# Towards Tackling MaxSAT by Combining Nested Monte Carlo with Local Search

Hui Wang[1], Abdallah Saffidine[2], and Tristan Cazenave[1]

[1] LAMSADE, University Paris Dauphine - PSL
[2] The University of New South Wales

**Abstract.** Recent work proposed the UCTMAXSAT algorithm to address Maximum Satisfiability Problems (MaxSAT) and shown improved performance over pure Stochastic Local Search algorithms (SLS). UCTMAXSAT is based on Monte Carlo Tree Search but it uses SLS instead of purely random playouts. In this work, we introduce two algorithmic variations over UCTMAXSAT. We carry an empirical analysis on MaxSAT benchmarks from recent competitions and establish that both ideas lead to performance improvements. First, a nesting of the tree search inspired by the Nested Monte Carlo Search algorithm is effective on most instance types in the benchmark. Second, we observe that using a static flip limit in SLS, the ideal budget depends heavily on the instance size and we propose to set it dynamically. We show that it is a robust way to achieve comparable performance on a variety of instances without requiring additional tuning.

## 1 Introduction

Maximum Satisfiability (MaxSAT) problem is an extension of Boolean Satisfiability (SAT) problem. For MaxSAT, the task is to find a truth value assignment for each literal which satisfies the maximum number of clauses [12]. Stochastic Local Search (SLS) algorithms like WalkSat [15] and Novelty [19] are well studied to solve MaxSAT problems. These methods can not find a provable optimal solution but are usually used to search for an approximate optimal solution especially for larger problem instances. However, SLS algorithms are easy to get stuck in a local optimal solution and it's hard for them to escape. Thus, it's important to find an effective way to get rid of the local optimal solution. As a well-known successful method to address this exploration-exploitation dilemma, Monte Carlo Tree Search (MCTS) with UCT formula [4] is an ideal algorithm to deal with MaxSAT problems.

MCTS has shown impressive performance on game playing (including perfect information games and imperfect information games) [11,8,25], probabilistic single-agent planning [23], as well as most of problems which can be formed as a sequential decision making process, also know as Markov Decision Process (MDP) [3]. Based on the UCT formula, MCTS can address the exploration and exploitation dilemma in a theoretically sound way because UCT provides a state-of-the-art way to build the search tree based on the previous search

records (including the node visited count and the node estimate values of the visit). Typically, the estimate method of the leaf node in the search tree is a random rollout policy. However, in a lot of applications, many other rollout policies are created to improve the accuracy of the leaf node value estimation. For MaxSAT problem, UCTMAXSAT (simply written as UCTMAX in the following parts) employs SLS algorithms to estimate the node value [12].

However, UCTMAX only runs MCTS for the root node to build a search tree until the time out, which may not sufficiently use the advantage of UCT reported by the Nested Monte Carlo Tree Search (NMCTS) [2]. NMCTS runs MCTS from root to the end or the time out. For each step, after performing the MCTS, it chooses the best assignment value for the current step and then enters into the next step and performs the MCTS again. In addition, UCTMAX employs a fixed flip limit for SLS algorithms. But in a UCT-style SLS, the number of the unassigned variables (literals below the search tree frontier are unassigned) will decreases along with the search tree deepens. Therefore, we design a novel computation called *Dynamic SLS*, see Equation 2, for Monte Carlo methods used in this paper. The experimental results show that for most of the MaxSAT instances [3], the Dynamic SLS way is more robust than the fixed way used for UCTMAX to achieve comparable performance on a variety of instances without extra tuning. Besides, the results show that the NMCTS is better than the UCTMAX on most instances with moderate improvement.

Moreover, Nested Monte Carlo Search (NMCS) method [5] and its variants [6,7,6] have been successfully applied to master many NP-hard combinatorial optimization problems, like Morpion Solitaire [9], and achieve impressive performance [5,27]. However, NMCS has not been investigated to deal with MaxSAT problems. Therefore, this paper further studies the effectiveness of NMCS (also using Dynamic SLS as the state estimate) for MaxSAT.

Overall, the main contribution of this paper can be summarized as follows:

1. We examine various Monte Carlo Search techniques for the domain of MaxSAT, especially rollout policies and high-level searches. Through an extensive empirical analysis, we establish that (a) Purely random or heuristic-based rollouts are weaker than a Stochastic Local Search policy. (b) An MCTS-based search is weaker than Nested MCTS, especially in larger instances. NMCTS with WalkSat is weaker than NMCS, but is stronger with Novelty.
2. We introduce Dynamic SLS, a new rollout policy that dynamically computes the flip budget available for a stochastic local search. We demonstrate that Monte Carlo algorithms building on Dynamic SLS achieve comparable performance on standard MaxSAT benchmarks with previously existing Monte Carlo approaches without extra tuning.

The rest of the paper is structured as follows. Before introducing preliminaries of this work in Sec. 3, we present an overview of the most relevant literature in Sect. 2. Then we present Dynamic SLS based Monte Carlo methods in Sect. 4.

[3] The instances are from *ms_random* benchmark:
http://www.maxsat.udl.cat/15/benchmarks/index.html

Thereafter, we illustrate the orientation experiments on a group of MaxSAT instances to finalize the structure of our proposed methods in Sect. 5. Then the full length experiments are presented in Sect. 6. Finally, we conclude our paper and discuss future work.

## 2 Related Work

There are a lot of solvers created to master MaxSAT problems [13,18,1,14]. Generally, these solvers can be categorized into two different types, i.e. *complete* solvers and *incomplete* solvers. Complete solvers can provide provable the best solution for the problem. Incomplete solvers start from a random assignment and continue to search for a better solution according to some strategies. Typically, Stochastic Local Search algorithms like WalkSat [15] and Novelty [19] are well studied on MaxSAT [21,16]. These *incomplete* solvers suffer from an exploration-exploitation dilemma. And MCTS has shown successful performance of dealing with this dilemma [4]. Therefore, Tompkins et al. implemented an experimentation environment for mastering SAT and MaxSAT, called UCBMAX [24]. Furthermore, Goffinet et al proposed UCTMAX algorithm to enhance the performance of SLS [12]. However, UCTMAX only performs UCT search once from the root, which may not sufficiently use the power of MCTS comparing to run UCT search for each step until to the terminal node or time out, which is known as Nested Monte Carlo Tree Search [2]. In addition to MCTS and NMCTS, NMCS [5] and its variations [7,22,6] also perform well especially for single agent NP-hard combinatorial problems, like Morpion Solitaire [9], where they achieve the best record which has not yet been improved even employing deep learning techniques [27,10]. Therefore, in this paper, we firstly employ NMCTS and NMCS to master MaxSAT problems with SLS methods.

## 3 Preliminaries

### 3.1 MaxSAT

In MaxSAT, like SAT, the problem is specified by a propositional formula described in conjunctive normal form (CNF) [20]. But unlike SAT which the aim is to find a truth assignment to satisfy all clauses, MaxSAT is just to find a truth assignment to satisfy the maximum number of clauses. For a set of Boolean variables $V = \{v_1, v_2, v_3, ..., v_i\}$, a literal $l_j$ is either a variable $v_j$ or its negation $\neg v_j$, $1 \leq j \leq i$. A clause is a disjunction of literals (i.e., $c_i = l_1 \vee l_2 \vee, ..., \vee l_j$ ). A CNF formula $F$ is a set of clauses as conjunctive normal form (i.e., $F = c_1 \wedge c_2 \wedge, ..., \wedge c_i$). MaxSAT instances written as CNF can be easily found in our tested benchmark.

### 3.2 Heuristics

In order to test the different rollout policies for Monte Carlo Methods, here we present 3 simple heuristics that commonly used for MaxSAT.

1. H1 is the heuristic which assigns the value *from the first variable to the last variable* and H1 sets 0 for a variable that its positive value occurs more times than its negative value in all clauses.
2. H2 is the heuristic which, for each step, assigns the *variable* first which occurs the most times and H2 sets 0 for a variable that its positive value occurs more times than its negative value in all clauses.
3. H3 is the heuristic which, for each step, assigns the *literal* first which occurs the most times and H3 sets 0 for a variable that its positive value occurs more times than its negative value in all clauses.

### 3.3 Stochastic Local Search

Based on [12], in this paper, we also investigate two well-studied Stochastic Local Search (SLS) algorithms to deal with MaxSAT problem, namely WalkSat and Novelty.

**Algorithm 1** Walksat

```
1: function WalkSat(s)
2:     assignment←InitAssignment()
3:     while fliptimes < f do
4:         if random() < ε1 then
5:             v ← random variable
6:         else
7:             v ← best unassigned variable
8:         assignment←flip(v)
       return assignment
```

**WalkSat** As it can be seen in Algorithm 1, the idea of WalkSat is to initialize a random assignment (basic version) or according to the current found best solution (enhanced version) for each variable. Then an unsatisfied clause is selected. Further step is to select a variable to flip which has the highest bonus after flipping in the selected unsatisfied clause. The bonus is the change of the number of satisfied clauses after flipping the variable.

**Novelty** Novelty is similar to WalkSat. The first step is also to initialize a random assignment (basic version) or according to the current found best solution (enhanced version). But differently, for each variable in all unsatisfied clauses, its bonus is computed. Then in order to avoid flipping in a dead loop, a variable which has the highest bonus but not selected in the most recent flipping is selected to flip. Simply, after line 7 in Algorithm 1, we add **If** $v = v_f$ **and random()** $< 1 - \epsilon_2$ **then** $v \leftarrow v_s$. $v_f$ is the most recent flipped variable and $v_s$ is the second best unassigned variable.

### 3.4 Monte Carlo Tree Search

**Algorithm 2** Monte Carlo Tree Search

**function** MCTS($s$)
  Search($s$)
  $\pi_s \leftarrow$normalize($Q(s,\cdot)$)
  **return** $\pi_s$
**function** SEARCH($s$)
  **if** $s$ is a terminal state **then**
    $v \leftarrow v_{end}$
    **return** $v$
  **if** $s$ is not in the Tree **then**
    Add $s$ to the Tree, initialize $Q(s,\cdot)$ and $N(s,\cdot)$ to 0
    Run rollout policy and get the solution score $v_{rollout}$
    $v \leftarrow v_{rollout}$
    **return** $v$
  **else**
    Select an action $a$ with highest UCT value
    $s' \leftarrow$getNextState($s$, $a$)
    $v \leftarrow$Search($s\prime$)
    $Q(s,a) \leftarrow \frac{N(s,a)*Q(s,a)+v}{N(s,a)+1}$
    $N(s,a) \leftarrow N(s,a)+1$
  **return** $v$

According to [26,28,29], a recursive MCTS pseudo code is given in Algorithm 2. For each search, the rollout value is returned (or the game termination score). For each visit of a non-leaf node, the action with the highest UCT value is selected to investigate next [4]. After each search, the average win rate value $Q(s,a)$ and visit count $N(s,a)$ for each node in the visited trajectory is updated correspondingly. The UCT formula is as follows:

$$U(s,a) = Q(s,a) + c\sqrt{\frac{ln(N(s,\cdot))}{N(s,a)+1}} \tag{1}$$

The Nested Monte Carlo Tree Search (Due to the high computation, we only investigate level 1 for NMCTS in this paper) calls MCTS for each step of the assignment process.

### 3.5 Nested Monte Carlo Search

According to [5], the Nested Monte Carlo Search algorithm employs nested calls with rollouts and the record of the best sequence of moves with different levels. The basic level only performs random moves. Since a nested search may obtain worse results than a previous lower level search, recording the currently found

**Algorithm 3** Nested Monte Carlo Search

```
1: function NMC(s, level)
2:     chosenSeq←[], bestScore← −∞, bestSeq←[]
3:     while s is not terminal do
4:         for each m in legalMoves(s) do
5:             s′ ← PerformMove(s, m)
6:             if level = 1 then
7:                 (score, seq) ← run rollout policy
8:             else
9:                 (score, seq) ← NMC(s′, level-1)
10:        highScore ← highest score of the moves from s
11:        if highScore > bestScore then
12:            bestScore ← highScore
13:            chosenMove ← m associated with highScore
14:            bestSeq ← seq associated with highScore
15:        else
16:            chosenMove ← first move in bestSeq
17:            bestSeq ← remove first move from bestSeq
18:        s ← perform chosenMove to s
19:        chosenSeq ← append chosenMove to chosenSeq
20:    return (bestScore, chosenSeq);
```

best sequence and following it when the searches result in worse results than the best sequence is important. Therefore, we present the pseudo code for the basic Monte Carlo Search algorithm as Algorithm 3. In order to estimate the leaf nodes from themselves instead of their children, we further test a variant of NMCS, named ZNMCS (Zero Nested Monte Carlo Search), where in Algorithm 3, line 4 is changed to **for** $i = 0, i < t, i++$ **do**, in our experiments, $t = 10$. In addition, line 5 has been removed. And line 9 is changed to (score, seq)← ZNMCS($s$, level-1).

## 4 Dynamic SLS Based Monte Carlo Methods

This section proposes the Dynamic SLS method with MCTS and NMCS. Since the number of the unassigned variables decreases as the search tree deepens, we propose a Dynamic SLS to avoid redundant flips and enlarge search tree to improve the performance within a fixed time budget. The flip limit (written as $f$) is simply computed according to the following Equation:

$$f = w \times u \tag{2}$$

$w$ is a weight number, $u$ is the number of the unassigned variables which can be flipped. Considering MCTS, in the search tree, the variables, upon the leaf nodes, have already been assigned to a value, so they can not be flipped anymore. We also tested several exponent values powered by $u$ and finally found exponent equals 1 is the best.

In this work, we insert Dynamic SLS to replace rollout policy for MCTS (line 11 in Algorithm 2) and NMCS (line 7 in Algorithm 3, same to ZNMCS). In addition, according to [12], it is reported that using square number of the score is the best for UCTMAX, so in this work, for MCTS, we also replace the value calculation in line 7 and line 12 in Algorithm 2 to $v = pow(v_{end}, 2)$ and $v = pow(v_{dsls}, 2)$.

## 5 Orientation Experiments

### 5.1 Trial with Different Rollout

There are several ways to estimate the state value for Monte Carlo methods. One typical way is to simply run random simulations to get approximate values. In addition, for MaxSAT, there are many well designed heuristics to assign the truth values, based on the assignment, a proper value can be obtained. Besides, there are also several well studied SLS algorithms which can be applied to estimate the state value. Therefore, in order to determine which way is the best for the state estimate function, we use different ways to work together with NMCTS and NMCS to process our test setting (50 different instances, 70 variables each). The NMCTS simulation is set as 100. Time cost for each run is 50 seconds. each setting runs 10 repetitions. The results are shown in Table 1. We see that the heuristics all outperform random rollout, H3 is better than H2 and H2 is better than H1. Importantly, SLS methods perform significantly the best. So we adopt WalkSat and Novelty as the rollout policies for the further experiments. In addition, WalkSat for NMCS is better than NMCTS, but NMCTS with Novelty is the best.

Table 1: Results for Max3Sat Instances (70 variables) Using Different Rollout Policies for MCTS, NMCS. Results are average number of unsatisfied clauses on tested group instances, same to the following results.

| Method | NMCTS | NMCS | | |
|---|---|---|---|---|
| Level | | - playout | level 1 | level 2 |
| Random | 81.4 | 125.2 | 80.8 | 80.5 |
| H1 | 56.1 | 70.0 | 54.4 | 53.7 |
| H2 | 55.1 | 69.5 | 54.6 | 53.8 |
| H3 | 53.2 | 64.4 | 52.2 | 52.2 |
| WalkSat | 47.9 | 52.0 | **47.4** | **47.7** |
| Novelty | **47.7** | **51.9** | 48.8 | 49.0 |

### 5.2 UCTMAX vs NMCTS

Since [12] only investigated the UCTMAX with one time MCTS from the root until the time out. However, it does not perform an action to enter next state and

run UCT again like game playing. To this end, the NMCTS [2] method should be further investigated. We let the MCTS simulation as a fixed value (set as 100) so that each step will stop and get a search tree. Based on this search tree, a best action can be found and performed to enter to next state. Then it runs another UCT process until the time out or the termination. The results show that the NMCTS performs clearly better than the UCTMAX way. In order to enlarge the result difference for different settings, we use larger instances (50 instances, each has 140 variables. [12] also used 140 as the test instance size, but they only tested on one instance, we test on 50 different instances with this size to reduce the noise.) for this experiment and the following orientation experiments.

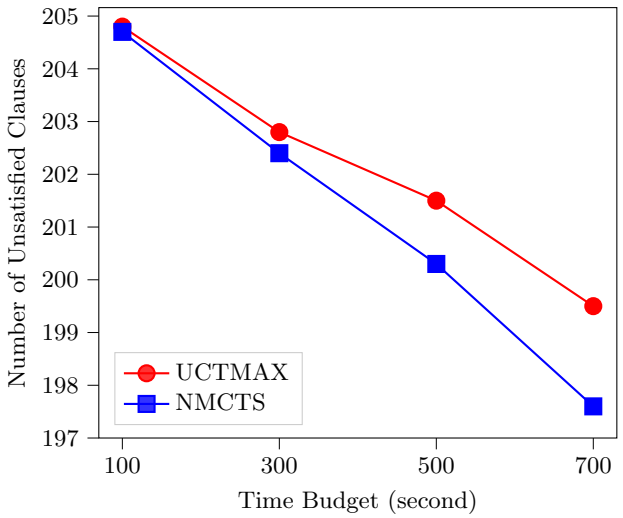



Fig. 1: Comparison of UCTMAX with NMCTS. NMCTS outperforms UCTMAX on 50 instances which has 140 variables each. For both UCTMAX and NMCTS, the $f$ is set as 2000 which is reported as the best.

### 5.3 Current Global Best Solution

Based on [12] and [5], we know that it is the key to keep the global best solution (the best of the local solutions from all steps) found so far and initialize the SLS algorithms with this global best solution. We still do not know whether it is also important in our Nested Monte Carlo Methods with SLS. Therefore, we design different combinations to show the importance.

The results are shown in Table 2, we see that with a small time budget (100 seconds), for NMCTS, keeping the global best records has shown the advantage, and initializing based on the global best records is also better than not but with small improvements. For NMCS, with 100 seconds, although we still find

Table 2: Impact of Random variable initialization and of keeping the global best solution on the performance of NMCTS and NMCS. Fixed number of flips (2000), 50 instances, 140 variables each.

| Keep Global | No | | Yes | | |
|---|---|---|---|---|---|
| Initialization | Rand | Best | Rand | Local | Best |
| Time Budget | 100s | | | | |
| NMCTS | 221.2 | 220.8 | 204.8 | 205.1 | **204.6** |
| NMCS | 198.8 | 199.1 | 199.3 | 198.8 | **198.7** |
| Time Budget | 300s | | | | |
| NMCTS | 219.9 | 219.8 | 202.9 | 202.9 | **202.6** |
| NMCS | 195.3 | 195.6 | 195.3 | 195.6 | **193.1** |

that keeping the global best records and initializing with them is the best, but it's not very significant. However, we see a clear improvement with larger time budget (300 seconds). The reason that different initialization does not differ too much might be that the flip limit is set too big so even if it is initialized from random, it can also reach a global record level after flipping. From this experiment, we can conclude that keeping the global best records and initializing based on them for SLS (in this case, it is WalkSat) are both important to the nested search. NMCS works better than NMCTS with WalkSat on 140 variables instances.

### 5.4 Probabilistic SLS Initialization

In order to further investigate the contribution of initializing WalkSat based on the global best solution found so far, we adopt the simplest but commonly used way to balance the exploration and exploitation, $\epsilon$-greedy, to initialize the assignment.

From Fig 2, we see that $\epsilon$=0.1 performs best, which further shows the best initialization way is to set literal assignment based on the best solution found so far but with a small randomness to initialize randomly. Thus, our following experiments are done with the $\epsilon$ as 0.1.

### 5.5 Fixed Flip Limits vs Dynamic Flip Limits

Goffinet et al. [12] used the fixed flip limits, which we found can be implemented in a dynamical way. Therefore, in this section, we test different $w$ values (from 0.5 to 25, but finally we only present results of $w \in \{1, 2, 4\}$ as they are better) for dynamic flip limits calculation equation (see Equation 2). And we found generally for both NMCTS and NMCS with different budget, $w$=2 is the best (only the result of 300 second is weaker for NMCTS). In addition, we test fixed flip limit with 2000 (which is reported the best for UCTMAX tuned on a single instance)

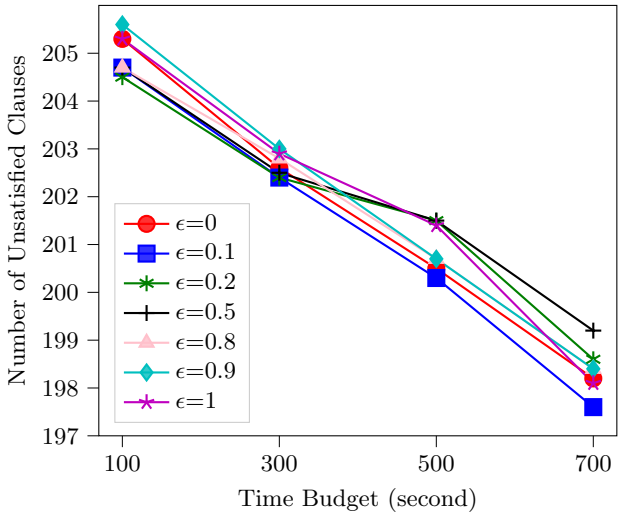


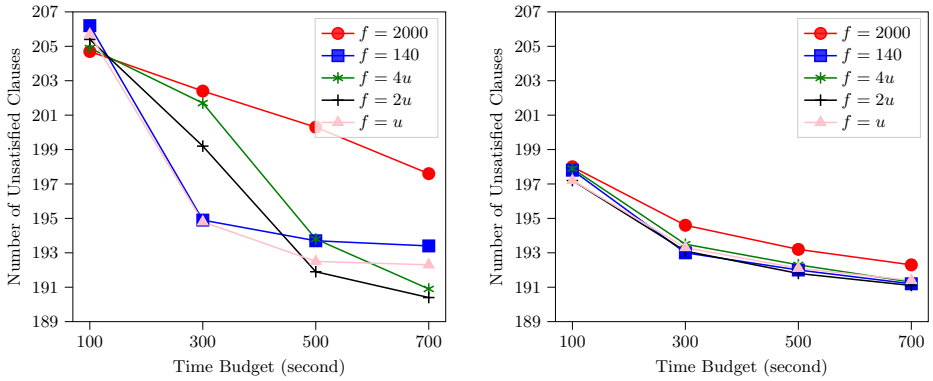

Fig. 2: Initializing Walksat Based on $\epsilon$-greedy for NMCTS on 50 instances with 140 variables each, $\epsilon$=0 means initializing WalkSat totally based on the global best solution. $\epsilon$=0.1 means there is 10% probability to take a random initialization for the literal, and so on. The $\epsilon$ equals 0.1 is the best.

and 140 (same as the average flip limit for each step with $w$=2). We found that with a fixed flip limit as 2000 is the worst and smaller limits increase the performance which shows that for Nested Monte Carlo methods, allocating time cost for relatively more steps contributes more.

Intuitively, even if a fine tuned fixed flip limit is found for a type of instances, it is not really applicable to set as the best for other instances. However, it is obviously that along with the increasing of sizes, the flip limit should also be larger. In order to test this assumption, we proposed the dynamic SLS and showed it works well for the category 140. Therefore, in order to show the adaptation of our Dynamic SLS method, after tuning the $w$ for Dynamic SLS, we further test the the best value we get for other larger instances which have 180 and 200 variables respectively, and compare the results with the fixed flip limit way (the best value is 140 for instances which have 140 variables). The results are presented in Fig 4. We see that $2u$ achieves better performance for both 180 and 200 variables categories, showing that our Dynamic SLS is more adaptive to other instances. Therefore, no redundant extra tuning cost is needed.

## 6 Experiments on Benchmark

In this section, we will show the experimental results on tested benchmark instances with aforementioned SLS based different Monte Carlo methods. The benchmark consists of 383 instances categorized by different numbers of vari-




(a) NMCTS (b) NMCS

Fig. 3: Comparison of Fixed SLS with Dynamic SLS for NMCTS and NMCS. In order to keep the $w$ consistent for all runs, considering the overall results, we decide setting the weight $w$ for Dynamic SLS flip limits as 2 is the best.

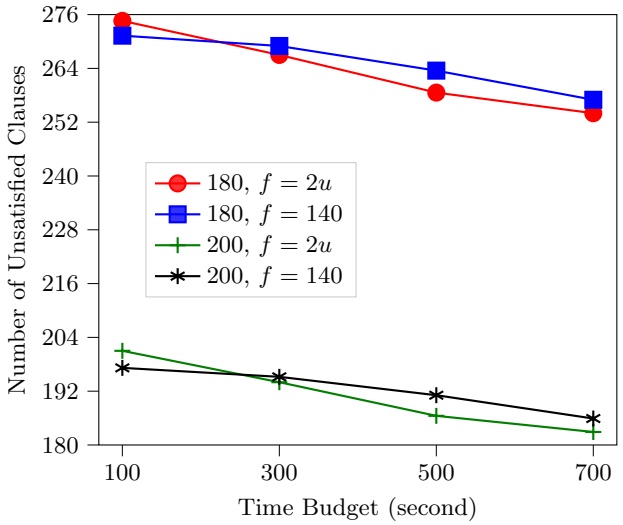



Fig. 4: Examples: Comparison of $2u$ and 140 flips for instances which have 180 and 200 variables respectively. NMCTS with Dynamic SLS is better than fixed flip limit on both 180 and 200 variables type, showing that our Dynamic SLS is more adaptive to other instances with different variable numbers.

ables. And for each category, there are a bunch of instances with different numbers of clauses.

Table 3: Results of MaxSat Instances Using WalkSat based UCTMAX, NMCTS, ZNMCS and NMCS respectively, with 300 seconds budget each run, 10 repetitions each.

| Benchmark | | Max_Walksat | | | | | | Known Optimal Solution |
|---|---|---|---|---|---|---|---|---|
| Vars | Instances | UCTMAX | NMCTS | ZNMCS | | NMCS | | |
| | | | $m = 100$ | round=5, level 1 | round=1, level 2 | round=5, level 1 | round=1, level 2 | |
| 70 | 50 | 47.7 | 47.8 | **47.1** | 47.2 | **47.1** | 47.7 | 46.8 |
| 80 | 50 | 27.3 | 27.4 | **27.1** | **27.1** | **27.1** | 27.5 | 26.9 |
| 120 | 50 | 223.3 | 219.1 | 219.2 | **218.7** | 219.0 | 221.0 | 196.1 |
| 140 | 50 | 201.8 | 199.2 | 194.0 | **193.1** | 195.3 | 195.9 | 184.8 |
| 160 | 42 | 257.7 | 256.4 | 246.1 | **243.1** | **243.1** | 246.6 | 227.6 |
| 180 | 44 | 248.2 | 247.4 | 237.7 | 235.9 | **235.4** | 238.5 | 220.6 |
| 200 | 49 | 195.7 | 195.2 | 186.2 | **184.5** | 184.9 | 187.6 | 171.0 |
| 250 | 24 | **7.7** | **7.7** | 8.2 | 8.6 | 8.5 | 8.7 | 5.5 |
| 300 | 24 | 9.3 | **9.1** | 9.8 | 10.2 | 10.1 | 10.5 | 6.3 |

Table 4: Results of MaxSat Instances Using Novelty based UCTMAX, NMCTS, ZNMCS and NMCS respectively, with 300 seconds budget each run, 10 repetitions each.

| Benchmark | | Max_Novelty | | | | | | Known Optimal Solution |
|---|---|---|---|---|---|---|---|---|
| Vars | Instances | UCTMAX | NMCTS | ZNMCS | | NMCS | | |
| | | | m=100 | round=5, level 1 | round=1, level 2 | round=5, level 1 | round=1, level 2 | |
| 70 | 50 | **47.1** | 47.4 | 48.6 | 48.0 | 47.9 | 47.9 | 46.8 |
| 80 | 50 | **27.4** | 27.8 | 28.9 | 28.4 | 28.2 | 28.2 | 26.9 |
| 120 | 50 | **212.7** | 212.8 | 213.6 | 213.2 | 213.1 | 213.2 | 196.1 |
| 140 | 50 | **185.7** | **185.7** | 186.6 | 186.0 | 186.1 | 186.1 | 184.8 |
| 160 | 45 | 228.9 | **228.8** | 229.8 | 229.2 | 229.1 | 229.3 | 227.6 |
| 180 | 44 | 222.4 | **222.2** | 223.2 | 222.4 | 222.5 | 222.6 | 220.6 |
| 200 | 49 | 173.2 | 173.2 | 173.8 | **173.1** | **173.1** | 173.3 | 171.0 |
| 250 | 24 | 11.6 | **11.2** | 12.3 | 13.0 | 12.7 | 13.1 | 5.5 |
| 300 | 24 | 14.4 | **14.0** | 14.7 | 15.3 | 15.0 | 15.4 | 6.3 |

From Table 3, we can see that with WalkSAT, Nested Monte Carlo methods perform better than UCTMAX. For smaller instances like 70 and 80 variables categories, ZNMCS and NMCS level 1 perform the best, and ZNMCS level 2 achieves similar scores. Interestingly, for categories from 120 to 200, the best performance is achieved by ZNMCS level 2. And for largest instances, NMCTS is the best. These results confirm that the high level nesting of Monte Carlo methods may lead to worse performance.

From Table 4, we still see that for Novelty, NMCTS performs the best for larger instances. But differently, for the small instances, UCTMAX achieves best scores. Only for type 200, ZNMCS achieves the best and the scores do not vary too much. Importantly, it is clear that for most instances, Comparing with

WalkSat, Novelty achieves better scores which are much more close to the known optimal solutions, which also shows that a better SLS estimate method achieves better performance together with Nested Monte Carlo. This also leads to that the improvements of NMCTS for Novelty are smaller than that for WalkSat, but we still see a possibility of increasing improvements along with the increasing of the instances sizes, which we should further investigate in future work.

In addition, the type 250 and 300 variables instances are different from others since their clauses are much more easy to be satisfied. In these cases, we find that the NMCTS performs much stably the best.

Therefore, for both WalkSat and Novelty, we can conclude that the nesting search improves the performance of Monte Carlo methods, especially for nesting the MCTS while dealing with larger instances and employing the better SLS method.

## 7 Conclusion and Future Work

In this paper, we first investigated different rollout policies (random, heuristics, SLS) for different Nested Monte Carlo methods, including NMCTS and NMCS to deal with MaxSAT problem. We found that heuristics are better than random, but SLS is the best rollout policy to work with Monte Carlo methods in the domain of MaxSAT. In addition, we confirmed that also for Nested Monte Carlo methods, SLS methods should also record the global best records and initialize assignment based on the found current best record. In order to further balance the exploration and exploitation, we employed $\epsilon$-greedy and found a proper $\epsilon$ value as 0.1 to randomly initialize the assignment for SLS, which improves the way that [12] initialized assignment fully based on the best record. The full benchmark experimental results show that for both WalkSat and Novelty based Monte Carlo methods, the nested tree search outperforms UCTMAX (Novelty in particularly performs better on larger instances), and NMCS with WalkSat also outperforms UCTMAX and even NMCTS. Therefore, we can conclude that nested search is important to deal with MaxSAT problems, especially for tree search on larger instances.

In the future, one way is to apply more powerful SLS algorithms together with Nested Monte Carlo methods like CCLS [17]. Besides, further investigation to find a light computation way for employing high level nested search is promising, especially for larger MaxSAT instances.

## References

1. Ansótegui, C., Gabas, J.: WPM3: an (in) complete algorithm for weighted partial MaxSAT. Artificial Intelligence **250**, 37–57 (2017)
2. Baier, H., Winands, M.H.: Nested Monte Carlo Tree Search for online planning in large mdps. In: ECAI. vol. 242, pp. 109–114 (2012)
3. Brechtel, S., Gindele, T., Dillmann, R.: Probabilistic MDP-behavior planning for cars. In: 2011 14th International IEEE Conference on Intelligent Transportation Systems (ITSC). pp. 1537–1542. IEEE (2011)

4. Browne, C.B., Powley, E., Whitehouse, D., Lucas, S.M., Cowling, P.I., Rohlfshagen, P., Tavener, S., Perez, D., Samothrakis, S., Colton, S.: A survey of Monte Carlo Tree Search methods. IEEE Transactions on Computational Intelligence and AI in games **4**(1), 1–43 (2012)
5. Cazenave, T.: Nested monte-carlo search. In: Twenty-First International Joint Conference on Artificial Intelligence (2009)
6. Cazenave, T.: Generalized nested rollout policy adaptation. In: Monte Carlo Search International Workshop. pp. 71–83. Springer (2020)
7. Cazenave, T., Teytaud, F.: Application of the nested rollout policy adaptation algorithm to the traveling salesman problem with time windows. In: International Conference on Learning and Intelligent Optimization. pp. 42–54. Springer (2012)
8. Cowling, P.I., Ward, C.D., Powley, E.J.: Ensemble determinization in Monte Carlo Tree Search for the imperfect information card game magic: The gathering. IEEE Transactions on Computational Intelligence and AI in Games **4**(4), 241–257 (2012)
9. Demaine, E.D., Demaine, M.L., Langerman, A., Langerman, S.: Morpion Solitaire. Theory of Computing Systems **39**(3), 439–453 (2006)
10. Doux, B., Negrevergne, B., Cazenave, T.: Deep reinforcement learning for Morpion Solitaire. In: Advances in Computer Games (2021)
11. Gelly, S., Silver, D.: Combining online and offline knowledge in UCT. In: Proceedings of the 24th international conference on Machine learning. pp. 273–280 (2007)
12. Goffinet, J., Ramanujan, R.: Monte-carlo tree search for the maximum satisfiability problem. In: International Conference on Principles and Practice of Constraint Programming. pp. 251–267. Springer (2016)
13. Heras, F., Larrosa, J., Oliveras, A.: MiniMaxSAT: An efficient weighted MaxSAT solver. Journal of Artificial Intelligence Research **31**, 1–32 (2008)
14. Ignatiev, A., Morgado, A., Marques-Silva, J.: RC2: an efficient MaxSAT solver. Journal on Satisfiability, Boolean Modeling and Computation **11**(1), 53–64 (2019)
15. Kautz, H., Selman, B., McAllester, D.: Walksat in the 2004 SAT Competition. In: Proceedings of the International Conference on Theory and Applications of Satisfiability Testing (2004)
16. Kroc, L., Sabharwal, A., Gomes, C.P., Selman, B.: Integrating systematic and local search paradigms: A new strategy for MaxSAT. In: Twenty-First International Joint Conference on Artificial Intelligence (2009)
17. Luo, C., Cai, S., Wu, W., Jie, Z., Su, K.: CCLS: an efficient local search algorithm for weighted maximum satisfiability. IEEE Transactions on Computers **64**(7), 1830–1843 (2014)
18. Martins, R., Manquinho, V., Lynce, I.: Open-WBO: A modular MaxSAT solver. In: International Conference on Theory and Applications of Satisfiability Testing. pp. 438–445. Springer (2014)
19. Menai, M.E.b., Batouche, M.: Efficient initial solution to extremal optimization algorithm for weighted MAXSAT problem. In: International Conference on Industrial, Engineering and Other Applications of Applied Intelligent Systems. pp. 592–603. Springer (2003)
20. Morgado, A., Heras, F., Liffiton, M., Planes, J., Marques-Silva, J.: Iterative and core-guided MaxSAT solving: A survey and assessment. Constraints **18**(4), 478–534 (2013)
21. Pelikan, M., Goldberg, D.E.: Hierarchical BOA solves ising spin glasses and MAXSAT. In: Genetic and Evolutionary Computation Conference. pp. 1271–1282. Springer (2003)

22. Rosin, C.D.: Nested rollout policy adaptation for Monte Carlo Tree Search. In: Twenty-Second International Joint Conference on Artificial Intelligence (2011)
23. Seify, A., Buro, M.: Single-agent optimization through policy iteration using Monte Carlo Tree Search. arXiv preprint arXiv:2005.11335 (2020)
24. Tompkins, D.A., Hoos, H.H.: UBCSAT: An implementation and experimentation environment for SLS algorithms for SAT and MAXSAT. In: International conference on theory and applications of satisfiability testing. pp. 306–320. Springer (2004)
25. Wang, H., Emmerich, M., Plaat, A.: Assessing the potential of classical Q-learning in General Game Playing. In: Benelux Conference on Artificial Intelligence. pp. 138–150. Springer (2018)
26. Wang, H., Emmerich, M., Preuss, M., Plaat, A.: Analysis of hyper-parameters for small games: Iterations or epochs in self-play? arXiv preprint arXiv:2003.05988 (2020)
27. Wang, H., Preuss, M., Emmerich, M., Plaat, A.: Tackling Morpion Solitaire with AlphaZero-like ranked reward reinforcement learning. In: 2020 22nd International Symposium on Symbolic and Numeric Algorithms for Scientific Computing (SYNASC). pp. 149–152. IEEE (2020)
28. Wang, H., Preuss, M., Plaat, A.: Warm-Start AlphaZero self-play search enhancements. In: Proceedings of the Parallel Problem Solving from Nature – PPSN XVI. pp. 528–542 (2020)
29. Wang, H., Preuss, M., Plaat, A.: Adaptive warm-start MCTS in AlphaZero-like deep reinforcement learning. In: Pacific Rim International Conference on Artificial Intelligence. pp. 60–71. Springer (2021)