ECAI 2024
U. Endriss et al. (Eds.)

doi:10.3233/FAIA240994

Scale-Adaptive Balancing of Exploration and Exploitation in Classical Planning

Stephen Wissow[a,*,1] **and Masataro Asai**[b,1]

[a]University of New Hampshire
[b]MIT-IBM Watson AI Lab

Abstract. Balancing exploration and exploitation has been an important problem in both adversarial games and automated planning. While it has been extensively analyzed in the Multi-Armed Bandit (MAB) literature, and the game community has achieved great success with MAB-based Monte Carlo Tree Search (MCTS) methods, the planning community has struggled to advance in this area. We describe how Upper Confidence Bound 1's (UCB1's) assumption of reward distributions with known bounded support shared among siblings (arms) is violated when MCTS/Trial-based Heuristic Tree Search (THTS) in previous work uses heuristic values of search nodes in classical planning problems as rewards. To address this issue, we propose a new Gaussian bandit, UCB1-Normal2, and analyze its regret bound. It is variance-aware like UCB1-Normal and UCB-V, but has a distinct advantage: it neither shares UCB-V's assumption of known bounded support nor relies on UCB1-Normal's conjectures on Student's t and χ^2 distributions. Our theoretical analysis predicts that UCB1-Normal2 will perform well when the estimated variance is accurate, which can be expected in deterministic, discrete, finite state-space search, as in classical planning. Our empirical evaluation confirms that MCTS combined with UCB1-Normal2 outperforms Greedy Best First Search (traditional baseline) as well as MCTS with other bandits.

1 Introduction

From the early history of AI and in particular of automated planning and scheduling, heuristic forward search has been a primary methodology for tacking challenging combinatorial problems. A rich variety of search algorithms have been proposed, including Dijkstra search [13], A*/ WA* [22], and Greedy Best First Search [8, GBFS]. They are divided into three categories: *optimizing*, which must guarantee the optimality of the output, *satisficing*, which may or may not attempt to minimize solution cost, and *agile*, which ignores solution cost and focuses on finding a solution quickly. This paper focuses on the agile setting.

Unlike optimizing search, theoretical understanding of satisficing and agile search has been limited. Recent theoretical work on GBFS [24, 26, 25, 35] refined the concept of search progress in agile search, but only based on a post hoc analysis that depends on oracular information, making their insights difficult to apply to practical search algorithm design, although it has been recently applied to a learning-based approach [16]. More importantly, their analysis is incompatible with a wider range of randomized algorithms [41, 28, 32, 55, 57, 58, 59, 60, 3, 35] that outperform the deterministic baseline with randomized explorations; as a result, their detailed theoretical properties are largely unknown except for probabilistic completeness [55]. It is unsurprising that analyzing randomized algorithms requires a statistical perspective, which is also growing more important due to recent advances in learned heuristic functions [54, 15, 51, 17, 47, 20, 19].

In this paper, we tackle the problem of balancing exploration and exploitation in classical planning through a statistical lens and from the perspective of MABs. Previous work showed that traditional forward search algorithms (A*, GBFS) can be seen as a form of MCTS, but we refine and recast this paradigm as a repeated process of collecting a reward dataset and exploring the environment based on estimates obtained from this dataset. This perspective reveals several theoretical issues in GreedyUCT [50], a MCTS modified for classical planning that uses UCB1. Among other things, the optimization objective of classical planning has no a priori known bound, which violates the bounded reward assumption of UCB1.

To apply MAB to classical planning correctly, we propose UCB1-Normal2, a new Gaussian bandit, and GreedyUCT-Normal2, a new agile planning algorithm that combines MCTS with UCB1-Normal2, and show that GreedyUCT-Normal2 outperforms traditional agile algorithms (GBFS), state-of-the-art diversified search (Softmin-Type(h) [35]), existing MCTS-based algorithms (GreedyUCT, GreedyUCT*), MCTS combined with other variance-aware bandits (UCB1-Normal and UCB-V [4]) or simple-regret bandits (TTTS [49]).

While most of our empirical analyses are based on Pyperplan-based implementation and focus on algorithmic efficiency rather than on low-level performance, we also re-implemented the algorithms in C++/Fast-Downward and performed evaluations in IPC 2018 satisficing instances, where GUCT-Normal2 outperformed Softmin-Type(h) on the number of instances solved.

In summary, our core contributions are as follows.

- We identify theoretical issues that arise when applying UCB1 to planning tasks.
- To address these issues, we present UCB1-Normal2, a new Gaussian bandit. We analyze its regret bound, which improves as the estimated variance is closer to the true variance, and is constant when they match. This makes it particularly powerful in a deterministic and finite state space such as classical planning.
- GreedyUCT-Normal2, a new forward search algorithm that combines UCB1-Normal2 with MCTS, outperforms existing algo-

* Corresponding Author. Email: swj@unh.edu
[1] Equal contribution.

rithms in agile classical planning.

The supplemental materials containing proofs and additional plots are available on arxiv.org/abs/2305.09840 [56].

2 Background

We define a propositional STRIPS Planning problem as a 4-tuple $[P, A, I, G]$ 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. Each action $a \in A$ is a 4-tuple $[\text{PRE}(a), \text{ADD}(a), \text{DEL}(a), \text{C}(a)]$ where $\text{C}(a) \in \mathbb{Z}^{0+}$ is a cost, $\text{PRE}(a) \subseteq P$ is a precondition and $\text{ADD}(a), \text{DEL}(a) \subseteq P$ are the add-effects and delete-effects. A state $s \subseteq P$ is a set of true propositions (all of $P \setminus s$ is false), an action a is *applicable* when $s \supseteq \text{PRE}(a)$ (read: s *satisfies* $\text{PRE}(a)$), and applying action a to s yields a new successor state $a(s) = (s \setminus \text{DEL}(a)) \cup \text{ADD}(a)$.

The task of classical planning is to find a sequence of actions called a *plan* $(a_1, \cdots, a_n)$ where, for $1 \leq t \leq n$, $s_0 = I$, $s_t \supseteq \text{PRE}(a_{t+1})$, $s_{t+1} = a_{t+1}(s_t)$, and $s_n \supseteq G$. A plan is *optimal* if there is no plan with lower cost $\sum_t \text{C}(a_t)$. A plan is otherwise called *satisficing*. In this paper, we assume unit-cost: $\forall a \in A; \text{C}(a) = 1$.

A domain-independent heuristic function h in classical planning is a function of a state s and the problem $[P, A, I, G]$, but the notation $h(s)$ usually omits the latter. It returns an estimate of the cumulative cost from s to one of the goal states (which satisfy G), typically through a symbolic, non-statistical means including problem relaxation and abstraction. Notable state-of-the-art functions that appear in this paper include $h^{\text{FF}}, h^{\max}, h^{\text{add}}$, and h^{GC} [27, 8, 18]. Their implementation details are beyond the scope of this paper, and are included in the appendix [56, Sec. S1].

2.1 Multi-Armed Bandit (MAB)

MAB [53, 48, 9] is a problem of finding the best strategy to choose from multiple unknown reward distributions. It is typically depicted by a row of K slot machines each with a lever or "arm." Each time the player plays one of the machines and pulls an arm (a *trial*), the player receives a reward sampled from the distribution assigned to that arm. Through multiple trials, the player discovers the arms' distributions and selects arms to maximize the reward.

The most common optimization objective of MAB is *Cumulative Regret* (CR) minimization. Let r_i $(1 \leq i \leq K)$ be a random variable (RV) for the reward that we would receive when we pull arm i. We call $p(\text{r}_i)$ an unknown *reward distribution* of i. Let t_i be a RV of the number of trials performed on arm i and $T = \sum_i \text{t}_i$ be the total number of trials across all arms.

Definition 1. *The* cumulative regret Δ *is the gap between the optimal and the actual expected cumulative reward:* $\Delta = T \max_i \mathbb{E}[\text{r}_i] - \sum_i \mathbb{E}[\text{t}_i]\mathbb{E}[\text{r}_i]$.

Algorithms whose regret per trial Δ/T converges to 0 with $T \to \infty$ are called *zero-regret*. Those with a logarithmically upper-bounded regret, $O(\log T)$, are also called *asymptotically optimal* because this is the theoretical optimum achievable by any algorithm [36]. Regret bounds tell the speed of convergence, thus its proof is stronger than that of the convergence proof.

Upper Confidence Bound 1 [6, UCB1] is a logarithmic CR MAB for rewards $\text{r}_i \in [0, c]$ with a known c. Let $r_{i1} \ldots r_{it_i} \sim p(\text{r}_i)$ be t_i i.i.d. samples obtained from an arm i. Let $\hat{\mu}_i = \frac{1}{t_i} \sum_{j=1}^{t_i} r_{ij}$. To minimize CR, UCB1 selects i with the largest Upper Confidence Bound defined below.

$$\begin{aligned} \text{UCB1}_i &= \hat{\mu}_i + c\sqrt{2 \log T / t_i} \\ \text{LCB1}_i &= \hat{\mu}_i - c\sqrt{2 \log T / t_i} \end{aligned} \tag{1}$$

For reward (cost) minimization, LCB1 instead selects i with the smallest Lower Confidence Bound defined above (e.g., in Kishimoto et al. [33]), but we may use the terms U/LCB1 interchangeably. UCB1's second term is often called an *exploration term*. Generally, an LCB is obtained by flipping the sign of the exploration term in a UCB. U/LCB1 refers to a specific algorithm while U/LCB refers to general confidence bounds. c is sometimes set heuristically as a hyperparameter called the *exploration rate*.

2.2 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 the open list, (2) (*expansion*) generate its successor nodes, (3) (*evaluation*) evaluate the successor nodes, and (4) (*queueing*) reinsert them into the open list. 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 the open list 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 search uses $f_{\text{Dijkstra}}(n) = g(n)$ (g-value), the minimum cost from the initial state I to the state s_n found so far. A* 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). GBFS uses $f_{\text{GBFS}}(n) = h(s_n)$. Forward best-first search that uses h is called forward *heuristic* best-first search. Dijkstra search is a special case of A* with $h(s) = 0$.

Typically, an open list is implemented as a priority queue ordered by NEC. Since the NEC can be stateful, e.g., $g(s_n)$ can update its value, a priority queue-based open list assumes monotonic updates to the NEC because it has an unfavorable time complexity for removals. A*, Dijkstra, and GBFS satisfy this condition because $g(n)$ decreases monotonically and $h(s_n)$ is constant.

MCTS is a class of forward heuristic best-first search that represents the open list as the leaves of a tree. We call the tree a *tree-based open list*. Our MCTS is based on the description in [30, 50]. 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 [21]. In recent work, the default policy is replaced by a learned policy [52].

Trial-based Heuristic Tree Search [30, 50, 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 reorder nodes efficiently under non-monotonic 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 [34, UCT] is a MCTS that uses UCB1 for action selection and became widely popular in adversarial games. Schulte and Keller [50] proposed several variants of UCT including GreedyUCT (GUCT), UCT*, and GreedyUCT* (GUCT*). We often abbreviate a set of algorithms to save space, e.g., [G]UCT[*] denotes $\{\mathrm{UCT}, \mathrm{GUCT}, \mathrm{UCT}^*, \mathrm{GUCT}^*\}$. In this paper, we mainly discuss GUCT[*] due to our focus on the agile satisficing setting that does not prioritize minimization of solution cost.

2.3 Base MCTS for Graph Search

Alg. 1 shows the pseudocode of MCTS adjusted for graph search, taken from [50]. Aside from what was described from the main section, it has a node-locking mechanism that avoids redundant effort.

Following THTS, our MCTS has a hash table that implements a *CLOSE* list and a *Transposition Table* (TT). A CLOSE list stores the generated states and avoids instantiating nodes with duplicate states. A TT stores various information about the states such as the parent information and the action used at the parent. The close list is implemented by a lock mechanism.

Since an efficient graph search algorithm must avoid visiting the same state multiple times, MCTS for graph search marks certain nodes as *locked*, and excludes them from the selection candidates. 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, (3) when all of its children are locked, or (4) when a node is a goal (relevant in an anytime iterated search setting [45, 46], but not in this paper). Thus, in the expansion step, when a generated node n has the same state as a node n' already in the search tree, MCTS discards n if $g(n) > g(n')$, else moves the subtree of n' to n and marks n' as locked. It also implicitly detects a cycle, as this is identical to the duplicate detection in Dijkstra/A*/GBFS.

The queueing step backpropagates necessary information from the leaf to the root. Efficient backpropagation uses a priority queue ordered by descending g-value. The queue is initialized with the expanded node p; each newly generated node n that is not discarded is inserted into the queue, and if a node n' for the same state was already present in the tree it is also inserted into the queue. In each backpropagation iteration, (1) the enqueued node with the highest g-value is popped, (2) its information is updated by aggregating its children's information (including the lock status), (3) and its parent is queued.

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

```
while True do
  Parent p ← r
  while not leaf p do      # Selection
    p ← arg min_{n∈S(p)} f(n)
  Q ← {p}
  for n ∈ S(p) do      # Expansion
    return n if n is goal.      # Early goal detection
    if ∃n' already in tree with same state s_{n'} = s_n then
      if g(n) > g(n') then
        continue
      Lock n', S(n) ← S(n'), Q ← Q ∪ {n, n'}
    else
      Compute h(s_n)      # Evaluation
      Q ← Q ∪ {n}
  while n ← Q.POPMAX() do      # Backpropagation
    Update n's statistics and lock status
    Q ← Q ∪ {n's parent}
```

3 Existing MCTS-based Classical Planning

We revisit GBFS implemented as THTS/MCTS from a MAB perspective. Let $S(n)$ be the set of successors of a node n, $L(n)$ be the set of leaf nodes in the subtree under n, and the NECs of GBFS as $f_{\mathrm{GBFS}}(n) = h_{\mathrm{GBFS}}(n)$. We expand the definition of the backup functions presented by Keller and Helmert [30] recursively down to the leaves, assuming $h_{\mathrm{GBFS}}(n) = h(s_n)$ if n is a leaf where h is a heuristic.

$$\begin{aligned} h_{\mathrm{GBFS}}(n) &= \min_{n' \in S(n)} [h_{\mathrm{GBFS}}(n')] \\ &= \min_{n' \in S(n)} [\min_{n'' \in S(n')} [h_{\mathrm{GBFS}}(n'')]] \\ &= \ldots = \min_{n' \in L(n)} [h(s_{n'})]. \end{aligned}$$

Keller and Helmert [30] called the min operator a *Full-Bellman* backup and compared it with *Monte-Carlo* backup in GUCT that uses the average, as expanded to the leaves below as well:

$$\begin{aligned} h_{\mathrm{GUCT}}(n) &= \tfrac{1}{|L(n)|} \textstyle\sum_{n' \in S(n)} |L(n')| h_{\mathrm{GUCT}}(n') \\ &= \tfrac{1}{|L(n)|} \textstyle\sum_{n' \in S(n)} \frac{\cancel{|L(n')|}}{\cancel{|L(n')|}} \sum_{n'' \in S(n')} |L(n'')| h_{\mathrm{GUCT}}(n'') \\ &= \ldots = \tfrac{1}{|L(n)|} \textstyle\sum_{n' \in L(n)} h(s_{n'}). \end{aligned}$$

To search, GUCT subtracts an exploration term from $h_{\mathrm{GUCT}}(n)$ based on LCB1, where p is a parent of n. $|L(p)|$ and $|L(n)|$ respectively correspond to T and t_i in Eq. 1.

$$f_{\mathrm{GUCT}}(n) = h_{\mathrm{GUCT}}(n) - c\sqrt{(2 \log |L(p)|)/|L(n)|}$$

While Keller and Helmert [30] managed to generalize various algorithms focusing on the procedural aspects (e.g., recursive backup from the children), we focus on its mathematical meaning. One key observation missing in Keller and Helmert [30] and is made clear by these expansions is that the set $L(n)$ of n's leaves is a *dataset*, the heuristic $h(n')$ at each leaf n' is a *reward sample*, and the NECs estimate its *statistic* such as the mean and the minimum. (The minimum is known as an *order statistic*; other order statistics include the top-k element, the q-quantile, and the median = 0.5-quantile.) Backpropagation from the expanded leaves to the root one step at a time is merely an efficient implementation detail that avoids computing the statistic (min,max,mean) over all leaves every time. Understanding

each $h(n)$ is a sample of a random variable representing a reward for MABs, we can focus on the theoretical efficiency guarantees and see how existing MCTS/THTS for classical planning fail to leverage it.

First, UCB1 assumes that all reward random variables, each associated with an arm, have a *shared, known bounds*, where each arm corresponds to each successor node during action selections. Heuristic values in classical planning lack such *a priori* known bounds, unlike adversarial games whose rewards are either +1/0 or +1/-1 representing a win/loss. Also, usually the range of heuristic values in each subtree of the search tree substantially differ from each other.

Although Schulte and Keller [50] claimed to have addressed this issue by modifying the UCB1, but their modification does not fully address the issue. Let us call their variant *GUCT-01*. It normalizes the first term of the NEC to $[0, 1]$ by taking the minimum and maximum among n's siblings sharing the parent p. Given $M = \max_{n' \in S(p)} h_{\mathrm{GUCT}}(n')$, $m = \min_{n' \in S(p)} h_{\mathrm{GUCT}}(n')$, and a hyperparameter c, GUCT-01 modifies f_{GUCT} into $f_{\text{GUCT-01}}$ (Eq. 2).

$$f_{\text{GUCT-01}}(n) = \frac{h_{\mathrm{GUCT}}(n) - m}{M - m} - c\sqrt{\frac{2\log|L(p)|}{|L(n)|}} \tag{2}$$

However, the node ordering by the GUCT-01's NEC is same when all arms are shifted and scaled by the same amount, thus GUCT-01 is identical to the standard UCB1 with a reward range $[0, c(M - m)]$ (Eq. 3); we additionally note that this version avoids a division-by-zero issue for $M - m = 0$.

$$\begin{aligned} &m + (M - m) f_{\text{GUCT-01}}(n) \\ &\quad = h_{\mathrm{GUCT}}(n) - c(M - m)\sqrt{\frac{2\log|L(p)|}{|L(n)|}} \end{aligned} \tag{3}$$

Here are two issues of GUCT-01: First, GUCT-01 does not address the fact that different subtrees have different ranges of heuristic values: When selecting an action, it assumes that all children have the same reward range $[0, c(M - m)]$. Although $M - m$ differs among parents, and thus it adjusts its exploration rate in each action selection at a different depth of the tree, it does not do so for each child, thus it is depth-aware but not breadth-aware. Second, we expect GUCT-01 to explore excessively, because the range $[0, c(M - n)]$ obtained from the data of the entire subtree of the parent is always broader than that of each child, since the parent's data is a union of those from all children.

Further, in an attempt to improve the performance of [G]UCT, Schulte and Keller [50] noted that using the average is "rather odd" for planning, and proposed UCT* and GreedyUCT* (GUCT*) which combines Full-Bellman backup with LCB1 without statistical justification.

Finally, these variants failed to improve over traditional algorithms (e.g., GBFS) unless combined with various other enhancements such as deferred heuristic evaluation (DE) and preferred operators (PO). The theoretical characteristics of these enhancements are not well understood, rendering their use ad hoc and the reason for GUCT-01's performance inconclusive, and motivating a better theoretical analysis.

4 Bandit for Unbounded Distributions

To handle reward distributions with unknown supports that differ across arms, we need a MAB that assumes an unbounded reward distribution spanning the real numbers. We use the Gaussian distribution here, although future work may consider other distributions. Formally, we assume each arm i has a reward distribution $\mathcal{N}(\mu_i, \sigma_i^2)$ for some unknown μ_i, σ_i^2. As σ_i^2 differs across i, the reward uncertainty differs across the arms. By contrast, the reward uncertainty of each arm in UCB1 is expressed by the range $[0, c]$, which is the same across the arms. We now discuss the shortcomings of MABs from previous work (Eq. 4-6), and present our new MAB (Eq. 7).

$$\text{UCB1-Normal}_i = \hat{\mu}_i + \hat{\sigma}_i\sqrt{(16\log T)/t_i} \tag{4}$$

$$\text{UCB1-Tuned}_1 = \hat{\mu}_i + c\sqrt{\min(1/4, \hat{\sigma}_i^2 + \sqrt{2\log T/t_i})\log T/t_i} \tag{5}$$

$$\text{UCB-V}_i = \hat{\mu}_i + \hat{\sigma}_i\sqrt{(2\log T)/t_i} + (3c\log T)/t_i \tag{6}$$

$$\text{UCB1-Normal2}_i = \hat{\mu}_i + \hat{\sigma}_i\sqrt{2\log T} \tag{7}$$

The UCB1-Normal MAB [6, Theorem 4], which was proposed along with UCB1 (idem, Theorem 1), is designed exactly for this scenario but is still unpopular. Given t_i i.i.d. samples $r_{i1} \dots r_{it_i} \sim \mathcal{N}(\mu_i, \sigma_i^2)$ from each arm i where $T = \sum_i t_i$, it chooses i that maximizes the metric shown in Eq. 4. To apply this bandit to MCTS, substitute $T = |L(p)|$ and $t_i = |L(n)|$, and backpropagate the statistics $\hat{\mu}_i, \hat{\sigma}_i^2$ (see the appendix [56, Sec. S4]). For minimization tasks such as classical planning, use the LCB. We refer to the GUCT variant using UCB1-Normal as *GUCT-Normal*. An advantage of UCB1-Normal is its logarithmic upper bound on regret [6, Appendix B]. However, it did not perform well in our empirical evaluation, likely because its proof relies on two conjectures which are explicitly stated by the authors as not guaranteed to hold.

Theorem 1 (From [6])**.** *UCB1-Normal has a logarithmic regret-per-arm* $256\frac{\sigma_i^2 \log T}{\Delta_i^2} + 1 + \frac{\pi^2}{2} + 8\log T$ ***if****, for a Student's t RV* X *with* s *degrees of freedom (DOF),* $\forall a \in [0, \sqrt{2(s+1)}]; P(X \geq a) \leq e^{-a^2/4}$*, and* ***if****, for a* χ^2 *RV* X *with* s *DOF,* $P(X \geq 4s) \leq e^{-(s+1)/2}$*.*

To avoid relying on these two conjectures, we need an alternate MAB that similarly adjusts the exploration rate based on the variance. Candidates include UCB1-Tuned [6] in Eq. 5, and UCB-V [4] in Eq. 6, but they all have various limitations. UCB1-Tuned assumes a bounded reward distribution, lacks a regret bound, and is outperformed by UCB-V. UCB-V improves UCB1-Tuned with a regret proof but it still assumes a bounded reward distribution.

We present UCB1-Normal2 (Eq. 7) and analyze its regret bound (*which is one of our main contributions*). To understand its behavior, see Fig. 1 which shows a MCTS selecting an action at a state S (which is equivalent to selecting a node from the open list, see Sec. 2.2). S has two successors A and B. The subtrees of A and B contain h-values $\{9, 2, 6, 3\}$ and $\{3, 7, 4, 6\}$, therefore their backpropagated statistics are $\mathcal{N}(\mu, \sigma) = \mathcal{N}(5, 3.16)$ and $\mathcal{N}(5, 1.83)$. A has a larger σ, thus the algorithm expects a higher chance of finding a lower h-value under A, as shown in the green plot. Although there is also a higher chance of finding a bad node (high h) under A, it avoids such branches during further recursions (e.g., prefer D over C). In other words, when the means are equal, it avoids low variances which is likely to contain mediocre heuristic values $h \approx 5$ all the time. This mechanism generalizes the concept of escaping heuristic plateaus during search [10]: The low σ of B indicates that its subtree contains a set of nodes with different but similar values. A heuristic plateau is a special case of $\sigma = 0$. In contrast, GUCT does not use σ and prefers A and B equally (both nodes have LCB $= \mu - c\sqrt{\frac{2\log T}{n_i}} = 5 - c\sqrt{\frac{2\log 8}{4}}$).

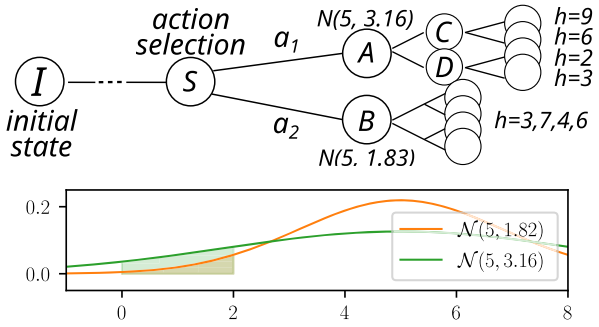

Figure 1. Search behavior. Larger σ assigns more probability on the lower h values we get in the next expansion.

Theorem 2 (Main Result)**.** *Let $\alpha \in [0, 1]$ be an unknown problem-dependent constant and $\chi^2_{1-\alpha,n}$ be the critical value for the tail probability of a χ^2 distribution with significance α and DOF n that satisfies $P(t_i\hat{\sigma}_i^2/\sigma_i^2 < \chi^2_{1-\alpha,t_i}) = \alpha$. UCB1-Normal2 has a following worst-case polynomial, best-case constant regret-per-arm where C is a finite constant if each arm is pulled at least $M = \inf\{n|8 < \chi^2_{1-\alpha,n}\}$ times.*

$$\frac{-4(\log\alpha)\sigma_i^2 \log T}{\Delta_i^2} + 1 + 2C + \frac{(1-\alpha)T(T+1)(2T+1)}{3} \xrightarrow{\alpha\to 1} 1 + 2C$$

Proof. (Sketch of the appendix [56, Sec. S3].) We use Hoeffding's inequality for sub-Gaussian distributions as Gaussian distributions belong to sub-Gaussian distributions. Unlike in UCB1 where the rewards have a fixed known support $[0, c]$, we do not know the true reward variance σ_i^2. Therefore, we use the mathematical truth *(not a conjecture)* that, when $r_{i1} \dots r_{it_i} \sim \mathcal{N}(\mu_i, \sigma_i^2)$ and ${\hat{\sigma}_i}^2 = \frac{1}{t_i}\sum_{j=1}^{t_i} r_{ij}^2 - (\frac{1}{t_i}\sum_{j=1}^{t_i} r_{ij})^2$, then $t_i{\hat{\sigma}_i}^2/\sigma_i^2$ follows a χ^2 distribution and $P(t_i{\hat{\sigma}_i}^2/\sigma_i^2 < \chi^2_{1-\alpha,t_i}) = \alpha$ for some α. We use union-bound to address the correlation and further upper-bound the tail probability. We also use $\chi^2_{1-\alpha,t_i} \geq \chi^2_{1-\alpha,2} = -2\log\alpha$ for $t_i \geq 2$. The resulting upper bound contains an infinite series C. Its convergence condition dictates the minimum pulls M that must be performed initially. □

Polynomial regrets are generally worse than logarithmic regrets of UCB1-Normal. However, UCB1-Normal relies on unproven conjectures and our experimental results shows UCB1-Normal is outperformed by UCB1-Normal2, suggesting that these conjectures do not hold. Our regret bound also improves over that of UCB1-Normal if T is small and $\alpha \approx 1$ ($\log\alpha \approx 0$ therefore $1 - \alpha \approx 0$). α represents the accuracy of the sample variance $\hat{\sigma}^2$ toward the true variance σ^2. In deterministic, discrete, finite state-space search problems like classical planning, α tends to be close to (or sometimes even match) 1 because $\sigma = \hat{\sigma}$ is achievable. Several factors of classical planning contribute to this. Heuristic functions in classical planning are deterministic, unlike rollout-based heuristics in adversarial games. This means $\sigma = \hat{\sigma} = 0$ when a subtree is linear due to the graph shape. Also, $\sigma = \hat{\sigma}$ when all reachable states from a node are exhaustively enumerated in its subtree. In statistical terms, this is because draws from heuristic samples are performed without replacements due to duplication checking in search algorithms.

Unlike UCB1-Normal, which pulls arms uniformly until all arms satisfy $t_i \geq \lfloor 8\log T \rfloor$, UCB1-Normal2 does not need such initialization pulls because every node is evaluated once and its heuristic value is used as a single sample. This means we assume $M = 1$, thus $\alpha > \mathrm{ERF}(2) > 0.995$ because $8 < \chi^2_{1-\alpha,1} \Leftrightarrow 1 - \alpha < \frac{\gamma(\frac{1}{2},\frac{8}{2})}{\Gamma(\frac{1}{2})} = 1 - \mathrm{ERF}(2)$. In classical planning, $\alpha > 0.995$ is more realistic than the conjectures used by UCB1-Normal.

Another explanation for the failure of UCB1-Normal is that its exploration term is made too confident / too small by $1/\sqrt{t_i}$ because it was derived with more assumptions (the χ^2 conjecture).

During the discussion, yet another potential explanation related to *the aleatoric and epistemic uncertainty* was suggested by one of the reviewers. The rewards in the typical bandit problems are "truly" stochastic, i.e., each arm always samples a different reward from the unknown fixed distribution, thus the uncertainty is aleatoric / an objective truth. However all rewards in classical planning are deterministic, and the uncertainty comes purely from the sampling (search behavior in the subtree), thus the uncertainty is epistemic / subjective to the agent. Under this interpretation, the standard regret analysis above (yielding polynomial regret) may tell little about the actual performance of the algorithms, potentially suggesting a new challenge for the bandit community.

5 Experimental Evaluation

We evaluated the algorithms over a subset of the International Planning Competition benchmark domains,[2] selected for compatibility with the set of PDDL extensions supported by Pyperplan [1]. We maintain the superset of the results of the experiments under a 10,000 node evaluation limit, a 4,000 node expansion limit, and a 300 second runtime limit, and then count the number of instances solved under each limit. We mainly focus on the node evaluations because heuristic computation is the main bottleneck in classical planning. See the appendix [56, Fig. S1-S3] for the results controlled by expansions and the runtime. Another reason for this focus is the fact that we used a python-based implementation (Pyperplan) for convenient prototyping. It is slower than C++-based state-of-the-art systems (e.g. Fast Downward [23]), but our focus on evaluations makes this irrelevant and also improves reproducibility by avoiding the effect of hardware differences and low-level implementation details.

In order to limit the length of the experiment, we also removed the problem instances which Pyperplan took more than 5 minutes and 2GB memory to parse and instantiate the input file. The instantiation limit removed 47 instances from freecell, 4 from logistics98, 2 from openstacks, and 24 from pipesworld-tankage. This resulted in 772 problem instances across 24 domains in total. We evaluated various algorithms with h^{FF}, h^{add}, $h^{\max}$, and h^{GC} (goal count) heuristics [18], and our analysis focuses on h^{FF}. We included h^{GC} because it can be used in environments without domain descriptions, e.g., in the planning-based approach [38] to the Atari environment [7]. We ran each configuration with 5 random seeds and report the average number of problem instances solved. To see the spread due to the seeds, see the cumulative histogram plots in the appendix [56, Fig. S1-S3].

We evaluated the following algorithms: **GBFS** is GBFS implemented in Pyperplan and FastDownward. We evaluated both implementations in order to compare the difference. **WA*** ($w = 5$) based on FastDownward is added because it outperformed GBFS in [50] in agile setting. **GUCT** is a GUCT based on the original UCB1. **GUCT-01** is GUCT with ad hoc $[0, 1]$ normalization of the mean [50]. **GUCT-Normal/-Normal2/-V** are GUCT variants using UCB1-Normal/UCB1-Normal2/UCB-V respectively. The * variants **GUCT*/-01/-Normal/-Normal2** are using full-bellman backup. For

[2] github.com/aibasel/downward-benchmarks

Table 1. The number of problem instances solved with less than 10,000 node evaluations; best configurations in **bold** for each heuristic; each number represents an average over 5 trials. We show results for both $c = 1.0$ and $c = 0.5$ ("best parameter" according to Schulte and Keller [50]) when the algorithm requires one. Algorithms in the bottom half have no hyperparameter. †: Data missing due to the lack of support of PO for GBFS in Pyperplan. ‡: Data missing because DE in Fast Downward measures node evaluations differently.

PO:Preferred Operators, DE:Deferred Evaluation.

$h =$	h^{FF}		h^{add}		$h^{\max}$		h^{GC}		h^{FF}+PO		h^{FF}+DE		h^{FF}+DE+PO	
$c =$	0.5	1	0.5	1	0.5	1	0.5	1	0.5	1	0.5	1	0.5	1
GUCT	442.8	412.0	435.8	397.8	237.0	228.4	306.6	285.2	484.6	454.0	455.8	389.2	497.4	439.4
*	542.0	458.6	529.2	480.8	248.4	242.2	317.8	310.4	591.6	495.8	480.2	423.6	527.4	471.0
-01	399.8	368.0	375.4	328.8	256.8	237.4	318.4	302.4	441.4	408.4	387.6	361.2	445.0	422.6
*-01	425.6	388.0	404.8	364.4	246.8	233.4	318.0	297.6	470.6	420.6	409.4	378.2	466.8	438.8
-V	361.2	317.4	354.0	310.6	226.2	208.6	278.4	255.4	427.0	389.6	370.8	344.6	431.2	421.0
-Normal	-	283.4	-	265.0	-	212.0	-	233.4	-	372.4	-	289.0	-	381.6
*-Normal	-	318.8	-	300.0	-	215.2	-	246.2	-	378.1	-	304.4	-	386.7
-Normal2	-	**581.8**	-	535.8	-	**316.6**	-	**379.0**	-	**621.0**	-	**518.0**	-	**578.0**
*-Normal2	-	567.2	-	533.8	-	263.0	-	341.0	-	618.0	-	511.4	-	567.8
TTTS-Normal	-	181.0	-	180.0	-	171.4	-	170.8	-	151.0	-	180.6	-	150.6
TTTS-Normal*	-	189.4	-	186.4	-	177.4	-	174.4	-	159.4	-	185.8	-	155.8
GBFS(Pyperplan/FastDownward)	538/539		-	518/517	-	224/226	-	354/349	-	†/539	-	489/‡	-	†/‡
WA* ($w = 5$) (FastDownward)	528		-	522	-	211	-	319	-	528	-	‡	-	‡
Softmin-Type(h) (FastDownward)	576.0		-	**542.6**	-	297.2	-	357.6	-	575.8	-	‡	-	‡

GUCT and GUCT-01, we evaluated the hyperparameter c with the standard value $c = 1.0$ and $c = 0.5$. The choice of the latter is due to Schulte and Keller [50], who claimed that GUCT [*]-01 performed the best when $0.6 < C = c\sqrt{2} < 0.9$, i.e., $0.4 < c < 0.63$. Our aim of testing these hyperparameters is to compare them against automatic exploration rate adjustments performed by UCB1-Normal[2]. Other algorithms are explained later.

Schulte and Keller [50] previously reported that two ad hoc enhancements to GBFS, PO and DE, also improve the performance of GUCT [*]-01. We evaluated our equivalent reimplementation. We did not evaluate PO with heuristics other than h^{FF}, which are not supported by Pyperplan.

In all comparisons between GUCT-Normal2 and other algorithms based on FF heuristics below, we performed Welch's unequal variances t-test on the coverage scores with 5 different random seeds, and confirmed $p < 0.001$ for all comparisons. Note that all algorithms evaluated in this paper are deterministic up to tie-breaking, including MCTS variants: With the same set of leafs, all NECs are deterministic, and thus the action selection is deterministic.

Detailed Ablation We first reproduced [50] and provide its more detailed ablation. Table 1 shows that GUCT [*]-01 is indeed significantly outperformed by the baseline algorithm GBFS, indicating that UCB1-based exploration is not beneficial for planning. Although this result disagrees with the final conclusion of their paper, their conclusion relied on incorporating the DE and PO enhancements, and these confounding factors impede conclusive analysis.

We also tested GUCT [*], which lacks the mean normalization (Eq. 2) of GUCT [*]-01, which was not previously evaluated. GUCT [*]-01 performs significantly worse, indicating that its normalization not only fails to address the unknown and different supports, but also harms the performance by excessive exploration, as predicted by our analysis in Sec. 3.

GUCT-Normal2 We then compared various algorithms. GUCT-Normal2 outperformed GBFS, GUCT/-01/-Normal/-V, and their * variants. The dominance against GUCT-Normal supports our analysis that in classical planning $\hat{\sigma}^2 \approx \sigma^2$, thus $P(t_i\hat{\sigma}^2/\sigma^2 < \chi^2_{1-\alpha,t_i}) = \alpha \approx 1$, overcoming the asymptotic deficit (the polynomial regret in GUCT-Normal2 vs. the logarithmic regret of GUCT-Normal). In other words, the logarithmic regret of UCB1-Normal does not hold in classical planning because the χ^2 conjectures tend to be violated.

While the * variants can be significantly better than the non-* variants at times, this trend was opposite in algorithms that perform better, e.g., GUCT*-Normal2 tend to be worse than GUCT-Normal2. This supports our claim that Full-Bellman backup proposed by [50] is theoretically unfounded and thus does not consistently improve search algorithms. Further theoretical investigation of Full-Bellman backup is an important avenue of future work.

The table also compares GUCT [*]-Normal[2], which do not require any hyperparameter, against GUCT [*][/-01/-V] with different c values. Although $c = 0.5$ improves the performance of GUCT [*]-01 as reported by Schulte and Keller [50], it did not improve enough to catch up with the adaptive exploration rate of GUCT [*]-Normal2. We also tested $c \in \{0.1, 0.3, 3.0\}$ (see the appendix [56, Sec. S6.3]). Results indicated that $c = 0.1$ tends to be better, but it still did not outperform GUCT-Normal2 (e.g., the best coverage among GUCT [*][-01] with FF/$c = 0.1$ was 561.8 by GUCT*, compared to 581.8 by GUCT-Normal2.) This is not surprising, as the limit of $c \to 0$ for GUCT* is GBFS, which also performs well (538).

Simple Regret *Cumulative regret* (CR, Eq. 1) bandits maximize the total rewards, *including those obtained during the experimentation*, while *simple regret* (SR) bandits / *best-arm identification* algorithms [5, 29] maximize the expected rewards of the *incumbent best arm* that is maintained during the experimentation and is returned at the end. SR MABs can explore more aggressively than CR MABs because the cost of the experimentation is free for SR. Feldman and Domshlak [14] showed that SR MABs are superior in *online* MDPs where the incumbent corresponds to the action to take next. The concept of incumbent also exists in anytime search [45], e.g., in LAMA [46], suggesting an interesting future direction. However, the incumbent does not exist in agile search, or in the first iteration of anytime search, so SR MABs are conceptually mismatched with offline agile search. Table 1 shows the performance of MCTS combined with a state-of-the-art SR MAB [49, TTTS]. TTTS-Normal/* respectively uses the Monte-Carlo/Full-Bellman backup. As expected, they are vastly outperformed by other algorithms. In online planning and acting, the justification for SR is that search is not a commitment, only the first action of the policy/plan is. *However, in agile planning, search is indeed a commitment to the computational resources (time, memory) which we minimize,* thus it justifies the CR objective.

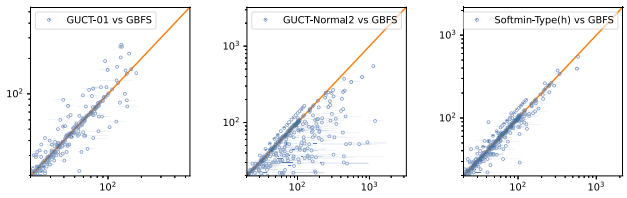

Figure 2. Comparing solution length of GUCT [-01/Normal2] and Softmin-Type(h) (x-axis) against GBFS (y-axis) using h^{FF}.

Preferred Operators In addition to the heuristic value of a state, some heuristic functions are able to return a list of actions called "helpful actions" [27] or "preferred operators" [44]. We reimplemented Schulte and Keller [50]'s strategy which limits the action selection to the preferred operators and falling back to the normal behavior if there are none. Table 1 shows that it also improves GUCT [*][-Normal2], consistent with the previous report on GUCT [*]-01.

Deferred Evaluation Table 1 shows the effect of deferred heuristic evaluation (DE) on search algorithms. *In this experiment,* DE should perform worse than eager evaluations because DE trades the number of calls to heuristics with the number of nodes inserted to the tree, which is limited to 10,000. When CPU time is the limiting resource, DE usually solves more instances, assuming the implementation is optimized for speed (e.g., using C++). However, our Python implementation (typically 100–1,000 times slower than C++) is not able to measure this effect because this low-level bottleneck could hide the effect of speed improvements. What we could learn from this experiment is therefore whether DE+PO is better than DE, and if GUCT [*]-Normal2+DE continues to dominate other algorithms with DE. Table 1 answers both questions positvely: DE+PO tends to perform better than DE alone, and GUCT [*]-Normal2 is still superior to other algorithms with DE and DE+PO.

Diversified Search We evaluated Softmin-Type(h) [35], a recent state-of-the-art diversified search algorithm for classical planning. We used the original C++ implementation based on Fast Downward. We excluded diversification methods that use state information, such as BFWS [37], as they are orthogonal concepts. Table 1 shows that UCB1-Normal2 outperforms Softmin-Type(h) with $h^{\mathrm{FF}}, h^{\max}, h^{\mathrm{GC}}, h^{\mathrm{FF}}$+PO. See the domain-wise comparisons in the appendix [56, Sec. S6.5].

Solution Quality Fig. 2 shows that GUCT-Normal2 and Softmin-Type(h) return longer solutions than GBFS does. GUCT-01 finds solutions with highly varying length, but overall they are not consistently longer or shorter than GBFS. See the appendix [56, Fig. S7-S10] for more plots. For agile search, we believe that a successful exploration must sacrifice the solution quality for faster search.

Runtime Comparison To assess the impact of the runtime overhead required by GUCT-Normal2 to maintain MCTS search tree, we reimplemented GUCT/-Normal/2 on Fast Downward and evaluated them on IPC 2018 satisficing instances. Table 2 shows that GUCT-Normal2 outperforms other algorithms.

6 Related Work

The idea of using variances to guide the search has been proposed as early as Crazy Stone for computer Go [12]. However, due to its focus on adversarial games, MCTS literature typically assumes a bounded reward setting (e.g., 0/1, -1/+1), making applications of UCB1-Normal scarce (e.g., Google Scholar returns 5900 vs. 60 for keyword "UCB1" and "UCB1-Normal", respectively) except a few model-selection applications [39]. While Gaussian Process MAB [42] has been used with MCTS for sequential decision making in continuous space search and robotics [31], it is significantly different from discrete search spaces like in classical planning.

MABs may provide a rigorous theoretical tool to analyze the behavior of a variety of existing randomized enhancements for agile/satisficing search that tackle the exploration-exploitation dilemma. ϵ-greedy GBFS was indeed inspired by MABs [55, Sec.2].

GUCT-Normal2 encourages exploration in nodes further from the goal, which tend to be close to the initial state. This behavior is similar to that of Diverse Best First Search [28], which stochastically enters an "exploration mode" that expands a node with a smaller g value more often. This reverse ordering is unique from other diversified search algorithms, including ϵ-GBFS, Type-GBFS [60], and Softmin-Type-GBFS [35], which selects g rather uniformly during the exploration.

Theoretical guarantees of MABs require modifications in tree-based algorithms (e.g. MCTS) due to non-i.i.d. sampling from the subtrees [11, 40]. Incorporating the methods developed in the MAB community to counter this bias in the subtree samples is an important direction for future work.

MDP and Reinforcement Learning literature often use discounting to avoid the issue of divergent cumulative reward: when the upper bound of step-wise reward is known to be R, then the maximum cumulative reward goes to ∞ with infinite horizon, while the discounting with γ makes it below $\frac{R}{1-\gamma}$, allowing the application of UCB1. Although it addresses the numerical issue and UCB1's theoretical requirement, it no longer optimizes the cumulative objective.

Table 2. IPC 2018 results (average number of instances solved over 3 seeds) using h^{FF} under 5 min time limit and 8GB memory limit. For caldera and organic-synthesis, we used their action-splitting variants [2] provided by the organizers. "Softmin" stands for Softmin-Type(h). Best results are highlighted in bold.

domain	GBFS	WA*	Softmin	GUCT	Normal	Normal2
agricola	9.0	4.0	9.0	8.0	1.0	**9.7**
caldera	4.0	2.0	**7.3**	6.0	6.7	6.7
data-net	4.0	5.0	9.0	4.0	2.0	**9.7**
flashfill	9.0	8.0	**9.0**	1.0	0.0	6.7
nurikabe	7.0	8.0	7.0	8.0	8.0	**8.3**
org-syn	9.0	10.0	9.3	10.0	**10.3**	9.7
settlers	0.0	4.0	5.3	**6.3**	5.3	2.3
snake	5.0	3.0	5.0	3.3	3.0	**16.7**
spider	8.0	**11.0**	8.7	8.7	9.0	9.3
termes	**12.0**	4.0	**12.0**	10.0	9.7	6.0
total	67.0	59.0	81.7	65.3	55.0	**85.0**

7 Conclusion

We examined the theoretical assumptions of existing bandit-based exploration mechanisms for classical planning, and showed that ad hoc design decisions can invalidate theoretical guarantees and harm performance. We presented GUCT-Normal2, a classical planning algorithm combining MCTS and our Gaussian bandit UCB1-Normal2, and analyzed it both theoretically and empirically. Future work includes combinations with other enhancements for agile search including novelty metric [37], lazy evaluations and preferred operators [43], and iterated anytime search [45].

Acknowledgments

This work was supported through DTIC contract FA8075-18-D-0008, Task Order FA807520F0060, Task 4 - Autonomous Defensive Cyber Operations (DCO) Research & Development (R&D).

References

[1] Y. Alkhazraji, M. Frorath, M. Grützner, M. Helmert, T. Liebetraut, R. Mattmüller, M. Ortlieb, J. Seipp, T. Springenberg, P. Stahl, and J. Wülfing. Pyperplan, 2020. URL https://doi.org/10.5281/zenodo.3700819.

[2] C. Areces, F. Bustos, M. A. Dominguez, and J. Hoffmann. Optimizing Planning Domains by Automatic Action Schema Splitting. In *Proc. of ICAPS*, 2014.

[3] M. Asai and A. Fukunaga. Exploration Among and Within Plateaus in Greedy Best-First Search. In *Proc. of ICAPS*, 2017.

[4] J.-Y. Audibert, R. Munos, and C. Szepesvári. Exploration–Exploitation Tradeoff using Variance Estimates in Multi-Armed Bandits. *Theoretical Computer Science*, 2009.

[5] J.-Y. Audibert, S. Bubeck, and R. Munos. Best Arm Identification in Multi-Armed Bandits. In *Proc. of COLT*, 2010.

[6] P. Auer, N. Cesa-Bianchi, and P. Fischer. Finite-Time Analysis of the Multiarmed Bandit Problem. *Machine Learning*, 2002.

[7] M. G. Bellemare, Y. Naddaf, J. Veness, and M. Bowling. The Arcade Learning Environment: An Evaluation Platform for General Agents (Extended Abstract). In *Proc. of IJCAI*, 2015.

[8] B. Bonet and H. Geffner. Planning as Heuristic Search. *Artificial Intelligence*, 2001.

[9] R. R. Bush and F. Mosteller. A Stochastic Model with Applications to Learning. *The Annals of Mathematical Statistics*, 1953.

[10] A. Coles and A. Smith. Marvin: A Heuristic Search Planner with Online Macro-Action Learning. *J. Artif. Intell. Res.(JAIR)*, 2007.

[11] P.-A. Coquelin and R. Munos. Bandit Algorithms for Tree Search. In *Proc. of UAI*, 2007.

[12] R. Coulom. Efficient Selectivity and Backup Operators in Monte-Carlo Tree Search. In *International Conference on Computers and Games*, 2006.

[13] E. W. Dijkstra. A Note on Two Problems in Connexion with Graphs. *Numerische mathematik*, 1959.

[14] Z. Feldman and C. Domshlak. Simple Regret Optimization in Online Planning for Markov Decision Processes. *J. Artif. Intell. Res.(JAIR)*, 2014.

[15] P. Ferber, M. Helmert, and J. Hoffmann. Neural Network Heuristics for Classical Planning: A Study of Hyperparameter Space. In *Proc. of ECAI*, 2020.

[16] P. Ferber, L. Cohen, J. Seipp, and T. Keller. Learning and Exploiting Progress States in Greedy Best-First Search. In *Proc. of IJCAI*, 2022.

[17] P. Ferber, F. Geißer, F. Trevizan, M. Helmert, and J. Hoffmann. Neural Network Heuristic Functions for Classical Planning: Bootstrapping and Comparison to Other Methods. In *Proc. of ICAPS*, 2022.

[18] R. E. Fikes, P. E. Hart, and N. J. Nilsson. Learning and Executing Generalized Robot Plans. *Artificial Intelligence*, 1972.

[19] C. R. Garrett, L. P. Kaelbling, and T. Lozano-Pérez. Learning to Rank for Synthesizing Planning Heuristics. In *Proc. of IJCAI*, 2016.

[20] C. Gehring, M. Asai, R. Chitnis, T. Silver, L. P. Kaelbling, S. Sohrabi, and M. Katz. Reinforcement learning for classical planning: Viewing heuristics as dense reward generators. In *Proc. of ICAPS*, 2022.

[21] S. Gelly and D. Silver. Monte-Carlo Tree Search and Rapid Action Value Estimation in Computer Go. *Artificial Intelligence*, 2011.

[22] P. E. Hart, N. J. Nilsson, and B. Raphael. A Formal Basis for the Heuristic Determination of Minimum Cost Paths. *Systems Science and Cybernetics, IEEE Transactions on*, 1968.

[23] M. Helmert. The Fast Downward Planning System. *J. Artif. Intell. Res.(JAIR)*, 2006.

[24] M. Heusner, T. Keller, and M. Helmert. Understanding the Search Behaviour of Greedy Best-First Search. In *Proc. of SOCS*, 2017.

[25] M. Heusner, T. Keller, and M. Helmert. Best-Case and Worst-Case Behavior of Greedy Best-First Search. In *Proc. of IJCAI*, 2018.

[26] M. Heusner, T. Keller, and M. Helmert. Search Progress and Potentially Expanded States in Greedy Best-First Search. In *Proc. of IJCAI*, 2018.

[27] J. Hoffmann and B. Nebel. The FF Planning System: Fast Plan Generation through Heuristic Search. *J. Artif. Intell. Res.(JAIR)*, 2001.

[28] T. Imai and A. Kishimoto. A Novel Technique for Avoiding Plateaus of Greedy Best-First Search in Satisficing Planning. In *Proc. of AAAI*, 2011.

[29] Z. Karnin, T. Koren, and O. Somekh. Almost Optimal Exploration in Multi-Armed Bandits. In *Proc. of ICML*, 2013.

[30] T. Keller and M. Helmert. Trial-Based Heuristic Tree Search for Finite Horizon MDPs. In *Proc. of ICAPS*, 2013.

[31] B. Kim, K. Lee, S. Lim, L. Kaelbling, and T. Lozano-Pérez. Monte Carlo Tree Search in Continuous Spaces using Voronoi Optimistic Optimization with Regret Bounds. In *Proc. of AAAI*, 2020.

[32] A. Kishimoto, R. Zhou, and T. Imai. Diverse Depth-First Search in Satisificing Planning. In *Proc. of SOCS*, 2012.

[33] A. Kishimoto, D. Bouneffouf, R. Marinescu, P. Ram, A. Rawat, M. Wistuba, P. Palmes, and A. Botea. Bandit Limited Discrepancy Search and Application to Machine Learning Pipeline Optimization. In *Proc. of AAAI*, 2022.

[34] L. Kocsis and C. Szepesvári. Bandit Based Monte-Carlo Planning. In *Proc. of ECML*, 2006.

[35] R. Kuroiwa and J. C. Beck. Biased Exploration for Satisficing Heuristic Search. In *Proc. of ICAPS*, 2022.

[36] T. L. Lai, H. Robbins, et al. Asymptotically Efficient Adaptive Allocation Rules. *Advances in Applied Mathematics*, 1985.

[37] N. Lipovetzky and H. Geffner. Best-First Width Search: Exploration and Exploitation in Classical Planning . In *Proc. of AAAI*, 2017.

[38] N. Lipovetzky, M. Ramírez, and H. Geffner. Classical Planning with Simulators: Results on the Atari Video Games. In *Proc. of IJCAI*, 2015.

[39] D. McConachie and D. Berenson. Estimating Model Utility for Deformable Object Manipulation using Multiarmed Bandit Methods. *IEEE Transactions on Automation Science and Engineering*, 2018.

[40] R. Munos et al. From Bandits to Monte-Carlo Tree Search: The Optimistic Principle Applied to Optimization and Planning. *Foundations and Trends® in Machine Learning*, 2014.

[41] H. Nakhost and M. Müller. Monte-Carlo Exploration for Deterministic Planning. In *Proc. of IJCAI*, 2009.

[42] S. Niranjan, A. Krause, S. M. Kakade, and M. W. Seeger. Gaussian Process Optimization in the Bandit Setting: No Regret and Experimental Design. In *Proc. of ICML*, 2010.

[43] S. Richter and M. Helmert. Preferred Operators and Deferred Evaluation in Satisficing Planning. In *Proc. of ICAPS*, 2009.

[44] S. Richter and M. Helmert. Preferred operators and deferred evaluation in satisficing planning. In *Proc. of ICAPS*, 2009.

[45] S. Richter, J. T. Thayer, and W. Ruml. The Joy of Forgetting: Faster Anytime Search via Restarting. In *Proc. of ICAPS*, 2010.

[46] S. Richter, M. Westphal, and M. Helmert. LAMA 2008 and 2011. In *Proc. of IPC*, 2011.

[47] O. Rivlin, T. Hazan, and E. Karpas. Generalized planning with deep reinforcement learning. In *Proc. of PRL*, 2019.

[48] H. Robbins. Some Aspects of the Sequential Design of Experiments. *Bulletin of the American Mathematical Society*, 1952.

[49] D. Russo. Simple Bayesian Algorithms for Best-Arm Identification. *Operations Research*, 2020.

[50] T. Schulte and T. Keller. Balancing Exploration and Exploitation in Classical Planning. In *Proc. of SOCS*, 2014.

[51] W. Shen, F. Trevizan, and S. Thiébaux. Learning Domain-Independent Planning Heuristics with Hypergraph Networks. In *Proc. of ICAPS*, 2020.

[52] D. Silver, A. Huang, C. J. Maddison, A. Guez, L. Sifre, G. Van Den Driessche, J. Schrittwieser, I. Antonoglou, V. Panneershelvam, M. Lanctot, et al. Mastering the Game of Go with Deep Neural Networks and Tree Search. *Nature*, 2016.

[53] W. R. Thompson. On the Likelihood that One Unknown Probability Exceeds Another in View of the Evidence of Two Samples. *Biometrika*, 1933.

[54] S. Toyer, F. Trevizan, S. Thiébaux, and L. Xie. Action Schema Networks: Generalised Policies with Deep Learning. In *Proc. of AAAI*, 2018.

[55] R. A. Valenzano, J. Schaeffer, N. R. Sturtevant, and F. Xie. A Comparison of Knowledge-Based GBFS Enhancements and Knowledge-Free Exploration. In *Proc. of ICAPS*, 2014.

[56] S. Wissow and M. Asai. Scale-Adaptive Balancing of Exploration and Exploitation in Classical Planning (Full version of this paper). *arXiv preprint arXiv:2305.09840*, 2023.

[57] F. Xie, H. Nakhost, and M. Müller. Planning Via Random Walk-Driven Local Search. In *Proc. of ICAPS*, 2012.

[58] F. Xie, M. Müller, and R. C. Holte. Adding Local Exploration to Greedy Best-First Search in Satisficing Planning. In *Proc. of AAAI*, 2014.

[59] F. Xie, M. Müller, R. C. Holte, and T. Imai. Type-Based Exploration with Multiple Search Queues for Satisficing Planning. In *Proc. of AAAI*, 2014.

[60] F. Xie, M. Müller, and R. C. Holte. Understanding and Improving Local Exploration for GBFS. In *Proc. of ICAPS*, 2015.