AlphaMapleSAT: An MCTS-based Cube-and-Conquer SAT Solver for Hard Combinatorial Problems

Piyush Jha[1] , **Zhengyu Li**[1] , **Zhengyang Lu**[2] , **Curtis Bright**[3] and **Vijay Ganesh**[1]

[1]Georgia Institute of Technology, USA
[2]University of Waterloo, Canada
[3]University of Windsor, Canada

Abstract

This paper introduces ALPHAMAPLESAT, a novel Monte Carlo Tree Search (MCTS) based *Cube-and-Conquer* (CnC) SAT solving method aimed at efficiently solving challenging combinatorial problems. Despite the tremendous success of CnC solvers in solving a variety of hard combinatorial problems, the *lookahead* cubing techniques at the heart of CnC have not evolved much for many years. Part of the reason is the sheer difficulty of coming up with new cubing techniques that are both low-cost and effective in partitioning input formulas into sub-formulas, such that the overall runtime is minimized.

Lookahead cubing techniques used by current state-of-the-art CnC solvers, such as March, keep their cubing costs low by constraining the search for the optimal splitting variables. By contrast, our key innovation is a deductively-driven MCTS-based lookahead cubing technique, that performs a deeper heuristic search to find effective cubes, while keeping the cubing cost low. We perform an extensive comparison of ALPHAMAPLESAT against the March CnC solver on challenging combinatorial problems such as the minimum Kochen–Specker and Ramsey problems. We also perform ablation studies to verify the efficacy of the MCTS heuristic search for the cubing problem. Results show up to 2.3× speedup in parallel (and up to 27× in sequential) elapsed real time.

1 Introduction

In recent years, we have witnessed many hard combinatorial problems such as the Boolean Pythagorean Triples [Heule *et al.*, 2016], Schur number five [Heule, 2018], and Lam's problem [Bright *et al.*, 2021] being solved by SAT-based techniques. Among these, the Cube-and-Conquer (CnC) SAT solving approach has emerged as the most dominant strategy [Heule *et al.*, 2011]. This method efficiently utilizes two distinct solvers: a lookahead "cubing or splitting solver" responsible for partitioning the input Boolean formula into sub-formulas using cubes (a conjunction of literals) and a subsequent "conquering or worker solvers" (e.g., a Conflict-driven Clause Learning SAT solver [Silva and Sakallah, 1996]) tasked with solving each sub-formula. CnC techniques [Biere *et al.*, 2021] have demonstrated superior performance compared to sequential, portfolio, or traditional divide-and-conquer solvers [Nejati, 2020] when dealing with hard combinatorial instances obtained from diverse applications in geometry [Bright *et al.*, 2021], physics [Li *et al.*, 2023], or cryptography [Zaikin, 2022].

The success of CnC solvers depends crucially on the order in which variables are split. Consequently, lookahead cubing solvers use empirically tested metrics (e.g., propagation rate) to rank variables in an input formula and choose the variable that ranks the best according to such metrics to then split the input formula into sub-formulas, with the goal that they can subsequently be handled relatively easily by conquering solvers. Note that a cubing solver may produce many cubes (each containing several literals) given an input formula. It does this typically by splitting on multiple variables and propagating them in a single run, resulting in these cubes.

Despite the dramatic success of CnC solvers, significant challenges remain. For example, the cubes created by traditional CnC solvers, such as March [Heule *et al.*, 2011], are far from optimal, where optimality for cubes can be defined in terms of minimizing the total elapsed real time of the solver.[1] Finding optimal cubes seems to require a deep lookahead search, which can get prohibitively expensive as the size of the input formula increases.

Problem Statement. Design and implement a (parallel) lookahead cubing solver that takes as input a Boolean formula F in conjunctive normal form (CNF), outputs a set $C = \{c_1, c_2, \ldots, c_k\}$ of cubes, where the sub-formulas $F \wedge c_i$ are solved by (parallel) conquering solver(s) s.t. the total CPU and elapsed real time for cubing and solving F is minimized.

ALPHAMAPLESAT: An MCTS-based CnC Solver. To address the above-stated problem, we present ALPHAMAPLE-

[1]We define total elapsed real time as the time taken by a (sequential or parallel) CnC solver to solve the input formula and produce a SAT/UNSAT result. This is probably the most relevant metric in the context of measuring the efficacy of a CnC solver, since users care the most about the latency of a system, the time elapsed from when a job is given to the time a result is obtained. While the definition of this term is simple enough in the sequential context, it can be complicated in a parallel setting depending on the kind of parallelization used. See Section 5.5 for a more formal definition.

SAT, a novel cubing solver based on the integration of Monte Carlo Tree Search (MCTS) [Coulom, 2006] with deductive rewards, resulting in a heuristic search technique that is both cost-effective and enables an informed exploration of the cubes. In recent years, the popularity of MCTS has grown significantly, especially because of its success in the context of hard combinatorial search for two-player games such as Go, Chess, and Shogi [Silver *et al.*, 2017]. This has prompted numerous endeavors to apply MCTS to tackle combinatorial optimization problems by transforming these problems into single-player games.

AlphaMapleSAT leverages the power of MCTS to peek ahead in the search space of splitting trees associated with the input formula, prioritizing splits that, while not necessarily offering the immediate highest reward, hold the potential to unlock better ones at subsequent splitting depths. AlphaMapleSAT makes use of a deductive reward signal using an automated reasoning tool, a departure from traditional reward signals/functions which are typically provided by human domain experts in the context of MCTS or reinforcement learning. Deductive rewards, derived from automated reasoning tools (e.g., solvers), are very general and can be used to construct richer reward functions by augmenting human understanding of a problem domain. This informed exploration, guided by deductive reward signals, steers the search process towards promising partitions, significantly reducing the total elapsed time spent on both cubing and solving.

The use of deduction-based metrics in the context of lookahead solvers is not new. In fact, March uses CDCL SAT solvers to compute propagation rate and other deductively-derived metrics to rank variables to split on. What is new here is the combination of deductive reward with MCTS during selection and rollout, resulting in a powerful splitting heuristic.

1.1 Contributions.

- We introduce AlphaMapleSAT, a novel CnC solver that integrates Monte Carlo Tree Search (MCTS) to overcome the limitations of both greedy (limited search used in current CnC solvers such as March, resulting in sub-optimal cubes) and exhaustive search approaches for cubing (search cost overwhelms any benefits derived from optimal cubes). MCTS strikes a balance between immediate rewards and the potential for future rewards, leading to significantly faster cubing and solving times while keeping the cost of searching for good cubes low.
- AlphaMapleSAT departs from traditional reward signals/functions in the context of MCTS, in that it uses an automated reasoning tool (a solver) to compute a deductive feedback reward signal (propagation rate). This novel approach is applied during the selection and rollout phases of MCTS, eliminating the use of neural networks commonly employed in AlphaGo-based learning frameworks. The use of automated reasoning, when applicable, can effectively complement reward functions based on human insight alone.
- We demonstrate the effectiveness of AlphaMapleSAT through extensive comparisons with the state-of-the-art March CnC solver on benchmark problems like the minimum Kochen–Specker and the Ramsey problem. We also perform ablation studies to verify the efficacy of the MCTS heuristic search for the cubing problem. Our results showcase up to 2.3× speedup in parallel (and up to 27× in sequential) CnC elapsed real time, highlighting AlphaMapleSAT's potential as a significant improvement over existing CnC SAT solvers.

2 Related Work

The utilization of Monte Carlo Tree Search (MCTS) gained momentum following its triumph in two-player games [Silver *et al.*, 2017]. This success prompted researchers to adapt MCTS for single-player games, leading to innovative approaches for problem-solving. MCTS has also been employed in the context of SAT solving by introducing a UCT for choosing branching variables in the DPLL algorithm [Previti *et al.*, 2011]. Subsequent studies extended and improved upon this approach by introducing reward functions based on learned clauses and activity scores [Schloeter, 2017; Keszocze *et al.*, 2020].

Most recently, Monte Carlo Forest Search [Cameron *et al.*, 2022] combines MCTS with neural networks to identify candidate search trees for DPLL branching policies through an offline learning approach to avoid the computational cost of the reward function. Moreover, MCTS has also found applications in finding backdoors to Mixed Integer Linear Programming (MILP) problems [Khalil *et al.*, 2022].

CombOpt Zero uses Graph neural networks within the AlphaZero framework to solve NP-hard problems [Abe *et al.*, 2019]. Graph neural networks-based MCTS have also been explored for solving the Quantified Boolean Formula Satisfaction (QSAT) problem [Xu and Lieberherr, 2022]. The combination of MCTS and neural networks has also been proposed to guide the search in Satisfiability Modulo Theories (SMT) solvers [Graham-Lengrand and Färber, 2018]. For additional details, we refer our readers to a recent review paper [Świechowski *et al.*, 2023], which provides a detailed overview of MCTS and its applications.

While MCTS is a heuristic search algorithm with roots in AI planning, in recent years, it has been closely aligned with reinforcement learning (RL) [Silver *et al.*, 2017]. The use of RL in the context of solvers is not new. Among the first solvers to effectively use RL is MapleSAT [Liang *et al.*, 2016], where the branching heuristic is modeled as a Multi-Armed Bandit (MAB) problem, and a deductive reward signal (conflict clauses) is used to guide the agent (the branching heuristic).

In our work, we introduce a novel application of MCTS for Cube-and-Conquer (CnC), employing a deductive reward signal in an online learning fashion during the selection and rollout phases of MCTS without the need for a neural network. This represents the first instance of utilizing MCTS (with deductive reward signal) in the context of CnC.

3 Background

3.1 MCTS

Monte Carlo Tree Search (MCTS) is an algorithm designed for navigating complex combinatorial spaces, particularly in

the context of search trees [Coulom, 2006]. It has proven effective in the context of reinforcement learning-based systems like AlphaGo [Silver *et al.*, 2016] and AlphaZero [Silver *et al.*, 2017] for games such as Go, Chess, and Shogi. In such two-player games, MCTS aims to identify the optimal action at each step of the game to maximize the probability of winning. It builds an n-ary tree where the root node represents the current game state, edges represent valid actions, and child nodes extend the parent's state by playing the corresponding action. Terminal states are associated with scalar reward values.

In more detail, MCTS iteratively runs a 4-step simulation: **(1) Selection:** Starting from the root of the tree, a tree-search policy traverses the tree until a leaf node is selected. Upper Confidence bound for Trees (UCT) [Kocsis and Szepesvári, 2006] algorithm is used to maintain an exploitation and exploration trade-off by balancing average rewards and visit counts. **(2) Expansion:** The tree is expanded from the leaf node by adding child node(s) based on the selected unexplored actions. **(3) Rollout:** If the selected leaf node is non-terminal, the simulation continues by subsequently choosing actions according to a rollout policy until it reaches a terminal state. **(4) Backup:** Rewards obtained at the terminal state from the last step are propagated back to the root node of the search tree.

This four-step loop is repeated until a termination condition (e.g., iteration budget), and the action corresponding to the highest reward or most visited child of the root node is played. The opponent responds, and the process repeats with an updated search tree representing the current game state. For additional details, we refer our readers to survey papers on MCTS [Browne *et al.*, 2012; Świechowski *et al.*, 2023].

3.2 Cube-and-Conquer (CnC) Solvers

The Cube-and-Conquer (CnC) SAT solving approach was introduced as a means to address challenging combinatorial problems [Heule *et al.*, 2011]. CnC solvers employ a two-step process: a *lookahead cubing solver* to partition the input SAT instance into distinct sub-problems, and a *conquering or worker solver* (e.g., a CDCL SAT solver) to solve each sub-problem with the goal of minimizing the total elapsed time required to partition and solve the input SAT instance.

Definition 1 (**Cube).** *A cube is a conjunction of literals, e.g., $x_1 \wedge \cdots \wedge x_n$, where x_i represents a literal in a given Boolean formula.*

Definition 2 (**Propagation rate).** *Given a conjunction of a formula F and a cube c, the propagation rate of $F \wedge c$ is defined as the ratio between the number of propagations derived by a Boolean Constraint Propagation (BCP) [Davis* et al.*, 1962] method on the input $F \wedge c$, and the size of c. This metric, widely recognized in the field of SAT solvers, serves as a fundamental gauge of solver performance.*

Definition 3 (**Lookahead heuristics).** *Lookahead heuristics iterate over all variables (or cubes) of a given formula, simplify the formula with respect to the given cubes, and probe them to decide the best variable to split on. Probing is the process of computing quality metrics (e.g., propagation rate) by running a SAT solver on the sub-formulas thus obtained.*

Definition 4 (**Cubing problem).** *Given a Boolean formula F in conjunctive normal form (CNF), the cubing solver outputs a set $C = \{c_1, c_2, \ldots, c_k\}$ of cubes. The resulting sub-formulas $F \wedge c_i$ are defined as the partitions or sub-formulas of the original formula F.*

Definition 5 (**Splitting or cubing tree).** *The splitting tree of a formula is a full binary tree where each node in the tree is a sub-formula, and the edges are marked by the values (True and False) assigned to the splitting variable. A path from the root of the splitting tree to the respective leaves represents a cube.*

Definition 6 (**Cubing solver).** *A cubing solver takes as an input a Boolean formula F in conjunctive normal form (CNF), explores the space of different cubing trees, and outputs the most rewarding cubing tree as output.*

The value of lookahead heuristics is that they provide a *global view* of the search space of an input formula, compared to CDCL solvers that analyze the formula in a very *local* fashion [Nejati, 2020], albeit at a higher cost in terms of searching for splitting variables. Empirical evidence has demonstrated the effectiveness of this CnC technique in solving hard SAT problems that have not been solved by any other known technique. We refer the readers to the Handbook of Satisfiability [Biere *et al.*, 2021] for an overview of CnC solvers.

4 The Design and Implementation of AlphaMapleSAT

In this section, we describe the design and implementation of the cubing solver in AlphaMapleSAT [2]. The challenge posed by the cubing problem involves navigating a vast search space of splitting trees of large depth, resembling the complexities found in strategic games like chess or Go. Exhaustive exploration of this extensive space is infeasible. Drawing inspiration from AlphaGo [Silver *et al.*, 2016], we address this issue by heuristically truncating the splitting trees, by approximating the rest of the tree via a *value function*. While AlphaGo utilizes a neural network during the selection and rollout steps of the MCTS, our strategy employs a solver, functioning as a deductive reasoning tool to calculate the propagation rate. While propagation rate has proven effective in our setting, we could easily extend this to other metrics that solvers compute. As the simulation progresses and the splitting trees get deeper during the cubing phase of the CnC solver, thus enabling the cubing solver to compute more accurate statistics, the MCTS policy is guided towards searching over splitting trees that contain *high reward* cubes.

4.1 Input and Output

The input to AlphaMapleSAT is a CNF Formula F. The output of the cubing solver is a set of k cubes which, respectively, in conjunction with F, produce sub-formulas $\{F_1, F_2, \ldots, F_k\}$ to be given to k (parallel) worker solvers. The original instance is considered solved if at least one of the sub-formulas is satisfiable or all of them are unsatisfiable.

[2] https://anonymous.4open.science/r/AlphaMapleSAT-B07C/

Instances	Tools	Cubing and simplification elapsed real time (s)	Total CPU time (s)	Total elapsed real time (s)	Total elapsed real time speedup (wrt next best)
Ramsey (3,8)	March (`eval_var`)	9,840	1,202,918	102,083	-
	March (`eval_cls`)	6,780	**1,087,694**	109,014	-
	ALPHAMAPLESAT	**4,200**	1,169,307	**78,749**	**1.30**×
KS 19	March (`eval_var`)	731	13,534	3,237	-
	March (`eval_cls`)	601	14,432	2,248	-
	ALPHAMAPLESAT	**516**	**7,756**	**1,559**	**1.44**×
KS 20	March (`eval_var`)	1,962	81,421	12,177	-
	March (`eval_cls`)	1,708	86,291	12,423	-
	ALPHAMAPLESAT	**988**	**57,899**	**7,759**	**1.57**×
KS 21	March (`eval_var`)	26,077	1,056,773	60,123	-
	March (`eval_cls`)	28,472	1,316,841	61,153	-
	ALPHAMAPLESAT	**2,170**	**673,365**	**25,693**	**2.34**×
KS 22	March (`eval_var`)	518,400	6,595,091	586,733	-
	March (`eval_cls`)	562,620	6,525,814	631,920	-
	ALPHAMAPLESAT	**202,200**	**3,385,952**	**260,635**	**2.25**×

Table 1: **Parallel Cubing and Parallel Solving:** Comparison results between ALPHAMAPLESAT and the two heuristics by the March solver across various benchmarks using parallel cubing and simplification with parallel solving and verification.

4.2 Problem Setup

We model the cubing problem as a sequential decision-making problem. We view cubing as a single-player, deterministic game. The goal is to find a set of cubes that maximizes the overall propagation rate. Thus, the cubing problem is modeled as a deterministic Markov Decision Process (MDP). More specifically, the given problem can be modeled as a tree MDP [Scavuzzo *et al.*, 2022]. In the context of a tree MDP, taking an action leads to the environment transitioning into two or more new child states. Notably, each of these child states adheres to the Markov property, where each child is solely dependent on its current state. Tree MDP policies encapsulate algorithms that systematically break down problems into two or more simpler, history-independent subproblems. For a more in-depth understanding of tree policies, interested readers can refer to Monte Carlo Forest Search (MCFS) [Cameron *et al.*, 2022], which models the DPLL tree search algorithm as tree MDPs. In our setting, the action is the variable to split on, the child states being the sub-formulas (formula in conjunction with the true and false directions of the selected variable, respectively), and the reward being the propagation rate.

4.3 Reward Function

Given a conjunction of a formula F and a cube c, the reward function associated with c is the propagation rate of $F \wedge c$ (Definition 2). We use MiniSAT to compute this metric.

4.4 Termination Condition

The termination condition of a cube is a user-defined parameter n, which denotes that the splitting process at a particular node must stop if at least n variables have been eliminated (through propagation or as part of splitting variables) in the cube. This choice in defining the termination condition is motivated by the objective of attaining balanced cubes within the solving process [Heule *et al.*, 2011].

4.5 Cubing Episode

The cubing process begins with a CNF formula that we aim to cube on. Each cubing episode involves multiple steps. Starting with the original CNF formula as the root node, we construct a splitting tree: a binary tree where each node represents a (sub-)formula, and the edges from the node are the true and false directions of the splitting variable. At each node, we invoke the Monte Carlo Tree Search (MCTS) simulation to identify the best variable for splitting. Upon determining the splitting variable, we create two new nodes by conjuncting the selected variable's true and false directions into the original formula, respectively. We then assess if the termination criteria for the current set of cubes have been met. If not, we iteratively call the MCTS simulation on the newly formed nodes, continuing the splitting process until the termination criteria are satisfied. At the end of the cubing episode, we obtain the set of cubes which, in conjunction with the original CNF formula, produce sub-formulas to be given to worker solvers in parallel.

4.6 MCTS Simulation

The cubing episode invokes the MCTS simulation, generating an MCTS tree with the root node representing the formula passed by the cubing episode. A user-defined simulation budget dictates the number of simulations that are being performed, with each simulation consisting of four steps:

Instances	Tools	Cubing elapsed real time (s)	Total CPU time (s)	Total elapsed real time (s)	Total elapsed real time speedup
KS 19	March (`eval_cls`)	7,055	10,695	7,155	-
	ALPHAMAPLESAT	**206**	**4,183**	**328**	**21.81**×
KS 20	March (`eval_cls`)	237,262	299,375	237,836	-
	ALPHAMAPLESAT	**8,184**	**55,873**	**8,676**	**27.41**×
KS 21	March (`eval_cls`)	Timeout	Timeout	Timeout	-
	ALPHAMAPLESAT	**11,010**	Timeout	Timeout	-
KS 22	March (`eval_cls`)	Timeout	Timeout	Timeout	-
	ALPHAMAPLESAT	**29,761**	Timeout	Timeout	-
Ramsey (3,8)	March (`eval_cls`)	Timeout	Timeout	Timeout	-
	ALPHAMAPLESAT	**15,129**	Timeout	Timeout	-

Table 2: **Sequential Cubing and Parallel Solving:** Comparison results between ALPHAMAPLESAT and the default heuristic employed by the March solver across all benchmarks using sequential cubing and parallel solving.

- **Selection.** In the selection phase, we employ the version of the PUCT algorithm [Rosin, 2011] used by AlphaGo [Silver *et al.*, 2016] but without the use of neural networks to provide an initial estimate of the available next set of actions. Instead, we use a solver to provide us with this estimate and help choose promising actions during the search process. First, to identify valid actions for our case (i.e., variables that have not been selected or propagated yet), we utilize Boolean Constraint Propagation (BCP) [Davis *et al.*, 1962]. Now, for each of these valid variables, we determine the number of propagations in both the true and false directions using BCP. These propagations from both directions are combined into a single scalar value for each variable using the expression $\text{prop}(x_i) \cdot \text{prop}(\neg x_i) + \text{prop}(x_i) + \text{prop}(\neg x_i)$ (where $\text{prop}(x_i)$ is the number of propagations after setting x_i to true), following the approach employed by March [Heule *et al.*, 2011]. The resulting score is then normalized to serve as the initial estimate $P(s, a)$ of taking an action a (selecting a variable) from the state s (formula).

  Now, introducing the PUCT formula, the action selection is guided by the following equations [Silver *et al.*, 2016]:

$$a_{\text{chosen}} = \arg\max_a (Q(s,a) + u(s,a)) \tag{1}$$

$$u(s,a) = c_{\text{puct}} \cdot P(s,a) \cdot \frac{\sqrt{\sum_b N(s,b)}}{1 + N(s,a)} \tag{2}$$

  In this formula, we use standard notations as used by AlphaGo [Silver *et al.*, 2016]. $Q(s, a)$ is the expected reward for taking action a from state s, and $N(s, a)$ is the number of times action a was selected from state s. c_{puct} is a constant that influences the trade-off between exploration and exploitation during the decision-making process. This search control strategy initially favors actions with high prior probability and low visit count, gradually shifting towards actions with high action value as the exploration progresses. The improved policy is determined by the variable that led to an overall higher propagation rate obtained so far. We chose the c_{puct} to be 5 after grid search on easier Kochen–Specker (KS) instances.

- **Expansion.** When encountering a new non-terminal node, the expansion step ensures that unexplored regions of the search space are systematically investigated. By selecting the variable with the best score (calculated using BCP), we prioritize exploring promising branches. We expand one level deeper in the MCTS tree by creating new child nodes.

- **Rollout.** Conducting random rollouts is resource-intensive, involving numerous calls to BCP at each subsequent node in order to determine valid actions and check termination conditions. Furthermore, as the depth increases, the creation of multiple child nodes at every level results in an exponential increase in BCP calls. We employ an alternative approach to mitigate this computational burden. Instead of relying on rewards at terminal nodes, we leverage rewards (propagation rate) at the current leaf nodes. Unlike other scenarios (Chess, Go, etc.), we have access to rewards at the intermediate levels. By incorporating rewards at intermediate stages, we strike a balance between computational efficiency and decision-making accuracy. This adjustment optimizes the rollout strategy within the MCTS framework, ensuring a more scalable and effective exploration of the problem space.

- **Backup.** The backup step in MCTS involves combining the scores from all the leaf nodes (which include the newly created child nodes) and iteratively calculating it all the way back to the root node. We combine the score of the two nodes using $\text{eval}(x_i) \cdot \text{eval}(\neg x_i) + \text{eval}(x_i) + \text{eval}(\neg x_i)$, where $\text{eval}(x_i)$ is the propagation rate at node x_i. This computation is inspired by the scoring methodology commonly employed in lookahead solvers [Heule *et al.*, 2011].

After completing the MCTS simulations (bounded by the specified simulation budget, which was selected to be 30 after grid search on easier KS instances), we chose the variable that has achieved the highest reward value.

Instances	Tools	Total CPU time (s)
Ramsey (3,8)	AMS + MCTS	**1,169,307**
	AMS − MCTS	1,290,851
KS 19	AMS + MCTS	**7,756**
	AMS − MCTS	8,370
KS 20	AMS + MCTS	**57,899**
	AMS − MCTS	74,291
KS 21	AMS + MCTS	**673,365**
	AMS − MCTS	777,031
KS 22	AMS + MCTS	**3,385,952**
	AMS − MCTS	5,274,015

Table 3: **Ablation Study with MCTS switched ON/OFF in AlphaMapleSAT:** Results of the ablation study evaluating the contribution of Monte Carlo Tree Search (MCTS) in AlphaMapleSAT (AMS) for performance across various benchmarks. A comparison is made between AMS with MCTS exploration enabled (AMS + MCTS) and when the exploration mechanism is disabled (AMS − MCTS).

5 Experimental Setup

5.1 Competing Tool

Our comparative analysis involves benchmarking our tool against the well-established Cube-and-Conquer (CnC) solver March [Heule *et al.*, 2011], which stands as the state-of-the-art in the field. For a comprehensive evaluation, we incorporate two distinct heuristics (`eval_cls` and `eval_var`) provided by March. These heuristics rely on calculating the number of propagated variables, along with some additional metrics, to find the best splitting variables. Remarkably, even after several years, March remains the dominant CnC solver and has been employed in various combinatorial settings [Heule, 2018; Bright *et al.*, 2022; Semenov *et al.*, 2023; Li *et al.*, 2023]. As detailed in Section 3.2, the CnC methodology, particularly through the March CnC solver, has demonstrated efficacy in resolving hard combinatorial problems in diverse fields.

Moreover, in recent SAT competitions (2022 and 2023), Paracooba [Heisinger, 2022] and MergeSAT [Manthey, 2023], both leveraging lookahead techniques, have incorporated March as their lookahead cubing solver. This shows that March remains the cutting-edge CnC SAT solver.

5.2 Benchmarks

Minimum Kochen–Specker problem

The minimum Kochen–Specker (KS) problem is of significant importance in quantum mechanics and has captivated the attention of physicists and mathematicians for decades. At its core, the KS problem is intricately tied to the fundamental principles of quantum mechanics, specifically addressing the notion of contextuality.

We use the SAT encodings and instances created by previous work in this domain [Li *et al.*, 2023] that used Cube-and-Conquer (using March as their cubing solver) and a SAT+CAS technique [Zulkoski *et al.*, 2015; Bright *et al.*, 2022] to solve this problem. We replace March with our AlphaMapleSAT solver for the cubing process for an apples-to-apples comparison. Moreover, we focus on CNF instances of KS order 19 (with 3,876 variables and 233,219 clauses), order 20 (with 4,560 variables and 408,455 clauses), order 21 (with 5,320 variables and 923,933 clauses), and order 22 (with 6,160 variables and 2,496,012 clauses) because of their computation tractability. For more details about the KS problem, we refer our readers to the previous work [Li *et al.*, 2023].

Ramsey problem

The foundation of Ramsey Theory was laid by Frank P. Ramsey [Ramsey, 1987], with Ramsey numbers standing as renowned and challenging problems. Only nine non-trivial Ramsey numbers are currently known. In this benchmark, we specifically focus on the Ramsey problem of $R(3, 8)$. This problem seeks to determine the smallest value of n for which every red/blue coloring of the complete graph on n vertices must contain either a blue triangle or a red 8-clique. The computationally established value for $R(3, 8)$ is 28 [McKay and Min, 1992].

Similar to the approach in solving the minimum Kochen–Specker problem discussed earlier, we employ the SAT+CAS technique to address the $R(3, 8)$ problem. The encoding of the Ramsey instance is adopted from [Fujita *et al.*, 2013], resulting in a CNF instance with 15,820 variables and 3,163,013 clauses. For more extensive literature on the Ramsey problem, readers can refer to a previous work [Radziszowski, 2011].

5.3 Implementation and Computational Environment

Our implementation is carried out in Python 3.10. We leverage MiniSAT's *propagate* functionality integrated with the PySAT library [Ignatiev *et al.*, 2018] to calculate the propagation rate through unit propagation.

Our experiments were conducted on a high-performance CentOS V7 cluster equipped with Intel E5-2683 v4 Broadwell processors running at 2.10 GHz, accompanied by 12 GiB of memory. For the parallel cubing phase, a dedicated CPU node featuring 16 cores was employed. Additionally, for the parallel solving phase, the number of single-core CPU nodes matched the count of cubes being generated, optimizing computational efficiency.

5.4 Cubing, Solving, and Verification using MathCheck

Our experimental setup makes use of the MathCheck [Li *et al.*, 2023] pipeline. MathCheck allows any lookahead solver to be used in a parallel cubing setting, significantly improving the efficiency of tackling combinatorial CNF instances. Moreover, instead of pre-generating all cubes, the cubing solver iteratively operates on the CNF instance. Subsequently, the conquering solver (SAT+CAS) [Zulkoski *et al.*, 2015; Bright *et al.*, 2022] solves each subproblem in parallel. This strategic modification ensures efficient parallel solving with proofs under 7 GiB. Additionally, for verification purposes, MathCheck employs DRAT-trim [Wetzler *et al.*, 2014].

5.5 Metrics

The metrics used by us have been widely used in this field [Heule *et al.*, 2011; Li *et al.*, 2023], providing a thorough assessment of the different tools. Our evaluation criteria include the following metrics:

- **Total CPU time:** Total time spent across all CPUs to perform cubing, simplification, solving, and verification. It offers insights into the computational efficiency and resource utilization of the (parallel) CnC solvers we compared in our study.
- **Elapsed real time:** End-to-end wall clock time spent by a solver to perform cubing, simplification, solving, and verification to produce a result for a given benchmark. This metric measures the latency or time-to-completion of the (parallel) CnC solvers we compared. (Note that elapsed real time does not include the job scheduling times of the widely-used cluster we employed.)

6 Experimental Results and Ablation Study

In this Section, we present a detailed experimental comparison of AlphaMapleSAT against March on the KS and Ramsey problems. We also present an ablation study where we switch our techniques (esp. MCTS) on and off in AlphaMapleSAT in order to properly study their efficacy.

6.1 Efficacy of AlphaMapleSAT against March using Parallel Cubing and Parallel Solving

We conducted a comprehensive evaluation of our tool by comparing it against two heuristics provided by the March CnC solver, as detailed in Section 5.1. This comparison was carried out on five challenging combinatorial instances, as described in Section 5.2.

We use the parallel cubing and parallel solving pipeline provided by MathCheck [Li *et al.*, 2023], and integrated AlphaMapleSAT's cubing solver into it. We used the same command-line arguments for all three tools for an apples-to-apples comparison. We present the results in terms of elapsed cubing and simplification time, total CPU time, and overall elapsed time spent in cubing, simplification, solving, and verification (Table 1).

We find that AlphaMapleSAT outperforms both the March heuristics on all the benchmarks (Table 1). The results are more pronounced on KS order 21, where AlphaMapleSAT takes substantially less amount of time in cubing as compared to the best March heuristic (12× speedup on KS order 21), and still emerges to be the fastest in terms of total elapsed time (2.34× speedup on KS order 21).

AlphaMapleSAT's success can be attributed to its utilization of Monte Carlo Tree Search (MCTS), enabling strategic exploration of the search space. In contrast, the March solver selects literals that locally optimize reductions without considering potential long-term consequences. AlphaMapleSAT, with its MCTS-guided exploration, proves adept at efficiently identifying superior cubes, contributing to its overall enhanced performance.

6.2 Efficacy of AlphaMapleSAT against March using Sequential Cubing and Parallel Solving

It's important to highlight that neither March nor our tool, AlphaMapleSAT, inherently offers parallel cubing functionality. In our evaluation in Section 6.1, we discuss the comparison between the parallel CnC versions of MathCheck with AlphaMapleSAT against March. By contrast, below we discuss the comparison of with AlphaMapleSAT vs. March on the same aforementioned benchmarks, in their default sequential cubing and solving setting.

We compare our tool against the default heuristic by March (`eval_cls`) in the sequential cubing followed by the parallel solving (using SAT+CAS) setting. We used identical command-line arguments for both tools to ensure a fair comparison. The evaluation metrics include elapsed cubing time, total CPU time, and total elapsed time (Table 2). Note that the reported total CPU and elapsed real time only includes cubing and solving. Simplification during cubing and verification using DRAT-trim were not performed for either solvers. Moreover, we set a timeout of 5 days for the cubing and solving process.

The results show that AlphaMapleSAT outperforms March significantly, even on relatively smaller instances (21.8× on KS order 19 and 27.4× on KS order 20), as detailed in Table 2. This performance distinction underscores the efficiency of AlphaMapleSAT in both parallel and sequential cubing settings.

March cubing solver experienced time-outs after 5 days on the remaining instances, whereas AlphaMapleSAT successfully produced the cubes albeit experiencing time-outs during the solving process. Note that we were able to obtain the results for all the benchmarks in the parallel cubing and parallel solving setting (Section 6.1) due to the incremental cubing functionality of MathCheck (Section 5.4).

6.3 Ablation Studies

To assess the impact of MCTS on the performance improvement observed earlier, we conducted an ablation study in the parallel cubing and parallel solving setting using the same benchmarks used in the previous sections. The objective is to determine whether MCTS contributes to the enhanced results achieved by AlphaMapleSAT.

To isolate the effect of MCTS, we turned off the MCTS exploration in AlphaMapleSAT. This means the cubing solver selects the variable with the highest propagation rate in the current splitting level without exploring deeper into the splitting tree.

The results of this ablation study are presented in Table 3, revealing that AlphaMapleSAT without MCTS exploration performs comparatively worse on every benchmark. This observation underscores the pivotal role of MCTS in the overall effectiveness of AlphaMapleSAT, emphasizing its contributions to our tool's improved performance.

7 Conclusion and Future Work

We have presented AlphaMapleSAT, a novel Monte Carlo Tree Search (MCTS) based Cube-and-Conquer (CnC) SAT solving technique. Using an informed exploration of the

search space, our tool overcomes the limitations of both greedy (limited search resulting in sub-optimal cubes) and exhaustive approaches where the search cost can overwhelm any benefits derived from the computation of optimal cubes. The deductive reward signal employed by ALPHAMAPLE-SAT offers a domain-agnostic metric for decision-making across diverse combinatorial problems. For example, we show experimental results on benchmarks from two different domains (the minimum KS and Ramsey problems) with a 2.3× speedup in parallel (and up to 27× in sequential) CnC elapsed real time, demonstrating the efficacy of our method compared to the state-of-the-art March CnC solver. We also conduct ablation studies to confirm the effectiveness of the MCTS heuristic search in addressing the cubing problem. In the future, we plan to explore the use of deep learning-based MCTS techniques for the cubing problem to adapt and learn across different instances from the same problem class, thereby potentially leading to more efficient cubing strategies.

References

[Abe *et al.*, 2019] Kenshin Abe, Zijian Xu, Issei Sato, and Masashi Sugiyama. Solving NP-hard problems on graphs with extended AlphaGo Zero. *arXiv preprint arXiv:1905.11623*, 2019.

[Biere *et al.*, 2021] Armin Biere, Marijn Heule, Hans van Maaren, and Toby Walsh. Look-ahead based SAT solvers. *Handbook of Satisfiability*, 336:183, 2021.

[Bright *et al.*, 2021] Curtis Bright, Kevin KH Cheung, Brett Stevens, Ilias Kotsireas, and Vijay Ganesh. A SAT-based resolution of Lam's problem. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 35, pages 3669–3676, 2021.

[Bright *et al.*, 2022] Curtis Bright, Ilias Kotsireas, and Vijay Ganesh. When satisfiability solving meets symbolic computation. *Communications of the ACM*, 65(7):64–72, 2022.

[Browne *et al.*, 2012] Cameron B Browne, Edward Powley, Daniel Whitehouse, Simon M Lucas, Peter I Cowling, Philipp Rohlfshagen, Stephen Tavener, Diego Perez, Spyridon Samothrakis, and Simon Colton. A survey of Monte Carlo tree search methods. *IEEE Transactions on Computational Intelligence and AI in games*, 4(1):1–43, 2012.

[Cameron *et al.*, 2022] Chris Cameron, Jason Hartford, Taylor Lundy, Tuan Truong, Alan Milligan, Rex Chen, and Kevin Leyton-Brown. Monte Carlo Forest Search: UNSAT solver synthesis via Reinforcement learning. *arXiv preprint arXiv:2211.12581*, 2022.

[Coulom, 2006] Rémi Coulom. Efficient selectivity and backup operators in Monte-Carlo tree search. In *International conference on computers and games*, pages 72–83. Springer, 2006.

[Davis *et al.*, 1962] Martin Davis, George Logemann, and Donald Loveland. A machine program for theorem-proving. *Communications of the ACM*, 5(7):394–397, 1962.

[Fujita *et al.*, 2013] Hiroshi Fujita, Miyuki Koshimura, and Ryuzo Hasegawa. SCSat: a soft constraint guided SAT solver. In *Theory and Applications of Satisfiability Testing–SAT 2013: 16th International Conference, Helsinki, Finland, July 8-12, 2013. Proceedings 16*, pages 415–421. Springer, 2013.

[Graham-Lengrand and Färber, 2018] Stéphane Graham-Lengrand and Michael Färber. Guiding SMT solvers with Monte Carlo tree search and neural networks. In *Third Conference on Artificial Intelligence and Theorem Proving (AITP'2018)*, 2018.

[Heisinger, 2022] Maximilian Levi Heisinger. Paracooba enters SAT competition 2022. *SAT Competition 2022*, page 42, 2022.

[Heule *et al.*, 2011] Marijn JH Heule, Oliver Kullmann, Siert Wieringa, and Armin Biere. Cube and conquer: Guiding CDCL SAT solvers by lookaheads. In *Haifa Verification Conference*, pages 50–65. Springer, 2011.

[Heule *et al.*, 2016] Marijn JH Heule, Oliver Kullmann, and Victor W Marek. Solving and verifying the Boolean Pythagorean triples problem via cube-and-conquer. In *International Conference on Theory and Applications of Satisfiability Testing*, pages 228–245. Springer, 2016.

[Heule, 2018] Marijn Heule. Schur number five. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 32, 2018.

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

[Keszocze *et al.*, 2020] Oliver Keszocze, Kenneth Schmitz, Jens Schloeter, and Rolf Drechsler. Improving SAT solving using Monte Carlo tree search-based clause learning. In *Advanced Boolean Techniques: Selected Papers from the 13th International Workshop on Boolean Problems*, pages 107–133. Springer, 2020.

[Khalil *et al.*, 2022] Elias B Khalil, Pashootan Vaezipoor, and Bistra Dilkina. Finding backdoors to integer programs: a Monte Carlo tree search framework. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 36, pages 3786–3795, 2022.

[Kocsis and Szepesvári, 2006] Levente Kocsis and Csaba Szepesvári. Bandit based Monte-Carlo planning. In *European conference on machine learning*, pages 282–293. Springer, 2006.

[Li *et al.*, 2023] Zhengyu Li, Curtis Bright, and Vijay Ganesh. A SAT solver and computer algebra attack on the minimum Kochen–Specker problem. *arXiv preprint arXiv:2306.13319*, 2023.

[Liang *et al.*, 2016] Jia Hui Liang, Vijay Ganesh, Pascal Poupart, and Krzysztof Czarnecki. Learning rate based branching heuristic for sat solvers. In *Theory and Applications of Satisfiability Testing–SAT 2016: 19th International Conference, Bordeaux, France, July 5-8, 2016, Proceedings 19*, pages 123–140. Springer, 2016.

[Manthey, 2023] Norbert Manthey. Parallel by default–mergesat and mergesat-pcasso. *SAT Competition 2023*, page 34, 2023.

[McKay and Min, 1992] Brendan D McKay and Zhang Ke Min. The value of the Ramsey number R (3, 8). *Journal of Graph Theory*, 16(1):99–105, 1992.

[Nejati, 2020] Saeed Nejati. *CDCL(Crypto) and machine learning based SAT solvers for cryptanalysis*. PhD thesis, University of Waterloo, 2020.

[Previti *et al.*, 2011] Alessandro Previti, Raghuram Ramanujan, Marco Schaerf, and Bart Selman. Monte-Carlo style UCT search for Boolean satisfiability. In *Congress of the Italian Association for Artificial Intelligence*, pages 177–188. Springer, 2011.

[Radziszowski, 2011] Stanisław Radziszowski. Small Ramsey numbers. *The electronic journal of combinatorics*, 1000:DS1–Aug, 2011.

[Ramsey, 1987] Frank P Ramsey. On a problem of formal logic. In *Classic Papers in Combinatorics*, pages 1–24. Springer, 1987.

[Rosin, 2011] Christopher D Rosin. Multi-armed bandits with episode context. *Annals of Mathematics and Artificial Intelligence*, 61(3):203–230, 2011.

[Scavuzzo *et al.*, 2022] Lara Scavuzzo, Feng Chen, Didier Chételat, Maxime Gasse, Andrea Lodi, Neil Yorke-Smith, and Karen Aardal. Learning to branch with tree MDPs. *Advances in Neural Information Processing Systems*, 35:18514–18526, 2022.

[Schloeter, 2017] Jens Schloeter. A Monte Carlo tree search based conflict-driven clause learning SAT solver. *INFORMATIK 2017*, 2017.

[Semenov *et al.*, 2023] Alexander Semenov, Kirill Antonov, Stepan Kochemazov, and Artem Pavlenko. Using linearizing sets to solve multivariate quadratic equations in algebraic cryptanalysis. *IEEE Access*, 2023.

[Silva and Sakallah, 1996] JP Marques Silva and Karem A Sakallah. GRASP-a new search algorithm for satisfiability. In *Proceedings of International Conference on Computer Aided Design*, pages 220–227. IEEE, 1996.

[Silver *et al.*, 2016] David Silver, Aja Huang, Chris J Maddison, Arthur Guez, Laurent Sifre, George Van Den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, Marc Lanctot, et al. Mastering the game of Go with deep neural networks and tree search. *nature*, 529(7587):484–489, 2016.

[Silver *et al.*, 2017] David Silver, Thomas Hubert, Julian Schrittwieser, Ioannis Antonoglou, Matthew Lai, Arthur Guez, Marc Lanctot, Laurent Sifre, Dharshan Kumaran, Thore Graepel, et al. Mastering chess and shogi by self-play with a general reinforcement learning algorithm. *arXiv preprint arXiv:1712.01815*, 2017.

[Świechowski *et al.*, 2023] Maciej Świechowski, Konrad Godlewski, Bartosz Sawicki, and Jacek Mańdziuk. Monte Carlo tree search: A review of recent modifications and applications. *Artificial Intelligence Review*, 56(3):2497–2562, 2023.

[Wetzler *et al.*, 2014] Nathan Wetzler, Marijn JH Heule, and Warren A Hunt Jr. DRAT-trim: Efficient checking and trimming using expressive clausal proofs. In *International Conference on Theory and Applications of Satisfiability Testing*, pages 422–429. Springer, 2014.

[Xu and Lieberherr, 2022] Ruiyang Xu and Karl Lieberherr. Towards tackling QSAT problems with Deep Learning and Monte Carlo tree search. In *Science and Information Conference*, pages 45–58. Springer, 2022.

[Zaikin, 2022] Oleg Zaikin. Inverting 43-step md4 via cube-and-conquer, 2022.

[Zulkoski *et al.*, 2015] Edward Zulkoski, Vijay Ganesh, and Krzysztof Czarnecki. Mathcheck: A math assistant via a combination of computer algebra systems and SAT solvers. In *Automated Deduction-CADE-25: 25th International Conference on Automated Deduction, Berlin, Germany, August 1-7, 2015, Proceedings 25*, pages 607–622. Springer, 2015.