, and applies `Mrs. Beaver` in maximization-preserving mode to each target. In this way it guarantees that the optimal solution for each target T_i is asserted after invocation i is completed.

`Oh Mrs. Beaver` invokes `Mrs. Beaver` as the underlying building block, but it could, in principle, use any maximization-preserving incremental Unweighted MaxSAT algorithm that allows the user to change the target. Unfortunately, no such algorithm exists in the literature. It is possible, however, to use a fresh

non-incremental Unweighted MaxSAT solver for every iteration i of Alg. 5 for the current target T_i. For that to work, one has to assert the cardinality constraint $\sum_{i=0}^{n-1} \neg T_i \leq \mathit{unsBits}(T_i)$ after each iteration i. Unfortunately, cardinality constraints are not part of the standard MaxSAT format. To evaluate the performance of non-incremental instantiations of `Oh Mrs. Beaver` with the different state-of-the-art Unweighted MaxSAT solvers, we encoded the cardinality constraints into clauses using the totalizer encoding.

Algorithm 5 `Oh Mrs. Beaver`

1: **function** SOLVE(CNF Formula F, Targets $T_{m-1}, T_{m-2}, \ldots, T_0$)
Require: F is satisfiable
2: **for** $i \leftarrow n-1$ **to** 0 **step** 1 **do**
3: $\mu :=$ `Mrs. Beaver` (F, T_i, *maxPreserving*)

7 Experimental Results

This section studies the performance of our algorithms. Sect. 7.1 analyzes the performance of Unweighted MaxSAT solvers run with settings that mimic the MaxSAT Evaluation 2017. Sect. 7.2 examines the performance of MaxSAT and BMO solvers on benchmarks we generated from our industrial application.

The benchmarks we generated, as well as the detailed results, are available in [31]. Unless specified differently, the experiments were executed on machines with 32Gb of memory running Intel® Xeon® processors with 3Ghz CPU frequency. The time is always shown in seconds.

7.1 Unweighted MaxSAT: MaxSAT Evaluation 2017

In this section we compare the performance of `Mrs. Beaver` to that of the winners of the MaxSAT Evaluation. In addition, we study the impact of `Mrs. Beaver`'s search mode (*SU* vs. *US*) and `Mrs. Beaver-Inc`'s three parameters, introduced in Sect. 5.2, on the performance of `Mrs. Beaver`.

We denote by $\{\texttt{ALG}, \mathit{itNum}, \mathit{obvConfThr}, \mathit{searchMode}\}$ a configuration of `Mrs. Beaver`, where the search mode *searchMode* is either *SU* or *US* and the incomplete preprocessor applies the algorithm `ALG` $\in \{\texttt{OBV-BS}, \texttt{UMS-OBV-BS}, \texttt{Mixed-OBV}\}$ using *itNum* iterations and the conflict threshold *obvConfThr* in `OBV-BS` and/or `UMS-OBV-BS`. We denote by the configurations $\{-, 1, 0, \mathit{SU}\}$ and $\{-, 1, 0, \mathit{US}\}$ the implementations of LSU and LUS, respectively, in the framework of `Mrs. Beaver` (i.e., a conflict threshold of 0 per bit means that the incomplete stage of `Mrs. Beaver` invokes SAT instead of `Mrs. Beaver-Inc`).

Recall that the MaxSAT Evaluation had two incomplete categories, with 60-second and 300-second timeouts, respectively, and one complete category with a 3600-second timeout.

Incomplete Categories We compared the performance of different configurations of `Mrs. Beaver` to that of the leading incomplete solvers `MaxHS`, `Open-WBO-LSU`, and `MaxRoster`. Based on preliminary experiments, we picked the following configuration as the baseline for `Mrs. Beaver` when the timeout is 60 second: $\{\texttt{Mixed-OBV}, 10^5, 10^4, SU\}$. To provide evidence that the baseline configuration is indeed the best one and to study the impact of the parameters, we also provide results for "neighbor" configurations, constructed by changing one of the parameters of the baseline configuration, and that of our linear search implementations ($\{-, 1, 0, SU\}$ and $\{-, 1, 0, US\}$).

Mimicking the MaxSAT Evaluation, our primary ranking criteria was average *score*. *score* for instance i and solver S is the ratio between the number of unsatisfied target bits in the best solution found by any of the solvers and the number of unsatisfied target bits in the best solution found by solver S. *score* is 0 if no solution was found by S. It holds that $score \in [0, 1]$.

Consider the upper part of Table 1 which displays the results for the 60-second timeout. Each row compares the results of a single configuration of `Mrs. Beaver`, shown in the first column, to those of the other solvers. Following the presentation style used in the MaxSAT Evaluation, we provide *score*, the number of solved instances (in columns titled #S) and the number of times each algorithm found the best solution (in columns titled #B). The best score in each row is highlighted. The table is sorted according to the score of the `Mrs. Beaver` configuration.

The best result was achieved by the baseline configuration $\{\texttt{Mixed-OBV}, 10^5, 10^4, SU\}$. It outperforms all the other solvers, including our LSU implementation ($\{-, 1, 0, SU\}$) and the LSU implementation in the MaxSAT evaluation winner `Open-WBO-LSU`. `Mrs. Beaver`'s performance is slightly better in the *SU* (SAT-UNSAT) mode.

Concerning the parameters of `Mrs. Beaver-Inc`, changing the conflict threshold or the number of iterations led to a mild deterioration of the score. Alternating between `OBV-BS` and `UMS-OBV-BS` yielded the best results. Using only `UMS-OBV-BS` ($\{\texttt{UMS-OBV-BS}, \infty, 10^5\}$) was insufficient for outscoring the other solvers.

The bottom part of Table 1 shows the results for the 300-second timeout. We found in preliminary experiments that the best-performing `Mrs. Beaver` configuration for the 300-second timeout is $\{\texttt{Mixed-OBV}, 10^4, 10^5, SU\}$; it is slightly different from that used for the 60-second timeout. As in the MaxSAT Evaluation, `MaxRoster` emerges as the best solver in this category. `Mrs. Beaver` comes out as the second best.

Complete Category Based on preliminary experiments, we picked the following configuration as the baseline for `Mrs. Beaver` for complete solving: $\{\texttt{UMS-OBV-BS}, 1, 10^3, US\}$. It applies one iteration of the preprocessor using `UMS-OBV-BS` and a relatively low conflict threshold of 10^3 as well as the *US* mode at the complete stage.

Mrs. Beaver Conf.	Mrs. Beaver			Open-WBO-LSU			MaxHS			MaxRoster		
	Score	#S	#B	Score	#S	#B	Score	#S	#B	Score	#S	#B
60-Second Timeout												
{Mixed-OBV, 10^5, 10^4, SU}	0.81792	182	56	0.73498	178	54	0.73017	192	37	0.67423	145	89
{Mixed-OBV, 10^4, 10^4, SU}	0.81787	182	55	0.73498	178	54	0.73017	192	37	0.67423	145	89
{Mixed-OBV, 10^5, 10^4, US}	0.81756	182	56	0.73498	178	54	0.73017	192	37	0.67423	145	89
{Mixed-OBV, 10^6, 10^4, SU}	0.81748	182	56	0.73498	178	54	0.73017	192	37	0.67423	145	89
{Mixed-OBV, 10^5, 10^5, SU}	0.81378	182	64	0.73902	178	50	0.73122	192	37	0.67094	145	87
{Mixed-OBV, 10^5, 10^3, SU}	0.79008	181	49	0.74207	178	59	0.73654	192	38	0.67593	145	89
{OBV-BS, 10^5, 10^4, SU}	0.7855	182	55	0.74274	178	57	0.73764	192	38	0.67765	145	92
{$-$, 1, 0, SU}	0.74531	182	65	0.75043	178	53	0.74577	192	45	0.68675	145	91
{UMS-OBV-BS, 10^5, 10^4, SU}	0.73236	181	39	0.74611	178	59	0.74121	192	44	0.67835	145	90
{$-$, 1, 0, US}	0.57173	182	8	0.77117	178	77	0.76302	192	47	0.69246	145	96
300-Second Timeout												
{Mixed-OBV, 10^4, 10^5, SU}	0.77807	183	39	0.71429	182	43	0.75285	194	55	0.87118	182	126
{Mixed-OBV, 10^4, 10^5, US}	0.77806	183	39	0.71429	182	43	0.75285	194	55	0.87118	182	126
{Mixed-OBV, 10^5, 10^5, SU}	0.77774	183	40	0.71424	182	43	0.75279	194	55	0.87112	182	126
{Mixed-OBV, 10^3, 10^5, SU}	0.77563	183	39	0.71429	182	43	0.75285	194	55	0.87118	182	126
{Mixed-OBV, 10^4, 10^4, SU}	0.77259	183	32	0.71354	182	44	0.75276	194	56	0.87191	182	128
{Mixed-OBV, 10^4, 10^6, SU}	0.75761	183	40	0.71442	182	43	0.75277	194	55	0.8715	182	127
{OBV-BS, 10^5, 10^4, SU}	0.72725	183	32	0.71503	182	44	0.75782	194	57	0.8768	182	129
{$-$, 1, 0, SU}	0.71329	184	36	0.7232	182	46	0.76508	194	62	0.88502	182	133
{UMS-OBV-BS, 10^5, 10^4, SU}	0.70314	183	32	0.71608	182	43	0.75515	194	60	0.87329	182	128
{$-$, 1, 0, US}	0.50183	184	3	0.72414	182	48	0.76543	194	62	0.88605	182	135

Table 1: MaxSAT Evaluation: Incomplete Categories

Consider Table 2. It displays the number of solved instances and the overall run-time of Open-WBO-RES, MaxHS, Open-WBO-LSU, our linear search implementations and several Mrs. Beaver configurations (MaxRoster cannot be applied for complete solving). The algorithms are sorted according to their performance.

Unsurprisingly, unlike in the incomplete categories, Mrs. Beaver did not perform as well as the leading solvers, Open-WBO-RES and MaxHS. Apparently, the reason is that Mrs. Beaver's top-performing complete algorithm relies merely on *US* linear search. Applying *SU* instead of *US* at the complete stage or changing the underlying algorithm at the incomplete stage results in a performance deterioration. Notably, the preprocessor allows Mrs. Beaver to solve 18 more instance as compared to the LUS implementation in our framework ({UMS-OBV-BS, 1, 10^3, US} vs. {$-$, 1, 0, US}).

Results			Results (Continued)		
Solver	#S	Overall Time	Solver	#S	Overall Time
MaxHS	655	927384.89	{UMS-OBV-BS, 1, 10^3, SU}	547	1251925.21
Open-WBO-RES	654	880493.49	{OBV-BS, 1, 10^3, US}	546	1281167
{UMS-OBV-BS, 1, 10^3, US}	572	1176926.57	{OBV-BS, 1, 10^3, SU}	543	1292191.23
Open-WBO-LSU	554	1209114.62	{$-$, 1, 0, SU}	541	1258922.31
{$-$, 1, 0, US}	554	1214299.29			

Table 2: MaxSAT Evaluation: Complete Category

7.2 Industrial Instances

For the experiments in this section, we generated 9 Weighted MaxSAT benchmarks that encode the industrial task of cleaning up soft design rules in Intel designs. We used the straightforward translation from BMO to Weighted MaxSAT, described in Sect. 6, for generating the benchmarks. The number of variables in the benchmarks ranges from 4,367,381 to 8,220,593, while the number of clauses ranges from 12,960,427 to 26,676,683. The number of target bit-vectors (before

applying the reduction from BMO to Weighted MaxSAT) is 44 in every benchmark.

The main goal of our experiments was to study the impact of enabling incrementality in MaxSAT solving on the performance of the Unweighted MaxSAT-based BMO algorithm in the context of our application. We compared `Oh Mrs. Beaver` against the non-incremental Unweighted MaxSAT flow with the following underlying solvers: `Open-WBO-RES`, `Open-WBO-LSU`, `MaxHS`, and `Mrs. Beaver`. We used the best performing configuration in the complete category for both `Oh Mrs. Beaver` and the non-incremental `Mrs. Beaver`. We used a timeout of 1800 seconds for `Oh Mrs. Beaver` and a timeout of 600 seconds for each invocation of a non-incremental solver.

The results are shown in Table 3. For each solver, the table shows the number of completed invocations (out of 44 incremental invocations, one for each target bit-vector) and the time. The last row shows the average number of completed invocations and the average time for each solver. `Oh Mrs. Beaver` is clearly the best solver. It solved all the benchmarks, being 6 times faster than the `Mrs. Beaver`-based non-incremental flow and 10 times faster than the `MaxHS`-based non-incremental flow. Neither `Open-WBO-RES` nor `Open-WBO-LSU` scaled to our instances. All in all, it pays off to apply *incremental* Unweighted MaxSAT solving to solve industrial instances of BMO.

In addition, for comparison, we ran the following Weighted MaxSAT solvers: `Open-WBO`, `MaxHS`, `MaxRoster`, `Clasp` [22] and `Sat4j` [6]. It turned out that only `Sat4j` was able to process our benchmarks successfully, since in our benchmarks the weight can be as high as 10^{129}, while the maximal weight in the MaxSAT format is restricted to 2^{63}. `Sat4j` timed-out on all the instances.

Finally, we translated our benchmarks to the BMO format, used during the Lion9 Challenge [7] (the only BMO competition ever held), and ran the following three dedicated BMO solvers, comprising all the participants in the challenge: `Sat4j` [6], `Open-WBO-SU`, and `Open-WBO-MSU3` (the latter two solvers were implemented in the `Open-WBO` framework). The results are shown in Table 4. Initially, when we invoked the BMO solvers on our standard machines (with 32Gb of memory), all three solvers failed to solve a single instance: `Sat4j` timed-out and both versions of `Open-WBO` came back with memory-outs. As a follow-up experiment, we ran `Oh Mrs. Beaver`, `Open-WBO-SU`, and `Open-WBO-MSU3` on a machine having 512Gb of memory. It turned out that both versions of `Open-WBO` slightly outperformed `Oh Mrs. Beaver` (by 1.2 times on average), but they used 50 times more memory (2Gb vs. 100Gb on average). Both `Open-WBO-SU` and `Open-WBO-MSU3` reduce BMO to iterative MaxSAT invocations, similarly to `Oh Mrs. Beaver` (using different MaxSAT algorithms underneath: LSU in the case of `Open-WBO-SU` and MSU3 [27] in the case of `Open-WBO-MSU3`). As for the huge difference in memory usage, it is difficult to determine the reason for this, as the source code of both `Open-WBO-SU` and `Open-WBO-MSU3` is unavailable.

#	Incr. Calls	Oh Mrs. Beaver		Open-WBO-RES		Open-WBO-LSU		MaxHS		Mrs. Beaver	
		Solved	Time	Solved	Time	Solved	Time	Solved	Time	Solved	Time
1	44	44	310	0	26400	0	26400	44	3060	44	1881
2	44	44	452	0	26400	0	26400	44	4343	44	2890
3	44	44	346	0	26400	0	26400	44	4141	44	2740
4	44	44	366	0	26400	0	26400	44	3404	44	2120
5	44	44	197	0	26400	0	26400	44	3635	44	2271
6	44	44	229	44	8883	44	8895	44	1998	44	1163
7	44	44	282	44	11483	44	11483	44	2336	44	1406
8	44	44	325	11	25828	8	25937	44	2737	44	1667
9	44	44	459	0	26400	0	26400	44	3159	44	1998
Avrg	44	44	330	11	22733	11	22746	44	3201	44	2015

Table 3: Evaluation on Industrial BMO Instances

#	Standard Settings (32Gb)			512Gb and 1.2Ghz CPU frequency					
	Sat4j	Open-WBO-SU	Open-WBO-MSU3	Oh Mrs. Beaver		Open-WBO-SU		Open-WBO-MSU3	
	Res	Res	Res	Time	Mem (Mb)	Time	Mem (Mb)	Time	Mem (Mb)
1	Timeout	Memout	Memout	413	1917	316	96881	333	96917
2	Timeout	Memout	Memout	630	2710	469	133578	451	133577
3	Timeout	Memout	Memout	469	2594	451	127032	298	127121
4	Timeout	Memout	Memout	356	2164	371	108399	367	108449
5	Timeout	Memout	Memout	438	2287	360	114650	463	114673
6	Timeout	Memout	Memout	275	1289	183	67374	235	67274
7	Timeout	Memout	Memout	298	1514	219	76320	249	74618
8	Timeout	Memout	Memout	349	1737	319	89581	296	89605
9	Timeout	Memout	Memout	437	2019	366	102081	359	102104
Avrg				407	2026	339	101766	339	101593

Table 4: Evaluation of BMO Solvers on Industrial BMO Instances

8 Conclusion

We explored how Unweighted MaxSAT solving can benefit from the recently introduced Optimization Modulo Bit-Vectors (OBV) algorithms. We proposed a new OBV-based Unweighted MaxSAT algorithm–`Mrs. Beaver`. `Mrs. Beaver` outscored the top solvers when run with the settings of the Incomplete-60-Second-Timeout Track of MaxSAT Evaluation 2017. Unlike the existing state-of-the-art algorithms, `Mrs. Beaver` was designed to be incremental in the sense that it can be reapplied with a different set of hard assumptions and soft clauses. We demonstrated that enabling incrementality in MaxSAT significantly improves the performance a MaxSAT-based BMO algorithm applied for solving a new critical industrial BMO application: cleaning-up weak design rule violations during the Physical Design stage of Computer-Aided-Design at Intel.

9 Acknowledgments

We are grateful to Ruben Martins for his valuable advice concerning different aspects of MaxSAT research. We thank Jessica Davies and Fahiem Bacchus for their help with `MaxHS`, and Vadim Ryvchin for suggesting the use of a conflict threshold in `OBV-BS`.

References

1. Carlos Ansotegui, Fahiem Bacchus, Matti Järvisalo, and Ruben 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.
2. Josep Argelich, Chu Min Li, Felip Manyà, and Jordi Planes. The first and second max-sat evaluations. *JSAT*, 4(2-4):251–278, 2008.
3. Olivier Bailleux. On the CNF encoding of cardinality constraints and beyond. *CoRR*, abs/1012.3853, 2010.
4. Olivier Bailleux and Yacine Boufkhad. Efficient CNF encoding of boolean cardinality constraints. In Francesca Rossi, editor, *Principles and Practice of Constraint Programming - CP 2003, 9th International Conference, CP 2003, Kinsale, Ireland, September 29 - October 3, 2003, Proceedings*, volume 2833 of *Lecture Notes in Computer Science*, pages 108–122. Springer, 2003.
5. Anton Belov, Marijn Heule, and João Marques-Silva. MUS extraction using clausal proofs. In Sinz and Egly [45], pages 48–57.
6. Daniel Le Berre and Anne Parrain. The sat4j library, release 2.2. *JSAT*, 7(2-3):59–6, 2010.
7. Daniel Le Berre, Olivier Roussel, Rémi Delmas, and Marie-Eléonore Marmion. Lion9 challenge. http://www.lifl.fr/LION9/challenge.php.
8. Armin Biere and Carla P. Gomes, editors. *Theory and Applications of Satisfiability Testing - SAT 2006, 9th International Conference, Seattle, WA, USA, August 12-15, 2006, Proceedings*, volume 4121 of *Lecture Notes in Computer Science*. Springer, 2006.
9. Armin Biere, Marijn Heule, Hans van Maaren, and Toby Walsh, editors. *Handbook of Satisfiability*, volume 185 of *Frontiers in Artificial Intelligence and Applications*. IOS Press, 2009.
10. Nikolaj Bjørner and Anh-Dung Phan. νz - maximal satisfaction with Z3. In Temur Kutsia and Andrei Voronkov, editors, *6th International Symposium on Symbolic Computation in Software Science, SCSS 2014, Gammarth, La Marsa, Tunisia, December 7-8, 2014*, volume 30 of *EPiC Series*, pages 1–9. EasyChair, 2014.
11. Nikolaj Bjørner, Anh-Dung Phan, and Lars Fleckenstein. νz - an optimizing SMT solver. In Christel Baier and Cesare Tinelli, editors, *Tools and Algorithms for the Construction and Analysis of Systems - 21st International Conference, TACAS 2015, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2015, London, UK, April 11-18, 2015. Proceedings*, volume 9035 of *Lecture Notes in Computer Science*, pages 194–199. Springer, 2015.
12. Aaron R. Bradley. Sat-based model checking without unrolling. In Ranjit Jhala and David A. Schmidt, editors, *Verification, Model Checking, and Abstract Interpretation - 12th International Conference, VMCAI 2011, Austin, TX, USA, January 23-25, 2011. Proceedings*, volume 6538 of *Lecture Notes in Computer Science*, pages 70–87. Springer, 2011.
13. Markus Büttner and Jussi Rintanen. Satisfiability planning with constraints on the number of actions. In Susanne Biundo, Karen L. Myers, and Kanna Rajan, editors, *Proceedings of the Fifteenth International Conference on Automated Planning and Scheduling (ICAPS 2005), June 5-10 2005, Monterey, California, USA*, pages 292–299. AAAI, 2005.
14. Jessica Davies. *Solving MAXSAT by Decoupling Optimization and Satisfaction*. Dissertation, University of Toronto, 2013.
15. Jessica Davies and Fahiem Bacchus. Exploiting the power of mip solvers in maxsat. In Matti Järvisalo and Allen Van Gelder, editors, *Theory and Applications of Satisfiability Testing - SAT 2013 - 16th International Conference, Helsinki, Finland, July 8-12, 2013. Proceedings*, volume 7962 of *Lecture Notes in Computer Science*, pages 166–181. Springer, 2013.

16. Jessica Davies and Fahiem Bacchus. Postponing optimization to speed up MAXSAT solving. In Christian Schulte, editor, *Principles and Practice of Constraint Programming - 19th International Conference, CP 2013, Uppsala, Sweden, September 16-20, 2013. Proceedings*, volume 8124 of *Lecture Notes in Computer Science*, pages 247–262. Springer, 2013.
17. Nachum Dershowitz, Ziyad Hanna, and Alexander Nadel. A scalable algorithm for minimal unsatisfiable core extraction. In Biere and Gomes [8], pages 36–41.
18. Niklas Eén, Alan Mishchenko, and Robert K. Brayton. Efficient implementation of property directed reachability. In Per Bjesse and Anna Slobodová, editors, *International Conference on Formal Methods in Computer-Aided Design, FMCAD '11, Austin, TX, USA, October 30 - November 02, 2011*, pages 125–134. FMCAD Inc., 2011.
19. Niklas Eén and Niklas Sörensson. An extensible SAT-solver. In Enrico Giunchiglia and Armando Tacchella, editors, *SAT*, volume 2919 of *Lecture Notes in Computer Science*, pages 502–518. Springer, 2003.
20. Daniel Frost and Rina Dechter. In search of the best constraint satisfaction search. In *AAAI*, pages 301–306, 1994.
21. Zhaohui Fu and Sharad Malik. On solving the partial MAX-SAT problem. In Biere and Gomes [8], pages 252–265.
22. Martin Gebser, Benjamin Kaufmann, and Torsten Schaub. Conflict-driven answer set solving: From theory to practice. *Artif. Intell.*, 187:52–89, 2012.
23. Pierre Hansen and Brigitte Jaumard. Algorithms for the maximum satisfiability problem. *Computing*, 44(4):279–303, Dec 1990.
24. Richard M. Karp. Reducibility among combinatorial problems. In Raymond E. Miller and James W. Thatcher, editors, *Proceedings of a symposium on the Complexity of Computer Computations, held March 20-22, 1972, at the IBM Thomas J. Watson Research Center, Yorktown Heights, New York.*, The IBM Research Symposia Series, pages 85–103. Plenum Press, New York, 1972.
25. Donald E. Knuth. *The Art of Computer Programming, Volume 4, Fascicle 6: Satisfiability*. Addison Wesley, December 2015.
26. João Marques-Silva, Josep Argelich, Ana Graça, and Inês Lynce. Boolean lexicographic optimization: algorithms & applications. *Ann. Math. Artif. Intell.*, 62(3-4):317–343, 2011.
27. João Marques-Silva and Jordi Planes. On using unsatisfiability for solving maximum satisfiability. *CoRR*, abs/0712.1097, 2007.
28. Ruben Martins, Saurabh Joshi, Vasco M. Manquinho, and Inês Lynce. Incremental cardinality constraints for maxsat. In O'Sullivan [39], pages 531–548.
29. António Morgado, Carmine Dodaro, and Joao Marques-Silva. Core-guided maxsat with soft cardinality constraints. In O'Sullivan [39], pages 564–573.
30. Matthew W. Moskewicz, Conor F. Madigan, Ying Zhao, Lintao Zhang, and Sharad Malik. Chaff: Engineering an efficient SAT solver. In *Proceedings of the 38th Design Automation Conference, DAC 2001, Las Vegas, NV, USA, June 18-22, 2001*, pages 530–535. ACM, 2001.
31. Alexander Nadel. Solving maxsat with bit-vector optimization: Benchmarks and detailed results. https://goo.gl/VS1WJg.
32. Alexander Nadel. Routing under constraints. In Ruzica Piskac and Muralidhar Talupur, editors, *2016 Formal Methods in Computer-Aided Design, FMCAD 2016, Mountain View, CA, USA, October 3-6, 2016*, pages 125–132. IEEE, 2016.
33. Alexander Nadel. A correct-by-decision solution for simultaneous place and route. In Rupak Majumdar and Viktor Kuncak, editors, *Computer Aided Verification -*

29th International Conference, CAV 2017, Heidelberg, Germany, July 24-28, 2017, Proceedings, Part II, volume 10427 of *Lecture Notes in Computer Science*, pages 436–452. Springer, 2017.

34. Alexander Nadel and Vadim Ryvchin. Efficient SAT solving under assumptions. In Alessandro Cimatti and Roberto Sebastiani, editors, *Theory and Applications of Satisfiability Testing - SAT 2012 - 15th International Conference, Trento, Italy, June 17-20, 2012. Proceedings*, volume 7317 of *Lecture Notes in Computer Science*, pages 242–255. Springer, 2012.
35. Alexander Nadel and Vadim Ryvchin. Bit-vector optimization. In Marsha Chechik and Jean-François Raskin, editors, *Tools and Algorithms for the Construction and Analysis of Systems - 22nd International Conference, TACAS 2016, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2016, Eindhoven, The Netherlands, April 2-8, 2016, Proceedings*, volume 9636 of *Lecture Notes in Computer Science*, pages 851–867. Springer, 2016.
36. Alexander Nadel, Vadim Ryvchin, and Ofer Strichman. Accelerated deletion-based extraction of minimal unsatisfiable cores. *JSAT*, 9:27–51, 2014.
37. Alexander Nadel, Vadim Ryvchin, and Ofer Strichman. Ultimately incremental SAT. In Sinz and Egly [45], pages 206–218.
38. Miguel Neves, Ruben Martins, Mikolás Janota, Inês Lynce, and Vasco M. Manquinho. Exploiting resolution-based representations for maxsat solving. In Marijn Heule and Sean Weaver, editors, *Theory and Applications of Satisfiability Testing - SAT 2015 - 18th International Conference, Austin, TX, USA, September 24-27, 2015, Proceedings*, volume 9340 of *Lecture Notes in Computer Science*, pages 272–286. Springer, 2015.
39. Barry 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*. Springer, 2014.
40. Ana Petkovska. *Exploiting Satisfiability Solvers for Efficient Logic Synthesis*. Dissertation, École Polytechnique Fédérale de Lausanne, 2017.
41. Ana Petkovska, Alan Mishchenko, Mathias Soeken, Giovanni De Micheli, Robert K. Brayton, and Paolo Ienne. Fast generation of lexicographic satisfiable assignments: enabling canonicity in sat-based applications. In Frank Liu, editor, *Proceedings of the 35th International Conference on Computer-Aided Design, ICCAD 2016, Austin, TX, USA, November 7-10, 2016*, page 4. ACM, 2016.
42. Knot Pipatsrisawat and Adnan Darwiche. A lightweight component caching scheme for satisfiability solvers. In *SAT*, pages 294–299, 2007.
43. Xujie Si, Xin Zhang, Vasco M. Manquinho, Mikolás Janota, Alexey Ignatiev, and Mayur Naik. On incremental core-guided maxsat solving. In Michel Rueher, editor, *Principles and Practice of Constraint Programming - 22nd International Conference, CP 2016, Toulouse, France, September 5-9, 2016, Proceedings*, volume 9892 of *Lecture Notes in Computer Science*, pages 473–482. Springer, 2016.
44. João P. Marques Silva and Karem A. Sakallah. GRASP: A search algorithm for propositional satisfiability. *IEEE Trans. Computers*, 48(5):506–521, 1999.
45. Carsten Sinz and Uwe Egly, editors. *Theory and Applications of Satisfiability Testing - SAT 2014 - 17th International Conference, Held as Part of the Vienna Summer of Logic, VSL 2014, Vienna, Austria, July 14-17, 2014. Proceedings*, volume 8561 of *Lecture Notes in Computer Science*. Springer, 2014.
46. Ofer Strichman. Tuning SAT checkers for bounded model checking. In E. Allen Emerson and A. Prasad Sistla, editors, *CAV*, volume 1855 of *Lecture Notes in Computer Science*, pages 480–494. Springer, 2000.

47. Takayuki Sugawara. Maxroster:solver description. In Ansotegui et al. [1], page 12.