Bilevel MCTS for Amortized O(1) Node Selection in Classical Planning

Masataro Asai
MIT-IBM Watson AI Lab
IBM Research Cambridge
masataro.asai@ibm.com

Abstract

We study an efficient implementation of Multi-Armed Bandit (MAB)-based Monte-Carlo Tree Search (MCTS) for classical planning. One weakness of MCTS is that it spends a significant time deciding which node to expand next. While selecting a node from an OPEN list with N nodes has $O(1)$ runtime complexity with traditional array-based priority-queues for dense integer keys, the tree-based OPEN list used by MCTS requires $O(\log N)$, which roughly corresponds to the search depth d. In classical planning, d is arbitrarily large (e.g., $2^k - 1$ in k-disk Tower-of-Hanoi) and the runtime for node selection is significant, unlike in game tree search, where the cost is negligible compared to the node evaluation (rollouts) because d is inherently limited by the game (e.g., $d \leq 361$ in Go). To improve this bottleneck, we propose a bilevel modification to MCTS that runs a best-first search from each selected leaf node with an expansion budget proportional to d, which achieves amortized $O(1)$ runtime for node selection, equivalent to the traditional queue-based OPEN list. In addition, we introduce Tree Collapsing, an enhancement that reduces action selection steps and further improves the performance.

1 Introduction

The exploration-exploitation trade-off has been a common research topic across AI literature, including Reinforcement Learning, Game Tree Search, and Classical Planning. Bandit-based Monte-Carlo Tree Search (MCTS) has been widely successful in addressing the trade-off in the first two areas, while it is not popular in the recent Classical Planning literature. As a result, successful methods (e.g., (Nakhost and Müller 2009; Xie et al. 2014; Kuroiwa and Beck 2022)) that try to address the trade-off in classical planning have commonly been approaches that are not backed by or benefiting from the statistical tools developed in the bandit community, even if they have their theoretical justifications.

Inspired by the initial work on applying MCTS to classical planning (Schulte and Keller 2014) which suggested that MCTS can implement traditional best-first search by viewing the tree as an OPEN list and modifying the backpropagation and selection rules, recent work (Wissow and Asai 2024) empirically showed that it can outperform queue-based diversified search as long as it uses the correct multi-armed bandit (MAB) algorithm that suits the classical planning context (UCB1-Normal2). However, the paper spends much of its space on the theoretical analysis and comparisons against simple baselines (e.g., GBFS) and other MCTS, making it unclear whether the approach can empirically compete against state-of-the-art planners such as LAMA (Richter, Westphal, and Helmert 2011).

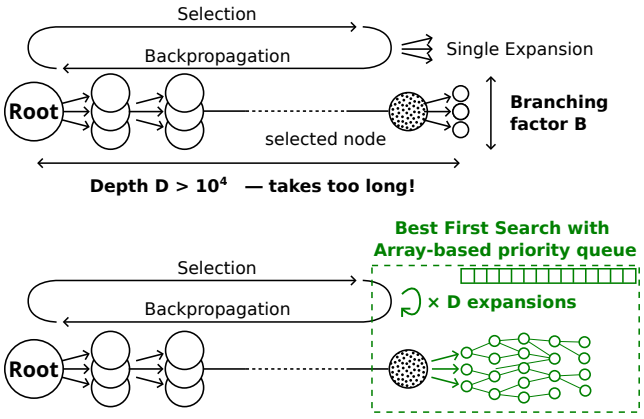

Figure 1: MCTS vs. Bilevel MCTS.

One weakness of MCTS is that it spends a significant amount of effort on node selection. While the traditional priority-queue-based OPEN list has $O(1)$ runtime for POP-MIN operation, the tree-based OPEN list used by MCTS requires $O(D)$ runtime for recursively descending the tree, where D is the depth of the path from the root to the selected leaf. In a perfectly balanced tree with a constant branching factor, D grows at an $O(\log N)$ rate for N leaf nodes and causes a significant bottleneck. This cost is significant in single-agent planning, unlike in game tree search, where the depth tends to be inherently limited by the game. For example, the maximum search depth of the Game of Go is $19 \times 19 = 361$, while the solution length of Tower of Hanoi with k disks is exponential ($2^k - 1$).

To improve this behavior, we propose Bilevel MCTS (Fig. 1) that runs a best-first search from each selected leaf with an expansion budget proportional to the depth of the node, achieving an amortized $O(1)$ runtime for selection per node. To further improve the performance, we propose an en-

hancement to MCTS called *Tree Collapsing*, which reduces the depth of the tree by letting the grandparents adopt their grandchildren as long as the number of children is below a threshold θ, thereby bypassing action selection steps.

Finally, to match the state of the art performance, we combine UCB1-Normal2 MAB, *novelty metric* (Lipovetzky and Geffner 2017), *alternation queue* (Röger and Helmert 2010), and *boosted preferred operators* (Richter and Helmert 2009) based on LAMA's configuration (Richter, Westphal, and Helmert 2011). The resulting configuration named **Nεbula** outperformed LAMA, SM-Type-LAMA (Kuroiwa and Beck 2022), and more recent advanced planners such as NOLAN (Corréa and Seipp 2025), DecStar (Gnad, Shleyfman, and Hoffmann 2018), and ApxNovelty (Singh et al. 2021).

This paper is organized in a way that alternates between progressively introducing improvements to the base algorithm and then performing a small experiment. All experiments are conducted on a compute cluster with Intel(R) Xeon(R) 6258R CPUs @ 2.70GHz in the Agile setting, i.e., under a 5-minute time limit and the 8GB memory limit (including the PDDL-to-SAS+ translation/grounding time), and with unit-cost conversion. Each algorithm is implemented in Fast Downward (Helmert 2006, FD) unless otherwise noted, and is evaluated on benchmark instances in IPC2018 and IPC2023. Randomized algorithms are evaluated with 5 seeds. Our code is going to be published in github.com/guicho271828/downward-mcts. Full version with appendix is available on arxiv:2508.08385.

2 Preliminary

We define a propositional STRIPS Planning problem as a 4-tuple $\langle P, A, I, G\rangle$ where P is a set of propositional variables, A is a set of actions, $I \subseteq P$ is the initial state, and $G \subseteq P$ is a goal condition. We omit the details of action applications as they are not important in this paper. It suffices to say an action $a \in A$ transitions from a state $s \subseteq P$ to a successor $s' = a(s) \subseteq P$. The task of classical planning is to find a sequence of actions called a *plan* $(a_1, \cdots, a_n)$ where, for $1 \leq t \leq n$, $s_0 = I$, $s_{t+1} = a_{t+1}(s_t)$, and $s_n \supseteq G$. A plan is *optimal* if there is no shorter plan. A plan is otherwise called *satisficing*. A problem setting that completely ignores the solution quality is called *agile* setting, while *satisficing* setting implies that the solver still attempts to find a shorter plan. This paper focuses on the *agile* setting.

A domain-independent heuristic function $h(s)$ returns an estimate of the cumulative cost from a state s to one of the goal states (states that satisfy G). Notable state-of-the-art functions that appear in this paper includes $h^{\mathrm{FF}}, h^{\mathrm{CEA}}, h^{\mathrm{CG}}$ (Hoffmann and Nebel 2001; Helmert and Geffner 2008; Helmert 2004).

2.1 Forward Heuristic Best-First Search

Classical planning problems are typically solved as a path finding problem defined over a state space graph induced by the transition rules, and the current dominant approach is based on *forward search*. Forward search maintains a set of search nodes called an *OPEN list*. They repeatedly (1) (*selection*) select a node from OPEN, (2) (*expansion*) generate its successor nodes, (3) (*evaluation*) evaluate the successor nodes, and (4) (*queueing*) reinsert them into OPEN. Termination typically occurs when a node is expanded that satisfies a goal condition, but a satisficing/agile algorithm can perform *early goal detection*, which immediately checks whether any successor node generated in step (2) satisfies the goal condition. Since this paper focuses on agile search, we use early goal detection for all algorithms.

Within forward search, forward *best-first* search defines a particular ordering in OPEN by defining *node evaluation criteria* (NEC) f for selecting the best node in each iteration. Let us denote a node by n and the state represented by n as s_n. As NEC, Dijkstra's search (Dijkstra 1959) uses $f_{\mathrm{Dijkstra}}(n) = g(n)$ (g-value), the minimum cost from the initial state I to the state s_n found so far. A* (Hart, Nilsson, and Raphael 1968) uses $f_{A^*}(n) = g(n) + h(s_n)$, the sum of g-value and the value returned by a heuristic function h (h-value). Greedy Best First Search (Doran and Michie 1966; Bonet and Geffner 2001) uses $f_{\mathrm{GBFS}}(n) = h(s_n)$. Forward best-first search that uses h is called forward *heuristic* best-first search.

MCTS is a class of forward heuristic best-first search that represents OPEN as the leaves of a tree. We call the tree a *tree-based open list*. Our MCTS is based on the description in (Keller and Helmert 2013; Schulte and Keller 2014). Overall, MCTS works in the same manner as other best-first search with a few key differences. (1) (*selection*) To select a node from the tree-based open list, it recursively selects an action on each branch of the tree, start from the root, using the NEC to select a successor node, descending until reaching a leaf node. (Sometimes the action selection rule is also called a *tree policy*.) At the leaf, it (2) (*expansion*) generates successor nodes, (3) (*evaluation*) evaluates the new successor nodes, (4) (*queueing*) attaches them to the leaf, and *backpropagates* (or *backs-up*) the information to the leaf's ancestors, all the way up to the root.

The evaluation obtains a heuristic value $h(s_n)$ of a leaf node n. In adversarial games like Backgammon or Go, it is obtained either by (1) hand-crafted heuristics, (2) *playouts* (or *rollouts*) where the behaviors of both players are simulated by uniformly random actions (*default policy*) until the game terminates, or (3) a hybrid *truncated simulation*, which returns a hand-crafted heuristic after performing a short simulation (Gelly and Silver 2011). In recent work, the default policy is replaced by a learned policy (Silver et al. 2016).

Trial-based Heuristic Tree Search (Keller and Helmert 2013; Schulte and Keller 2014, THTS), a MCTS for classical planning, is based on two key observations: (1) the rollout is unlikely to terminate in classical planning due to sparse goals, unlike adversarial games, like Go, which are guaranteed to finish in a limited number of steps with a clear outcome (win/loss); and (2) a tree-based open list can efficiently handle a large number of node reordering due to the updates to NEC, and thus is more flexible than a priority queue-based OPEN list, and can readily implement standard search algorithms such as A* and GBFS without significant performance penalty. In this paper, we use THTS and MCTS interchangeably.

Finally, Upper Confidence Bound applied to trees (Koc-

sis and Szepesvári 2006, UCT) is a MCTS that uses UCB1 (Auer, Cesa-Bianchi, and Fischer 2002) Multi-Armed Bandit (Thompson 1933; Robbins 1952; Bush and Mosteller 1953, MAB) algorithm for action selection and became widely popular in adversarial games. Schulte and Keller (2014) proposed several variants of UCT including GreedyUCT (GUCT), which differs from UCT in that the NEC assigned to the node is simply its heuristic value $h(s_n)$ just like in GBFS, rather than $g(n)+h(s_n)$ like in A*. This paper only discusses the greedy variants due to our focus on the agile planning.

Wissow and Asai (2024) demonstrated that, for a MAB-based MCTS to work on a particular task, the MAB algorithm must be chosen with careful theoretical considerations. They highlighted that the rewards based on heuristic values have no upper bound, unlike adversarial games which typically provide binary win (1) / loss (0) rewards, thus they do not meet the theoretical requirement to be used with the commonly used UCB1 MAB algorithm: To use UCB1, the reward distributions of all arms must have the common known finite support $[0, c]$. As a result, combining heuristics-based rewards with UCB1 means the algorithm does not guarantee any of UCB1's known theoretical properties, such as a logarithmic regret bound, leading to poor performance.

They then demonstrated that MAB algorithms that assume Gaussian reward distributions can handle heuristics-based rewards properly due to the assumption of infinite support $\mathbb{R}$. They used two Gaussian bandits, UCB1-Normal (Auer, Cesa-Bianchi, and Fischer 2002) and UCB1-Normal2 (Wissow and Asai 2024), where the latter outperformed the former as well as other finite-support bandits. UCB1-Normal2 is a frequentist version of Bayes-UCB2 (Tesauro, Rajan, and Segal 2010). We refer to the MCTS that uses UCB1-Normal2 as GreedyUCT-Normal2 (GUCTN2).

$$\text{UCB1}_i = \hat{\mu}_i + c\sqrt{(2\log T)/t_i} \tag{1}$$

$$\text{UCB1-Normal2}_i = \hat{\mu}_i + \hat{\sigma}_i\sqrt{2\log T} \tag{2}$$

Alg. 1 shows the pseudocode of MCTS adjusted for graph search, taken from (Schulte and Keller 2014). Since efficient graph search algorithms must avoid visiting the same state multiple times, Alg. 1 marks certain nodes as *locked*, and excludes them from the selection candidates. In an offline search, a node is locked either (1) when a node is a dead-end that will never reach a goal (detected by having no applicable actions, by a heuristic function, or other facilities), (2) when there is a node with the same state in the search tree with a smaller g-value, or (3) when all of its children are locked.

Alg. 1 differs slightly from existing work by omitting *tree grafting*, which involves moving a subtree to a new parent when it is *reopened* (reached via a path with a smaller g-value). While it could help satisficing search, it is generally detrimental in agile search. Appendix contains additional evaluations with this feature.

3 Bilevel MCTS

Theoretical regret bounds of MAB algorithms are often functions of the number of total pulls T, e.g., the regret of an *asymptotically optimal* MAB is logarithmically upper bounded by $O(\log T)$. In the application to the heuristic search, T corresponds to the total number of node evaluations in the current subtree. These theoretical guarantees *could* make MCTS efficient if the performance is measured by the number of node evaluations, but in reality it is a moot point because node evaluations are not the sole bottleneck in a search algorithm. Namely, under a simplified assumption, the runtime of *(1) selection* step in a tree-based OPEN list grows logarithmically to the size of OPEN.

Algorithm 1 High-level general MCTS. **Input**: Root node r, successor function S, NEC f, heuristic function h, ordered set B sorted by g. Initialize $\forall n; g(n) \leftarrow \infty$, $B \leftarrow \emptyset$.

```
1:  while True do
2:    Parent node p ← r
3:    while not leaf p do                          # Selection
4:      p ← arg min_{n∈S(p)} f(n)
5:    for a child node n ∈ S(p) do                 # Expansion
6:      return n if n is a goal.        # Early goal detection
7:      continue to line 5 if state s_n is not new
8:      compute h(s_n)                             # Evaluation
9:    B ← B ∪ {p}
10:   while not B.EMPTY() do                       # Backpropagation
11:     n ← B.POPMAX()              # largest g = depth first
12:     Update n's statistics and its lock status from S(n)
13:     if n's statistics or its lock status has changed then
14:       B ← B ∪ {n.PARENT}
```

Algorithm 2 Bilevel MCTS. (The difference from Alg. 1)

```
1:  while True do
2:    Parent node p ← r, Budget b ← 0
3:    while not leaf p do                          # Selection
4:      p ← arg min_{n∈S(p)} f(n), b ← b + 1
5:    Priority queue sorted by f: Q ← {p}
6:    while b > 0 and not Q.EMPTY() do                # BFS
7:      Parent node p ← Q.POPMIN(), b ← b − 1
8:      for a child node n ∈ S(p) do               # Expansion
9:        [identical to line 6-8 in Alg. 1]
10:       Q ← Q ∪ {n}
11:     p ← COLLAPSE(p, θ) if θ is optionally given
12:     B ← B ∪ {p}
13:   [identical to line 10-14 in Alg. 1]
```

Theorem 1. *Assume that the search space forms a tree with a constant branching factor B, and we have a tree-based OPEN list of depth D, containing $N = B^D$ leaves. The runtime of* selection *step is $BD = O(\log N)$.*

To demonstrate this, in the scatter plot in Fig. 2 (left), we compare the number of node evaluations per second using h^{FF} heuristics between GBFS and GUCTN2. The latter sometimes has more than three orders of magnitude slower evaluations compared to the former, indicating that action selections in MCTS can slow down the search significantly. (See appendix for larger plots with more heuristics.)

Further, to empirically validate that our cost model in Thm. 1 under the simplified assumption applies to the actual instances, we visualized the correlation between the node

processing speed and the average depth of the evaluated nodes on instances successfully solved by GUCTN2 with h^{FF}. Fig. 2 shows an inverse correlation between the search depth and the number of node evaluations per second. (See appendix for larger plots with more heuristics.)

Based on these theoretical and empirical observations, we propose *Bilevel MCTS* (Alg. 2), a modification to MCTS that performs a budgeted best-first search (BFS) from the leaf node reached by the action selection. It uses a priority queue Q (line 5) for the BFS, and the runtime complexity of its POPMIN operation affects the runtime of *selection* in MCTS. It has the following theoretical characteristics (Proofs in the appendix):

Theorem 2. *In the Bilevel MCTS, the amortized runtime of each* selection *is* $O(\log \log N)$ *if* Q *is a heap.*

Although this is already an improvement over the original $O(\log N)$, we are inspired by Burns et al. (2012) to improve it further by leveraging the unit cost structure of agile classical planning with array-based priority queues (Dial 1969) whose POPMIN have an $O(1)$ runtime.

Theorem 3. *In the Bilevel MCTS, the amortized runtime of each* selection *is* $O(1)$ *if* Q *is an array-based priority queue.*

Our bilevel modification *sacrifices search efficiency to gain low-level efficiency*, as it deviates from the policy suggested by the bandit. Therefore, we must empirically verify that the trade-off pays off in finding the goal faster.

We compared Bilevel GUCTN2 with GUCTN2, as well as baseline GBFS and a state-of-the-art diversified search algorithm Softmin-type(h) (Kuroiwa and Beck 2022) by measuring the number of instances solved within the time limit (# solved) and the Agile IPC score $\sum_i \min\{1, 1 - \log t_i / \log 300\}$ where $t_i > 0$ is the runtime for solving instance i in seconds. We used h^{FF}, h^{CG}, and h^{CEA} heuristics for this evaluation in order to demonstrate the heuristic independence. Table 1 shows that Bilevel GUCTN2 (base) significantly outperforms the standard GUCTN2 (base) and GBFS. We also observed that GUCTN2 struggles in IPC23, which Bilevel GUCTN2 improves significantly. As a result, Bilevel GUCTN2 outperforms Softmin in the IPC18+23 total # solved and IPC score in h^{CEA} and h^{FF}. Bilevel GUCTN2 did not outperform Softmin in a relatively cheap h^{CG} heuristic, which still emphasizes the remaining bottleneck in MCTS, but it narrowed the gap significantly compared to GUCTN2. The detailed domain-wise breakdown is available in the Appendix. Fig. 2 (right) also shows that Bilevel GUCTN2 significantly improves the node processing speed over GUCTN2, especially in a deeper search.

This tradeoff is surprising because the majority of its search is performed by the BFS when the search is deep. In other words, with a strong exploration policy (UCB1-Normal2), such a small amount of exploration is often sufficient to gain significant performance improvements.

3.1 Depth-Based vs. Fixed Budget Strategy

The budget b of BFS in Bilevel MCTS is automatically adjusted by the depth of the selected node (line 4). This dynamic budgeting is indeed crucial in the runtime complexity analysis (Thm. 2-3), but it also has a significant empirical effect.

To demonstrate this, we evaluated a variant named Fixed Budget Bilevel MCTS, where the BFS budget b is a hyperparameter $\in \{10, 100, 1000, 10000\}$. This variant spends too much time on the BFS when the tree is shallow, and too little when the tree is deep. Table 1 (rightmost columns) shows that, while Fixed Budget Bilevel GUCTN2 improved the #solved over the base GUCTN2 largely thanks to h^{FF}+IPC2023 configurations (+34 instances), it performed much worse overall compared to Bilevel GUCTN2 whose budget is dynamic. The Agile IPC scores also tend to show inferior performance.

3.2 Tree Collapsing

We propose a further modification to Bilevel-MCTS, called *Tree Collapsing*. The number of NEC computations grows proportionally to the depth of the tree. However, many action selection steps are not informative enough because some nodes contain only a small number of children. Since each selection in MCTS narrows down the set of candidate leaf nodes for expansion, the nodes with fewer children fails to reduce the subset *fast enough*.

We address this issue by collapsing the tree (see Fig. 3 and Alg. 3) when the width of the tree is below a certain threshold θ given as a hyperparameter. Let p be an expanded node, $S(p)$ be its successors, and p' be the parent of p. Before inserting p into the backpropagation queue B (Alg. 2, line 11), we compute $S(p') + S(p) - 1$. If it is less than θ, we deem that p' should have more children. Then we connect each $n \in S(p)$ directly to p' and discard p. Note that this happens separately from the table that tracks the predecessor of each node, which is necessary for extracting the final plan.

In addition, we remove the need for hyperparameter tuning with *Dynamic Tree Collapsing (DTC)*. DTC is inspired by the success of dynamic budgeting — The collapsing threshold θ is dynamically set to the depth of the node being collapsed.

We evaluated Tree Collapsing with various hyperparameter θ, for both the base GUCTN2 and the Bilevel GUCTN2. The result (Table 1) shows that it generally improves Bilevel in both # solved and the IPC scores. We also observed improvements in the node/sec, supporting our hypothesis. Due to space, we included the plots in the appendix. However, it tends to have a negligible impact when applied to the base GUCTN2, except in h^{FF}+IPC 2023 configuration where both the #solved and the IPC scores have improved.

This indicates a synergetic effect between Bilevel and Tree Collapsing, which is explained by its interaction with the queue operations in the backpropagation as follows. In Bilevel MCTS, backpropagation is paused during the BFS, and the expanded nodes wait in the backpropagation queue B. Meanwhile, COLLAPSE often returns the same grandparent node which are detected by B as duplicates. This results in a significant reduction of the size of B, as well as the runtime for the backpropagation. This additional effect is not present in the base MCTS because backpropagation happens after every expansion.

Next, we evaluated DTC. It managed to closely match the performance of the best hyperparameter $\theta = 40$, demonstrating its ability to eliminate the need for hyperparameter tuning. To avoid the tuning, we use the DTC exclusively hereafter.

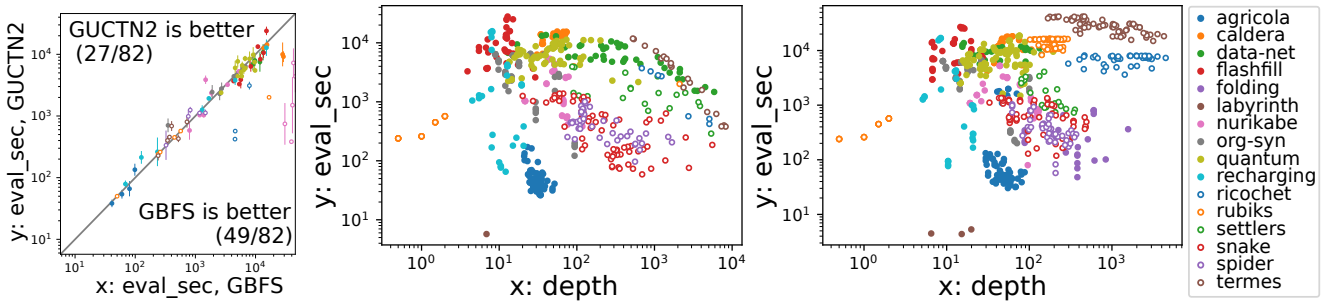

Figure 2: **Left:** Comparing the number of node evaluations per second on IPC instances solved by both GBFS (x-axis) vs. GUCTN2 (y-axis) within the limit, using h^{FF} heuristics. The points below the diagonal indicate that GUCTN2 has significantly slower node evaluations. **Middle, Right:** Log-log plots comparing the number of node evaluations per second (y-axis) versus the average depth of the nodes evaluated during the search (x-axis) for h^{FF}. GUCTN2 (middle) shows that the search becomes slower as it goes deeper. Bilevel GUCTN2 (right) explores deeper yet shows less degradation in the node/sec. The effect is pronounced in **termes**, **ricochet**, **rubiks**, **data-network**.

Algorithm 3 COLLAPSE(Parent p, threshold θ)

if p is not root **then**
  Grandparent $p' \leftarrow p$.PARENT
  if $|S(p')| + |S(p)| - 1 < \theta$ **then**
    $S(p') \leftarrow (S(p') \setminus \{p\}) \cup S(p)$; $p \leftarrow p'$
return p

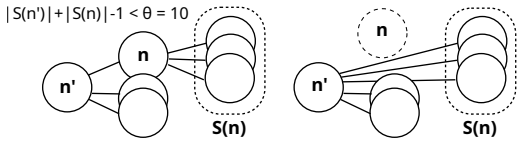

Figure 3: Example illustrating Tree Collapsing with $\theta = 10$.

3.3 Lazy Evaluation

Finally, we combined GUCTN2 with lazy evaluation, a well-known enhancement that trades heuristic accuracy with speed by assigning the parent's heuristic value to the children, instead of the children's own value. The results are primarily negative, and we discuss this in the appendix.

4 Orthogonality from Existing Techniques

Existing State-of-the-Art planners have employed a number of enhancements to achieve the performance, including *width-based search* (Lipovetzky and Geffner 2017), *alternation queue* (Röger and Helmert 2010), and *boosted preferred operators* (Richter and Helmert 2009). Interestingly, these techniques all address different aspects of exploration (i.e., deviates from the best-first behavior based on cost-to-go estimates). Width-based search encourages exploring states that are dissimilar to states visited before, the alternation queue tries to diversify the heuristics to use, and bandits in MCTS perform statistical reasoning on a single heuristic to decide whether to trust it more (exploit) or less (explore).

Since each technique targets a different aspect of exploration, we naturally expect them to offer orthogonal and modular performance improvement, which we demonstrate in this section through extensive evaluations.

Novelty Metric Before evaluating our novelty-enhanced MCTS later, we must first compare our novelty metric implementation with existing work in order to distinguish the effect of the implementation difference.

Novelty metric is a metric can be seen as a soft approximation of close list: It tries to rank states by checking how dissimilar a newly generated state is from the set of states that have been generated. Best-first search algorithms that incorporate the novelty metric as part of their sorting criteria are generally referred to as Best-First Width Search (BFWS). The metric is defined as the minimum size of the conjunctions of propositions that have never been true in a set of states visited before. The set is divided into subsets by their heuristic values, and the metric is measured for each subset. Since computing the novelty with large conjunctions requires an exponentially large storage, practical implementations often restrict the size of conjunctions (thus the maximum value of the metric) to $w_{\max} = 2$, or try to approximate the computation through a bloom filter (Singh et al. 2021, ApxNovelty). More recently, NOLAN (Corréa and Seipp 2025) proposed selecting $w_{\max} = 1$ when the number $|P|$ of propositions exceeds a threshold $V = 100$, and $w_{\max} = 2$ otherwise. We reimplemented the metric in FD. For a controlled experiment, all configurations use the FD-based grounder, although Lapkt-based planners could use FF- or Tarski-based grounders.

Lipovetzky and Geffner claimed that the open-list configuration which lexcographically sorts the nodes with cheap heuristics $\boldsymbol{f}_5 = [w_{\#g,\#r}, \#g]$ achieved a good result, where $\#g$ denotes goal-count heuristics (Fikes and Nilsson 1972) and $\#r$ counts how many propositions in the relaxed plan's actions' preconditions became true so far since the last time $\#g$ decreased in the path from the root. The low computational cost makes $\boldsymbol{f}_5$ an ideal baseline for comparing the low-level efficiency. Table 3 shows that our reimplementation of $\boldsymbol{f}_5$ (appendix for implementation details) was faster than the original Lapkt-based BFWS. Both implementations use the maximum novelty 2. NOLAN's dynamic $w_{\max}$ degraded the performance (appendix), presumably because their $V = 100$

	h, year	GB FS	Soft min	base $\theta=0$	GUCTN2 10	GUCTN2 20	GUCTN2 40	GUCTN2 80	GUCTN2 160	base $\theta=0$	Bilevel GUCTN2 10	Bilevel GUCTN2 20	Bilevel GUCTN2 40	Bilevel GUCTN2 80	Bilevel GUCTN2 160	Fixed Budget Bilevel DTC	Fixed Budget Bilevel $b=10$	Fixed Budget Bilevel 10^2	Fixed Budget Bilevel 10^3	Fixed Budget Bilevel 10^4
# solved	h^{CEA}, *18*	40	48.4	50.2	53.4	51.4	50	51.2	52.6	53.8	56.4	**61.4**	58.6	58.8	57.8	**61**	57.6	51.8	49	46.6
	h^{CEA}, *23*	30	34.2	25.8	26.6	27	27.2	25.4	26.4	30.8	32	32.2	34.2	33	**34.4**	33.4	32.4	**36.8**	33.8	33.6
	h^{CG}, *18*	35	59.2	42.4	43.2	44.6	44.4	45.6	41.4	56.2	58.4	59.6	**60.2**	**59.8**	56.2	57.6	51.6	55.8	48.6	48.8
	h^{CG}, *23*	36	39.4	25	25.4	27.2	26.8	28	26.8	41.6	41.6	42	**45.8**	45.4	43.4	**46.4**	34.2	42.2	44	42.2
	h^{FF}, *18*	60	80.2	79.8	82	79.8	82.8	77.8	81.4	87.4	87.2	88.6	90.4	**90.6**	**91.4**	90.2	85.8	83.4	82.2	72.8
	h^{FF}, *23*	51	55.4	23.8	24.6	27.2	27.2	25.8	24.8	58.2	56.8	57.8	58	56.2	55	58.4	45.4	56.8	**60**	**58.6**
	total	252	316.8	247	255.2	257.2	258.4	253.8	253.4	328	332.4	341.6	**347.2**	343.8	338.2	**347**	307	326.8	317.6	302.6
Agile IPC Score	h^{CEA}, *18*	17.7	20.6	23.8	23.4	**24.5**	23.8	23.7	24.1	22.4	23.4	**25.2**	23.6	23.9	23.3	24.4	23.1	22.1	20.8	18.8
	h^{CEA}, *23*	23.1	24.1	21.5	21.5	22.3	22.3	20.9	21.3	23.8	24.3	24.3	**25.3**	23.9	25.2	24.2	25.0	**26.3**	24.8	23.7
	h^{CG}, *18*	18.4	27.2	19.9	19.7	21.1	21.6	21.5	20.4	26.1	27.6	28.6	**30.0**	**28.7**	27.7	28.0	23.9	25.1	23.3	21.8
	h^{CG}, *23*	23.6	24.8	16.4	16.9	18.6	18.4	18.0	17.8	25.5	23.7	24.9	**25.9**	25.7	25.3	**26.5**	22.0	24.4	24.0	24.6
	h^{FF}, *18*	30.4	37.7	40.4	41.4	41.7	41.1	40.1	39.8	43.0	44.2	44.6	44.9	**45.7**	**46.4**	45.5	44.3	43.8	39.5	35.6
	h^{FF}, *23*	31.0	**35.5**	19.0	19.1	21.5	20.6	19.9	20.0	32.9	32.7	33.8	**34.4**	34.4	32.2	33.7	27.3	34.0	32.7	33.4
	total	144.2	169.9	141.0	142.1	149.7	147.8	144.0	143.3	173.8	176.0	181.4	**184.1**	182.3	180.1	**182.4**	165.6	175.7	165.2	158.1

Table 1: **Synopses**: Evaluating the effect of MCTS modifications (Bilevel, Tree Collapsing, Dynamic Tree Collapsing, Fixed Budget Bilevel) and the baselines (GBFS, Softmin-type(h)) over multiple heuristics $h \in \{h^{\text{CEA}}, h^{\text{CG}}, h^{\text{FF}}\}$. Each cell shows the number of IPC18+23 instances solved within 5 min. and 8GB, or the Agile IPC score $\sum_i \min\{1, 1 - {\log t_i}/{\log 300}\}$ where $t_i > 0$ is the runtime for instance i in seconds. **Bold scores** indicate top-2 (including ties). **Summary**: Bilevel generally improves GUCTN2. Tree Collapsing generally improves Bilevel. Tree Collapsing does not significantly improve GUCTN2 by itself except h^{FF} in IPC 2023 (thus overall better total). The best θ tends to be between 40 and 80, but depends on the domain and h. DTC avoids hyperparameter tuning and is overall in the second place. The fixed budget bilevel improved over GUCTN2, but performed significantly worse than the dynamic, depth-based budget strategy.

LAMA	$[h^{\text{FF}}, \text{PR}(h^{\text{FF}}), \#l, \text{PR}(\#l)]$
LAMA+SM	$[h^{\text{FF}}, \text{PR}(h^{\text{FF}}), \text{SM}([h^{\text{FF}}, g]), \#l, \text{PR}(\#l), \text{SM}([\#l, g])]$
NOLAN	$[h^{\text{FF}}, \text{PR}(h^{\text{FF}}), \#l, \text{PR}(\#l), [w_{\#l}, \#l]]$
LAMAe+BFWS	$[[w_{\#l,h^{\text{FF}}}, h^{\text{FF}}], \text{PR}(h^{\text{FF}}), [w_{\#l,h^{\text{FF}}}, \#l], \text{PR}(\#l)]$
N2DTC+BFWS	$\text{N2}([w_{h^{\text{FF}}}, h^{\text{FF}}])$
N2DTC+LAMAe	$[\text{N2}([h^{\text{FF}}]), \text{PR}(h^{\text{FF}}), \text{N2}([\#l]), \text{PR}(\#l)]$
Nɛbula	$[\text{N2}([w_{\#l,h^{\text{FF}}}, h^{\text{FF}}]), \text{PR}(h^{\text{FF}}), \text{N2}([w_{\#l,h^{\text{FF}}}, \#l]), \text{PR}(\#l)]$

Table 2: Summary of the configurations. LAMAe denotes the same configuration as LAMA, but with eager evaluations.

was tuned from pre-IPC14 domains. We also observed that their $\boldsymbol{f}_4 = [w_{\#l,h^{\text{FF}}}, \#l, h^{\text{FF}}]$ performed better overall (Table 3), where $\#l$ is landmark-sum heuristic (Büchner et al. 2023). Based on these results, we use $w_{\max} = 2$ and $\boldsymbol{f}_4$ as the base of our novelty-enhanced MCTS. We also reproduced $\boldsymbol{f}_6$, the second phase of Dual-BFWS (Lipovetzky and Geffner 2017), which underperformed (appendix).

Alternation with Boosted Preferred Operators *Alternation queue* (Röger and Helmert 2010) is a method for combining multiple heuristics by expanding nodes from different queues in a round-robin manner. A *preferred operator queue* is a priority queue that is populated only by the nodes generated by the operators that are marked by a heuristic function as preferred, and is often used as a member of alternated queues. When a preferred operator queue is *boosted*, the alternation queue watches the smallest value returned by each heuristics observed so far. When the value is updated, it expands from the preferred operator queues exclusively for a fixed number of iterations (the boost value). Let $\text{PR}(h)$ denote a preferred operator queue for a heuristic h. LAMA (Richter, Westphal, and Helmert 2011), which won IPC11, alternates 4 queues using the lazy evaluation. Kuroiwa and Beck (2022) added two Softmin-Type(h) queue (denoted as SM) to LAMA, resulting in a 6-queue SM-Type-LAMA that outperformed LAMA. Both configurations use boost=1000. See Table 2 for a summary of configurations. Note that, due to the agile setting, LAMA and variants evaluated here perform only the first iteration of the anytime search.

Nɛbula We combined Bilevel GUCTN2 + DTC, **n**ovelty **B**FWS, and **LA**MA into a configuration named *Nɛbula*. Let $\text{N2}([w_{\#l,h^{\text{FF}}}, h])$ denote a tree-based OPEN list that back-propagates the minimum $w_{\#l,h^{\text{FF}}}$ (Full-Bellman backup) and the average and the standard deviation of h (Monte-Carlo Backup) from the children. In each action selection, it selects the child with the best $w_{\#l,h^{\text{FF}}}$, then breaks ties with UCB1-Normal2$_i$ (Eq. 2).

Due to the negative result of lazy evaluations (Sec. 3.3) with MCTS, Nɛbula is based on **LAMAe** which uses eager evaluations and boost=10 (eager evaluation can rely less on preferred operators due to the accurate estimates.) It also avoids the issue of evaluating the novelty metric lazily, whose effect has not been studied in the literature (outside the scope of this paper). Then Nɛbula replaces the standard best-first queues in LAMAe with $\text{N2}([w_{\#l,h^{\text{FF}}}, h])$, rather than adding new queues like in SM-Type-LAMA, resulting in a simpler 4-queues configuration.

We compared Nɛbula with **ApxNovelty**, BFWS variants, **LAMA**, **SM-Type-LAMA**, **NOLAN**, as well as **DecStar** (Gnad, Shleyfman, and Hoffmann 2018) which won IPC2023 Agile. **NOLAN** adds a $\boldsymbol{f}_2$ BFWS configuration by Lipovetzky and Geffner (2017) to LAMA's queue. (We use $w_{\max} = 2$; appendix .)

		IPC year	Lapkt-BFWS Apx^{fd}	Lapkt-BFWS f_5^{fd}	FD-BFWS f_4	FD-BFWS f_5	LAMA	LAMA +SM	Dec Star	NO LAN	LAMAe	LAMAe +BFWS	N2DTC +BFWS	N2DTC +LAMAe	Nebula
5 min. agile	# solved	ipc18	99	83	109	92	111	**120.4**	99	105	98	100	107	94.6	**122**
		ipc23	67	67	55	71	66	66	63	**72**	61	**72**	60.4	61.6	70.2
		total	166	150	164	163	177	**186.4**	162	177	159	172	167.4	156.2	**192.2**
	score	ipc18	51.3	41.4	53.9	45.3	57.1	**60.3**	51.1	47.5	47.2	45.8	52.1	44.3	**60.0**
		ipc23	39.0	39.9	36.4	**43.9**	38.3	39.0	42.6	**45.2**	38.2	39.3	34.7	34.8	39.4
		total	90.3	81.3	90.2	89.2	95.4	**99.3**	93.7	92.8	85.4	85.2	86.8	79.1	**99.4**
30 min. extended	# solved	ipc18	118	90	132	102	123	133.4	110	133	117	**140**	126.8	111.6	**145**
		ipc23	83	69	76	72	81	80	70	**86**	69	**86**	80.4	73.4	85.6
		total	201	159	208	174	204	213.4	180	219	186	**226**	207.2	185	**230.6**
	score	ipc18	65.4	52.5	69.8	58.1	71.0	**76.3**	63.6	65.0	62.1	63.5	67.8	58.4	**77.9**
		ipc23	48.5	46.8	43.8	**50.4**	46.6	47.2	48.4	**53.7**	45.0	49.2	44.0	42.7	48.9
		total	113.9	99.2	113.7	108.6	117.6	**123.4**	112.0	118.7	107.1	112.8	111.8	101.0	**126.8**

Table 3: Top-2 (including ties) are highlighted in **bold**. Synopsis: Apx^{fd} = Approximate Novelty with FD-based grounder, f_5^{fd} = The original Lapkt-based BFWS with FD-based grounder, f_4, f_5 = Our BFWS reimplementations, +SM = Softmin-Type-LAMA.

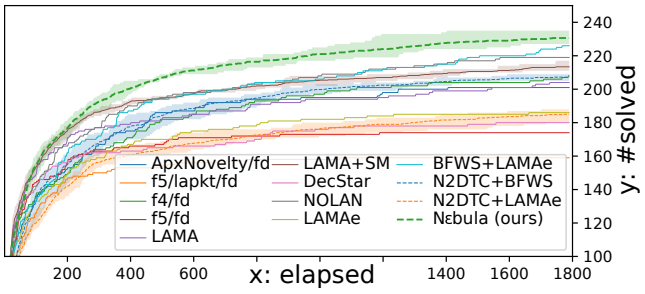

Figure 4: A histogram plot of the number of IPC18+IPC23 instances solved within 30 minutes. Lines indicate the average of 5 seeds, while the bands indicate the max/min.

The 5-minute agile result (Table 3, top) shows that Nebula outperforms them in terms of # solved, and is on par with SM-Type-LAMA in the IPC score. It outperformed ApxNovelty with the same FD-based grounder despite using a simpler novelty metric. Reimplementing approximate novelty in Fast Downward and combine it with Nebula is an interesting avenue of future work. Despite not an IPC participant, SM-Type-LAMA was also quite strong, outperforming ApxNovelty without novelty metric. Domain-wise results and solution cost comparisons are included in the appendix .

As Nebula combines 3 techniques, we perform an ablation study removing one of the 3 components. In Table 3, **LAMAe+BFWS** replaces the best-first queues in LAMAe with tiebreaking queues $[w_{\#l,h^{\mathrm{FF}}}, h]$ for $h \in \{\#l, h^{\mathrm{FF}}\}$. **N2DTC+LAMAe** replaces the best-first queues in LAMAe with $\mathrm{N2}([h])$ for $h \in \{\#l, h^{\mathrm{FF}}\}$. **N2DTC+BFWS** is a single queue configuration using $\mathrm{N2}([w_{h^{\mathrm{FF}}}, h^{\mathrm{FF}}])$. All configurations use the eager search. Appendix also contains other extensive evaluations, including the result of varying the collapsing parameter θ in Nebula.

Extended 30 Minutes Run Bilevel MCTS is still a computationally demanding algorithm in terms of the runtime and the memory in comparison to simple queue-based search because of the tree management cost. Its true potential is unlocked in the longer, extended 30 minutes runs. Fig. 4 shows a coverage histogram plot which demonstrates this. Table 3 (bottom half) also shows the instances solved and the extended IPC score $\sum_i \min\{1, 1 - \log t_i / \log 1800\}$ which mimics the Agile IPC score. Nebula significantly exceeds existing work (230.6 instances solved), including once closely matched SM-Type-LAMA (213.4). However, near the end of 30 minute time limit, the space for trees caused memory exhaustion, which allowed queue-based LAMAe+BFWS to catch up (226). Micro-optimization of the tree management could widen the gap. See Appendix for the similar histogram plots using the memory usage and the number of node evaluations.

5 Related Work

MCTS variants that perform Bilevel search are common in the game literature (Kishimoto et al. 2012), such as PN^2 (Kishimoto et al. 2012). However, the main motivation of existing work tends to be improving the memory usage, much like linear-memory graph search algorithms (e.g. IDA* (Korf 1985), RBFS (Korf 1993)), and they discard the nodes explored by the second-level search to release the memory, unlike our Bilevel MCTS.

Df-UCT (Yoshizoe et al. 2011) similarly aims at improving the action selection performance, but with the main focus on improving the communication overhead in distributed search.

6 Conclusion

Classical planning tasks have a significantly longer horizon compared to game tree search, which makes the application of naive MCTS impractical. We proposed Bilevel MCTS, a modification that reduces the action selection overhead in MCTS by running a best-first search from the leaf nodes. We also proposed tree-collapsing to reduce the overhead further. Combining them with proven techniques in the literature (novelty, alternation with boosted preferred operator queue) yielded a state-of-the-art planner, Nebula, that outperforms existing work.

References

Auer, P.; Cesa-Bianchi, N.; and Fischer, P. 2002. Finite-Time Analysis of the Multiarmed Bandit Problem. *Machine Learning*, 47(2-3): 235–256.

Bonet, B.; and Geffner, H. 2001. Planning as Heuristic Search. *Artificial Intelligence*, 129(1): 5–33.

Büchner, C.; Keller, T.; Eriksson, S.; and Helmert, M. 2023. Landmark progression in heuristic search. In *Proc. of the International Conference on Automated Planning and Scheduling (ICAPS)*, volume 33, 70–79.

Burns, E. A.; Hatem, M.; Leighton, M. J.; and Ruml, W. 2012. Implementing Fast Heuristic Search Code. In *Proc. of Annual Symposium on Combinatorial Search*.

Bush, R. R.; and Mosteller, F. 1953. A Stochastic Model with Applications to Learning. *The Annals of Mathematical Statistics*, 559–585.

Corréa, A. B.; and Seipp, J. 2025. Alternation-Based Novelty Search. In *Proc. of the International Conference on Automated Planning and Scheduling (ICAPS)*.

Dial, R. 1969. Shortest Path Forest with Topological Ordering. *CACM*, 12(11): 632–633.

Dijkstra, E. W. 1959. A Note on Two Problems in Connexion with Graphs. *Numerische mathematik*, 1(1): 269–271.

Doran, J. E.; and Michie, D. 1966. Experiments with the Graph Traverser Program. *Proceedings of the Royal Society of London. Series A. Mathematical and Physical Sciences*, 294(1437): 235–259.

Fikes, R. E.; and Nilsson, N. J. 1972. STRIPS: A New Approach to the Application of Theorem Proving to Problem Solving. *Artificial Intelligence*, 2(3): 189–208.

Gelly, S.; and Silver, D. 2011. Monte-Carlo Tree Search and Rapid Action Value Estimation in Computer Go. *Artificial Intelligence*, 175(11): 1856–1875.

Gnad, D.; Shleyfman, A.; and Hoffmann, J. 2018. DecStar–STAR-topology DECoupled Search at its best. In *Proc. of the International Planning Competition*, 42–46.

Hart, P. E.; Nilsson, N. J.; and Raphael, B. 1968. A Formal Basis for the Heuristic Determination of Minimum Cost Paths. *Systems Science and Cybernetics, IEEE Transactions on*, 4(2): 100–107.

Helmert, M. 2004. A Planning Heuristic Based on Causal Graph Analysis. In *Proc. of the International Conference on Automated Planning and Scheduling (ICAPS)*, 161–170.

Helmert, M. 2006. The Fast Downward Planning System. *J. Artif. Intell. Res.(JAIR)*, 26: 191–246.

Helmert, M.; and Geffner, H. 2008. Unifying the Causal Graph and Additive Heuristics. In *Proc. of the International Conference on Automated Planning and Scheduling (ICAPS)*, 140–147.

Hoffmann, J.; and Nebel, B. 2001. The FF Planning System: Fast Plan Generation through Heuristic Search. *J. Artif. Intell. Res.(JAIR)*, 14: 253–302.

Keller, T.; and Helmert, M. 2013. Trial-Based Heuristic Tree Search for Finite Horizon MDPs. In *Proc. of the International Conference on Automated Planning and Scheduling (ICAPS)*.

Kishimoto, A.; Winands, M. H.; Müller, M.; and Saito, J.-T. 2012. Game-Tree Search using Proof Numbers: The First Twenty Years. *ICGA journal*, 35(3): 131–156.

Kocsis, L.; and Szepesvári, C. 2006. Bandit Based Monte-Carlo Planning. In *Proc. of the European Conference on Machine Learning and Principles and Practice of Knowledge Discovery in Databases*, 282–293. Springer.

Korf, R. E. 1985. Depth-First Iterative-Deepening: An Optimal Admissible Tree Search. *Artificial Intelligence*, 27(1): 97–109.

Korf, R. E. 1993. Linear-Space Best-First Search. *Artificial Intelligence*, 62(1): 41–78.

Kuroiwa, R.; and Beck, J. C. 2022. Biased Exploration for Satisficing Heuristic Search. In *Proc. of the International Conference on Automated Planning and Scheduling (ICAPS)*.

Lipovetzky, N.; and Geffner, H. 2017. Best-First Width Search: Exploration and Exploitation in Classical Planning . In *Proc. of AAAI Conference on Artificial Intelligence*.

Nakhost, H.; and Müller, M. 2009. Monte-Carlo Exploration for Deterministic Planning. In *Proc. of International Joint Conference on Artificial Intelligence (IJCAI)*.

Richter, S.; and Helmert, M. 2009. Preferred Operators and Deferred Evaluation in Satisficing Planning. In *Proc. of the International Conference on Automated Planning and Scheduling (ICAPS)*, 273–280.

Richter, S.; Westphal, M.; and Helmert, M. 2011. LAMA 2008 and 2011. In *Proc. of the International Planning Competition*, 117–124.

Robbins, H. 1952. Some Aspects of the Sequential Design of Experiments. *Bulletin of the American Mathematical Society*, 58(5): 527–535.

Röger, G.; and Helmert, M. 2010. The More, the Merrier: Combining Heuristic Estimators for Satisficing Planning. In *Proc. of the International Conference on Automated Planning and Scheduling (ICAPS)*.

Schulte, T.; and Keller, T. 2014. Balancing Exploration and Exploitation in Classical Planning. In *Proc. of Annual Symposium on Combinatorial Search*.

Silver, D.; Huang, A.; Maddison, C. J.; Guez, A.; Sifre, L.; Van Den Driessche, G.; Schrittwieser, J.; Antonoglou, I.; Panneershelvam, V.; Lanctot, M.; et al. 2016. Mastering the Game of Go with Deep Neural Networks and Tree Search. *Nature*, 529(7587): 484–489.

Singh, A.; Lipovetzky, N.; Ramírez, M.; and Segovia-Aguas, J. 2021. Approximate Novelty Search. In *Proc. of the International Conference on Automated Planning and Scheduling (ICAPS)*, volume 31, 349–357.

Tesauro, G.; Rajan, V.; and Segal, R. 2010. Bayesian Inference in Monte-Carlo Tree Search. In *Proc. of the International Conference on Uncertainty in Artificial Intelligence (UAI)*, 580–588.

Thompson, W. R. 1933. On the Likelihood that One Unknown Probability Exceeds Another in View of the Evidence of Two Samples. *Biometrika*, 25(3-4): 285–294.

Wissow, S.; and Asai, M. 2024. Scale-Adaptive Balancing of Exploration and Exploitation in Classical Planning. In *Proc. of European Conference on Artificial Intelligence*.

Xie, F.; Müller, M.; Holte, R. C.; and Imai, T. 2014. Type-Based Exploration with Multiple Search Queues for Satisficing Planning. In *Proc. of AAAI Conference on Artificial Intelligence*.

Yoshizoe, K.; Kishimoto, A.; Kaneko, T.; Yoshimoto, H.; and Ishikawa, Y. 2011. Scalable Distributed Monte-Carlo Tree Search. In *Proc. of Annual Symposium on Combinatorial Search*, volume 2, 180–187.