Cubing for Tuning

Haoze Wu[1 2], **Clark Barrett**[3], **Nina Narodytska**[2]
[1]Amherst College
[2]VMware Research by Broadcom
[3]Stanford University
hwu@amherst.edu, barrettc@stanford.edu, n.narodytska@gmail.com

Abstract

We are exploring the problem of building an automated reasoning procedure that adaptively tunes the high-level solving strategy for a given problem. There are two main distinctive characteristics of our approach: tuning is performed solely online, unlike the common use of tuning as an offline process; and tuning data comes exclusively from the given instance, so we do not rely on the availability of similar benchmarks and can work with unique challenging instances. Our approach builds on top of the divide-and-conquer paradigm that naturally serves partitioned sub-problems for an automated tuning algorithm to obtain a good solving strategy. We demonstrate performance improvement on two classes of important problems – SAT-solving and neural network verification – and show that our method can learn unconventional solving strategies in some cases.

1 Introduction

Algorithmic tuning is a common practice among automated reasoning practitioners seeking to optimize a solver's performance on a specific problem domain. Traditionally, tuning is regarded as an offline process, where a set of benchmarks is selected and evaluated against a set of candidate solving strategies, with the goal of learning a good solving strategy to be used for solving new problems. However, offline tuning may not always be feasible – for example, if the solver is used on a single, unique problem instance, or if the practitioner lacks the expertise to navigate the space of possible candidate strategies. Motivated by this observation, we ask the following question: *can the automated reasoning tool find a good solving strategy for a given problem using the problem itself as a source of learning?*

Our key high-level idea is to view the given problem as a generator of representative sub-problems and use them to perform strategy tuning. To perform sub-problem generation, we take advantage of the powerful divide-and-conquer paradigm, which is the state-of-the-art approach for solving challenging problems in many domains. For example, in SAT-solving, a technique called cube-and-conquer (CNC) (Heule et al. 2011) operates by partitioning a challenging SAT problem into thousands of sub-problems with look-ahead techniques and solving the sub-problems in a parallel manner. We exploit the synergy between algorithmic strategy tuning and the cube-and-conquer approach to answer our main question. First, the formula is partitioned into a set of sub-formulas. Next, small subsets of suitable sub-formulas are selected for tuning and validating the solving strategies. Finally, the learned strategy is used for solving the remaining sub-problems.

We present our approach, termed TACO (Tuning Algorithm Configuration Online), as a general solving procedure. We instantiate TACO for the solving of SAT formulas and neural network verification queries, and show that our approach consistently boosts the performance of a CNC-based solving procedure on a range of challenging benchmarks. Additionally, on those benchmarks, our prototype compares favorably against state-of-the-art solvers (Saoudi et al. 2024; Wu et al. 2024) in respective domains, contributing several unique solutions. A preview of our results on a single benchmark is shown in Figure 1. Each point corresponds to a single sub-problem (i.e., cube). We see that TACO discovered a parameter setting that reduces the number of conflicts on a small set of selected cubes (left figure); the conflict-reducing effect of TACO generalizes to the remaining unsolved sub-problems (middle figure); and this reduction in conflicts translates to a reduction in runtime (right figure).

To summarize, the contribution of this paper includes:

- a general methodology, TACO, that customizes the solving strategy of an automated reasoning tool online by learning solely from the given formula;
- instantiation of the methods in two automated reasoning settings – SAT-solving and neural network verification;
- experiments that show the practical benefit of TACO; and
- discussions of unexpected instance-specific strategies discovered by TACO.

2 Tuning Solver Configuration Online

This section describes our cubing-based strategy learning approach. Before we start, we briefly recall the *cube-and-conquer* (CNC) technique, as TACO is based on it. CNC partitions the input formula f into a set of sub-formulas of the the form $f \wedge c$, where c, referred to as a *cube*, is a conjunction of *literals*. The goal of CNC is to create a number of easier sub-problems that can be processed in parallel. For example, if f is a SAT formula and p_1, p_2 are propositional variables,

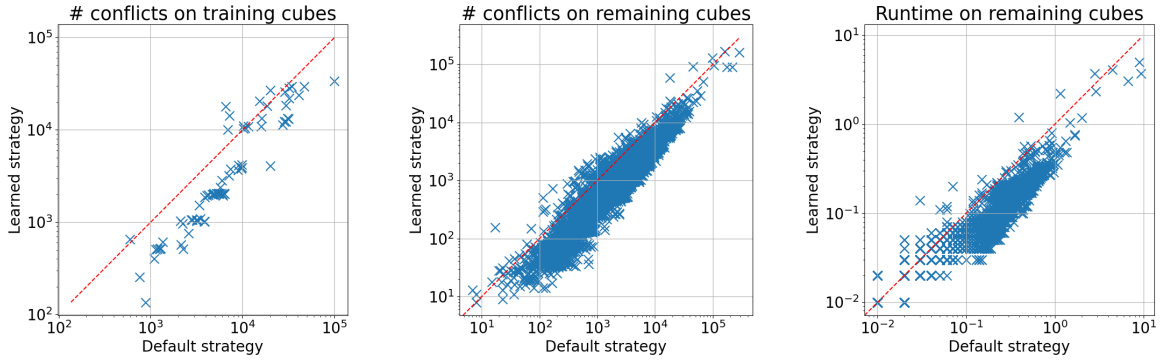

Figure 1: TACO learned a new strategy on a challenging SAT benchmark (`eq.atree.braun.13`). CNC with TACO solved the benchmark in 139 seconds, while it took plain CNC, PAINLESS, and sequential KISSAT 243, 1088, 5279 seconds, respectively. The benchmark deals with multiplier verification and was found especially suitable to be tackled by CNC (Heule et al. 2011).

then a cube c is $(p_1 \wedge p_2)$ and the corresponding sub-formula is $\{f \wedge (p_1 \wedge p_2)\}$. The set of obtained sub-formulas has the property that their disjunction is equi-satisfiable with the original formula, i.e., f is satisfiable iff $\left(\bigvee_{c \in C} f \wedge c\right)$ is satisfiable, where C denotes the set of cubes produced by the partitioning strategy. One valid partition over p_1 and p_2 contains four cubes with the following sub-formulas: $\{f \wedge (p_1 \wedge p_2), f \wedge (p_1 \wedge \neg p_2), f \wedge (\neg p_1 \wedge p_2), f \wedge (\neg p_1 \wedge \neg p_2)\}$. Partitioning strategies for different logical theories have been studied in the past in the context of divide-and-conquer-based SAT/SMT-solving.

2.1 Overview

Figure 2 provides an overview of the proposed workflow.

Cubing. In the cubing stage, the input formula f is partitioned into a set of sub-formulas. For SAT-solving, a popular cubing technique is look-ahead (Heule and van Maaren 2009), which is intended for creating sub-problems that are both easier and *balanced* in terms of difficulty. Upon generating sub-formulas, a CNC-based solver would typically proceed to the solving stage, where pre-determined solving strategies are used to solve each of the sub-formulas. In contrast, we observe that cubing opens up the possibilities for tuning the solver on the fly. Based on this observation, we introduce an online strategy learning phase.

Strategy learning. The strategy learning stage consists of two phases: tuning and validation. We start by describing the tuning phase (the **tuning** box in Figure 2). To perform strategy learning, first, we need a set of cubes to evaluate different candidate solving strategies. Importantly, the cubes need to be *sufficiently challenging* – to differentiate solving strategies, but also *sufficiently easy* – to incur only small overhead when evaluated repeatedly. Cube difficulty is measured by a cost metric (e.g., time or number of conflicts it takes to solve the cube). We propose a procedure COLLECT to obtain such a set of cubes in Sec. 2.2. At this stage, the COLLECT procedure outputs a set of cubes C_{tune} and passes them to the tuning procedure **tune**. The goal of **tune** is to optimize the solving strategy for C_{tune} by searching over a strategy space $\mathcal{V}$ with respect to a cost metric. We focus on cases where $\mathcal{V}$ is formed using a set of configurable parameters for the solving

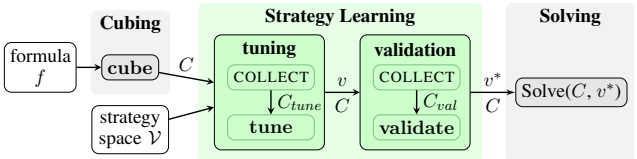

Figure 2: Overview of the TACO-based solving procedure.

procedure. As a result of this phase, we obtain an optimized strategy v. We describe **tune** in Sec. 2.4.

Next, in the validation phase (the **validation** box in Figure 2), we collect a different subset of cubes C_{val} with COLLECT and use C_{val} to validate the learned strategy v. Depending on the validation results, **validate** either accepts the strategy v found in the tuning phase as the final solving strategy v^*, or rejects v and sets v^* to a fallback strategy. We chose to fallback to a sequential portfolio strategy that includes v and the default strategy, but other plausible choices exist, as discussed in Sec. 2.3.

Solving. In the final solving stage, currently untackled cubes C are solved using the learned strategy v^*. If all the sub-formulas are unsatisfiable, then the original formula is unsatisfiable; and if one of the sub-formulas is satisfiable, then the original formula is satisfiable.

2.2 Collecting Suitable Cubes

Designing a robust method to collect a suitable set of cubes for strategy learning is not straightforward. Simply taking a random subset of the initial partition can run into a number of pitfalls. First, if all the sampled cubes are trivial (e.g., can be solved with very low cost), then they cannot effectively distinguish different solving strategies, as each strategy would have a similar (if not the same) cost. On the other hand, if the sampled cubes contain very challenging cubes, then solving them repeatedly with different solving strategies would incur a large overhead. We argue that an ideal cube collection procedure should take the cube difficulty (as defined by a cost range) into account and *guarantee* to return a given number of cubes of the specified difficulties. Such cubes are informative enough to distinguish between candidate strategies yet efficient to evaluate. Alg. 1 describes such

Algorithm 1 Identifying a set of suitable cubes

1: **Input:** Formula f, initial partition C, number of cubes t, set of strategies $\mathcal{V}$
2: **Parameters:** number of cubes to create during re-partitioning $\mathbf{k}$, minimal cost $\mathbf{l}$, maximal cost $\mathbf{u}$
3: **Output:** $\langle \texttt{SAT/UNSAT/UNKNOWN}, C' \rangle$; C' is a set of t cubes if the first output is UNKNOWN, and empty otherwise.
4: **function** COLLECT($f, C, t, \mathcal{V}$)
5: $\quad C' \leftarrow \varnothing$
6: $\quad$ **while** $|C'| < t$ **do**
7: $\quad\quad c \leftarrow sample(C)$
8: $\quad\quad v \leftarrow sample(\mathcal{V})$
9: $\quad\quad res \leftarrow \mathbf{check}(f \wedge c, v, \mathbf{u})$
10: $\quad\quad$ **if** $res = \texttt{SAT}$ **then return** $\langle \texttt{SAT}, \varnothing \rangle$
11: $\quad\quad$ **else if** $res = \texttt{UNSAT}$ and $\mathbf{eval}(f \wedge c, v) \geq \mathbf{l}$ **then**
12: $\quad\quad\quad C' \leftarrow C' \cup \{c\}$
13: $\quad\quad$ **else if** $res = \texttt{UNKNOWN}$ **then**
14: $\quad\quad\quad C \leftarrow C \cup \{c' \wedge c \,|\, c' \in \mathbf{cube}(f \wedge c, \mathbf{k})\}$
15: $\quad\quad C.remove(c)$
16: $\quad\quad$ **if** $|C| = 0$ **then return** $\langle \texttt{UNSAT}, \varnothing \rangle$
17: $\quad$ **return** $\langle \texttt{UNKNOWN}, C' \rangle$

a cube collection procedure, referred to as COLLECT.

The COLLECT procedure takes as input $\langle f, C, t, \mathcal{V} \rangle$, where f is a formula, C is the current set of unsolved cubes, $t \in \mathbb{N}$ is the target number of cubes, and $\mathcal{V}$ is a set of strategies. The procedure makes use of the following sub-procedures: 1) $\mathbf{check}(f, v, u)$ solves the formula f with solving strategy v and cost budget u; the method returns SAT/UNSAT if the f is solved, and UNKNOWN otherwise; 2) $\mathbf{eval}(f, v)$ returns the cost of solving the formula f with solving strategy v; and 3) $\mathbf{cube}(f, k)$ partitions the formula f into k cubes ($k > 1$). We assume the $\mathbf{cube}$ procedure is *sound*, i.e., f and $(\bigvee_{c \in \mathbf{cube}(f,k)} f \wedge c)$ are equi-satisfiable. The COLLECT procedure is also parameterized by $\mathbf{k}$–the number of cubes to create during re-partitioning ($\mathbf{k} > 1$), and $\mathbf{l}$/$\mathbf{u}$–the minimal/maximal cost. The procedure gradually grows a set of suitable cubes C' (initialized at line 5 as an empty set) and terminates either when sufficient cubes that fall in the desired cost range have been collected (i.e., $|C'| = t$), or when the original formula is solved in this process.

Concretely, COLLECT repeatedly removes an unsolved cube c from C (line 7), selects a strategy $v \in \mathcal{V}$ (line 8), and attempts to solve the cube using the strategy with cost budget $\mathbf{u}$ (line 9). If the $\mathbf{check}$ method returns SAT, then the process terminates and returns $\langle \texttt{SAT}, \varnothing \rangle$, concluding that the original formula is satisfiable (line 10); if the $\mathbf{check}$ method returns UNSAT and the cost of the strategy v on solving the cube c is larger than the minimal cost $\mathbf{l}$, then the cube c is added to the collection of suitable cubes C' (line 12); if c is not solved within the cost budget, then c is further partitioned into smaller cubes using the function $\mathbf{cube}$ (line 14) and those cubes are added back to C, which maintains the current set of unsolved cubes. If there is no unsolved cube left, then the procedure returns $\langle \texttt{UNSAT}, \varnothing \rangle$, concluding that the original formula is unsatisfiable (line 16). Finally, if the procedure exited the while loop without terminating, the procedure returns $\langle \texttt{UNKNOWN}, C' \rangle$.

The following proposition characterizes the functionality of Alg. 1 and holds by construction:

Proposition 1. *If the procedure* COLLECT$(f, C, t, \mathcal{V})$ *returns* $\langle \textit{UNKNOWN}, C' \rangle$*, then* $|C'| = t$ *and for each cube* c *in* C'*, there is some solving strategy* $v \in \mathcal{V}$ *such that* $\mathbf{l} \leq \mathbf{eval}(f \wedge c, v) \leq \mathbf{u}$.

The following proposition states the soundness of Alg. 1 and holds because re-partitioning maintains equi-satisfiability:

Proposition 2. *Given a set of cubes* C *such that* $(\bigvee_{c \in C} f \wedge c)$ *and* f *are equi-satisfiable, and a sound* $\mathbf{cube}$ *procedure, the formula* f *is satisfiable if* COLLECT$(f, C, t, \mathcal{V})$ *returns* $\langle \textit{SAT}, \varnothing \rangle$*, and unsatisfiable if it returns* $\langle \textit{UNSAT}, \varnothing \rangle$.

To summarize, given a partitioning of the formula f as a set of cubes C, COLLECT either solves the formula or returns a given number of cubes of specified difficulties.

The termination of Alg. 1 is not always guaranteed, even if all of its sub-routines terminate. For example, if the $\mathbf{check}$ method keeps returning UNKNOWN on all cubes, then neither will $|C'|$ grow, nor will $|C|$ reduce to 0. Conceptually, termination relies on the choices of $\mathcal{V}$, the cost metric, and the cubing procedure $\mathbf{cube}$. We specify a practical sufficient condition for the termination of Alg. 1. The proof is in the extended version of the paper (Wu, Barrett, and Narodytska 2025).

Definition 1 (Cubing trace)**.** *A* cubing trace t *starting from formula* f *with respect to* $\mathbf{cube}(\cdot, \mathbf{k})$ *is defined as a cube sequence* $c_0,\ c_1,\ \ldots$ *where* $c_0 = \top$ *and* $c_{i+1} \in \mathbf{cube}(f \wedge c_i, \mathbf{k})$*. We say the trace* t reaches *cube* c_i *in* i *steps.*

Theorem 1 (Termination via bounded cost reduction)**.** *Assume that all sub-routines of* COLLECT *terminate. And assume that the cubing procedure* $\mathbf{cube}(\cdot, \mathbf{k})$ *satisfies the following property with respect to a set of strategies* $\mathcal{V}$*, and a maximal cost* $\mathbf{u}$*:*

(Bounded cost reduction) *For any formula* f*, every cubing trace starting from* f *with respect to* $\mathbf{cube}(\cdot, \mathbf{k})$ *will reach, in a finite number of steps, a cube* c *such that* $\max_{v \in \mathcal{V}} \mathbf{eval}(f \wedge c, v) < \mathbf{u}$.

Then COLLECT$(f, C, t, \mathcal{V})$ *terminates.*

Intuitively, the bounded cost reduction assumption requires that $\mathbf{cube}$ must lead to sub-formulas that are sufficiently easy to solve by any of the solving strategies in $\mathcal{V}$.

Efficiently implementing Alg. 1. There are a few things to note when it comes to implementing Alg. 1. First, the procedure is highly parallelizable: the input cubes C can be viewed as a dynamically shrinking and growing work queue to be processed in parallel. Second, eagerly invoking the $\mathbf{cube}$ method (line 14) can unnecessarily incur time and memory overhead when there are still unexamined cubes in C. Therefore, one could lazily re-partition UNKNOWN cubes only when all the cubes in C have been examined, but not enough cubes have been collected. Finally, to further reduce the number of $\mathbf{cube}$ calls, one could consider adaptively increasing (e.g., multiplying by some factor r) the value of $\mathbf{u}$ for re-partitioned cubes. Thus, the cost budget for any cube

Algorithm 2 Cube-and-conquer with self-driven algorithmic configuration

1: **Input:** Formula f, strategy space $\mathcal{V}$ (with a default strategy v_0)
2: **Parameters:** number of cubes to create initially $\mathbf{n}$, number of cubes for tuning $\mathbf{T_1}$, number of cubes for validation $\mathbf{T_2}$
3: **Output:** SAT or UNSAT
4: **function** CNC-TACO($f, \mathcal{V}$)
5: $C \leftarrow \mathbf{cube}(f, \mathbf{n})$ ▷ Initial partition
6: $v^* \leftarrow v_0$
7: $res, C_{tune} \leftarrow$ COLLECT$(f, C, \mathbf{T_1}, \mathcal{V})$
8: **if** $res \in \{$SAT, UNSAT$\}$ **then return** res
9: $v \leftarrow \mathbf{tune}(\{f \wedge c \,|\, c \in C_{tune}\}, \mathcal{V})$
10: **if** $v \neq v_0$ **then**
11: $res, C_{val} \leftarrow$ COLLECT$(f, C, \mathbf{T_2}, \{v\})$
12: **if** $res \in \{$SAT, UNSAT$\}$ **then return** res
13: $v^* \leftarrow \mathbf{validate}(\{f \wedge c \,|\, c \in C_{val}\}, v, v_0)$
14: **for** each $c \in C$ **do**
15: **if** $\mathbf{check}(f \wedge c, v^*, \infty) =$ SAT **then return** SAT
16: **return** UNSAT

Strategy Learning

c is $\mathbf{u} \cdot r^n$, where n is the number of re-splits that leads to c ($n = 0$ for original cubes). With this approach, Thms. 2 and 1 still holds, and Thm. 1 holds by changing $\mathbf{u}$ to $\mathbf{u} \cdot r^n$.

2.3 Extending CnC with Online Learning

Alg. 2 introduces an extension of the traditional CNC solving procedure with an online learning phase (CNC-TACO). The goal is to dynamically optimize solving strategies based on fractions of the original formula.

The CNC-TACO procedure takes as input a formula f and a strategy space $\mathcal{V}$ (with a default strategy v_0), and returns either SAT or UNSAT. Apart from the cube collection procedure COLLECT, CNC-TACO also depends on the following sub-procedures: 1) **check** is the same as described in the previous section; we assume that when the cost budget $\mathbf{u}$ is unlimited (∞), **check** returns either SAT or UNSAT; 2) $\mathbf{tune}(F, \mathcal{V})$ optimizes the solving strategy with respect to a set of formulas F over a strategy space $\mathcal{V}$; and 3) $\mathbf{validate}(F, v, v_0)$ compares the performance (i.e., total cost) of a new strategy v with that of the default strategy v_0 on a set of formulas F and returns the final strategy used for solving the unsolved cubes.

We now describe the execution of CNC-TACO. The procedure first creates an initial partition C like traditional CNC, and then enters the strategy learning phase. During this phase, CNC-TACO first attempts to collect $\mathbf{T_1}$ cubes from C for tuning (line 7). Importantly, we assume C is *passed by reference*, meaning that after COLLECT terminates, C contains the latest set of cubes yet to be solved. If the original formula is not solved by COLLECT, the **tune** method is invoked on the set of formulas $F = \{f \wedge c \,|\, c \in C_{tune}\}$ to optimize the solving strategy (line 9). If a better strategy is not found, the strategy learning phase terminates and the remaining cubes are solved with the default strategy v_0. Otherwise, a different set of $\mathbf{T_2}$ cubes, denoted C_{val}, is collected to validate the new strategy v (line 11). Similarly, the original formula might be solved during this second round of cube collection. If this is not the case, the **validate** method is invoked to decide on the final solving strategy v^* (line 13), which is then used to solve the remaining cubes.

The **validate** method is instantiated as follows: if the new strategy v performs better on C_{val}, then return v, otherwise, solve the cube using v with a cost budget of $\mathbf{u}$, and fallback to the default strategy v_0. Alternatively, one could simply fall back to v_0 instead of the sequential portfolio strategy, or even select a different candidate strategy by repeating the tuning phase with a larger $\mathbf{T_1}$.

Design choice: strategies for collection. We would like to call attention to a nuance in the choice of the fourth input to COLLECT, which specifies the set of candidate strategies to pick from when examining a cube. For the tuning cubes C_{tune}, CNC-TACO uses $\mathcal{V}$ as this set of strategies (line 7), which means it is *unbiased* towards any strategies and the collected cubes fall in the suitable range for different strategies. Another plausible design choice is to always collect cubes with the default strategy $\{v_0\}$. We did not opt for the latter since C_{tune} might be biased towards cubes that v_0 already performs well on, leaving less room for improvement. On the other hand, when collecting the validation cubes C_{val}, CNC-TACO only uses the newly learned strategy v (line 11). This means CNC-TACO is *optimistic* and only rejects v if the default strategy v_0 performs better than v on a set of cubes that v performs well on.

2.4 Optimizing the Solving Strategy on the Cubes

We now describe our instantiation of the **tune** procedure (line 9, Alg. 2). Given a set of collected cubes C_{tune}, the goal of **tune** is to optimize the solving strategy in a given strategy space $\mathcal{V}$:

$$\underset{v \in \mathcal{V}}{\text{minimize}} \quad cost(v) := \Sigma_{c \in C_{tune}} \mathbf{eval}(f \wedge c, v).$$

When $\mathcal{V}$ is small, one could evaluate all strategies in $\mathcal{V}$. But this is unaffordable when $|\mathcal{V}|$ is large, especially in the online strategy learning setting. We consider the scenario where one stochastically optimizes the solving strategy while being only allowed to examine $\mathbf{h}$ strategies in the strategy space $\mathcal{V}$, with $\mathbf{h} \ll |\mathcal{V}|$. In principle, any stochastic local search approach can be applied in this setting, and one standard approach with a convergence guarantee is Markov-Chain Monte-Carlo (MCMC) sampling. In our setting, MCMC can be used to generate a sequence of solving strategies with the desirable property that the sequence converges to the strategies with the lowest cost.

A widely-used MCMC method is the Metropolis-Hastings (M-H) Algorithm (Chib and Greenberg 1995), instantiated in the context of TACO as follows: 1. Choose a current strategy v; 2. Propose to replace the current strategy with a new one v', sampled from a *proposal distribution* $q(v'|v)$; 3. If $cost(v') \leq cost(v)$, accept v' as the current strategy; 4. Otherwise, accept v' as the current strategy with some probability $a(v \rightarrow v')$ (e.g., a probability inversely proportional to the increase in the cost); 5. Go to step 2.

This process is repeated until $\mathbf{h}$ samples are drawn. Intuitively, under this scheme, a better proposal is always accepted, while a proposal that increases the cost may still be

accepted. In other words, the algorithm greedily navigates towards a better strategy whenever possible, but can also overcome local minima. In our implementation, the acceptance probability is computed using a standard method (Kass et al. 1998). First, $cost(v)$ is transformed into a probability distribution $p(v) \propto \exp(-\beta \cdot cost(v))$, where $\beta > 0$ is a configurable parameter. The acceptance probability is then computed as:

$$a(v \to v') = \min\left(1, \frac{p(v')}{p(v)}\right) = \min\left(1, \exp\left(\beta \cdot (cost(v) - cost(v'))\right)\right).$$

Under this acceptance probability, the larger the total cost increases going from the current strategy v to the proposed strategy v', the lower the probability of accepting v' as the new current strategy. On the other hand, the larger β is, the more reluctant we are to move to a worse proposal.

Design choice: proposal distribution. To ensure the aforementioned convergence property of MCMC, the proposal distribution must be both *symmetric* and *ergodic*.[1] For discrete search spaces (which is the setting we consider in this paper), a common proposal distribution is the symmetric random walk, which moves to one of the *neighbors* of the current sample at uniform random. This proposal distribution is both symmetric and ergodic. We define the neighbors of a strategy as all strategies for which $\leq k$ parameter values are different. We use $k = 2$ in our implementation.

Design choice: MCMC initialization. The better (i.e., low-cost) the initial strategy is, the shorter it takes for MCMC-sampling to converge (Roy 2020). While a natural choice of the initial strategy is the default strategy v_0, we found that it is empirically advantageous to "probe" every 1-neighbor of v_0 and initialize the MCMC-sampling with the best strategy seen in this process.

While the presented MCMC-sampling method is empirically effective in our setting, we note that alternative solver tuning methods have been studied in the literature (Hoos, Hutter, and Leyton-Brown 2021) and TACO can naturally leverage developments in the stochastic optimization space for further performance improvement.

3 Instantiations of TACO

In this section, we present the instantiation of TACO for two types of logical formulas: SAT formulas and neural network verification (NNV) queries. We prove in the extended version of the paper (Wu, Barrett, and Narodytska 2025) that the bounded cost reduction assumption that guarantees termination can be satisfied for both cases. Our prototype, implemented in PYTHON3, takes as input a formula and runs CNC on the formula with or without TACO.

[1]A proposal distribution q is symmetric if $q(v'|v) = q(v|v')$ for any $v, v' \in \mathcal{V}$ and is ergodic if there is a non-zero probability of reaching a strategy $v \in \mathcal{V}$ from any other strategy $v' \in \mathcal{V}$ in a finite number of steps.

SAT-solving. For SAT-solving, the prototype uses MARCH_CU (Heule et al. 2011) as the cuber and KISSAT (Biere et al. 2024) as the solver, and takes formulas in CNF format. MARCH_CU is a popular cubing tool used in previous CNC work (Heule et al. 2011; Ozdemir, Wu, and Barrett 2021; Heisinger, Fleury, and Biere 2020), and KISSAT is a state-of-the-art SAT-solver. We use the number of conflicts as the cost metric, which has been used as a proxy for runtime in prior offline tuning work (Beskyd and Surynek 2022). The strategy space consists of combinations of possible values of 9 KISSAT command-line parameters that we believe have a significant impact on the branching decisions. We allow at most four parameters to deviate from the default value, resulting in a total of 349 unique parameter settings. Tab. 1 shows the parameters and their candidate values. We also evaluate the effect of using a different strategy space in the extended version of the paper (Wu, Barrett, and Narodytska 2025).

Parameter	default	alts	Parameter	default	alts
`bump`	1	0	`phase`	1	0
`bumpreasons`	1	0	`stable`	1	0, 2
`chrono`	1	0	`target`	1	0
`eliminate`	1	0	`tumble`	1	0
`forcephase`	0	1			

Table 1: Parameters in the strategy space for KISSAT

Neural Network Verification. For NNV, the prototype uses the MARABOU (Wu et al. 2024) verifier as both the cuber and the solver, and takes MARABOU's input query format. MARABOU is a state-of-the-art SMT-based neural network verification tool. We chose NNV with MARABOU as the second case study because CNC-like approach has been shown to improve MARABOU's performance (Katz et al. 2019; Wu et al. 2020). Since MARABOU performs DPLL(T)-like search without conflict-driven clause learning, we use the number of decisions as the cost metric instead of the number of conflicts. Two parameters are chosen for tuning: `pl-split-freq` and `branch`. The former specifies the frequency of performing a case-split on internal neurons when input-splitting is enabled (e.g., 1 means always splitting on internal neurons; 5 means splitting on an internal neuron every 4 input-splittings). The latter determines the heuristics for selecting which internal neuron to branch on. The strategy space is described in Tab. 2. In total, $\mathcal{V}_{\text{MARABOU}}$ contains 12 candidate solving strategies.

Parameter	default	alts
`pl-split-freq`	10	1, 2, 5
`branch`	pseudo-impact	babsr, polarity

Table 2: Parameters in the strategy space for MARABOU

Our prototype could be extended to handle other logical formulas by providing the corresponding implementations of the **cube**, **check**, and **eval** methods. Other implementation details are described in the extended version of the paper (Wu, Barrett, and Narodytska 2025).

Choice of the cost metric. Importantly, the assumption that the numbers of conflicts or decisions are good proxies for runtime is not always true. For example, if we turn off learned clause deletion, the number of conflicts would likely reduce, but the runtime can significantly increase due to overhead in unit propagation. This fact needs to be taken into account when choosing which parameter to include in the strategy space: the parameter's impact on the cost metric and its impact on the runtime must *align*. If a parameter setting reduces the cost metric but simultaneously increases runtime, the tuning procedure would be misled to favor the parameter setting. We believe that coming up with a deterministic, easy-to-measure, and consistently faithful performance proxy is a highly non-trivial research topic.

4 Experimental Evaluation

We conduct a number of experiments in the two aforementioned automated reasoning applications to evaluate the proposed method. The main questions we investigate are:

1. Can TACO lead to performance gain? [Yes]
2. Can TACO offer new formula-solving insights? [In several cases]

4.1 Case study: SAT-Solving

In SAT-solving, CNC is recognized as a method for tackling challenging instances that cannot be efficiently solved by sequential solver or a portfolio strategy. To study the effectiveness of TACO, we focus on benchmarks studied in existing CNC literature, which were known to be challenging for state-of-the-art SAT-solvers:

- `CnC`: benchmarks studied in the original CNC work (Heule et al. 2011);
- `cruxmiter`: an arithmetic verification benchmark family (Ritirc, Biere, and Kauers 2017) that was studied in a distributed CNC work PARACOOBA (Heisinger, Fleury, and Biere 2020).

In addition, to investigate the effectiveness of TACO on the latest hard problems, we propose to evaluate CNC-based techniques on a new set of benchmarks that consists of unsolved problems from recent SAT competitions:

- `SC-hard`: benchmarks from recent SAT Competitions (22–24) that were not solved by any participating tools.

We perform a direct comparison between CNC with and without TACO. We use MARCH_CU to partition a given benchmark with maximal cube depth 15. This means at most 32768 cubes are generated, a number on par with what was used in prior work (Heisinger, Fleury, and Biere 2020). In addition, we also compare the performance of our prototype with the state-of-the-art parallel SAT solver PAINLESS, which won the parallel track of SAT Competition 2024 (Saoudi et al. 2024). Experiments were performed on a cluster equipped with Dell PowerEdge R6525 CPU servers featuring 2.6-GHz AMD CPU cores. All three configurations were given 24 cores (spawning 23 workers) and 192GB of memory. For `CnC` and `cruxmiter` benchmarks, we used a wall-clock timeout of 12 hours. For `SC-hard` benchmarks, we used a wall-clock timeout of 1 hour. Additional details about the experimental setup are provided in the extended version of the paper (Wu, Barrett, and Narodytska 2025).

		CNC			CNC+TACO			PAINLESS
Benchmark	Res.	t_c	t_s	$cost$	$t_l + t_s$	$cost$	t_l	t_{total}
9dlx_vliw_at_b_iq8	UNS	409.4	19716	1.16e+09	**17171**	**4.18e+08**	1084	182
9dlx_vliw_at_b_iq9	UNS	625.9	24502	1.43e+09	**21438**	**5.27e+08**	1365	275
AProVE07-25	UNS	3.5	203	7.02e+07	**152**	**5.87e+07**	25	486
dated-5-19-u	UNS	74.7	261	2.05e+07	**198**	**1.04e+07**	57	522
eq.atree.braun.12	UNS	1.0	70	4.25e+07	**64**	**1.59e+07**	64	282
eq.atree.braun.13	UNS	1.1	243	1.56e+08	**142**	**6.20e+07**	54	1088
gss-24-s100	SAT	12.8	1557	1.99e+08	**1325**	**1.66e+08**	99	272
gss-26-s100	SAT	7.4	10939	2.23e+09	**8718**	**1.82e+09**	185	578
gus-md5-14	UNS	152.9	8746	1.89e+08	**6956**	**1.724e+8**	408	41486
ndhf_xits_09_UNS	UNS	16.0	518	1.10e+08	**511**	**9.89e+07**	52	820
rbcl_xits_09_UNK	UNS	6.4	291	1.10e+08	**283**	**1.00e+08**	29	684
rpoc_xits_09_UNS	UNS	7.3	**305**	1.04e+08	347	**9.83e+07**	36	273
total-10-17-u	UNS	168.4	592	3.06e+05	**251**	**2.92e+05**	169	268
total-5-15-u	UNS	104.8	4610	2.30e+08	**2345**	**1.75e+08**	60	2075
cruxmiter32seed0	UNS	0.2	112	1.35e+8	**101**	**1.30e+8**	9	329
cruxmiter32seed1	UNS	0.2	111	1.35e+8	**100**	**1.30e+8**	9	299
cruxmiter32seed2	UNS	0.2	165	1.77e+8	**161**	**1.75e+8**	11	190
cruxmiter32seed3	UNS	0.2	442	3.69e+8	**285**	**3.56e+8**	12	603
cruxmiter32seed4	UNS	0.2	703	5.05e+8	**511**	**4.53e+8**	12	1149
cruxmiter32seed5	UNS	0.2	159	2.10e+8	**142**	**1.91e+8**	12	360
cruxmiter32seed6	UNS	0.2	195	2.87e+8	**161**	**2.69e+8**	13	453
cruxmiter32seed7	UNS	0.2	258	3.23e+8	**212**	**2.98e+8**	12	138
cruxmiter32seed8	UNS	0.2	308	4.19e+8	**290**	**4.19e+8**	12	854
cruxmiter32seed9	UNS	0.2	524	5.12e+8	**320**	**4.73e+8**	11	2549

Table 3: Results on the `CnC` and `cruxmiter` benchmarks. t_c, t_l, and t_s denotes cubing, strategy learning, and solving time; the total runtime is $t_c + t_s$ for CNC, and $t_c + t_l + t_s$ for CNC+TACO. $cost$ denotes the total conflicts on cubes commonly solved by CNC and CNC+TACO. The asterisk (*) near a benchmark means re-partitioning happened; for other benchmarks, CNC and CNC+TACO solved the same set of cubes. The last column shows the runtime of PAINLESS.

The `CnC` and `cruxmiter` benchmarks. Tab. 3 compares the performance of CNC and CNC+TACO on the first two benchmark sets. We omitted one satisfiable `CnC` benchmark on which both CNC and CNC+TACO timed out. We report the time to create the initial cubes (t_c). For CNC, we report the time to solve the cubes (t_s); for CNC+TACO, we report the total time of the strategy learning phase and the solving phase ($t_l + t_s$). We also report the total number of conflicts on cubes that were commonly solved by CNC and CNC+TACO using different solving strategies ($cost$). We report PAINLESS's runtime in the last column.

Impressively, TACO was able to find a new solving strategy that reduces the total number of conflicts on all CNC and `cruxmiter` benchmarks. Moreover, the reduction in total cost consistently translates to a reduction in runtime. The reduction is over 15% in 12 out of the 24 benchmarks and in several cases exceeds 40% (e.g., `total-5-15-u`). Upon closer examination, on all but two instances (`rbcl_xits_09_UNK` and

			CNC		CNC+TACO			PAINLESS
Benchmark	Res.	t_c	t_s	$cost$	$t_l + t_s$	$cost$	t_l	t_{total}
W4.2.90.10.1SS	UNS	1.3	183	1.096e+8	**153**	**8.083e+7**	14	TO
W4.3.90.10.1DA	UNS	5.1	441	1.310e+8	**253**	**8.857e+7**	23	TO
W5.3.50.10.2DA	UNS	9.2	2058	5.279e+8	**1015**	**3.490e+8**	24	TO
grs_192_256	UNS	178.4	**1390**	9.760e+7	1963	**6.926e+7**	166	TO
multiplier.14.14	UNS	0.4	**165**	2.005e+7	183	**1.890e+7**	59	512
php15_mixed_15	UNS	0.3	TO	-	**1150**	-	11	2852
sin.depth.miter.1	UNS	1503.1	1117	2.396e+7	1127	**2.382e+7**	1127	TO

Table 4: Results on unsolved benchmarks from SC'22-24 that were solved by either CNC or CNC+TACO with 24 cores and a one-hour wall-clock timeout.

`rpoc_xits_09_UNS`), the new strategy found by tuning was validated and used to solve the remaining cubes.

Another interesting result is that CNC+TACO and PAINLESS exhibit complementary performance. CNC+TACO outperformed PAINLESS on 7 out of the 14 `CnC` benchmarks and dominated PAINLESS on all `cruxmiter` benchmarks. This suggests CNC is competitive to state-of-the-art solver PAINLESS on benchmark sets considered in CNC literature.

Discussion of tuned strategies. We found that there are several cases where TACO uncovers unexpectedly useful strategies. One interesting example comes from the two benchmarks with the prefix `eq.atree.braun`, where TACO learns to set `bump=0` and `tumble=0`, reducing the runtime on `eq.atree.braun.13` by more than 40%. Upon closer examination, this parameter setting results in a completely static branching order that splits on the first variables in the original encoding—essentially mimicking a BDD (Binary Decision Diagram) ordering. This represents a novel insight into solving this benchmark. We discuss other observations in the extended version of the paper (Wu, Barrett, and Narodytska 2025).

Unsolved benchmarks from SC'22-24. We now discuss results on `SC-hard`, a total of 73 benchmarks unsolved during recent SAT Competitions. Most of those benchmarks remain out-of-reach for CNC and CNC+TACO with the allocated computational resources. On 16 benchmarks, our prototype ran out of time or memory during the initial cubing phase. Nonetheless, CNC and CNC+TACO combined solved 7 out of the 57 remaining benchmarks. The results are shown in Tab. 4. We see that CNC+TACO consistently reduces the total number of conflicts and solves one benchmark which CNC alone is unable to tackle. Benchmark `grs_192_256` is an interesting case. While the total number of conflicts is significantly reduced by TACO, its runtime increases. Upon closer examination, we discovered that while CNC+TACO did solve most cubes faster, there is one cube that took the new strategy over 1000 seconds and the default strategy less than 300 seconds. We hypothesize that the solver got to this cube late in the solver process, leading to a slow-down in the solving time. In the future, it could be interesting to also learn cube-ordering on the fly so that hard cubes could be front-loaded for better load balancing.

In the extended version (Wu, Barrett, and Narodytska 2025), we have also conducted ablation studies to examine the effect of 1) reducing the number of tuning cubes, 2) removing validation, 3) using a larger strategy space, and 4) removing parameter probing for MCMC initialization.

4.2 Case Study: Neural Network Verification

We now shift our attention to a second automated reasoning task: solving neural network verification (NNV) queries. We used two of the four benchmark sets used for evaluating the latest version of MARABOU (Wu et al. 2024):

- `NAP`: 235 verification benchmarks concerned with checking whether certain neuron activation pattern implies certain classification (Gopinath et al. 2019); and
- `AltLoop`: 259 benchmarks concerned with checking the existence of infinite loops in a robot navigation system (Amir et al. 2023).

We omitted the two other benchmark sets, which contain queries that are easily solvable.

In addition to CNC and CNC+TACO, we also ran the parallel solving mode of MARABOU (Wu et al. 2020) (DNCMARABOU), which performs CNC-based parallel-solving with dynamic re-partitioning. Each job is assigned 8 cores (thus 7 workers are spawned), 64GB of memory, and a one-hour wall-clock timeout.

Evaluation on `NAP` and `AltLoop`. Tab. 5 summarizes the runtime performance of CNC and CNC+TACO. Since cubing with MARABOU took less than 1 second on those benchmarks, we did not split the runtime into cubing time and solving time. We report the total solving time (t_{total}), and for CNC+TACO, we also report the time spent on strategy learning (t_l), the number of benchmarks on which a new solving strategy has been found (v^*), and the percentage of overall runtime reduction on those benchmarks.

CNC+TACO exhibits interesting behaviors on both benchmark sets. On the `NAP` benchmarks, CNC+TACO solves significantly more benchmarks than CNC. Upon closer examination, this significant performance gain is *not* due to new strategies learned, but in fact due to satisfiable cubes discovered during the cube collection process. While default CNC attempts to solve each cube before moving on to the next cube, the COLLECT method creates a *depth-limited* (or more precisely, cost-limited) breadth-first search pattern, where the solver scrolls through and attempts at cubes with a low-cost budget. This turns out to be the preferable approach for finding satisfying assignments on this benchmark set. DNCMARABOU also solved 216 `NAP` benchmarks, though with a higher average solving time. Upon closer examination, CNC+TACO and DNCMARABOU together solved 230 (all `SAT`) out of the 235 benchmarks, while CNC does not contribute any unique solutions. To our knowledge, this is the first study to show that almost none of the activation patterns in the NAP benchmark set can actually guarantee a fixed prediction, despite their reported statistical correlation with specific outputs (Gopinath et al. 2019). As for the `AltLoop` benchmarks, both CNC and CNC+TACO solved all the instances, while CNC+TACO is overall more efficient.

Discussion of tuned strategies. We found that for the vast majority of the benchmarks, TACO did not find a different

	CNC		CNC+TACO					DNCMARABOU	
Family (#)	Slv.	t_{total}	Slv.	t_{total}	t_l	v^*	% Red.	Slv.	t_{total}
NAP (235)	163	1206538	**216**	**33606**	8955	0	-	216	420312
AltLoop (259)	259	93764	259	**66532**	2489	53	44%	258	108459

Table 5: Effect of TACO on NNV benchmarks.

solving strategy, suggesting that the default solving strategy is already reasonable. However, there are 53 benchmarks where a new solving strategy is found. On those benchmarks, CNC+TACO reduces the total runtime by 44% (from 46761 seconds to 26161 seconds). Examining the new strategies found by TACO in those 53 cases, we discovered that they always involves performing less input splitting (and more case splits on internal neurons instead). Interestingly, this deviates from the conventional wisdom (Wu et al. 2020; Bunel et al. 2020; Jia and Rinard 2021) that input splitting is preferable for neural networks with low input dimensions (AltLoop benchmarks have input dimension $\leq$ 20). In fact, 45 out of the 53 new strategies completely turn off input splitting. This suggests that even on benchmarks from the "same distribution," there might not be a uniformly optimal solving strategy.

5 Related Work

Tuning of automated reasoning tools. Our work is motivated by the success of offline meta-algorithmic design approaches such as *automated configuration* (Hutter et al. 2007; Hutter, Hoos, and Stützle 2007; KhudaBukhsh et al. 2016; Hutter et al. 2014; Ansótegui et al. 2015) and *per-instance solver selection* (Xu et al. 2008; Xu, Hoos, and Leyton-Brown 2010; Singh et al. 2009; Scott et al. 2021) in the automated reasoning domain. Unlike both approaches, TACO customizes the solving strategy for an automated reasoning tool from a pool of candidate strategies without any offline work. In addition, compared to offline per-instance algorithmic selection, TACO does not require feature extraction (i.e., representing a formula as a set of features), which is in itself a complex problem.

Online learning. The general idea of online learning is not new in automated reasoning. For example, learning-based branching and restart heuristics have been devised in MAPLESAT (Liang et al. 2016, 2018). More broadly, algorithms with adaptive components are common in automated reasoning (Moskewicz et al. 2001; Li, Wei, and Zhang 2007; Biere 2008; Katz et al. 2017; Cherif, Habet, and Terrioux 2021) and are adopted in modern tools including KISSAT and MARABOU. The concurrent work by Shi et al. is along this direction, where a selected, typically binary, solver parameter is adjusted during the solving process. In contrast, TACO focuses on optimizing over a (potentially large) collection of solving strategies, instead of designing adaptive low-level heuristics. Our experiments demonstrate that TACO can result in performance gain on top of the dynamic low-level heuristics in the solver. There has also been work on choosing solving strategies online when solving a sequence of related problems (Pimpalkhare et al. 2021; Wu et al. 2023). In contrast, our approach operates on individual formulas.

Looking beyond automated reasoning, online tuning methods such as DAC (Adriaensen et al. 2022) have shown success in other AI subfields. DAC uses reinforcement learning to dynamically adjust hyperparameters of an algorithm during execution. Adapting DAC-like techniques to state-of-the-art automated reasoning tools is highly non-trivial: modern solvers employ a complex scheduling of pre-processing and in-processing techniques that have to be aligned with the learnt policy, which is a challenging research topic on its own; in addition, one needs to devise solver-specific reward functions for the strategy adjustment, which is known to be hard as a candidate strategy performance cannot be easily approximated ahead of time (Adriaensen et al. 2022).

Partitioning strategies. Our work leverages partitioning strategies devised in cube-and-conquer-based automated reasoning. Cube-and-conquer was originally designed for tackling challenging SAT problems (Heule et al. 2011; Heule, Kullmann, and Marek 2016), by partitioning a problem into easier and balanced sub-problems using look-ahead techniques (Heule and van Maaren 2009). Various other partitioning strategies for SAT and SMT have been devised (Nejati, Le Frioux, and Ganesh 2020; Nejati et al. 2017; Heule et al. 2011; Wilson et al. 2023; Hyvärinen, Marescotti, and Sharygina 2015; Wu et al. 2020; Zhao, Cai, and Qian 2024; Hyvärinen, Marescotti, and Sharygina 2021).

6 Conclusion and Next Steps

We explored the feasibility of customizing the solving strategy for a given problem by learning from the problem itself. We introduced TACO, an online learning methodology that extends a traditional cube-and-conquer solving procedure. Experimental evaluation in two automated reasoning applications showed that TACO can consistently boost the performance of CNC, uncover effective instance-specific solving strategy, and in many cases outperform state-of-the-art solvers in the respective domains.

Our long-term vision is that *intelligent agents should both efficiently conduct low-level reasoning and adaptively shift their high-level solving strategy*—this work is one step towards that direction. We believe the application of TACO to more automated reasoning tools (e.g., general-purpose SMT solvers) is a promising next step and might reveal that new design choices are required. Progress has already been made progress on this front (Wilson et al. 2025). It would also be interesting to study the applicability of TACO in the cloud setting, where the efficient sharing of training data and strategies becomes less straightforward. Finally, we believe deep integration of TACO-like approaches into the search procedure of modern solvers would be a significant next challenge to tackle.

Acknowledgments

This work was funded in part by the National Science Foundation (No. 2211505) and was performed using high-performance computing equipment obtained under National Science Foundation Grant (No. 2117377). We thank Armin Biere, Gagandeep Singh, and the anonymous reviewers for their constructive feedback.

References

Adriaensen, S.; Biedenkapp, A.; Shala, G.; Awad, N.; Eimer, T.; Lindauer, M.; and Hutter, F. 2022. Automated dynamic algorithm configuration. *Journal of Artificial Intelligence Research*, 75: 1633–1699.

Amir, G.; Corsi, D.; Yerushalmi, R.; Marzari, L.; Harel, D.; Farinelli, A.; and Katz, G. 2023. Verifying learning-based robotic navigation systems. In *International Conference on Tools and Algorithms for the Construction and Analysis of Systems*, 607–627. Springer.

Ansótegui, C.; Malitsky, Y.; Samulowitz, H.; Sellmann, M.; Tierney, K.; et al. 2015. Model-Based Genetic Algorithms for Algorithm Configuration. In *IJCAI*, 733–739.

Beskyd, F.; and Surynek, P. 2022. Domain dependent parameter setting in sat solver using machine learning techniques. In *International Conference on Agents and Artificial Intelligence*, 169–200. Springer.

Biere, A. 2008. Adaptive restart strategies for conflict driven SAT solvers. In *Theory and Applications of Satisfiability Testing–SAT 2008: 11th International Conference, SAT 2008, Guangzhou, China, May 12-15, 2008. Proceedings 11*, 28–33. Springer.

Biere, A.; Faller, T.; Fazekas, K.; Fleury, M.; Froleyks, N.; and Pollitt, F. 2024. CaDiCaL, Gimsatul, IsaSAT and Kissat Entering the SAT Competition 2024. In Heule, M.; Iser, M.; Järvisalo, M.; and Suda, M., eds., *Proc. of SAT Competition 2024 – Solver, Benchmark and Proof Checker Descriptions*, volume B-2024-1 of *Department of Computer Science Report Series B*, 8–10. University of Helsinki.

Bunel, R.; Lu, J.; Turkaslan, I.; Torr, P. H.; Kohli, P.; and Kumar, M. P. 2020. Branch and bound for piecewise linear neural network verification. *Journal of Machine Learning Research*, 21(42): 1–39.

Cherif, M. S.; Habet, D.; and Terrioux, C. 2021. Combining vsids and chb using restarts in sat. In *27th International Conference on Principles and Practice of Constraint Programming*.

Chib, S.; and Greenberg, E. 1995. Understanding the metropolis-hastings algorithm. *The american statistician*, 49(4): 327–335.

Gopinath, D.; Converse, H.; Pasareanu, C.; and Taly, A. 2019. Property inference for deep neural networks. In *2019 34th IEEE/ACM International Conference on Automated Software Engineering (ASE)*, 797–809. IEEE.

Heisinger, M.; Fleury, M.; and Biere, A. 2020. Distributed cube and conquer with paracooba. In *Theory and Applications of Satisfiability Testing–SAT 2020: 23rd International Conference, Alghero, Italy, July 3–10, 2020, Proceedings 23*, 114–122. Springer.

Heule, M. J.; Kullmann, O.; and Marek, V. W. 2016. Solving and verifying the boolean pythagorean triples problem via cube-and-conquer. In *International Conference on Theory and Applications of Satisfiability Testing*, 228–245. Springer.

Heule, M. J.; Kullmann, O.; Wieringa, S.; and Biere, A. 2011. Cube and conquer: Guiding CDCL SAT solvers by lookaheads. In *Haifa Verification Conference*, 50–65. Springer.

Heule, M. J.; and van Maaren, H. 2009. Look-ahead based SAT solvers. In *Handbook of satisfiability*, 155–184. IOS Press.

Hoos, H. H.; Hutter, F.; and Leyton-Brown, K. 2021. Automated configuration and selection of SAT solvers. In *Handbook of Satisfiability*, 481–507. IOS Press.

Hutter, F.; Babic, D.; Hoos, H. H.; and Hu, A. J. 2007. Boosting verification by automatic tuning of decision procedures. In *Formal Methods in Computer Aided Design (FMCAD'07)*, 27–34. IEEE.

Hutter, F.; Hoos, H. H.; and Stützle, T. 2007. Automatic algorithm configuration based on local search. In *Aaai*, volume 7, 1152–1157.

Hutter, F.; Xu, L.; Hoos, H. H.; and Leyton-Brown, K. 2014. Algorithm runtime prediction: Methods & evaluation. *Artificial Intelligence*, 206: 79–111.

Hyvärinen, A. E.; Marescotti, M.; and Sharygina, N. 2015. Search-space partitioning for parallelizing SMT solvers. In *Theory and Applications of Satisfiability Testing–SAT 2015: 18th International Conference, Austin, TX, USA, September 24-27, 2015, Proceedings 18*, 369–386. Springer.

Hyvärinen, A. E.; Marescotti, M.; and Sharygina, N. 2021. Lookahead in partitioning SMT. In *2021 Formal Methods in Computer Aided Design (FMCAD)*, 271–279. IEEE.

Jia, K.; and Rinard, M. 2021. Verifying low-dimensional input neural networks via input quantization. In *Static Analysis: 28th International Symposium, SAS 2021, Chicago, IL, USA, October 17–19, 2021, Proceedings 28*, 206–214. Springer.

Kass, R. E.; Carlin, B. P.; Gelman, A.; and Neal, R. M. 1998. Markov chain Monte Carlo in practice: a roundtable discussion. *The American Statistician*, 52(2): 93–100.

Katz, G.; Barrett, C.; Dill, D. L.; Julian, K.; and Kochenderfer, M. J. 2017. Reluplex: An efficient SMT solver for verifying deep neural networks. In *Computer Aided Verification: 29th International Conference, CAV 2017, Heidelberg, Germany, July 24-28, 2017, Proceedings, Part I 30*, 97–117. Springer.

Katz, G.; Huang, D. A.; Ibeling, D.; Julian, K.; Lazarus, C.; Lim, R.; Shah, P.; Thakoor, S.; Wu, H.; Zeljić, A.; et al. 2019. The marabou framework for verification and analysis of deep neural networks. In *Computer Aided Verification: 31st International Conference, CAV 2019, New York City, NY, USA, July 15-18, 2019, Proceedings, Part I 31*, 443–452. Springer.

KhudaBukhsh, A. R.; Xu, L.; Hoos, H. H.; and Leyton-Brown, K. 2016. SATenstein: Automatically building local search SAT solvers from components. *Artificial Intelligence*, 232: 20–42.

Li, C. M.; Wei, W.; and Zhang, H. 2007. Combining adaptive noise and look-ahead in local search for SAT. In *Theory and Applications of Satisfiability Testing–SAT 2007: 10th International Conference, Lisbon, Portugal, May 28-31, 2007. Proceedings 10*, 121–133. Springer.

Liang, J. H.; Ganesh, V.; Poupart, P.; and Czarnecki, K. 2016. Learning Rate Based Branching Heuristic for SAT Solvers. In Creignou, N.; and Berre, D. L., eds., *Theory and Applications of Satisfiability Testing - SAT 2016 - 19th International Conference, Bordeaux, France, July 5-8, 2016, Proceedings*, volume 9710 of *Lecture Notes in Computer Science*, 123–140. Springer.

Liang, J. H.; Oh, C.; Mathew, M.; Thomas, C.; Li, C.; and Ganesh, V. 2018. Machine learning-based restart policy for CDCL SAT solvers. In *Theory and Applications of Satisfiability Testing–SAT 2018: 21st International Conference, SAT 2018, Held as Part of the Federated Logic Conference, FloC 2018, Oxford, UK, July 9–12, 2018, Proceedings 21*, 94–110. Springer.

Moskewicz, M. W.; Madigan, C. F.; Zhao, Y.; Zhang, L.; and Malik, S. 2001. Chaff: Engineering an efficient SAT solver. In *Proceedings of the 38th annual Design Automation Conference*, 530–535.

Nejati, S.; Le Frioux, L.; and Ganesh, V. 2020. A machine learning based splitting heuristic for divide-and-conquer solvers. In *Principles and Practice of Constraint Programming: 26th International Conference, CP 2020, Louvain-la-Neuve, Belgium, September 7–11, 2020, Proceedings 26*, 899–916. Springer.

Nejati, S.; Newsham, Z.; Scott, J.; Liang, J. H.; Gebotys, C.; Poupart, P.; and Ganesh, V. 2017. A propagation rate based splitting heuristic for divide-and-conquer solvers. In *Theory and Applications of Satisfiability Testing–SAT 2017: 20th International Conference, Melbourne, VIC, Australia, August 28–September 1, 2017, Proceedings 20*, 251–260. Springer.

Ozdemir, A.; Wu, H.; and Barrett, C. 2021. Sat solving in the serverless cloud. In *2021 Formal Methods in Computer Aided Design (FMCAD)*, 241–245. IEEE.

Pimpalkhare, N.; Mora, F.; Polgreen, E.; and Seshia, S. A. 2021. MedleySolver: online SMT algorithm selection. In *Theory and Applications of Satisfiability Testing–SAT 2021: 24th International Conference, Barcelona, Spain, July 5-9, 2021, Proceedings 24*, 453–470. Springer.

Ritirc, D.; Biere, A.; and Kauers, M. 2017. Column-wise verification of multipliers using computer algebra. In *2017 Formal Methods in Computer Aided Design (FMCAD)*, 23–30. IEEE.

Roy, V. 2020. Convergence diagnostics for markov chain monte carlo. *Annual Review of Statistics and Its Application*, 7(1): 387–412.

Saoudi, M.; Lejemble, T.; Baarir, S.; and Sopena, J. 2024. PL-PRS-BVA-KISSAT in SAT Competition 2024.

Scott, J.; Niemetz, A.; Preiner, M.; Nejati, S.; and Ganesh, V. 2021. MachSMT: A machine learning-based algorithm selector for SMT solvers. In *International Conference on Tools and Algorithms for the Construction and Analysis of Systems*, 303–325. Springer.

Shi, Z.; Jiang, W.; Zhang, X.; Luo, J.; Liang, Y.; Chu, Z.; and Xu, Q. 2025. Dynamicsat: Dynamic configuration tuning for sat solving. In *31st International Conference on Principles and Practice of Constraint Programming (CP 2025)*, 34–1. Schloss Dagstuhl–Leibniz-Zentrum für Informatik.

Singh, R.; Near, J. P.; Ganesh, V.; and Rinard, M. 2009. AvatarSAT: An auto-tuning boolean SAT solver. Technical report, Technical Report MIT-CSAIL-TR-2009-039. Massachusetts Institute of Technology.

Wilson, A.; Narodytska, N.; Barrett, C.; and Wu, H. 2025. Per-Instance Subproblem Generation for Strategy Selection in SMT. In *Formal Methods in Computer Aided Design (FMCAD)*, 94–103. TU Wien Academic Press.

Wilson, A.; Noetzli, A.; Reynolds, A.; Cook, B.; Tinelli, C.; and Barrett, C. W. 2023. Partitioning Strategies for Distributed SMT Solving. In *FMCAD*, 199–208.

Wu, H.; Barrett, C.; and Narodytska, N. 2025. Cubing for Tuning. *arXiv preprint arXiv:2504.19039*.

Wu, H.; Hahn, C.; Lonsing, F.; Mann, M.; Ramanujan, R.; and Barrett, C. 2023. Lightweight Online Learning for Sets of Related Problems in Automated Reasoning. In *2023 Formal Methods in Computer-Aided Design (FMCAD)*, 1–11. IEEE.

Wu, H.; Isac, O.; Zeljić, A.; Tagomori, T.; Daggitt, M.; Kokke, W.; Refaeli, I.; Amir, G.; Julian, K.; Bassan, S.; et al. 2024. Marabou 2.0: a versatile formal analyzer of neural networks. In *International Conference on Computer Aided Verification*, 249–264. Springer.

Wu, H.; Ozdemir, A.; Zeljic, A.; Julian, K.; Irfan, A.; Gopinath, D.; Fouladi, S.; Katz, G.; Pasareanu, C.; and Barrett, C. 2020. Parallelization techniques for verifying neural networks. In *2020 Formal Methods in Computer Aided Design (FMCAD)*, volume 1, 128–137. TU Wien Academic Press.

Xu, L.; Hoos, H.; and Leyton-Brown, K. 2010. Hydra: Automatically configuring algorithms for portfolio-based selection. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 24, 210–216.

Xu, L.; Hutter, F.; Hoos, H. H.; and Leyton-Brown, K. 2008. SATzilla: portfolio-based algorithm selection for SAT. *Journal of artificial intelligence research*, 32: 565–606.

Zhao, M.; Cai, S.; and Qian, Y. 2024. Distributed SMT Solving Based on Dynamic Variable-Level Partitioning. In *International Conference on Computer Aided Verification*, 68–88. Springer.