Aperture: an Anytime, Complete, and Incremental Solver for MaxSAT and Beyond (Work-in-Progress)

Alexander Nadel
Faculty of Data and Decision Sciences, Technion, Haifa, Israel

Yam Slonimski
Faculty of Data and Decision Sciences, Technion, Haifa, Israel

Ofer Strichman
Faculty of Data and Decision Sciences, Technion, Haifa, Israel

Abstract

Anytime MaxSAT solving is indispensable in applications that require finding near-optimal solutions within a flexible, user-controlled time budget. Recent years have seen substantial progress in anytime MaxSAT algorithms, including the introduction of SAT-based local search, the development of efficient approximation techniques, and significant advances in classical local search methods. However, all leading anytime solvers—including the winners of the three most recent MaxSAT Evaluations (MSEs) in the anytime categories—reuse the code base of a single solver, `tt-open-wbo-inc`, which, while highly efficient, lacks an in-memory API, is non-incremental and incomplete, and contains a substantial amount of dead and legacy code.

We introduce `Aperture`, a new solver for anytime and incremental SAT-based optimization, including MaxSAT, written from scratch in C++ with in-memory APIs for C++ (IPAMIR-compliant) and Python. For MaxSAT, `Aperture` follows a similar high-level algorithmic approach to `tt-open-wbo-inc`, but employs a unified flow for both weighted and unweighted MaxSAT and guarantees completeness.

Performance-wise, `Aperture` is highly competitive as a MaxSAT solver. Under the benchmark sets and evaluation conditions of MSE 2024, it outperforms both the MSE 2024 winner `SPB-MaxSAT-c` and `tt-open-wbo-inc` in all four anytime categories, and in 8 out of 12 anytime categories across the three most recent MSEs (2022–2024). Moreover, `Aperture` outperforms leading non-anytime MaxSAT solvers on 2 out of the 5 incremental benchmark sets from MSE 2022, an evaluation setting in which—unlike the anytime categories—both incrementality and completeness are mandatory.

Beyond MaxSAT, `Aperture` also supports two other paradigms for optimizing an objective function under Boolean constraints, namely Optimization Modulo Bitvectors (OBV) and black-box optimization. Across all three paradigms, the solver is incremental and anytime, while it is complete for MaxSAT and OBV but incomplete for black-box optimization.

2012 ACM Subject Classification Mathematics of computing → Solvers

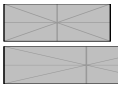
Keywords and phrases MaxSAT, Anytime MaxSAT, Incremental solving, SAT-based Optimization

Digital Object Identifier 10.4230/LIPIcs...

1 Introduction

This paper is work in progress and describes ongoing work; the MaxSAT part is under conference submission.

MaxSAT [2] is an optimization extension of SAT [18] that optimizes a linear Pseudo-Boolean objective subject to Boolean constraints in Conjunctive Normal Form (CNF). Many MaxSAT applications [46, 49, 27, 17] require *anytime* [12, 50] solvers, which are expected to return a feasible solution quickly and then generate progressively improving solutions.


© Alexander Nadel, Yam Slonimski, Ofer Strichman;
licensed under Creative Commons License CC-BY 4.0

Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

Anytimeness is crucial in industrial settings, as it enables obtaining near-optimal solutions for very difficult instances within a reasonable user-controlled time budget [34].

The importance of anytime MaxSAT has driven substantial algorithmic progress in recent years along three complementary directions: significant advancements in classical local search (`SatLike` [7], `NuWLS` [9], `USW` [10], `SPB` [24], `DeepDist` [22]), the development of effective approximation techniques (MrsBeaver [32, 33] and clustering [25]), and the introduction of SAT-based local search (Polosat [35]).

`tt-open-wbo-inc` [36, 38] (hereafter abbreviated as `tt`), derived from `open-wbo-inc` [25], which in turn is based on `open-wbo` [30], is the solver in which these advances have been implemented. Algorithmically, `tt` proceeds in three stages: first, a SAT solver is used to obtain an initial feasible solution; second, classical local search is applied to improve this solution; and finally, a SAT-based approximation phase further refines the solution by repeatedly invoking SAT-based local search or plain SAT as an oracle, and may, under certain conditions, transition to a complete optimization stage. The winners of the three most recent MaxSAT Evaluations (MSEs) in the anytime categories all reuse the `tt` code base, differing only in their classical local search component.

Despite its empirical success, `tt` has limitations that severely restrict its applicability and complicate maintenance and extensibility. In particular, it lacks an in-memory API, does not support incremental MaxSAT solving [32, 41] (even though its underlying SAT-based algorithms are natively incremental), is incomplete, and contains substantial dead and legacy code. Furthermore, it maintains separate and partially duplicated flows for weighted and unweighted MaxSAT.

We introduce `Aperture`[1], a new open-source solver for anytime and incremental SAT-based optimization, including MaxSAT. `Aperture` is written from scratch in C++. For MaxSAT, `Aperture` follows essentially the same high-level algorithmic flow as `tt`, but resolves all of the aforementioned shortcomings. Unlike existing anytime MaxSAT solvers, which are neither complete nor incremental, `Aperture` is both, achieving completeness by guaranteeing a transition to the complete stage once progress in the incomplete stage stalls. It further provides native in-memory APIs in C++ (including IPAMIR [41] compliance) and Python[2], and employs a unified control flow for weighted and unweighted MaxSAT. Tables 1 and 2 compare `tt` and `Aperture` in terms of features and API, respectively.

Table 1 Feature Comparison for MaxSAT Solving

Solver	Anytime	Complete	Incremental	Unified-Flow
`tt`	✓	✗	✗	✗
`Aperture`	✓	✓	✓	✓

Table 2 API Comparison for MaxSAT Solving

Solver	CLI (File)	Native C++	IPAMIR C++	Python
`tt`	✓	✗	✗	✗
`Aperture`	✓	✓	✓	✓

Moreover, `Aperture` supports a broad set of backend SAT and local-search solvers, as shown in Table 3.

Empirically, as a MaxSAT solver, `Aperture` is highly competitive with the state of the art. Under the benchmark sets and evaluation conditions of MSE'24, it outscores both the

[1] Source code: `https://github.com/YamSln/Aperture`
[2] Available on PyPI: `https://pypi.org/project/aperture-solver/` (install with `pip install aperture-solver`)

Table 3 Backends (`Aperture` defaults are underlined)

Solver	SAT Solvers	Local Search Solvers
`tt`	`Glucose`, `IntelSAT`	`USW`
`Aperture`	`Glucose`, `IntelSAT`, `CaDiCaL`, `AE-Kissat-MAB`	`SPB-Band`, `USW`, `DeepDist`

MSE'24 anytime winner `SPB-MaxSAT-c` and `tt` in all four anytime categories, and in 8 out of 12 such categories across the three most recent MSEs (2022–2024). Moreover, although `Aperture` is anytime, it outperforms leading non-anytime MaxSAT solvers on 2 out of the 5 incremental benchmark sets from MSE'22, an evaluation setting in which—unlike in anytime categories—both incrementality and completeness are mandatory.

Beyond MaxSAT, `Aperture` also supports two additional paradigms for optimizing an objective function under Boolean constraints, namely Optimization Modulo Bitvectors (OBV) [40] and black-box optimization [31, 35]. Across all these settings, the solver is incremental and anytime; it is complete for MaxSAT and OBV, and incomplete for black-box optimization.

The remainder of the paper is organized as follows. Sect. 2 introduces preliminaries. Sect. 3 presents the two additional optimization paradigms, OBV and black-box optimization. Sect. 4 describes `Aperture`'s API, Sect. 5 its algorithmic flow, Sect. 6 the experimental results, Sect. 7 concludes, and Appendix 7 provides an ablation study.

2 Preliminaries

A *literal* is either a Boolean variable x or its negation $\neg x$. A *clause* is a finite set (disjunction) of pairwise distinct literals. Given a set of clauses H, a (non-incremental) SAT solver either finds a satisfying assignment (known as a *model* or *solution*) for H or proves that H is unsatisfiable.

Incremental SAT solving [14, 39] is widely used in practice. An incremental SAT solver can be queried multiple times; between queries, the user may add clauses. For each query, the solver receives a set of *assumption literals* (assumptions) A, which apply only to the current query. The solver then determines whether $F \wedge A$ is satisfiable, where F consists of all clauses added so far.

(Non-incremental) MaxSAT extends SAT to the optimization of a linear Pseudo-Boolean (PB) function as follows. Given a set of propositional clauses H and a *target bit-vector (target)* $T = \{t_n, t_{n-1}, \ldots, t_1\}$, where each *target bit* t_i is a Boolean variable associated with a strictly positive integer weight w_i, and assuming that H is satisfiable, MaxSAT finds a model σ of H that minimizes the objective function $\Psi_T(\sigma) = \sum_{i=1}^{n} \sigma(t_i) \times w_i$. A MaxSAT instance is *unweighted* iff all weights are 1; otherwise, it is *weighted*. *Anytime* MaxSAT solvers, as evaluated at MSEs since 2011, produce over time an *improving* sequence of models $\{\mu_1, \mu_2, \ldots, \mu_n\}$ such that for all $j > i$, $\Psi_T(\mu_j) < \Psi_T(\mu_i)$. A *complete* MaxSAT solver, given enough time, is guaranteed to return a model σ of H that is *globally optimal*, i.e., such that $\Psi(\sigma) = \min_{\sigma' \models H} \Psi(\sigma')$.

3 Beyond MaxSAT: Additional Paradigms in Boolean Optimization

Beyond MaxSAT, `Aperture` supports two additional optimization paradigms over Boolean constraints and reuses the corresponding algorithms as key building blocks inside its MaxSAT flow.

3.1 Optimization Modulo Bitvectors (OBV)

Optimization Modulo Bitvectors (OBV) [40] (also known as LEXSAT [44]) is an optimization problem that shares the same input format as unweighted MaxSAT but differs in the semantics of the optimization objective. In OBV, the objective is to minimize the *value* of the target T interpreted as an unsigned integer. OBV has been applied to industrial placement fixing [40, 35], canonicity and canonical SOP generation in synthesis [44, 43], computing lexicographically smallest finite models [21], and encoding lexicographic constraints in SMT-based constraint solving [16]. Intuitively, unlike in unweighted MaxSAT, in OBV the order of bits in T matters: bit i is strictly more important than all bits 0 through $i-1$. This makes OBV considerably easier to solve in practice than MaxSAT. The OBV-BS algorithm [40, 28] exploits this by using incremental SAT queries to progressively fix target bits from the most to the least significant. It has been adopted and generalized within OMT(BV/FP) engines such as OptiMathSAT [45, 47].

3.2 Black-box Optimization

In *black-box optimization* [31, 35], the objective function is not required to have an explicit algebraic or structural representation. Instead, the solver can only access the objective by querying a user-given callback on assignments. Formally, given a satisfiable propositional formula H over variables V and an objective function $\Psi\colon \{0,1\}^{|V|} \to \mathbb{R}$ provided as a black-box callback, the goal is to find a model μ of H that minimizes $\Psi(\mu)$. In this setting, the solver may invoke Ψ at any time on assignments, but cannot reason about the internal structure of Ψ. This abstraction allows optimizing arbitrary Pseudo-Boolean functions, including functions whose encoding into Boolean constraints would be prohibitively heavy, or those available only via querying external tools. Polosat [35] is a state-of-the-art algorithm for black-box optimization. It has been shown to be practically effective for solving the industrial placement problem [35, 11]. Polosat is an incomplete algorithm that can be viewed as a SAT-based local search method exploring the neighborhood of the current best model using incremental SAT queries.

3.3 OBV and Black-box Optimization for Anytime MaxSAT Solving

Beyond their stand-alone interest, OBV-BS and Polosat have become crucial algorithmic subcomponents of state-of-the-art anytime MaxSAT solvers. In particular, `tt` uses OBV-BS within the MrsBeaver algorithm (Sect. 5.2) to approximate unweighted MaxSAT by OBV. It also uses Polosat as a SAT-based local-search oracle during its incomplete optimization stages. `Aperture` uses OBV-BS-based MrsBeaver for *both* unweighted and weighted MaxSAT, and additionally exposes OBV-BS and Polosat via its OBV and BlackBox APIs (Sect. 4).

4 Application Programming Interface (API)

The main idea of `Aperture`'s API is to seamlessly extend the API of an incremental SAT solver to MaxSAT, OBV and black-box optimization.

The API, available in C++ and Python, supports an incremental MaxSAT invocation that, in addition to the assumptions A, receives a weighted target T. It then solves the resulting MaxSAT instance with clauses $F \wedge A$ and T as the current optimization target, where F is the accumulated set of user-provided clauses and A and T are query-specific parameters. Furthermore, `Aperture` implements the standard incremental MaxSAT C++ interface IPAMIR [41], where the mapping between the native API and IPAMIR is straightforward.

Furthermore, Aperture exposes in-memory APIs for OBV and black-box optimization.

For OBV, it receives a target bit-vector T and assumptions A, and minimizes the unsigned integer value of T under $F \wedge A$.

For black-box optimization, it takes assumptions A, a set of observables O (i.e., literals on which the objective function depends), together with a user-defined objective callback that evaluates candidate assignments. The solver then performs incremental anytime optimization of the provided objective over the models of $F \wedge A$.

Finally, Aperture can run standalone from the command line, accepting standard WCNF or native-format input files.

4.1 Essentials of the In-Memory Native API

Fig. 1 summarizes the essential functions of Aperture's native in-memory API.

At a high level, users first create or incrementally update the instance by declaring variables and constraints (Sect. 4.1.1), then invoke one of the solving routines for SAT, MaxSAT, OBV, or black-box optimization (Sect. 4.1.2), optionally configuring MaxSAT-specific behaviour via the additional parameters of **MaxSAT** (Sect. 4.1.3), and finally query the solver for the best solution and its status (Sect. 4.1.4). The API is designed for incremental use, so this flow can be repeated multiple times as the problem evolves. We briefly describe the aforementioned steps below.

4.1.1 Instance Creation

Clauses are permanently added using **add_clause** on top of variables created with **new_var**. In addition, the API allows users to add cardinality and pseudo-Boolean (PB) constraints via **add_cardinality_at_most** and **add_pseudo_boolean_at_most**, respectively; these are internally translated into clauses. Both cardinality and PB constraints can optionally be made *conditional*, in which case they are enforced only if a user-provided selector literal is satisfied.

Below, we let F denote the conjunction of all clauses in the instance, including those obtained by translating cardinality and PB constraints.

4.1.2 Solving

Four solving functions are available. Besides being callable directly via the API, the last two functions, **OBV** and **BlackBox**, are also used internally within Aperture's MaxSAT flow (Sect. 5.2). The functions are as follows:

1. **SAT** runs a SAT query over F under the given assumptions A.
2. **MaxSAT** runs a MaxSAT query over F, optimizing the weighted target T under A. The parameters of **MaxSAT** are discussed in Sect. 4.1.3.
3. **OBV** solves an OBV query by minimizing the unsigned-integer value of T under A using OBV-BS.
4. **BlackBox** performs black-box optimization using a user-provided objective callback that is expected to be strictly monotone in the provided O (the *observables* of [35]): the objective depends only on the restriction of an assignment to O, and flipping any bit of O from 1 to 0 strictly decreases its value. In practice, this monotonicity restriction can be relaxed by including, for each variable of interest, both the variable and its negation in O. **BlackBox** applies Polosat.

4.1.3 MaxSAT's Parameters

MaxSAT has two parameters besides the assumptions A and target T: sol_cb and fix_sol_val?.

sol_cb is a user-provided callback function invoked by the current **MaxSAT** invocation when a new improving solution is found. This enables users to observe intermediate solutions and implement custom stopping criteria: if sol_cb returns $\top$, **MaxSAT** returns immediately; otherwise, it continues.

fix_sol_val? is Boolean. If set to $\top$, the solver attempts to guarantee that, once the current query completes with μ^* as the latest and best model of $F \wedge A$ with respect to T, any returned model σ satisfies $\Psi_T(\sigma) \leq \Psi_T(\mu^*)$. This feature is designed for incremental flows, where preserving or improving solution quality across a sequence of related queries is often desirable. The fix-solution request is guaranteed to succeed if the solver finds an optimal solution; however, it may fail if the query is terminated early by the user.

4.1.4 Getting Status

is_best_sol_val_fixed reports whether the fix-solution request (if any) succeeded for the latest **MaxSAT**, whereas **get_best_solution** returns the current best solution, and **is_best_sol_optimal** indicates whether this solution is guaranteed to be optimal.

```
                                                                                    Instance Creation
new_var() → lit
add_clause(clause: Vec(lit)) → bool
add_cardinality_at_most(T: Vec(lit), k: u_int, [optional] selector: lit) → bool
add_pseudo_boolean_at_most(T: Vec(lit, weight), k: u_int, [optional] selector: lit) → bool
                                                                                    Solving
SAT(A: Vec(lit)) → Status
MaxSAT(A: Vec(lit), T: Vec(lit, weight), fix_sol_val?: bool, sol_cb: Func) → Status
BlackBox(A: Vec(lit), O: Vec(lit), T_func: Func, sol_cb: Func) → Status
OBV(A: Vec(lit), T: Vec(lit), sol_cb: Func) → Status
                                                                                    Getting Status
get_best_solution() → Vec(lit)
is_best_sol_optimal() → bool
is_best_sol_val_fixed() → bool
```

Figure 1 Aperture's Main API Functions

5 Algorithmic Flow for MaxSAT

Fig. 2 illustrates the overall algorithmic flow of Aperture, with a detailed presentation of the MaxSAT-specific flow.

The solver starts in the IDLE stage, where it can receive calls to the API functions.

A call to **MaxSAT** triggers a transition to the INITSAT stage, which queries SAT over the provided clauses to obtain an initial feasible model. For INITSAT, the user may choose a SAT backend from Table 3 that is different from the one used in the remaining stages (among the non-incremental AE-Kissat-MAB [13] and the incremental Glucose [1], IntelSAT [37], and CaDiCaL [5]). If the same (incremental) solver is selected for all stages, its instance is reused; otherwise, once INITSAT completes, the initial solver is destroyed and a new instance of the other solver is created.

The subsequent stages improve the best model so far in an anytime fashion, with each stage starting from the most recent model produced by the previous one. The LS and MRSBEAVER stages are incomplete, whereas the final stage, LINEARSU, guarantees completeness.

This high-level flow is similar to that of tt for unweighted solving, with the key difference that Aperture guarantees that the complete stage is always eventually reached. For

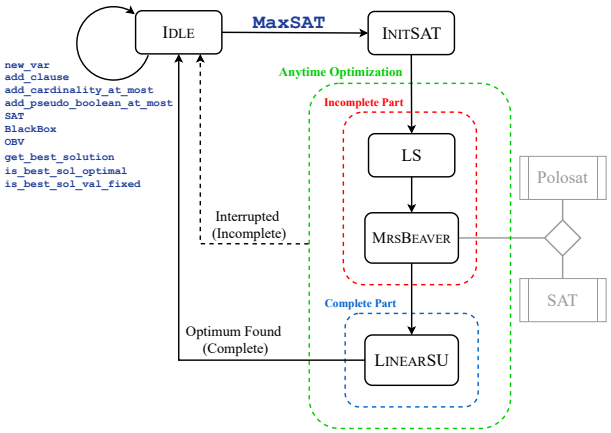


Figure 2 `Aperture`'s MaxSAT solving flow

weighted MaxSAT, `tt` relies on clustering [25] to approximate weighted MaxSAT, whereas `Aperture` applies MrsBeaver [32, 33] to both unweighted and weighted MaxSAT within a single, unified flow.

Employing optimization-oriented polarity and variable selection heuristics in the backend SAT solvers is crucial for the efficiency of anytime MaxSAT [33, 36]. Similarly to `tt`, throughout all SAT stages, `Aperture` overrides the default SAT polarity selection with the *Target-Optimum-Rest-Conservative (TORC)* heuristic and applies the *Target-Score-Bump (TSB)* variable selection heuristic [33]. The intuition is to bias each new solution so that it stays in the neighborhood of the previous best one (Rest–Conservative) while still improving it (Target–Optimum).

Sect. 5.1 presents the LS stage, followed by descriptions of MRSBEAVER and LINEARSU in Sects. 5.2 and 5.3.

5.1 LS: Incomplete Local Search

Classical local search for MaxSAT has a long history [19, 23, 20]. A major breakthrough that established it as a core inprocessing component of state-of-the-art anytime MaxSAT solvers—used to rapidly improve the initial model found by SAT—was achieved with SATLike [29, 8]. Unlike earlier approaches that prioritized feasibility over optimization, SATLike initially emphasizes objective minimization—even allowing constraint violations—and gradually tightens the search to produce an anytime sequence of feasible solutions.

`tt` with SATLike won 3 out of the 4 anytime categories at MSE'21. In the three subsequent MSEs, including the latest MSE'24, replacing SATLike in `tt` with increasingly more efficient local-search approaches enabled the resulting solver to win all four anytime categories at MSE'22 (`NuWLS` [9]), MSE'23 (`USW` [10]), and MSE'24 (`SPB` [24]).

As shown in Table 3, `Aperture` supports several local-search backends, namely `USW`, `SPB` (`SPB-Band` variant), and the recent `DeepDist` [22], with `SPB-Band` used by default.

As in `tt`, `Aperture` runs local search up to a 10^7 flip limit or a problem-dependent timeout (15–25 seconds), extending both limits whenever an improving solution is found.

5.2 MrsBeaver: Incomplete SAT-based Optimization

The MRSBEAVER stage applies the incomplete phase of the MrsBeaver algorithm [32, 33].

Specifically, MrsBeaver rapidly improves the best known solution by repeatedly applying OBV-BS under different target-bit orders. To better approximate unordered solutions, it shuffles or reverses the bit order between iterations and may move zero-valued bits toward more significant positions.

As in `tt`, `Aperture` uses Polosat as the SAT oracle within MrsBeaver and switches back to plain SAT when the number of improving solutions per second drops below 2.5 (unweighted), or when unit propagations per improving solution exceed 10^7 (weighted).

A critical performance detail is that, similarly to `tt`, all the SAT queries within MrsBeaver are always bounded by a conflict threshold of 1000, to avoid getting stuck and to quickly move on to the next query.

In `Aperture`, the transition from the incomplete MrsBeaver stage to LinearSU is governed by the following stopping conditions. The algorithm terminates MrsBeaver when either (i) the expected number of CNF clauses required to encode the Pseudo-Boolean or cardinality constraints used in LinearSU falls below 10^6 for unweighted MaxSAT or $3.5 \cdot 10^6$ for weighted MaxSAT (the *size threshold*), (ii) a global iteration threshold of $2^{31} - 1$ is reached, or (iii) 50 (weighted) or 80 (unweighted) consecutive iterations occur without an improving solution.

The first two conditions are inherited from `tt`; in practice, the global iteration threshold is never reached. The crucial difference in `Aperture` is the third condition, which enables early exit from MrsBeaver when progress stalls, ensuring completeness. This early-exit mechanism is possible in `Aperture` because, unlike `tt`, it applies a compact binary encoding in the LinearSU stage when the unary encoding would be too large, as detailed in Sect. 5.3.

5.3 LinearSU: Complete SAT-based Optimization

Finally, the LinearSU stage applies the complete Linear SAT–UNSAT (LSU) algorithm [4]. Starting from the current best model μ, LSU iteratively blocks all solutions of weight $\geq \Psi(\mu)$—using PB constraints in the weighted case or cardinality constraints in the unweighted case—and searches for improving models using SAT, stopping when the SAT solver returns UNSAT, at which point the last model is optimal.

`Aperture` encodes these PB or cardinality constraints either with unary-sum (totalizer) encodings—the standard [3, 6] or generalized [26] totalizer—or with binary-sum (adder) encodings [48, 15]. When LinearSU is triggered by the size threshold, we use the arc-consistent and propagation-efficient unary-sum encoding; otherwise, we choose the memory-efficient binary-sum encoding.

6 Experimental Results

This section presents an empirical evaluation of the MaxSAT capabilities of `Aperture`, while an evaluation of its stand-alone OBV and black-box optimization capabilities is work in progress.

We first describe how we tested its correctness (Sect. 6.1), then compare it with state-of-the-art anytime MaxSAT solvers on the anytime tracks of the MaxSAT Evaluations 2022–2024 (Sect. 6.2), and finally assess its performance on the incremental benchmarks from the MaxSAT Evaluation 2022 (Sect. 6.3).

All experiments were run on machines with 57 GB of memory and Intel® Xeon® processors at 2.10 GHz.

6.1 Testing Aperture

To verify the correctness of Aperture in non-incremental mode, including its completeness guarantees, we used the off-the-shelf MaxSAT fuzzing suite from [42].

Since incrementality is not supported by [42], we tested Aperture's incremental mode by extending IntelSAT's fuzzer [37] to generate incremental MaxSAT benchmarks in Aperture's file format. Correctness was verified by reducing each incremental check to a sequence of non-incremental checks of the fuzzing suite in [42].

6.2 Anytime MaxSAT Evaluations

To assess the anytime performance of Aperture, we evaluated it under the benchmark sets and evaluation conditions of the anytime tracks of the three most recent MSEs (2022–2024). We compared Aperture against the winners of the anytime categories of MSE'24, SPB-MaxSAT-c-Band and SPB-MaxSAT-c-FPS, as well as against tt in its two configurations with Glucose and IntelSAT as backend SAT solvers. Since both SPB and tt were evaluated in two variants, we likewise include two configurations of Aperture, using Glucose and IntelSAT, to ensure a fair comparison.

For weighted MaxSAT only, we used AE-Kissat-MAB as the SAT solver for the initial SAT query.

Following the anytime tracks of recent MSEs, we computed, for each solver S, time interval T, and instance, a score $c \in [0, 1]$ defined as $c = (1 + w_{\min})/(1 + w_S(T))$, where $w_{\min}$ is the minimal known weight for the instance and $w_S(T)$ is the weight found by S within time interval T. We used two time intervals, $T = 60\,\text{s}$ and $T = 300\,\text{s}$, for both unweighted and weighted MaxSAT, and compared solvers using the overall average of these scores.

Table 4 summarizes the results. Overall, Aperture exhibits strong performance, outperforming the leading anytime solvers in all four MSE'24 categories and in 5 out of 6 unweighted and 3 out of 6 weighted categories across the three evaluations. Aperture's performance on unweighted benchmarks (where all solvers use MrsBeaver [32, 33]) is consistently superior or close to that of the other solvers, but it occasionally lags behind on weighted benchmarks.

Our analysis shows that this gap stems from the effectiveness of the clustering approximation [25] employed by all other tt-based solvers in weighted mode, which Aperture does not currently use. Instead, Aperture also relies on MrsBeaver in weighted mode. While our results indicate that MrsBeaver is competitive—indeed, sufficient to achieve a winning performance on the MSE'24 benchmarks—we plan to investigate how clustering can be integrated into Aperture's workflow as part of future work.

6.3 Incremental MaxSAT Evaluation 2022

We further evaluated Aperture on the incremental track benchmarks from MSE'22, an evaluation setting in which, unlike for the anytime categories, both incrementality and completeness are mandatory.

In this setting, each benchmark family consists of 100 incremental instances, and a solver is credited with solving an instance only if it correctly solves the entire sequence of related MaxSAT queries in that instance. The timeout for each instance is 7200 seconds, and timed-out instances are not counted toward the average-time calculation.

We ran all solvers that participated in this track, none of which is anytime. To the best of our knowledge, this is the first time that an anytime MaxSAT solver has been systematically evaluated against exact solvers implementing fundamentally different algorithmic families [2].

Table 4 Anytime MaxSAT Evaluations

MaxSAT Evaluation 2024	Unweighted		Weighted	
Solver	60s	300s	60s	300s
Aperture-Glucose	0.571	0.647	0.879	0.932
Aperture-IntelSAT	0.545	0.641	0.872	0.919
SPB-MaxSAT-c-FPS	0.553	0.619	0.864	0.913
SPB-MaxSAT-c-Band	0.562	0.637	0.855	0.906
tt-Glucose	0.559	0.636	0.857	0.915
tt-IntelSAT	0.549	0.642	0.861	0.911
MaxSAT Evaluation 2023	Unweighted		Weighted	
Solver	60s	300s	60s	300s
Aperture-Glucose	0.810	0.882	0.796	0.881
Aperture-IntelSAT	0.809	0.873	0.800	0.888
SPB-MaxSAT-c-FPS	0.786	0.869	0.807	0.886
SPB-MaxSAT-c-Band	0.797	0.873	0.816	0.902
tt-Glucose	0.815	0.880	0.796	0.856
tt-IntelSAT	0.799	0.873	0.813	0.891
MaxSAT Evaluation 2022	Unweighted		Weighted	
Solver	60s	300s	60s	300s
Aperture-Glucose	0.854	0.910	0.801	0.870
Aperture-IntelSAT	0.840	0.913	0.785	0.837
SPB-MaxSAT-c-FPS	0.850	0.896	0.788	0.872
SPB-MaxSAT-c-Band	0.849	0.901	0.789	0.876
tt-Glucose	0.853	0.906	0.791	0.899
tt-IntelSAT	0.835	0.907	0.780	0.849

Table 5 reports, for each benchmark family and solver, the number of solved instances and the average solving time over solved ones.

The main result is that `Aperture` outperforms the existing incremental solvers on 2 out of the 5 benchmark families (`BiOptSat` and `AdaBoost`). This demonstrates that anytime algorithms can be useful and complementary to exact algorithms even when completeness is required.

7 Conclusion

We presented `Aperture`, a new anytime and incremental solver for SAT-based optimization that addresses key architectural and usability limitations of existing state-of-the-art anytime MaxSAT solvers, and extends the available SAT-based optimization API to Optimization Modulo Bitvectors (OBV) and black-box optimization. `Aperture` offers native in-memory APIs in C++ and Python, a unified flow for weighted and unweighted MaxSAT, and guarantees completeness for MaxSAT and OBV without sacrificing anytime performance, while also supporting incomplete black-box optimization.

Empirically, as a MaxSAT solver, `Aperture` is highly competitive with the best anytime solvers, outperforming them in all anytime categories of MSE'24 and 8 / 12 categories across MSEs 2022–2024. It also outperforms exact incremental solvers on 2 of 5 families in the Incremental track of MSE'22, showing for the first time that anytime algorithms can complement exact ones— excelling on some families and lagging on others—even when completeness is mandatory.

Future work includes exploiting `Aperture`'s extensive MaxSAT, OBV, and black-box optimization capabilities—including anytimeness and incrementality—to boost the performance of constraint solving on real-world optimization problems. We hope that `Aperture` will serve as a platform for researchers and practitioners to take advantage of SAT-based optimization in solving a wide range of problems.

Table 5 Incremental MaxSAT Evaluation 2022

BiOptSat Solver	# Solved (100)	Average Time (s)
Aperture	53	668
iMaxHS	52	841
UwrMaxSAT	52	1216
UwrMaxSATScip	48	1580
EvalMaxSAT	32	925
SeeSaw Solver	**# Solved (100)**	**Average Time (s)**
EvalMaxSAT	19	946
UwrMaxSATScip	19	1057
UwrMaxSAT	8	2290
iMaxHS	4	2149
Aperture	–	–
ExtEnf. Solver	**# Solved (100)**	**Average Time (s)**
UwrMaxSATScip	47	661
EvalMaxSAT	41	564
UwrMaxSAT	38	691
iMaxHS	37	579
Aperture	14	780
AdaBoost Solver	**# Solved (100)**	**Average Time (s)**
Aperture	36	1307
UwrMaxSATScip	20	896
EvalMaxSAT	19	1324
UwrMaxSAT	18	2984
iMaxHS	17	1129
PDR Solver	**# Solved (100)**	**Average Time (s)**
EvalMaxSAT	46	920
iMaxHS	40	996
UwrMaxSATScip	37	684
UwrMaxSAT	33	739
Aperture	24	1538

Appendix A: Ablation Study

To justify our choice of Aperture's backend SAT and local-search solvers, we conducted an ablation study over the MSE'24 anytime benchmark sets. In all experiments, we varied independently each of the three solvers (or *axes*) applied by the MaxSAT flow, as described in Sect. 5, using the backends summarized in Table 3:

1. the SAT solver used for the initial SAT query (InitSAT), chosen among all SAT backends in Table 3, including the non-incremental AE-Kissat-MAB,
2. the local-search solver used in the LS stage, chosen among the LS backends in Table 3,
3. the main incremental SAT backend used in the remaining stages (all SAT backends except the non-incremental AE-Kissat-MAB).

Recall from Sect. 5 that if the same incremental SAT solver is selected for all stages, its instance is reused; otherwise, once InitSAT completes, the initial solver is destroyed and a new instance of the other SAT solver is created.

For each setting (unweighted and weighted), we start from a fixed default configuration and modify exactly one axis at a time, keeping the other two axes at their default values, thereby performing a single-axis ablation. In the suffix of each Aperture configuration name, each letter encodes the choice along one axis, in the order (initial SAT solver, local-search solver, main SAT backend). The letter corresponding to the modified axis in each configuration is highlighted in bold. The notation is as follows: K = AE-Kissat-MAB, G = Glucose, I = IntelSAT, C = CaDiCaL, B = SPB-Band, U = USW, D = DeepDist.

Unweighted Anytime MSE'24

For unweighted MaxSAT, our default configuration is `Aperture-G-B-G`, which corresponds to: `G` = `Glucose` for the initial SAT query, `B` = `SPB-Band` as the local-search solver, `G` = `Glucose` as the main SAT backend, matching the default choices in Table 3.

The evaluated configurations and their corresponding results on the MSE'24 anytime unweighted benchmarks are presented in Table 6. For context, the table also reports the scores of the two variants of the MSE'24 winner `SPB-MaxSAT-c` and of `tt`.

The baseline `Aperture-G-B-G` attains the best average scores at both 60s and 300s, while single-axis modifications degrade performance. This supports our choice of `Aperture-G-B-G` as the default unweighted configuration.

Weighted Anytime MSE'24

For weighted MaxSAT, our default configuration is `Aperture-K-B-G`, which uses: `K` = `AE-Kissat-MAB` for the initial SAT query, `B` = `SPB-Band` as the local-search solver, `G` = `Glucose` as the main SAT backend,

The resulting configurations and their performance, as well as that of `SPB-MaxSAT-c` and `tt`, on the MSE'24 anytime weighted benchmarks are shown in Table 7. Here, `Aperture-K-B-G` provides a strong baseline: at 60s it achieves the best average score among all `Aperture` configurations, while at 300s two variants (`Aperture-K-D-G` and `Aperture-I-B-G`) perform slightly better. However, these variants lag behind `Aperture-K-B-G` by a larger margin at 60s, so none of them dominates across both time limits. This trade-off motivates our choice of `Aperture-K-B-G` as the default weighted configuration, balancing short- and longer-time performance on the MSE'24 benchmarks.

Table 6 Anytime MSE'24: Unweighted

Solver	60s	300s
`Aperture-G-B-G`	0.571	0.647
`Aperture-`**`K`**`-B-G`	0.542	0.633
`Aperture-`**`C`**`-B-G`	0.521	0.630
`Aperture-`**`I`**`-B-G`	0.527	0.612
`Aperture-G-`**`D`**`-G`	0.569	0.645
`Aperture-G-`**`U`**`-G`	0.568	0.645
`Aperture-G-B-`**`C`**	0.547	0.619
`Aperture-G-B-`**`I`**	0.541	0.627
`SPB-MaxSAT-c-FPS`	0.553	0.619
`SPB-MaxSAT-c-Band`	0.562	0.637
`tt-Glucose`	0.559	0.636
`tt-IntelSAT`	0.549	0.642

Table 7 Anytime MSE'24: Weighted

Solver	60s	300s
`Aperture-K-B-G`	0.879	0.932
`Aperture-`**`G`**`-B-G`	0.872	0.928
`Aperture-`**`C`**`-B-G`	0.872	0.925
`Aperture-`**`I`**`-B-G`	0.866	0.933
`Aperture-K-`**`U`**`-G`	0.871	0.923
`Aperture-K-`**`D`**`-G`	0.870	0.938
`Aperture-K-B-`**`C`**	0.856	0.922
`Aperture-K-B-`**`I`**	0.872	0.919
`SPB-MaxSAT-c-FPS`	0.864	0.913
`SPB-MaxSAT-c-Band`	0.855	0.906
`tt-Glucose`	0.857	0.915
`tt-IntelSAT`	0.861	0.911

References

1 Gilles Audemard and Laurent Simon. On the glucose SAT solver. *Int. J. Artif. Intell. Tools*, 27(1):1840001:1–1840001:25, 2018. doi:10.1142/S0218213018400018.

2 Fahiem Bacchus, Matti Järvisalo, and Ruben Martins. Maximum satisfiability. In Armin Biere, Marijn Heule, Hans van Maaren, and Toby Walsh, editors, *Handbook of Satisfiability - Second Edition*, Frontiers in Artificial Intelligence and Applications, pages 929–991. IOS Press, 2021. doi:10.3233/FAIA201008.

3 Olivier Bailleux and Yacine Boufkhad. Efficient CNF encoding of boolean cardinality constraints. In *CP 2003*, pages 108–122, 2003. doi:10.1007/978-3-540-45193-8_8.

4 Daniel Le Berre and Anne Parrain. The sat4j library, release 2.2. *JSAT*, 7(2-3):59–6, 2010. URL: http://jsat.ewi.tudelft.nl/content/volume7/JSAT7_4_LeBerre.pdf.

5 Armin Biere, Tobias Faller, Katalin Fazekas, Mathias Fleury, Nils Froleyks, and Florian Pollitt. Cadical 2.0. In Arie Gurfinkel and Vijay Ganesh, editors, *Computer Aided Verification - 36th International Conference, CAV 2024, Montreal, QC, Canada, July 24-27, 2024, Proceedings, Part I*, Lecture Notes in Computer Science, pages 133–152. Springer, 2024. doi:10.1007/978-3-031-65627-9_7.

6 Markus Büttner and Jussi Rintanen. Satisfiability planning with constraints on the number of actions. In *ICAPS 2005*, pages 292–299, 2005. URL: http://www.aaai.org/Library/ICAPS/2005/icaps05-030.php.

7 Shaowei Cai and Zhendong Lei. Old techniques in new ways: Clause weighting, unit propagation and hybridization for maximum satisfiability. *Artificial Intelligence*, 287:103354, 2020.

8 Shaowei Cai and Zhengyu Lei. Towards bridging local search for sat and maxsat. *Artificial Intelligence*, 285:103291, 2020.

9 Yi Chu, Shaowei Cai, and Chuan Luo. Nuwls: Improving local search for (weighted) partial maxsat by new weighting techniques. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 37, pages 3915–3923, 2023.

10 Yi Chu, Chu-Min Li, Furong Ye, and Shaowei Cai. Enhancing maxsat local search via a unified soft clause weighting scheme. In Supratik Chakraborty and Jie-Hong Roland Jiang, editors, *27th International Conference on Theory and Applications of Satisfiability Testing, SAT 2024, Pune, India, August 21-24, 2024*, LIPIcs, pages 8:1–8:18. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2024. URL: https://doi.org/10.4230/LIPIcs.SAT.2024.8, doi:10.4230/LIPICS.SAT.2024.8.

11 Aviad Cohen, Alexander Nadel, and Vadim Ryvchin. Local search with a SAT oracle for combinatorial optimization. In Jan Friso Groote and Kim Guldstrand Larsen, editors, *Tools and Algorithms for the Construction and Analysis of Systems - 27th International Conference, TACAS 2021, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2021, Luxembourg City, Luxembourg, March 27 - April 1, 2021, Proceedings, Part II*, Lecture Notes in Computer Science, pages 87–104. Springer, 2021. doi:10.1007/978-3-030-72013-1_5.

12 Thomas L Dean, Mark S Boddy, et al. An analysis of time-dependent planning. In *AAAI*, volume 88, pages 49–54, 1988.

13 Hang Ding, Mao Luo, Chu-Min Li, Shunwei Li, Runyao Chen, Caiquan Xiong, and Xinyun Wu. AE-Kissat-MAB: Automated evolution of kissat with multi-armed bandits. In Cayden Codel, Katalin Fazekas, Marijn J. H. Heule, and Markus Iser, editors, *Proceedings of SAT Competition 2025: Solver and Benchmark Descriptions*. TU Wien, 2025. URL: https://repositum.tuwien.at/handle/20.500.12708/218424.

14 Niklas Eén and Niklas Sörensson. An extensible sat-solver. In *International conference on theory and applications of satisfiability testing*, pages 502–518. Springer, 2003.

15 Niklas Eén and Niklas Sörensson. Translating pseudo-boolean constraints into SAT. *J. Satisf. Boolean Model. Comput.*, 2(1-4):1–26, 2006. URL: https://doi.org/10.3233/sat190014, doi:10.3233/SAT190014.

16 Hani Abdalla Muftah Elgabou. Encoding the lexicographic ordering constraint in satisfiability modulo theories. MSc by research, University of York, York, UK, 2015. URL: https://etheses.whiterose.ac.uk/10387/1/Encoding%20The%20Lexicographic%20Ordering%20Constraint%20in%20Satisfiability%20Modulo%20Theories.pdf.

17 Yu Feng, Osbert Bastani, Ruben Martins, Isil Dillig, and Saswat Anand. Automated synthesis of semantic malware signatures using maximum satisfiability. *arXiv preprint arXiv:1608.06254*, 2016.

18 John Franco and John Martin. A history of satisfiability. In Armin Biere, Marijn Heule, Hans van Maaren, and Toby Walsh, editors, *Handbook of Satisfiability - Second Edition*, Frontiers in Artificial Intelligence and Applications, pages 3–74. IOS Press, 2021. doi:10.3233/FAIA200984.

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

20 Holger H. Hoos and Thomas Stützle. *Stochastic Local Search: Foundations and Applications*. Morgan Kaufmann, 2005.

21 Petr Jarušek and Peter Jipsen. Sat-based techniques for lexicographically smallest finite models. In *Proceedings of the 8th International Workshop on Boolean Problems*, pages 95–102, 2018.

22 Menghua Jiang, Haokai Gao, Shuhao Chen, and Yin Chen. Enhancing local search for maxsat with deep differentiation clause weighting. *arXiv preprint arXiv:2512.05619*, 2025.

23 Yuhui Jiang, Henry Kautz, and Bart Selman. Solving problems with hard and soft constraints using a stochastic algorithm. In *Proceedings of the International Joint Workshop on Artificial Intelligence Workshop on Soft Constraints*, 1995.

24 Mingming Jin, Kun He, Jiongzhi Zheng, Jinghui Xue, and Zhuo Chen. Combining bandmaxsat and fps with spb-maxsat-c. *MaxSAT Evaluation 2024: Solver and Benchmark Descriptions*, pages 23–24, 2024.

25 Saurabh Joshi, Prateek Kumar, Sukrut Rao, and Ruben Martins. Open-wbo-inc: Approximation strategies for incomplete weighted maxsat. *Journal on Satisfiability, Boolean Modelling and Computation*, 11(1):73–97, 2019.

26 Saurabh Joshi, Ruben Martins, and Vasco M. Manquinho. Generalized totalizer encoding for pseudo-boolean constraints. In *CP 2015*, pages 200–209, 2015. doi:10.1007/978-3-319-23219-5_15.

27 Sepideh Khoshnood, Markus Kusano, and Chao Wang. Concbugassist: constraint solving for diagnosis and repair of concurrency bugs. In *Proceedings of the 2015 international symposium on software testing and analysis*, pages 165–176, 2015.

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

29 Zhengyu Lei and Shaowei Cai. Satlike: A fast local search algorithm for maxsat. In *Proceedings of the 27th International Joint Conference on Artificial Intelligence (IJCAI)*, pages 177–183, 2018.

30 Ruben Martins, Vasco Manquinho, and Inês Lynce. Open-wbo: A modular maxsat solver,. In 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*, Lecture Notes in Computer Science, pages 438–445. Springer, 2014. doi:10.1007/978-3-319-09284-3_33.

31 Igor S. Masich and Lev A. Kazakovtsev. A branch-and-bound algorithm for a pseudo-boolean optimization problem with black-box functions. *Facta Universitatis, Series: Mathematics and Informatics*, 33(2):337 – 360, 2018. URL: http://casopisi.junis.ni.ac.rs/index.php/FUMathInf/article/view/3713, doi:10.22190/FUMI1802337M.

32 Alexander Nadel. Solving maxsat with bit-vector optimization. In *International Conference on Theory and Applications of Satisfiability Testing*, pages 54–72. Springer, 2018.

33 Alexander Nadel. Anytime weighted maxsat with improved polarity selection and bit-vector optimization. In *2019 Formal methods in computer aided design (FMCAD)*, pages 193–202. IEEE, 2019.

34 Alexander Nadel. Anytime algorithms for maxsat and beyond. In *2020 Formal Methods in Computer Aided Design, FMCAD 2020, Haifa, Israel, September 21-24, 2020*, page 1. IEEE, 2020. URL: https://doi.org/10.34727/2020/isbn.978-3-85448-042-6_1, doi:10.34727/2020/ISBN.978-3-85448-042-6_1.

35 Alexander Nadel. On optimizing a generic function in sat. In *2020 Formal Methods in Computer Aided Design (FMCAD)*, pages 205–213. IEEE, 2020.

36 Alexander Nadel. Polarity and variable selection heuristics for sat-based anytime maxsat. *J. Satisf. Boolean Model. Comput.*, 12(1):17–22, 2020. URL: https://doi.org/10.3233/sat-200126, doi:10.3233/SAT-200126.

37 Alexander Nadel. Introducing intel (r) sat solver. In *25th International Conference on Theory and Applications of Satisfiability Testing (SAT 2022)*, pages 8–1. Schloss Dagstuhl–Leibniz-Zentrum für Informatik, 2022.

38 Alexander Nadel. Tt-open-wbo-inc: an efficient anytime maxsat solver. *Journal on Satisfiability, Boolean Modelling and Computation*, 15(1):1–7, 2024.

39 Alexander Nadel and Vadim Ryvchin. Efficient SAT solving under assumptions. In *Theory and Applications of Satisfiability Testing - SAT, Proceedings*, 2012. doi:10.1007/978-3-642-31612-8_19.

40 Alexander Nadel and Vadim Ryvchin. Bit-vector optimization. In *International Conference on Tools and Algorithms for the Construction and Analysis of Systems*, pages 851–867. Springer, 2016.

41 Andreas Niskanen, Jeremias Berg, and Matti Järvisalo. Incremental maximum satisfiability. In *25th International Conference on Theory and Applications of Satisfiability Testing (SAT 2022)*, pages 14–1. Schloss Dagstuhl–Leibniz-Zentrum für Informatik, 2022.

42 Tobias Paxian and Armin Biere. Maxsat fuzzing and delta debugging. *J. Artif. Intell. Res.*, 85, 2026. URL: https://doi.org/10.1613/jair.1.19716, doi:10.1613/JAIR.1.19716.

43 Ana Petkovska. *Exploiting Satisfiability Solvers for Efficient Logic Synthesis*. PhD thesis, École Polytechnique Fédérale de Lausanne (EPFL), Lausanne, Switzerland, 2019.

44 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. URL: http://doi.acm.org/10.1145/2966986.2967040, doi:10.1145/2966986.2967040.

45 Roberto Sebastiani and Patrick Trentin. Optimathsat: A tool for optimization modulo theories. In *Proceedings of the 27th International Conference on Computer Aided Verification (CAV 2015)*, Lecture Notes in Computer Science, pages 447–454. Springer, 2015.

46 Xujie Si, Xin Zhang, Radu Grigore, and Mayur Naik. Maximum satisfiability in software analysis: Applications and techniques. In Rupak Majumdar and Viktor Kuncak, editors, *Computer Aided Verification - 29th International Conference, CAV 2017, Heidelberg, Germany, July 24-28, 2017, Proceedings, Part I*, Lecture Notes in Computer Science, pages 68–94. Springer, 2017. doi:10.1007/978-3-319-63387-9_4.

47 Patrick Trentin and Roberto Sebastiani. Optimization modulo the theories of signed bit-vectors and floating-point numbers. *Journal of Automated Reasoning*, 63(1):229–263, 2019.

48 Joost P. Warners. A linear-time transformation of linear inequalities into conjunctive normal form. *Information Processing Letters*, 68(2):63–69, 1998. doi:10.1016/S0020-0190(98)00104-4.

49 Charlie Shucheng Zhu, Georg Weissenbacher, and Sharad Malik. Post-silicon fault localisation using maximum satisfiability and backbones. In *2011 Formal Methods in Computer-Aided Design (FMCAD)*, pages 63–66. IEEE, 2011.

50 Shlomo Zilberstein. Using anytime algorithms in intelligent systems. *AI magazine*, 17(3):73–73, 1996.