A Bayesian Approach to Online Planning

Nir Greshler [1] **David Ben Eli** [1] **Carmel Rabinovitz** [1] **Gabi Guetta** [1] **Liran Gispan** [1] **Guy Zohar** [1] **Aviv Tamar** [2]

Abstract

The combination of Monte Carlo tree search and neural networks has revolutionized online planning. As neural network approximations are often imperfect, we ask whether uncertainty estimates about the network outputs could be used to improve planning. We develop a Bayesian planning approach that facilitates such uncertainty quantification, inspired by classical ideas from the meta-reasoning literature. We propose a Thompson sampling based algorithm for searching the tree of possible actions, for which we prove the first (to our knowledge) finite time Bayesian regret bound, and propose an efficient implementation for a restricted family of posterior distributions. In addition we propose a variant of the Bayes-UCB method applied to trees. Empirically, we demonstrate that on the ProcGen Maze and Leaper environments, when the uncertainty estimates are accurate but the neural network output is inaccurate, our Bayesian approach searches the tree much more effectively. In addition, we investigate whether popular uncertainty estimation methods are accurate enough to yield significant gains in planning. Our code is available at: https://github.com/nirgreshler/bayesian-online-planning.

1. Introduction

Online planning is fundamental to various decision making problems, ranging from game playing, such as Chess and Go (Silver et al., 2018), to robotic manipulation and navigation (Finn & Levine, 2017; Shim et al., 2003), autonomous driving (Williams et al., 2017; Cesari et al., 2017), and more

---
[1]General Motors, Advanced Technical Center, Israel [2]Department of Electrical and Computer Engineering, Technion - Israel Institute of Technology, Haifa, Israel. Correspondence to: Nir Greshler <nir.greshler@gm.com>, Aviv Tamar <avivt@technion.ac.il>.
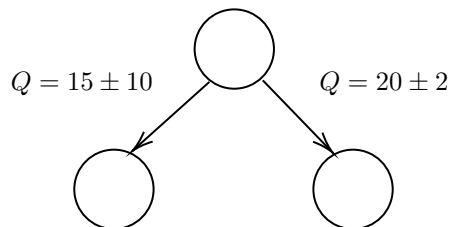
Figure 1. Example of value estimation errors during search

recently, planning with large language models (Zhang et al., 2023; Mankowitz et al., 2023). In the standard problem setting, as we consider here, a model of the world is known, its state is fully observed, and an agent must sequentially take actions that yield a high cumulative reward.

For almost all realistic problems, calculating the optimal sequence of actions is intractable, and some approximations must be made. For the past two decades, the dominant approximation approach has been Monte-Carlo Tree Search (MCTS) – a stochastic traversal of the search tree that balances exploration and exploitation using an upper confidence bound (UCB) (Kocsis & Szepesvári, 2006). Breakthrough performance in several games was achieved by the seminal AlphaZero, an extension of MCTS with neural network approximations of a value function and a policy, which considerably cut down the search effort (Silver et al., 2018).

When the online search has a limited budget, as is the case in any real-time application, errors in the neural network approximations can be problematic for MCTS. As an example, consider the situation in Figure 1, where the search has reached a state with two possible actions, with action-value estimates of 15 and 20. MCTS will choose the second, higher value action repeatedly, until the low visit count of the unexplored action in the UCB term will dominate, and only then will it explore the first action. However, if we were to know in advance that the uncertainty in the action-value estimates are ± 10 and ± 2, respectively, then we should explore the first action much more frequently, as its low-value estimate may likely be wrong. Unfortunately, MCTS, and its AlphaZero variants are frequentist methods, and do not naturally take such uncertainty information into account.[1]

---
[1]We are mainly interested in *epistemic* uncertainty here (Der Kiureghian & Ditlevsen, 2009). For *aleatoric* uncertainty, frequentist methods such as Audibert et al. (2009) are well suited.

We advocate here a Bayesian approach to online search. Our premise is that the Bayesian method can naturally exploit uncertainty estimates of the neural network approximations, to yield better performance, especially under modest search budgets and inaccurate neural network predictions.

Bayesian search algorithms were explored in the planning literature already in the 1990s (Russell & Wefald, 1991b; Dearden et al., 1998). For single stage decision making, Bayesian optimization (Shahriari et al., 2015; Frazier, 2018; Chen et al., 2018) can be seen as a modern incarnation of similar ideas, and meta reinforcement learning is essentially a Bayesian approach to multi-task RL (Zintgraf et al., 2019). The work of Tesauro et al. (2010) pioneered the Bayesian approach to MCTS. However, to our knowledge, during the recent deep learning-fueled revival of online planning (Silver et al., 2017; Anthony et al., 2017; Silver et al., 2018; Schrittwieser et al., 2020), Bayesian methods have so far been ignored. In this work, we aim to rectify this matter.

We develop both the fundamental and practical aspects of a Bayesian approach to online planning and learning. Our first contribution is a Bayesian formulation of the tree search problem, and a corresponding Thompson sampling based tree search algorithm. We establish a Bayesian regret bound for our algorithm, based on modern analysis techniques (Russo & Van Roy, 2016), which to our knowledge is the first regret analysis of a Bayesian tree search approach. Importantly, our bound shows that when the Shannon entropy of the prior is small (equivalent to high certainty in the neural net approximation), the expected regret is small. Our second contribution is a practical implementation of Thompson sampling tree search, by incorporating efficient methods for sampling from and updating the posterior. Interestingly, our methods bear resemblance to techniques suggested in previous works such as Tesauro et al. (2010); our formulation establishes them as concrete instances of the Thompson sampling method. In addition, we propose an adaptation of the Bayes-UCB method of Kaufmann et al. (2012) to tree search, which we find to work very well in practice. Finally, in the spirit of AlphaZero and Expert Iteration (Anthony et al., 2017; Silver et al., 2018), we incorporate deep learning of value functions into our approach using self play. Different from prior work, however, our Bayesian planning algorithms make explicit use of *uncertainty estimates* about the neural network predictions, and we discuss how such could be obtained.

We evaluate our method on procedurally generated Maze and Leaper environments from the ProcGen benchmark (Cobbe et al., 2020). In the setting we investigate, the agent is tested on domains it has not been trained on, and therefore we expect some errors in its neural network approximations. With access to accurate uncertainty estimates (which can easily be computed for the maze domain),

our Bayesian approaches significantly outperform MCTS, validating our main premise. However, with two popular methods for *learning* the epistemic uncertainty, we could not obtain predictions accurate enough to translate to performance gains in planning, suggesting that more research is required to fully harness the potential of the Bayesian paradigm.

2. Bayesian Online Planning

We consider an agent that sequentially interacts with a dynamic environment in discrete time steps. At each time step, the agent observes the current environment state and performs an action. Subsequently, the environment transitions to a new state according to some transition law. We assume that the agent has a model of the environment, and at each step can use the model to plan the next course of action. For the sake of planning, we assume that the environment model is deterministic. However, we note that since the agent replans at each time step, our solution can also be applied to non-deterministic systems (Yoon et al., 2007). In the following, we focus on the planning problem that needs to be solved at each time step, and propose a Bayesian approach for it.

2.1. Bayesian Tree Search

Consider a deterministic decision process $\mathcal{T}$ with an initial (root) state s_0, and a finite action set A. Let $s_{n+1} = f(s_n, a_n)$ denote the deterministic dynamics, and let $r(s_n, a_n) \in [-R_{max}, R_{max}]$ denote a *deterministic* reward for a state-action pair. We consider decision processes of depth H, that is, we wish to maximize:

$$\max_{a_0,\ldots,a_{H-1}} \sum_{n=0}^{H-1} r(s_n, a_n), \tag{1}$$
$$\text{s.t. } s_{n+1} = f(s_n, a_n), \quad \forall n \in 0, \ldots, H-1.$$

A decision process is equivalent to a tree of depth H, and henceforth we will refer to it as such. While we do not denote it explicitly, we assume that states at different levels of the tree are distinct (e.g., by having the level of the tree be part of the state). Naively, one can solve (1) by evaluating all the A^H possible H-length action sequences (e.g., using breadth first search). We will be interested in problems where A and H are such that this approach is not tractable.

Intuitively, we would like to focus the search on the more promising parts of the search tree, assuming that we have some prior knowledge about where the optimal solution may lie. In the following, we cast this idea within a formal probabilistic interpretation. Our main insight, inspired by the meta reasoning literature (Russell & Wefald, 1991b), is that each edge expansion during the tree search is equivalent to querying the rewards for the actions of a state. Thus,
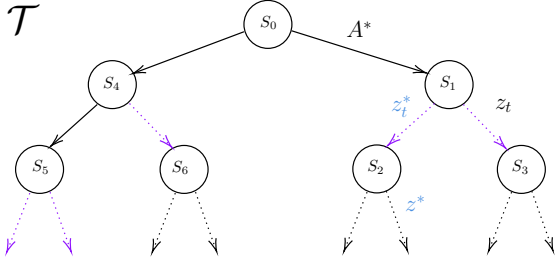
Figure 2. Illustration of the formulation in Section 2.1. A tree $\mathcal{T}$ of depth $H = 3$ is shown. Let the action $\{L, R\}$ correspond to the left and right transitions, respectively. Assume that the optimal branch is $(S_0, R) \to (S_1, L) \to (S_2, R)$. Then, $z^* = (S_2, R)$. At time $t = 4$, the state-action pairs that have already been explored are marked in solid line, and the next state-action to be explored is $z_t = (S_1, R)$. The set $\mathcal{Z}_t$ is marked in purple. Note that z^* is indicative of the optimal branch, and also of z_t^*, and of the optimal action at the root, A^*.

in terms of the number of computations required, finding the optimal action sequence when rewards are known is equivalent to identifying the rewards of the optimal action sequence when rewards are not known. The latter, however, is much more convenient to interpret probabilistically.

A tree is completely characterized by the rewards $r(s, a)$. We assume a prior distribution over the rewards, which induces a distribution over the trees, which we denote as $P(\mathcal{T})$.[2] For a tree $\mathcal{T}$, let $Q_n(s, a; \mathcal{T})$ denote its corresponding state-action value function, defined as follows:

$$Q_n(s, a) = \max_{a_{n+1}, \ldots, a_{H-1}} \sum_{\tau=n+1}^{H-1} r(s_\tau, a_\tau),$$

$$\text{s.t.} \quad s_n = s, \quad a_n = a,$$
$$s_{\tau+1} = f(s_\tau, a_\tau), \quad \forall \tau \in n, \ldots H - 1.$$

The prior distribution over value functions is given by $P(Q_n(s, a) = \alpha) = \sum_{\mathcal{T}} P(\mathcal{T}) \mathbf{1}\{Q_n(s, a; \mathcal{T}) = \alpha\}$.

We consider a sequential and stochastic discovery of the tree that takes place over T iterations (T will be termed the *search budget*), where at each iteration $t \in \{1, \ldots, T\}$, a reward for a particular leaf state-action pair $z_t = (s_t, a_t)$ is revealed; an illustration is provided in Figure 2. Recall that this sequential discovery relates to the planning that happens *at each* time step of the online planning scheme described above, and should yield the optimal action to take at the root node, which is the current state of the environment. Let $\mathcal{Z}$ denote the set of all state-action pairs in the tree, and let $\mathcal{Z}_t$ denote the set of *leaf* state-action pairs in the tree that has been discovered up to iteration t. That is, we have that for all t, $z_t \in \mathcal{Z}_t$, and $\mathcal{Z}_t \subset \mathcal{Z}$.

---

[2]For simplicity, we assume that $P(\mathcal{T})$ is a discrete distribution, but our derivations extend to continuous distributions by replacing sums with integrals.

Let $O_t = \{(s_t, a_t), r(s_t, a_t)\} \equiv \{z_t, r(z_t)\}$ denote the observation at time t, and let $\mathcal{F}_t = \{O_0, \ldots, O_{t-1}\}$, where $O_0 = \{\}$, denote the history (σ-algebra) of the discovery process at time t.[3] We will consider the posterior distributions $P(\mathcal{T}|\mathcal{F}_t)$ and $P(Q_n(s, a)|\mathcal{F}_t)$, which are well defined.

Given $\mathcal{T}$, an optimal action at the root is well defined:

$$A^* \in \arg\max_{a \in A} Q_0(s_0, a; \mathcal{T}). \tag{2}$$

For simplicity, we will assume that for any possible tree, the optimal action at the root is unique. For a leaf state-action pair z_t that is explored at iteration t, let $A_{\text{root}}(z_t) \in A$ denote the action at the root that leads to z_t. Also, let z_t^* denote the leaf available at time t that is on the optimal branch (if the branch is unique, then the leaf is unique), and let z^* denote the leaf of the complete tree on the optimal branch. Note that $A_{\text{root}}(z_t^*) = A_{\text{root}}(z^*) = A^*$.

We define the T period *regret* of the sequence of state-action pairs $z_1, \ldots, z_T$ as the random variable, $\text{Regret}(T) = \sum_{t=1}^{T} [Q_0(s_0, A^*) - Q_0(s_0, A_{\text{root}}(z_t))]$. Note that minimizing regret is equivalent to minimizing the error due to a suboptimal action *at the root*, which, as explained above, is what ultimately matters for the online planning scheme. We shall study the expected regret, a.k.a. Bayesian regret,

$$\mathbb{E}\left[\text{Regret}(T)\right] = \mathbb{E}\left[\sum_{t=1}^{T} [Q_0(s_0, A^*) - Q_0(s_0, A_{\text{root}}(z_t))]\right], \tag{3}$$

where the expectation is taken over the randomness in the action selection, and over the prior distribution over $\mathcal{T}$.

We shall now propose a general tree search algorithm, and then analyse its Bayesian regret. Our tree search algorithm is based on the Thompson Sampling idea (Thompson, 1933), and selects leaves according to their posterior probability of being on the optimal branch:

$$P(z_t = z|\mathcal{F}_t) = P(z_t^* = z|\mathcal{F}_t). \tag{4}$$

For a random variable X, let $\mathcal{H}(X)$ denote its Shannon entropy. The next theorem bounds the regret of our algorithm.

Theorem 1. *The regret of the leaf selection rule defined in Eq. (4) satisfies:* $\mathbb{E}\left[\text{Regret}(T)\right] \leq H R_{max} \sqrt{\frac{1}{2}|\mathcal{Z}|\mathcal{H}(z^*)T}$.

Note the dependence on $\mathcal{H}(z^*)$ – if the prior over z^* has low entropy, i.e., it is an *informative* prior, the problem becomes easier. This is a property of the Bayesian analysis. To further demonstrate this point, the following example compares the performance of Thompson Sampling based tree search to a search of the tree that is agnostic to the prior (e.g., breadth first search or depth first search).

---

[3]While we assume here that rewards are deterministic, our derivation extends to stochastic rewards, where each observation reveals an i.i.d. sample from the reward.

Example 1. *Consider a distribution of trees where each tree has exactly one state-action pair with non-zero reward, equal to* 1. *Without loss of generality, at each time step* t *before the non-zero reward is found, the agnostic search has a probability of* $1/(|\mathcal{Z}|-t+1)$ *to discover the reward, and the regret is* 1. *After the reward has been found, the regret is* 0. *This leads to expected regret* $\mathbb{E}\left[Regret(T)\right] = T(1 - \frac{T}{2|\mathcal{Z}|})$. *In particular, for* $T = |\mathcal{Z}|$, *we have* $\mathbb{E}\left[Regret(T = |\mathcal{Z}|)\right] = \frac{|\mathcal{Z}|}{2}$. *The bound in Theorem* 1, *on the other hand, gives* $\mathbb{E}\left[Regret(T = |\mathcal{Z}|)\right] = |\mathcal{Z}|\sqrt{\frac{\mathcal{H}(z^*)}{2}}$, *where we ignore the* HR_{max} *terms as each trajectory can have at most reward* 1. *When the prior is uniform,* $\mathcal{H}(z^*) = \log|\mathcal{Z}|$, *giving a worse regret than for the agnostic search. However, if the prior is such that* $\mathcal{H}(z^*) < 0.5$, *Thompson sampling will exploit this structure to obtain a lower regret.*

The proof of Theorem 1 builds on the information theoretic analysis of Thompson Sampling for multi armed bandits (MABs) by Russo & Van Roy (2016), and is detailed in Section G. We adapt their analysis to our case by defining a MAB problem that has similar regret as (3), but with non i.i.d. rewards, and exploit the fact that their analysis does not depend on the rewards being i.i.d.[4]

2.2. Practical Thompson Sampling Tree Search

The previous section established that the Thompson sampling strategy is a sound exploration method for tree search. Practically, however, each iteration of Thompson sampling involves sampling a leaf in the tree from the posterior $P(z_t^*|\mathcal{F}_t)$. In general, computing the posterior probability and sampling from it can be computationally demanding. In the following, we propose an efficient method for the special family of independent Q value posteriors.

We shall use the following notation. For independent random variables $X_1, \ldots, X_n$ with distributions $P(X_1), \ldots, P(X_n)$ we denote by $P\left(\max_{i \in 1, \ldots, n}\{X_i\}\right)$ the distribution of their maximum order statistic. For a scalar b, we denote by $P(X_1 + b)$ the distribution of the random variable $X_1 + b$.

Given $\mathcal{T}$, Bellman's optimality equation states that, $Q_n(s, a; \mathcal{T}) = r(s, a) + \max_{a'}\{Q_{n+1}(f(s, a), a'; \mathcal{T})\}$.

Consider that at iteration $t - 1$ of our Thompson sampling algorithm, action a_{t-1} was chosen at state s_{t-1}, and $r(s_{t-1}, a_{t-1})$ was revealed. Assume that the posteriors for the next state and action values $P(Q_{n+1}(f(s_{t-1}, a_{t-1}), a')|\mathcal{F}_t)$ are independent (with respect to the different actions). Then, we have that the poste-

---

Algorithm 1 Thompson Sampling Tree Search

Require: $P_{\text{query}}(Q(s, a))$, $r_{\text{query}}(s, a)$.
  Init known set $S_{\text{known}} := \{s_0\}$
  Init value $P(Q(s_0, a)) := P_{\text{query}}(Q(s_0, a)) \quad \forall a \in A$.
  for t = 1,2,...,T **do**
                                    # Forward sampling
    Set $s := s_0$, $s' := \emptyset$
    while True **do**
      For each $a \in A$, sample $Q(s, a) \sim P(Q(s, a))$ [*]
      Set $s' := f(s, \arg\max_{a \in A} Q(s, a))$ [**]
      if $s' \notin S_{\text{known}}$ **then**
        Set $S_{\text{known}} := S_{\text{known}} \cup \{s'\}$
        Break
      end if
      Set $s := s'$
    end while
    Set value distribution for leaf $P(Q(s', a')) = P_{\text{query}}(Q(s', a')) \quad \forall a' \in A$.
                                    # Max-Backup
    while True **do**
      Set $(s, a) := f^{-1}(s')$
      Set $P(Q(s, a)) := P(r(s, a) + \max_{a' \in A} Q(s', a'))$
      if $s = s_0$ **then**
        Break
      end if
      Set $s' := s$
    end while
  end for

---

rior for $Q_n(s_{t-1}, a_{t-1})$ is given by,[5]

$$P(Q_n(s_{t-1}, a_{t-1})|\mathcal{F}_t) = \qquad (5)$$
$$P\left(r(s_{t-1}, a_{t-1}) + \max_{a'}\{Q_{n+1}(f(s_{t-1}, a_{t-1}), a')\}\,\Big|\,\mathcal{F}_t\right).$$

If we further assume that the posterior for branches that do not involve (s_{t-1}, a_{t-1}) does not change, we can apply the rule in (5) recursively to update all the posteriors in the tree. We refer to this update as the *max-backup* method.

After updating the posterior Q values, sampling from $P(z_t^*|\mathcal{F}_t)$ can be done by noting that for any state s, we have that $P((s, a) \in \text{optimal branch}) = P(Q(s, a) > Q(s, \tilde{a}) \quad \forall \tilde{a} \neq a)$. Therefore, we can sample from the optimal branch distribution by sequentially sampling Q values, and choosing the optimal action w.r.t. the sampled Q. We term this the *forward sampling* method.

The Thompson Sampling Tree Search method (TSTS) in Algorithm 1 combines forward sampling and max-backup into a complete tree search routine. Figure 3 further illustrates the different steps in the algorithm. Our algorithm requires that when we explore a leaf (s, a), we can directly

---

[4]While Russo & Van Roy (2016) do consider a case of i.i.d. rewards, their analysis does not actually use this property at all. We note that this fact was previously observed in other studies such as Bubeck & Eldan (2016).

[5]Eq. (5) also holds for dependent posteriors. For ease of exposition, however, we focus on the independent case.

query its reward and next state-action value posteriors, denoted $r_{\text{query}}(s,a)$ and $P_{\text{query}}(Q(s,a))$, respectively.[6] In the sequel, we will realize the posterior query using various learned neural network approximations. We mention that the max-backup rule was proposed by Tesauro et al. (2010), but without a formal derivation. In our formulation, max-backup, when combined with forward sampling, and under the independent posterior distribution assumption, emerges as a natural implementation of the TSTS method. We establish this formally in the following proposition.

Proposition 1. *Assume that at each iteration t, the posterior of values for state-actions on the leaves are independent, i.e., $P(Q(s,a) : (s,a) \in \mathcal{Z}_t | \mathcal{F}_t) = \prod_{(s,a) \in \mathcal{Z}_t} P(Q(s,a) | \mathcal{F}_t)$. Then Algorithm 1 samples leaves from the TS distribution, $P(z_t^* | \mathcal{F}_t)$.*

Let us discuss the validity of the independent posteriors assumption. Indeed, it is easy to imagine problems where inferring a reward in a particular state action is informative about the value of different actions in the same state, or even about the value of other states in the tree. Unfortunately, designing an efficient posterior update and sampling method for this case is non-trivial, and we leave it as an open problem for future work. Empirically, we have found that even under the independent posteriors assumption, our Bayesian approach can yield significant performance improvements.

2.3. Improved Exploration via Bayes-UCB

The TSTS method resolves the exploration-exploitation tradeoff through posterior sampling. In practice, other exploration methods may perform better, as posterior sampling is not necessarily Bayes-optimal. In this section we propose a different Bayesian exploration strategy that we found to empirically perform very well.

We propose Bayes-UCB Tree Search (BTS) – an exploration method based on the idea of optimism in the face of uncertainty, inspired by the Bayes-UCB algorithm of Kaufmann et al. (2012). The idea is to choose actions at each state proportional to the *quantiles* of posterior state-action values.

For a random variable X with distribution $P(X)$, let $\rho(\alpha, P)$ be its α-quantile, such that $P(X \leq \rho(\alpha, P)) = \alpha$. In BTS, we replace the action selection rule in TSTS's Forward Sampling method (lines marked by [*] and [**] in Algorithm 1) with the following:

$$a^* := \arg\max_a \rho(\alpha(s), P(Q(s,a))),$$
$$s' := f(s, a^*), \tag{6}$$

where the quantile level is given by $\alpha(s) = 1 - (1 - \alpha_0) \cdot e^{-\frac{N(s)-1}{\beta}}$, where $N(s)$ is the number of visits to state s,

and the initial quantile α_0 and rate coefficient β are tunable hyper-parameters. Full pseudo-code is provided in Appendix A.

The Bayes-UCB algorithm of Kaufmann et al. (2012) applied a selection rule $\alpha(s) = 1 - \frac{\beta}{N(s)}$ in the MAB setting, where $P(Q(s,a))$ is replaced with the posterior reward probability for each arm, and $N(s)$ is replaced with the iteration number. Under the assumptions of Proposition 1, the Max-Backup method leads to the correct state-action value posterior in each state, and hence BTS applies Bayes-UCB with a different quantile schedule to tree search by applying it to each state, similarly to the way UCB is adapted to tree search in UCT (Kocsis & Szepesvári, 2006). The Bayes-UCT2 rule of Tesauro et al. (2010) selects an action that maximizes $\mathbb{E}[P(Q(s,a))] + \sqrt{2 \ln N(s) \text{Var}[P(Q(s,a))]}$, which for a Gaussian posterior is equivalent to a quantile schedule $0.5 + 0.5 \, \text{erf}(\sqrt{\ln N(s)})$. Intuitively, in all three schedules, as a node is visited more often, the action selection is more optimistic (higher quantile), exploring actions that have some chance of turning out to be better than the action that currently yields the highest expected return. This intuition is shown in Kaufmann et al. (2012) to yield asymptotically optimal Bayesian regret bounds for the MAB problem with binary rewards. In our experiments, we found the BTS schedule to outperform both Bayes-UCB and Bayes-UCT2. Adapting the analysis of Kaufmann et al. (2012) to the tree search setting is not trivial, and left to future work.

2.4. Action Commitment in Online Planning

To connect our Bayesian tree search methods to the online planning scheme, note that TSTS and BTS return the posterior state-action value distribution at the root state, and also the tree discovered during search, both of which can be used by the online planning scheme to select an action to perform in the environment. We shall term this action selection step as *action commitment*, different from the action selection during tree search.

We found that using the posterior state-action value distribution for action commitment often yielded unfavorable results, as the max-backup tends to inflate the value of states down the tree with high uncertainty. An alternative, which we shall term *MCTS action commitment*, is to select an action that corresponds to the branch with the highest expected return (sum of rewards $r_{\text{query}}(s,a)$ and $\mathbb{E}[P_{\text{query}}(Q(s,a))]$ at the leaf). We note that due to the deterministic dynamics and reward, this strategy is equivalent to a standard MCTS algorithm that commits to the action with the highest backed-up value at the root. An even safer strategy is to select an action that corresponds to the branch with the highest α-quantile of the return. A different method is to choose actions stochastically, according to a SoftMax over the backed-up value at the root; this method is helpful when the agent can get

---

[6] We omit the conditioning of the posteriors r_{query} and $P_{\text{query}}(Q(s,a))$ on history to ease notation.

'stuck' by committing to a wrong action over and over again. We explore these commitment strategies in our experiments.

2.5. Learning in Bayesian Tree Search

After describing the fundamentals of Bayesian tree search, we are finally ready to combine TSTS/BTS with deep learning of the state-action value function distribution.

The basic component in our method is a neural network prediction for the posterior value distribution $P_{\text{query}}(Q(s,a)) = \mathcal{N}(\mu_\theta(s,a), \sigma_\theta^2(s,a))$, where $\mu_\theta(s,a)$ is the output of a neural network with parameters θ, and $\sigma_\theta(s,a)$ is an estimate of the uncertainty in $\mu_\theta(s,a)$. The literature on estimating uncertainty in neural network predictions is extensive (Gawlikowski et al., 2021); here, we focus on two simple methods: maximum likelihood estimation (MLE) and ensembles. In the MLE method, the neural network has an additional head for $\log \sigma_\theta(s,a)$ (Kendall et al., 2018), and the loss function for a data sample $\widehat{Q}(s,a)$ is the negative log likelihood:
$\mathcal{L} = \frac{1}{2}\log\left(\sigma_\theta(s,a)^2\right) + \frac{\left(\mu_\theta(s,a)-\widehat{Q}(s,a)\right)^2}{2\sigma_\theta(s,a)^2}.$

In the ensemble method, we have an ensemble of K neural networks for $\mu(s,a)$, $\mu_{\theta_1}(s,a), \ldots, \mu_{\theta_K}(s,a)$, each trained using the mean-squared error loss $\mathcal{L} = \sum_{i=1}^{K}\left(\mu_{\theta_i}(s,a) - \widehat{Q}(s,a)\right)^2$, but with different (random) initial weights. The output of the ensemble is the average, $\mu_\theta(s,a) = \frac{1}{K}\sum_{i=1}^{K}\mu_{\theta_i}(s,a)$, and the uncertainty estimate is the empirical standard deviation, $\sigma_\theta^2(s,a) = \frac{1}{K-1}\sum_{i=1}^{K}\left(\mu_{\theta_i}(s,a) - \mu_\theta(s,a)\right)^2$.

Our method proceeds in rounds similar to AlphaZero and Expert Iteration (Anthony et al., 2017; Silver et al., 2018), where in each round the initial state is reset to s_0, and online planning with current neural network parameters θ is performed for k time steps (or until a terminal state is reached). The learning targets for each root state and actions visited in the online trajectory are the expected posterior Q values at the root per each search.

3. Related Work

Meta-reasoning is the study of allocating computational resources in artificial intelligence (Russell & Wefald, 1991a; Griffiths et al., 2019). Selecting which actions to explore during search is known as the *metalevel* decision problem (Hay et al., 2012; Russell & Wefald, 1991b), and is related to Howard's value of information (Howard, 1966) and the Bayesian ranking and selection problem (Frazier & Powell, 2010). In its Bayesian formulation, the optimal sequence of actions is well defined and its computation is equivalent to solving a partially observed MDP (Hay et al., 2012), thus finding it is generally intractable. Approximate solutions include Thompson sampling (Thompson, 1933)

and the knowledge gradient (Ryzhov et al., 2012), which is related to the value of perfect information (VPI) heuristic (Baum & Smith, 1997; Dearden et al., 1998; Russell & Wefald, 1991b), and the expected improvement heuristic in Bayesian optimization (Frazier, 2018).

Several studies applied a Bayesian metalevel decision making approach in MCTS. Mern et al. (2021) use Gaussian processes for MCTS with continuous actions, and apply the expected improvement heuristic for selecting actions. Bai et al. (2013) replace the UCB selection rule in each node of the MCTS search tree with Thompson sampling, assuming that Q values are distributed as a mixture-of-Gaussians, and Bai et al. (2014; 2018) further extends this approach to planning in partially observable problems. Tolpin & Shimony (2012); Hay et al. (2012) replace MCTS's action selection at the root node by approximations to the value of perfect information, and with a UCB update suited for the simple regret. Tesauro et al. (2010) propose a backup of the maximum order statistic, similarly to our max-backup, and used it within an ad hoc UCB-style selection rule. Closely related to our work, the recent study by Dam et al. (2023) models the value distributions along the search tree using Gaussians, and uses the power mean to backup values and their uncertainties, and propose both a UCT and Thompson sampling strategies for action selection. Bai et al. (2013) and Tesauro et al. (2010) claim asymptotic convergence of their methods to the optimal action in the limit $T \to \infty$, and Dam et al. (2023) also provide an *asymptotic* polynomial convergence rate. Different from the works above, we effectively apply Thompson sampling to the *branches* in the search tree, which allows us to obtain the first *finite-sample regret* guarantees for Thompson sampling tree search. In addition, we investigate the *learning* setting, by connecting the Bayesian posterior to neural network uncertainty estimates.

Lan et al. (2021) estimate the neural network uncertainty, and use it to stop the MCTS search when it is certain, reducing average computation time. In contrast, we exploit the uncertainty to design a different search procedure, which is orthogonal to early stopping. Danihelka et al. (2021) improve Alpha Zero's search by replacing UCB at the root, which minimizes *cumulative* regret, with sequential halving (Karnin et al., 2013) – a frequentist algorithm for minimizing the *simple* regret. We compare our approach with different action selection and backup procedures based on Danihelka et al. (2021), Bai et al. (2013), and Dam et al. (2023) in our experiments.

Jin & Keutzer (2015) is, to our knowledge, the only previous work that used neural networks with a Thompson sampling-based MCTS algorithm. In that work, a policy network was employed for selecting actions during rollouts. In our work, following the successful AlphaZero methodology (Silver et al., 2018), we use neural networks for value estimates.
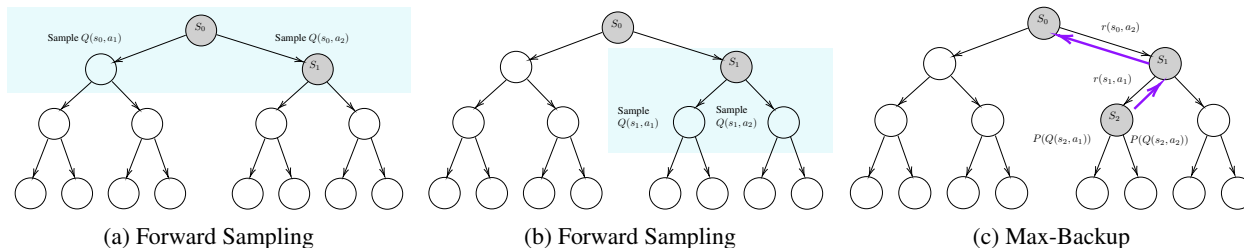
Figure 3. TSTS Algorithm Schematic. Plots (a) and (b) show two successive iterations of forward sampling, where states in S_{known} are marked in gray. Subsequently, in Plot (c), state s_2 is added to S_{known}, and the max-backup routine is performed to update the posteriors.

4. Experiments

We aim to demonstrate the potential of using uncertainty estimates in online planning. However, as the online planning and learning procedure involves several components, our consideration was to design experiments where the contribution of individual components could be clearly teased out and investigated. To this end, we focus on two tasks from the ProcGen suite of procedurally generated game environments (Cobbe et al., 2020), Maze and Leaper. We report an extensive investigation on Maze in the main text, and present similar results on Leaper in the supplementary material Section D. Designed as a benchmark for zero-shot generalization in deep RL, ProcGen presents a challenge in dealing with epistemic uncertainty, which we hope to mitigate using our Bayesian approach. In addition, the maze and leaper domains allow us to calculate ground-truth values for the Q functions and consequently, also for the uncertainty in the neural-network approximation. We begin by describing our experimental setup; comprehensive technical details are provided in Appendix B.

Online planning in ProcGen: ProcGen games are deterministic, with a finite action space. The game state is not directly accessible, but the agent observes a rendering of the game state as an image. The simulator state can be saved, and reset to a saved state, allowing us to implement a model-based planning scheme without access to the true state transitions, by querying the simulator for the image that would be observed upon taking an action. At each time step $i = 1, \ldots, k$, the agent is in state s_i and allowed a search budget of T tree search iterations, after which it must commit to an action a_i, and the game proceeds to the next time step. We evaluate the agent by whether it reached a rewarding terminal state or not; evaluation by the accumulated reward in the environment gave similar results.

Learning in ProcGen: ProcGen procedurally generates game levels, and we let ℓ denote a specific level instance (in practice, the random seed used to generate this level). We consider a set of N_{train} training levels and disjoint set of N_{test} test levels. We train our neural networks on the training levels, and evaluate their performance on the test levels. Pre-

vious work (Cobbe et al., 2020) has already established that in the Maze game, for a moderate N_{train} there is a significant generalization gap, indicating high relevance for epistemic uncertainty. We emphasize that our goal is not to reduce this uncertainty, but only to mitigate its effect on planning. Therefore, we adopt the Impala neural network architecture that was used in previous studies (Espeholt et al., 2018), and a moderate $N_{\text{train}} = 150$. Further details regarding the training of the neural network are given in Appendix B.2.

Evaluation: Different planning algorithms can be evaluated using the same neural network $\mu_\theta(s, a)$. We differentiate between the planner used for collecting the data for learning, termed the *annotator*, and the planner used for evaluation. In our experiments, we evaluate different planners on a network trained with a single annotator, allowing us to compare different planners on *the same* neural network. In addition, we note that some planners are inherently stochastic (e.g., TSTS), while some are deterministic (e.g., UCT). In domains such as mazes, where repeatedly choosing a wrong action would get the agent stuck, stochasticity can be an advantage. To fairly compare planners in such domains, we fix the random seed of a stochastic planner A to be the same at all time steps, and denote such a planner as A_{det}. Finally, we specify the action commitment strategies we use for each experiment in the experiment description.

Ground Truth Values and Uncertainties: In a Maze task, it is straightforward to calculate a ground truth value of $Q(s, a)$, denoted $Q^{\text{GT}}(s, a)$ using algorithms for shortest paths on graphs (Even, 2011). While our agents do not have direct access to the graph that underlies the observed image, we can use the ground truth value to obtain a ground truth estimate of the neural network uncertainty,

$$\sigma_\theta^{\text{GT}}(s, a) = |\mu_\theta(s, a) - Q^{\text{GT}}(s, a)|. \tag{7}$$

We emphasize that in any realistic problem, $Q^{\text{GT}}(s, a)$ and $\sigma_\theta^{\text{GT}}(s, a)$ would not be available at test time, and we use them here only to demonstrate the potential of our algorithms when uncertainty estimation is perfect.

Baselines and Ablations: We compare TSTS and BTS with the following baselines. N-MCTS: a neural MCTS al-

gorithm based on Anthony et al. (2017); Silver et al. (2017). For a fair comparison with our methods, we train a single Q-network, and use it both for value estimation and a SoftMax policy for searching the tree using P-UCT[7]. The SoftMax temperature was set to 2.0 using a hyper-parameter search. `SH-N-MCTS`: a variant of `N-MCTS` inspired by Danihelka et al. (2021), where exploration at the root is done using sequential halving instead of P-UCT. Note that `N-MCTS` is deterministic while `SH-N-MCTS` is stochastic. `B-UCT2` and `B-UCB`: the `BTS` algorithm, but with the Bayes-UCT2 (Tesauro et al., 2010) and the Bayes-UCB (Kaufmann et al., 2012) action selection rules, respectively. In all algorithms, best hyper-parameters were searched for; we report on the sensitivity to hyper-parameters in Appendix E. In addition, in Appendix E.5 we compare our algorithm with the methods of Dam et al. (2023) and Bai et al. (2013).

4.1. Results

In Figure 4 we compare various deterministic Bayesian planners with `N-MCTS`, under different search budgets. In this experiment, the neural network was trained using an `N-MCTS` annotator for 250 epochs. When training the head for predicting $\sigma_\theta(s, a)$ we kept the rest of the network frozen, therefore the `N-MCTS` results are the same as would have been obtained without learning the uncertainty. Our comparison uses the same $\mu_\theta(s, a)$ in all planners. Furthermore, in this experiment we chose MCTS action commitment for all planners. Thus, **the only difference between the planners is how they search the tree**. We make several observations. Clearly, the results on training domains and test domains are markedly different, validating our hypothesis that epistemic uncertainty can significantly affect planning. On training domains, all methods are comparable, and achieve significantly better results than on test, where network predictions can be significantly less accurate. We next discuss the comparison on test domains. First, with ground truth uncertainty, all Bayesian methods (`TSTS` and `BTS` variants) significantly outperform `N-MCTS`, validating our main premise – *accounting for epistemic uncertainty during the search leads to more informative search trees* (see Appendix F for a detailed analysis including illustration of the search trees). Second, we observe that `BTS` outperforms `B-UCT2` and `B-UCB`, which outperform `TSTS`. Thus, in contrast to the MAB setting (Chapelle & Li, 2011), it appears that in tree search Thompson sampling is significantly outperformed by (Bayesian) exploration bonuses. Third, we observe that the learned uncertainty estimates are not accurate enough to yield improvement over `N-MCTS`.

In Figure 4c we show similar results for stochastic planners: `TSTS` and `SH-N-MCTS`, in which the search process is

stochastic, and also deterministic `N-MCTS` and `BTS` with a stochastic SoftMax action commitment. Stochasticity helps avoid recurring mistakes and improves performance, yet the relative ordering between the algorithms is similar to the deterministic case. For `SH-N-MCTS`, we noticed significantly worse performance than all other methods in this domain. In Appendix Figure 7 we show a similar comparison and similar conclusions using neural network ensembles.

We next study how accurate uncertainty prediction needs to be to yield improvements over `N-MCTS`. We add $\rho\%$ error to each ground truth uncertainty estimate in Eq. 7 by multiplying it by $1 + U$, where $U \sim \text{Uniform}(-0.01\rho, 0.01\rho)$. Our results in Figure 5a show that with up to 20% error `BTS` still significantly outperforms `N-MCTS`.

In Figure 5b we investigate the action commitment during online planning, using `BTS` with ground truth uncertainty and different quantiles (cf. Section 2.4). We observe that committing to *risk-averse* actions significantly improves performance, while the results above show that during the exploration of the search tree being risk seeking is beneficial. Thus, maintaining posterior distributions has additional benefits in online planning beyond the improved search trees.

5. Discussion

The hypothesis in this paper is that uncertainty estimates can benefit the search in online planning with neural networks. We developed the fundamentals of a Bayesian tree search that facilitates such estimates, and proposed several practical algorithms. Our experimental results are mixed: on the one hand, with ground truth uncertainty estimates, we observed a dramatic improvement over state-of-the-art frequentist methods. On the other hand, ground truth estimates are not practical, and our efforts to learn uncertainty estimates proved too inaccurate to yield significant gains in planning.

Our results are specific to the ProcGen maze and leaper environments, and it is possible that domains with different reward structures and dynamics will be more or less sensitive to the uncertainty estimation accuracy. Nevertheless, we conclude that there is great potential to studying methods for estimating neural network uncertainty in the sequential decision making setting, reinforcing a similar conclusion made by Riquelme et al. (2018) for Bayesian bandits. Some promising recent developments include methods based on conformal prediction (Angelopoulos & Bates, 2021) and epistemic neural networks (Osband et al., 2023).

Another interesting direction is to learn posteriors that depend on the complete search history, instead of the independent neural network estimates used here, for example, using transformer models that learn complete decision making strategies (Laskin et al., 2022). The Bayesian view offers a natural framework for such methods.
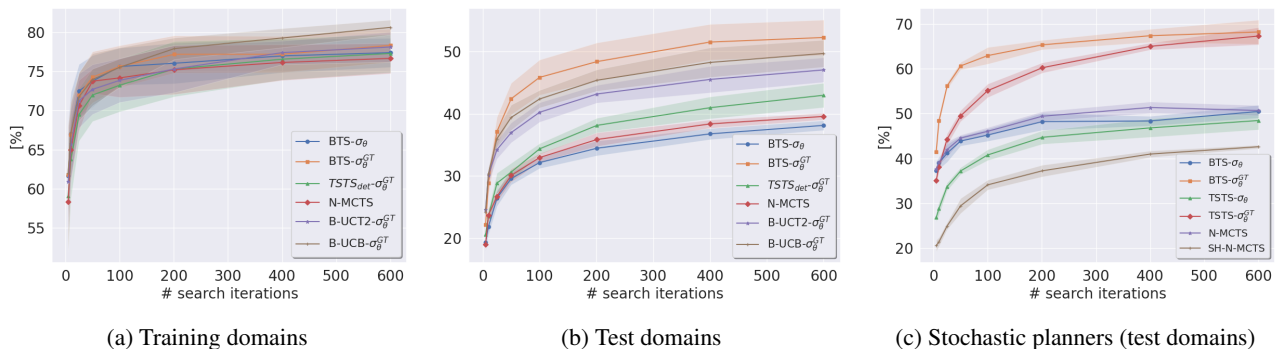
---

[7]The alpha-zero implementation in (Silver et al., 2017), for example, had different networks for the policy and the value functions, making it harder to compare with `TSTS`.

(a) Training domains   (b) Test domains   (c) Stochastic planners (test domains)

Figure 4. Success rate of different planners on ProcGen maze. Left + Middle: deterministic planners. Right: stochastic planners. Error bars are over 6 neural networks obtained from independent training runs. See Section 4.1 for more details.
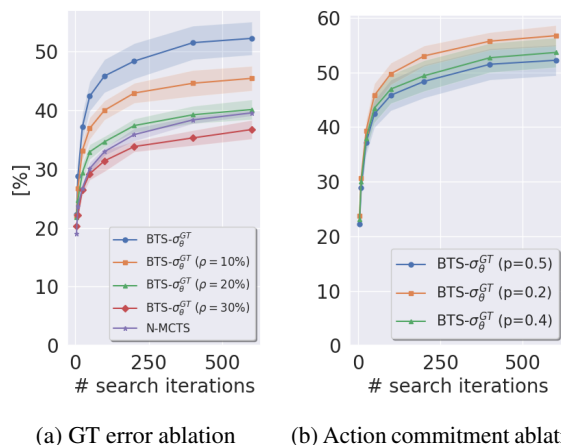

(a) GT error ablation   (b) Action commitment ablation

Figure 5. Ground truth uncertainty error and action commitment ablations, per Section 4.1 in the text.

Impact Statement

This paper presents work whose goal is to advance the field of Machine Learning. There are many potential societal consequences of our work, none which we feel must be specifically highlighted here.

References

Angelopoulos, A. N. and Bates, S. A gentle introduction to conformal prediction and distribution-free uncertainty quantification. *arXiv preprint arXiv:2107.07511*, 2021.

Anthony, T., Tian, Z., and Barber, D. Thinking fast and slow with deep learning and tree search. *Advances in Neural Information Processing Systems*, 30, 2017.

Audibert, J.-Y., Munos, R., and Szepesvári, C. Exploration–exploitation tradeoff using variance estimates in multi-armed bandits. *Theoretical Computer Science*, 410(19): 1876–1902, 2009.

Bai, A., Wu, F., and Chen, X. Bayesian mixture modelling and inference based thompson sampling in monte-carlo

tree search. *Advances in neural information processing systems*, 26, 2013.

Bai, A., Wu, F., Zhang, Z., and Chen, X. Thompson sampling based monte-carlo planning in pomdps. In *Proceedings of the International Conference on Automated Planning and Scheduling*, volume 24, pp. 29–37, 2014.

Bai, A., Wu, F., and Chen, X. Posterior sampling for monte carlo planning under uncertainty. *Applied Intelligence*, 48(12):4998–5018, 2018. URL https://aijunbai.github.io/publications/AI18-Bai.pdf.

Baum, E. B. and Smith, W. D. A bayesian approach to relevance in game playing. *Artificial Intelligence*, 97 (1-2):195–242, 1997.

Bubeck, S. and Eldan, R. Multi-scale exploration of convex functions and bandit convex optimization. In *Conference on Learning Theory*, pp. 583–589. PMLR, 2016.

Cesari, G., Schildbach, G., Carvalho, A., and Borrelli, F. Scenario model predictive control for lane change assistance and autonomous driving on highways. *IEEE Intelligent transportation systems magazine*, 9(3):23–35, 2017.

Chapelle, O. and Li, L. An empirical evaluation of thompson sampling. *Advances in neural information processing systems*, 24, 2011.

Chen, Y., Huang, A., Wang, Z., Antonoglou, I., Schrittwieser, J., Silver, D., and de Freitas, N. Bayesian optimization in alphago. *arXiv preprint arXiv:1812.06855*, 2018.

Cobbe, K., Hesse, C., Hilton, J., and Schulman, J. Leveraging procedural generation to benchmark reinforcement learning. In *International conference on machine learning*, pp. 2048–2056. PMLR, 2020.

Dam, T., Stenger, P., Schneider, L., Pajarinen, J., D'Eramo, C., and Maillard, O.-A. Monte-carlo tree search with

uncertainty propagation via optimal transport. *arXiv preprint arXiv:2309.10737*, 2023.

Danihelka, I., Guez, A., Schrittwieser, J., and Silver, D. Policy improvement by planning with gumbel. In *International Conference on Learning Representations*, 2021.

Dearden, R., Friedman, N., and Russell, S. Bayesian q-learning. In *Aaai/iaai*, pp. 761–768, 1998.

Der Kiureghian, A. and Ditlevsen, O. Aleatory or epistemic? does it matter? *Structural safety*, 31(2):105–112, 2009.

Espeholt, L., Soyer, H., Munos, R., Simonyan, K., Mnih, V., Ward, T., Doron, Y., Firoiu, V., Harley, T., Dunning, I., et al. Impala: Scalable distributed deep-rl with importance weighted actor-learner architectures. In *International conference on machine learning*, pp. 1407–1416. PMLR, 2018.

Even, S. *Graph algorithms*. Cambridge University Press, 2011.

Finn, C. and Levine, S. Deep visual foresight for planning robot motion. In *2017 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 2786–2793. IEEE, 2017.

Frazier, P. I. A tutorial on bayesian optimization. *arXiv preprint arXiv:1807.02811*, 2018.

Frazier, P. I. and Powell, W. B. Paradoxes in learning and the marginal value of information. *Decision Analysis*, 7 (4):378–403, 2010.

Gawlikowski, J., Tassi, C. R. N., Ali, M., Lee, J., Humt, M., Feng, J., Kruspe, A., Triebel, R., Jung, P., Roscher, R., et al. A survey of uncertainty in deep neural networks. *arXiv preprint arXiv:2107.03342*, 2021.

Griffiths, T. L., Callaway, F., Chang, M. B., Grant, E., Krueger, P. M., and Lieder, F. Doing more with less: meta-reasoning and meta-learning in humans and machines. *Current Opinion in Behavioral Sciences*, 29: 24–30, 2019.

Hay, N., Russell, S., Tolpin, D., and Shimony, S. E. Selecting computations: Theory and applications. In *28th Conference on Uncertainty in Artificial Intelligence, UAI 2012*, pp. 346–355, 2012.

Howard, R. A. Information value theory. *IEEE Transactions on systems science and cybernetics*, 2(1):22–26, 1966.

Jin, P. H. and Keutzer, K. Convolutional monte carlo rollouts in go. *arXiv preprint arXiv:1512.03375*, 2015.

Karnin, Z., Koren, T., and Somekh, O. Almost optimal exploration in multi-armed bandits. In *International Conference on Machine Learning*, pp. 1238–1246. PMLR, 2013.

Kaufmann, E., Cappé, O., and Garivier, A. On bayesian upper confidence bounds for bandit problems. In *Artificial intelligence and statistics*, pp. 592–600. PMLR, 2012.

Kendall, A., Gal, Y., and Cipolla, R. Multi-task learning using uncertainty to weigh losses for scene geometry and semantics. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 7482–7491, 2018.

Kocsis, L. and Szepesvári, C. Bandit based monte-carlo planning. In *European conference on machine learning*, pp. 282–293. Springer, 2006.

Lan, L.-C., Wu, T.-R., Wu, I.-C., and Hsieh, C.-J. Learning to stop: Dynamic simulation monte-carlo tree search. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 35, pp. 259–267, 2021.

Laskin, M., Wang, L., Oh, J., Parisotto, E., Spencer, S., Steigerwald, R., Strouse, D., Hansen, S., Filos, A., Brooks, E., et al. In-context reinforcement learning with algorithm distillation. *arXiv preprint arXiv:2210.14215*, 2022.

Mankowitz, D. J., Michi, A., Zhernov, A., Gelmi, M., Selvi, M., Paduraru, C., Leurent, E., Iqbal, S., Lespiau, J.-B., Ahern, A., et al. Faster sorting algorithms discovered using deep reinforcement learning. *Nature*, 618(7964): 257–263, 2023.

Mern, J., Yildiz, A., Sunberg, Z., Mukerji, T., and Kochenderfer, M. J. Bayesian optimized monte carlo planning. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 35, pp. 11880–11887, 2021.

Osband, I., Wen, Z., Asghari, S. M., Dwaracherla, V., Ibrahimi, M., Lu, X., and Van Roy, B. Approximate thompson sampling via epistemic neural networks. *arXiv preprint arXiv:2302.09205*, 2023.

Riquelme, C., Tucker, G., and Snoek, J. Deep bayesian bandits showdown: An empirical comparison of bayesian deep networks for thompson sampling. *arXiv preprint arXiv:1802.09127*, 2018.

Russell, S. and Wefald, E. Principles of metareasoning. *Artificial intelligence*, 49(1-3):361–395, 1991a.

Russell, S. J. and Wefald, E. *Do the right thing: studies in limited rationality*. 1991b.

Russo, D. and Van Roy, B. An information-theoretic analysis of thompson sampling. *The Journal of Machine Learning Research*, 17(1):2442–2471, 2016.

Ryzhov, I. O., Powell, W. B., and Frazier, P. I. The knowledge gradient algorithm for a general class of online learning problems. *Operations Research*, 60(1):180–195, 2012.

Schrittwieser, J., Antonoglou, I., Hubert, T., Simonyan, K., Sifre, L., Schmitt, S., Guez, A., Lockhart, E., Hassabis, D., Graepel, T., et al. Mastering atari, go, chess and shogi by planning with a learned model. *Nature*, 588(7839): 604–609, 2020.

Shahriari, B., Swersky, K., Wang, Z., Adams, R. P., and De Freitas, N. Taking the human out of the loop: A review of bayesian optimization. *Proceedings of the IEEE*, 104 (1):148–175, 2015.

Shim, D. H., Kim, H. J., and Sastry, S. Decentralized nonlinear model predictive control of multiple flying robots. In *42nd IEEE International Conference on Decision and Control (IEEE Cat. No. 03CH37475)*, volume 4, pp. 3621–3626. IEEE, 2003.

Silver, D., Schrittwieser, J., Simonyan, K., Antonoglou, I., Huang, A., Guez, A., Hubert, T., Baker, L., Lai, M., Bolton, A., et al. Mastering the game of go without human knowledge. *nature*, 550(7676):354–359, 2017.

Silver, D., Hubert, T., Schrittwieser, J., Antonoglou, I., Lai, M., Guez, A., Lanctot, M., Sifre, L., Kumaran, D., Graepel, T., et al. A general reinforcement learning algorithm that masters chess, shogi, and go through self-play. *Science*, 362(6419):1140–1144, 2018.

Tesauro, G., Rajan, V., and Segal, R. Bayesian inference in monte-carlo tree search. In *Proceedings of the Twenty-Sixth Conference on Uncertainty in Artificial Intelligence*, pp. 580–588, 2010.

Thompson, W. R. On the likelihood that one unknown probability exceeds another in view of the evidence of two samples. *Biometrika*, 25(3-4):285–294, 1933.

Tolpin, D. and Shimony, S. Mcts based on simple regret. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 26, pp. 570–576, 2012.

Williams, G., Wagener, N., Goldfain, B., Drews, P., Rehg, J. M., Boots, B., and Theodorou, E. A. Information theoretic mpc for model-based reinforcement learning. In *2017 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 1714–1721. IEEE, 2017.

Yoon, S. W., Fern, A., and Givan, R. Ff-replan: A baseline for probabilistic planning. In *ICAPS*, volume 7, pp. 352–359, 2007.

Zhang, S., Chen, Z., Shen, Y., Ding, M., Tenenbaum, J. B., and Gan, C. Planning with large language models for code generation. *arXiv preprint arXiv:2303.05510*, 2023.

Zintgraf, L., Shiarlis, K., Igl, M., Schulze, S., Gal, Y., Hofmann, K., and Whiteson, S. Varibad: A very good method for bayes-adaptive deep rl via meta-learning. In *International Conference on Learning Representations*, 2019.

A. Bayes-UCB Tree Search Pseudo-code

We provide a complete pseudo-code for `BTS`. Differences from `TSTS` are highlighted.

Algorithm 2 Bayes-UCB Tree Search

Require: $P_{\text{query}}(Q(s,a))$, $r_{\text{query}}(s,a)$, c.
  Init known set $S_{\text{known}} := \{s_0\}$
  Init state visit counters $N(s) := 0$ for all $s \in S$
  Init value $P(Q(s_0,a)) := P_{\text{query}}(Q(s_0,a)) \quad \forall a \in A.$
  for t = 1,2,...,T **do**

                                                                  # Forward sampling
    Set $s := s_0$, $s' := \emptyset$
    while True **do**
      Set $N(s) := N(s) + 1$
      Set $a^* := \arg\max_a \rho(1 - (1 - \alpha_0) \cdot e^{-\frac{N(s)-1}{\beta}}, P(Q(s,a)))$
      Set $s' := f(s,a^*)$
      if $s' \notin S_{\text{known}}$ **then**
        Set $S_{\text{known}} := S_{\text{known}} \cup \{s'\}$
        Break
      end if
      Set $s := s'$
    end while
    Set value distribution for leaf $P(Q(s',a')) = P_{\text{query}}(Q(s',a')) \quad \forall a' \in A.$

                                                               # Max-Backup
    while True **do**
      Set $(s,a) := f^{-1}(s')$
      Set $P(Q(s,a)) := P(r(s,a) + \max_{a' \in A} Q(s',a'))$
      if $s = s_0$ **then**
        Break
      end if
      Set $s' := s$
    end while
  end for
  return $P(Q(s_0,a))$

B. Implementation Details

We detail technical points in our implementation of training and evaluation.

B.1. ProcGen Maze Environment

Throughout our implementation, we did not use a discount factor. To account for this, we modified the ProcGen reward function such that the reward for each time step until reaching the goal is -1.0. This induces short paths to the goal without discounting.

We used random seeds to generate different maze environments for training and testing. In particular, we use a dataset of 150 samples for training, and also present the results of an evaluation on this set (see Figure 4a.). For testing, we use a disjoint set of 500 samples to evaluate the different planners (see Figure 4).

B.2. Neural Network Training Parameters

Our neural network model was trained using the following parameters:

The optimizer is an Adam optimizer with fixed learning rate of 0.001, β coefficients of 0.9 and 0.999 for running averages of gradient and its square respectively, and ϵ of 1e-8.

We expand on our data collection and loading. We maintain a buffer of N_{buffer} samples. Each sample contains a state s, and 4 targets $Q(s, a)$, one for each action at that state. During training, in each epoch we sample N_{bs} batches, each with size 32 samples, from the buffer for training the neural network. We fill the buffer in a FIFO manner using the annotator. The annotator is run on 150 different mazes, each for up to 200 environment steps (early stopping if reaching the goal), and each step has a search budget of 250. After each search, the annotator converts the $Q(s, a)$ values at the root (in the case of Bayesian annotator, we take the expected Q values) to probabilities by using SoftMax with a temperature scaling of 10, and samples an action commitment according to these probabilities. The committed action is used to advance the state of the world, and a new tree search is started over the new state, and this process is repeated. The Q values at each root state during the interaction are inserted as targets to the buffer.

In our experiments we set $N_{\text{buffer}} = 40000$, and $N_{bs} = 200$.

We will release checkpoints of the trained networks to reproduce the figures in the paper.

Figure 6 shows the success rate of the different planners on a subset of the training set, during the training procedure, using a `BTS` annotator. In the experiments in the main text we only used an `N-MCTS` annotator. Note that we show training results on a small subset of training domains, for fast evaluations of the planners during training, therefore the results in the figure are not comparable to the results in the main text.
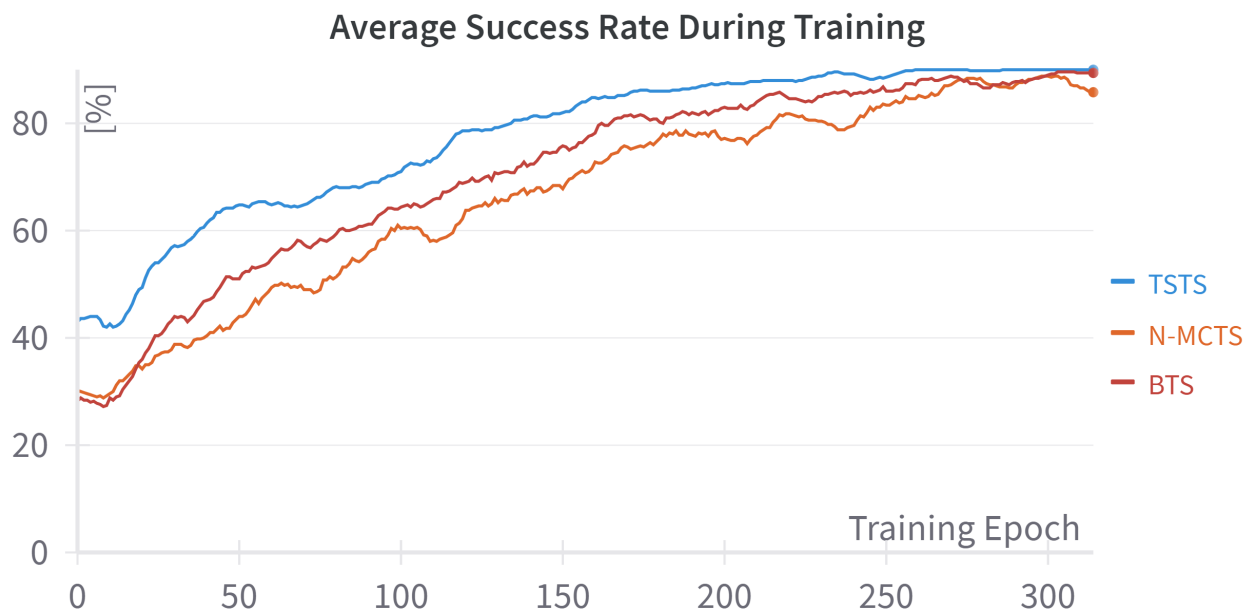

Figure 6. Success rate on train domains during the training

B.3. Computing the Max-Backup

The calculation of the distribution of max of several independent random variables is given by

$$P(\max(X_1, ..., X_n) \leq a) = P(X_1 \leq a \text{ ... and } X_n \leq a) = P(X_1 \leq a, \text{ ... } X_n \leq a) = \prod_{i=1}^{n} P(X_i \leq a),$$

where in the last equality we use the independence assumption.

To perform this calculation in practice we use Algorithm 3 with $M = 50$ bins linearly spaced starting for each individual CDF from 0.001 and ending at 0.999, which in the case of normal distribution covers ± 3 standard deviations around the mean of the distribution.

Algorithm 3 Max-Backup

Require: individual cdf distributions $\{C_i\}_{i=1}^N$, given at M bins $b_i = \{b_1^i, \ldots, b_M^i\}$ for each C_i.
    first_bin := $\max_i (b_1^i)$
    last_bin := $\max_i (b_M^i)$
    all_bins := linspace(first_bin, last_bin, M)

    for i = 1,2,...,N **do**
        $C_i^{interp} :=$ interpolate(C_i over points all_bins)
    end for

    MaxDistributionCdf := $\prod_{i=1}^N C_i^{interp}$
    return all_bins, MaxDistributionCdf

B.4. Approximations

When performing the forward sampling in TSTS one has to sample a general pdf $P(Q)$ resulting from the distribution of the max-backup. An approximate calculation is to sample a Gaussian distribution with the expectation and variance of $P(Q)$. Comparing the performance we see little difference between the exact and approximate sampling, so we opted to use the approximate sampling.

We adopt a similar approximation also for `BTS` where instead of calculating the exact percentile $\rho(\alpha, P)$ for a general $P(Q)$ we calculate it assuming a Gaussian distribution with the expectation and variance of $P(Q)$. Comparison with the exact calculation showed little difference in results.

C. Standard Deviation Estimation via Ensemble of Neural Networks

In this section we report in Figure 7 the results using an ensemble of neural networks to estimate the uncertainty (see Section 2.5) for details. For this evaluation, we used an ensemble $K = 5$ models, where all models are initialized with a random set of weights, and trained similarly.

The results using this method are similar to the ones reported in 4.1. We do note that using the ensemble improves the predictions of $Q(s, a)$ (by taking the average of the different models), which mainly improves the performance of `N-MCTS` on test domains.
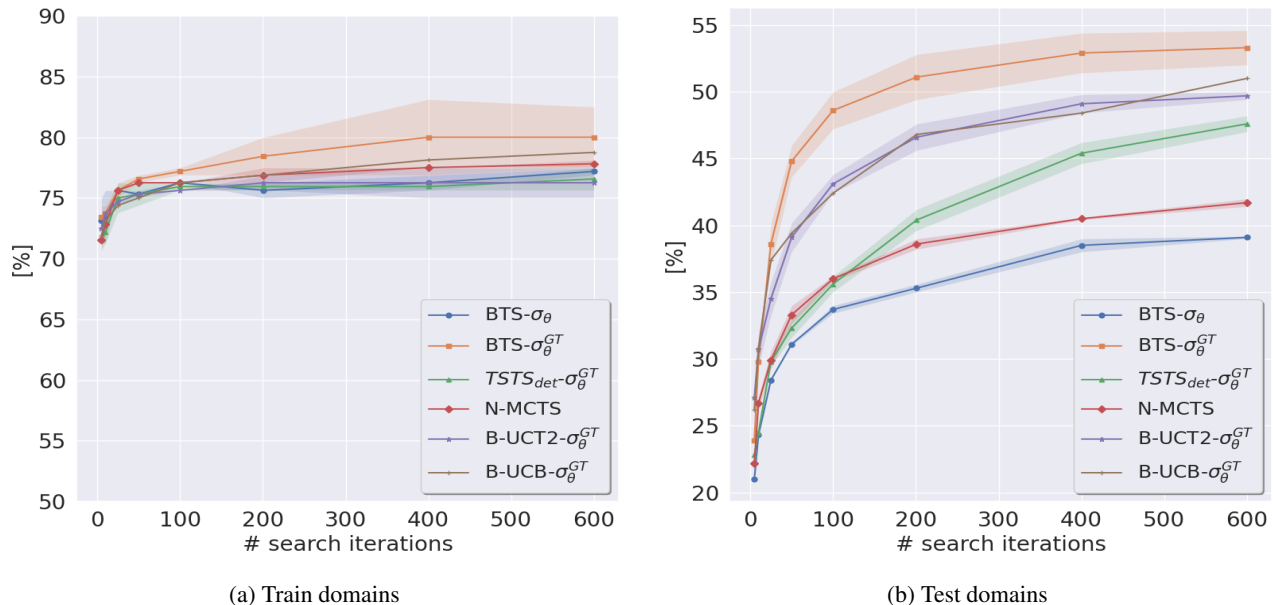
(a) Train domains

(b) Test domains

Figure 7. Comparison of Different Planners Using an Ensemble of NN

D. Evaluation on ProcGen Leaper

In this section we describe additional results on another ProcGen environment, the Leaper environment. In this game, depicted in Figure 8, a frog starts at the bottom of the screen and must get to the finish line. To achieve that, it has to pass road lanes while not being hit by passing cars, and then cross the river by jumping onto wooden logs. If the frog is hit by a car or falls into the river the game terminates, while reaching the finish line yields a reward of 10. Since we set a discount factor of 1 in all of our experiments, we modified the Leaper default reward function such that an *Up* action receives a reward of -0.1, while all other actions receive a reward of -0.2 each time step until the game terminates. This modification is done to encourage the agent to reach the goal quickly, instead of stalling and assuming it will obtain the same accumulated reward in the future. Unlike the maze, this environment is dynamic, hence the frog also has a *Wait* action in addition to moving in four directions.

The original Leaper domain exhibited a very small train-test gap in previous work (Cobbe et al., 2020). To generate a significant train-test gap, we manually divided Leaper instances into train and test datasets, such that train instances have at most two road lanes and two river lanes (see an example in Figure 8a), while test instances can have more (Figure 8b). We used $N_{train} = 100$ for training of the neural network and $N_{test} = 100$ for evaluation. We trained the NN for 60 epochs in a similar manner to the described in Section B.2.

We tested `BTS` against `N-MCTS` on the test instances, where each planner is evaluated for $k = 25$ time steps, where at each time step a search is performed and an action is committed and executed, according to the online planning scheme described in Section 2. To solve an instance, the frog must reach the finish line within the $k = 25$ time steps (and obviously not get hit by a car or fall into the river before that). To estimate GT uncertainty values for `BTS` we use an A^* search from each state where the vertical distance of the agent from the finish line is used as an admissible heuristic.

Figure 9 shows the success rate (i.e., the percentage of solved instances) of `BTS` with and without GT uncertainty estimates against `N-MCTS`. Even without GT estimates, `BTS` significantly outperforms `N-MCTS`, where when incorporating GT, `BTS` can achieve a success rate $> 85\%$ on the test set.

Notably, the neural network uncertainty estimates on Leaper are good enough to yield a significant improvement, differently from the maze domain. We explain this by observing that in Leaper, some of the uncertainty is aleatoric, for example, the uncertainty about whether a log is going to be spawned in the next frame or not. This uncertainty is similar in training and testing, and is easier to capture by training the neural network using the MLE loss.
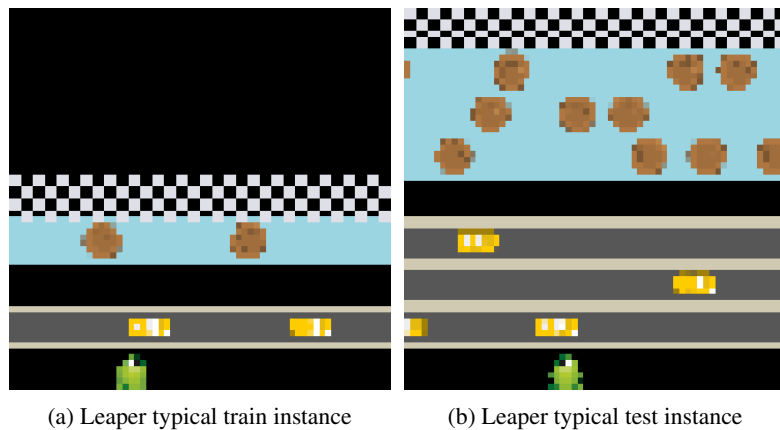
(a) Leaper typical train instance

(b) Leaper typical test instance

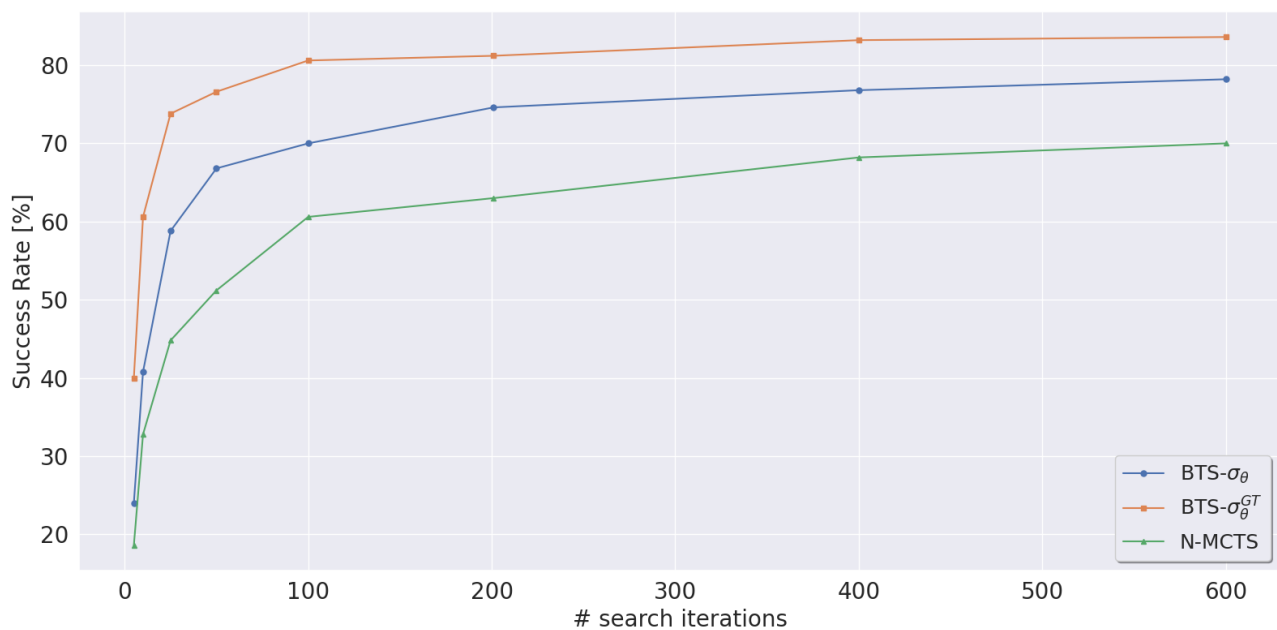*Figure 8.* ProcGen Leaper environment

Figure 9. Test Results on ProcGen Leaper

E. Sensitivity to Hyper-Parameters

In this section we provide the results of an ablation studying the hyper parameters for the different planners, and choosing the best ones for each planner for a fare comparison.

E.1. B-UCB

The B-UCB algorithm selects actions to explore in the tree search, based on quantiles of the $Q(s, a)$ posterior distributions, according to formula 6. The quantile level $\alpha(s)$ depends on the number of visits $N(s)$ and given by:

$$\alpha(s) = 1 - \frac{\beta}{N(s)}.$$

We tested several values of β as depicted in Figure 10, and found out that $\beta = 0.5$ gives the best results.
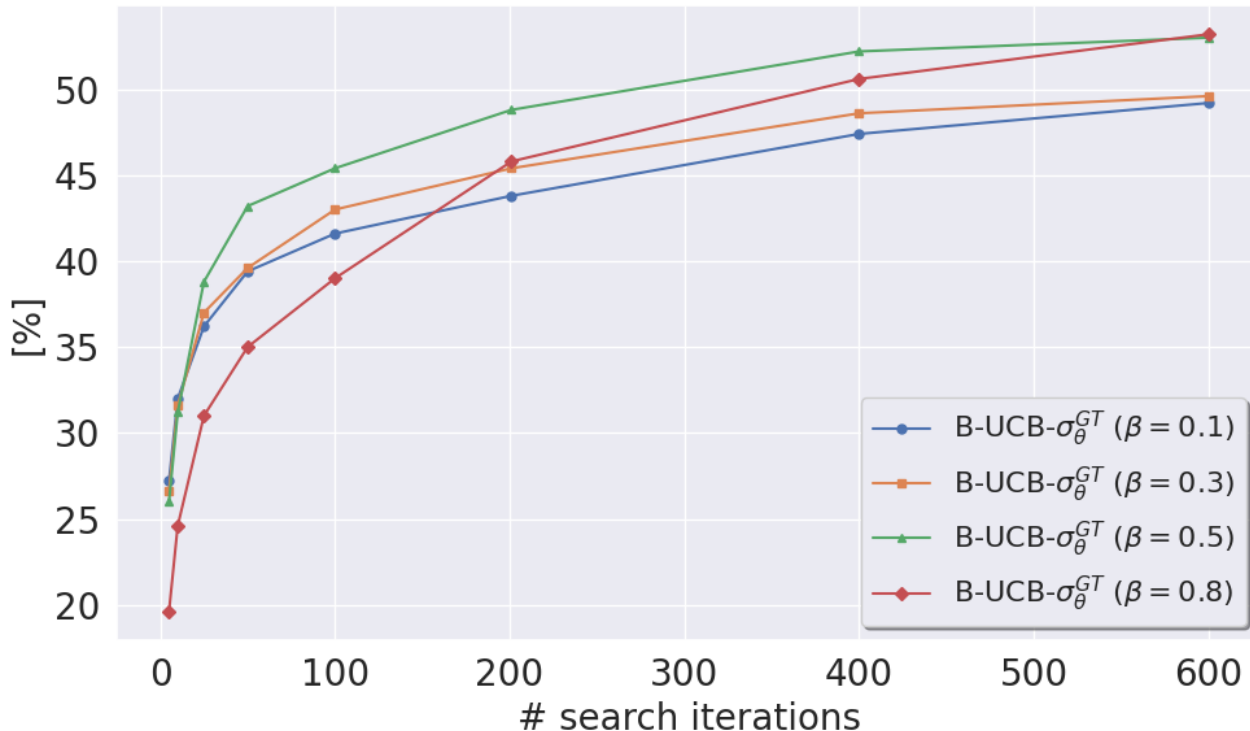
Figure 10. `B-UCB` β ablation

E.2. `BTS`

Our suggested `BTS` algorithm also selects actions based on quantiles of the $Q(s, a)$ posterior distributions, according to formula 6. However, here the quantile level $\alpha(s)$ depends on two hyper-parameters α_0 and β by:

$$\alpha(s) = 1 - (1 - \alpha_0) \cdot e^{-\frac{N(s)-1}{\beta}}.$$

We tested the cross product of the following values for each parameter: $\alpha_0 = [0.1, 0.3, 0.5, 0.8]$, $\beta = [0.5, 1, 3, 8]$ (i.e., 16 different choices) and report the results in Figure 11. We found out that $\alpha_0 = 0.5$, $\beta = 3$ gives the best results.

E.3. `N-MCTS` with stochastic SoftMax action commitment

In SoftMax action commitment, we choose an action to perform in the environment by converting $Q(s, a)$ values to probabilities using a SoftMax operation. We tested several values for the SoftMax temperature and report the results in Figure 12. We found that using a temperature of 2.0 yields the best performance.

E.4. `BTS` with stochastic SoftMax action commitment

Similarly, we tested several values of the temperature when using SoftMax action commitment with `BTS`. Results are shown in Figure 13. We found that the best temperature value in this case is 1.0, however for simplicity we used the value 2.0 here as well (similar to `N-MCTS` with SoftMax action commitment), and note that using a value of 1.0 would improve the results of `BTS` shown in Figure 4c.

E.5. Comparison with `W-MCTS` and `DNG-MCTS`

In this section we provide a comparison between our suggested algorithms, and two previously suggested methods that share similar ideas to ours. The first method is the Dirichlet-NormalGamma MCTS (`DNG-MCTS`) algorithm suggested by Bai et al. (2013). In `DNG-MCTS`, the selection rule of each node of the MCTS search tree is replaced with Thompson
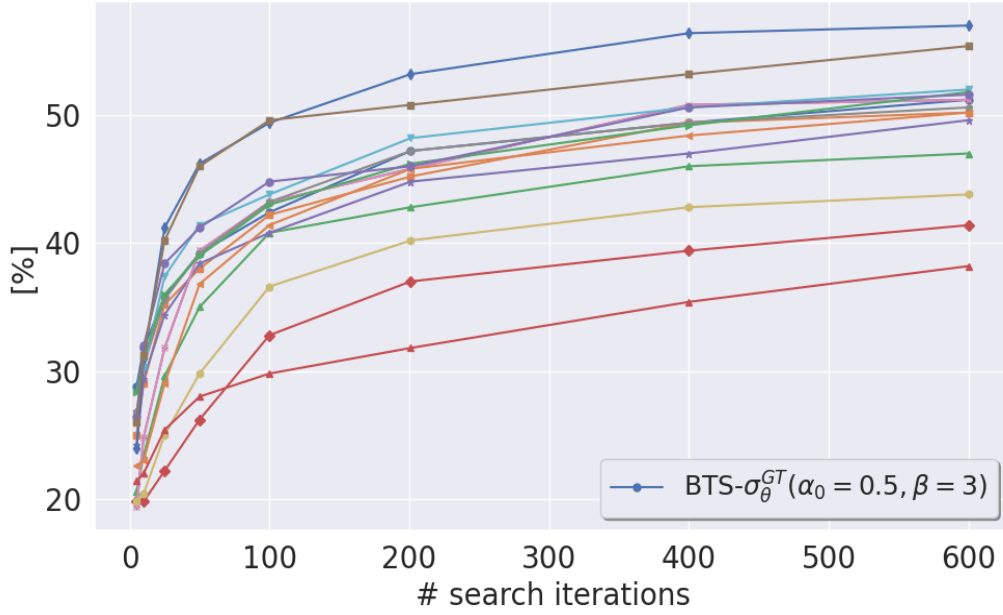
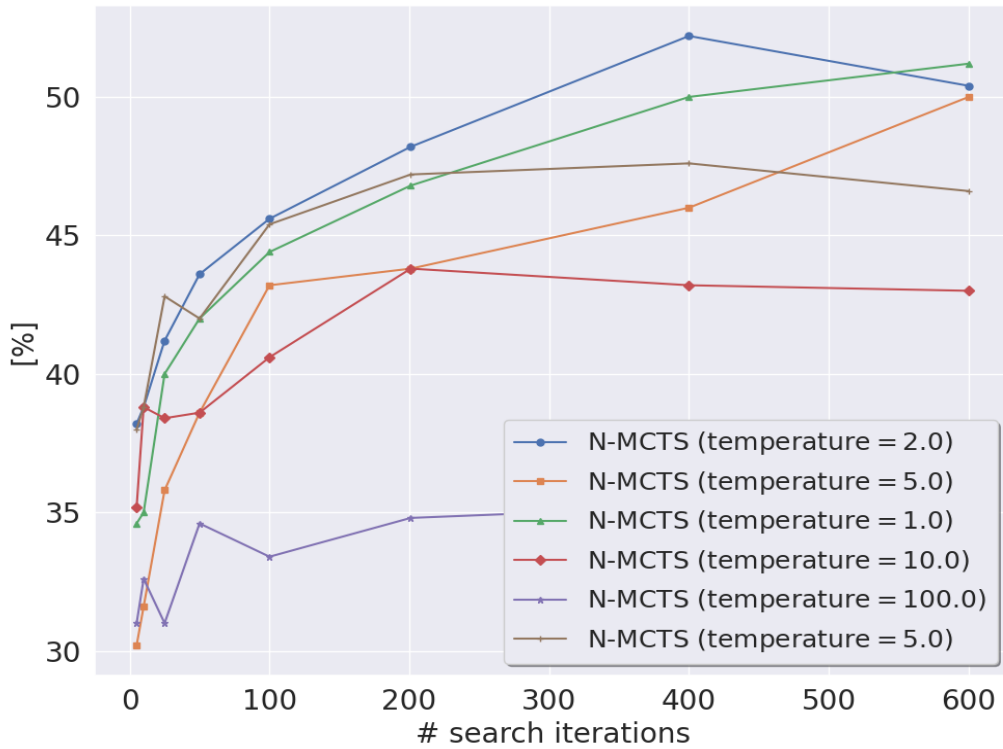*Figure 11.* `BTS` α_0 and β ablation

Figure 12. `N-MCTS` sensitivity to SoftMax temperature

sampling, assuming that Q values are distributed as a mixture-of-Gaussians, i.e., $N(\mu, \frac{1}{\tau})$, where $\tau = \frac{1}{\sigma^2}$ and (μ, τ) follows
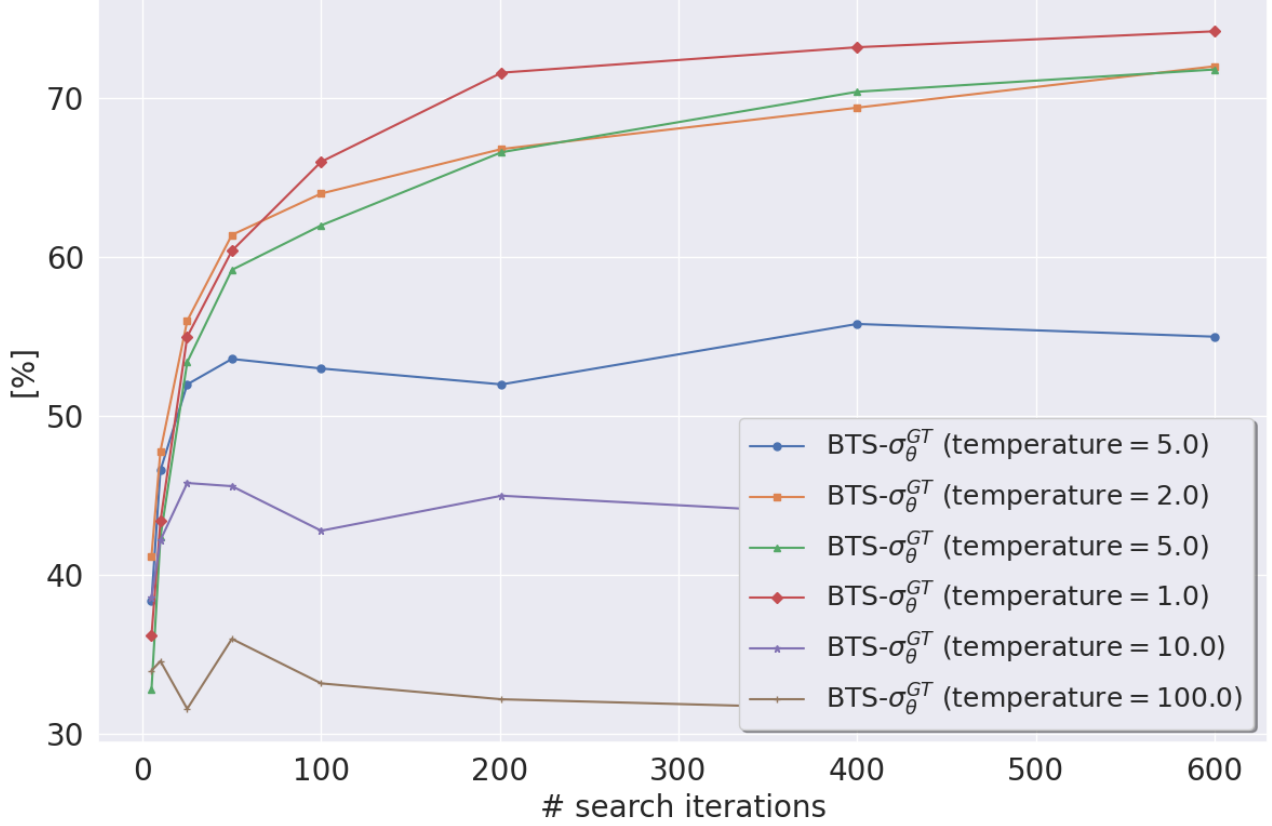
Figure 13. `BTS` sensitivity to SoftMax temperature

a NormalGamma distribution defined by parameters $\langle\mu_0, \lambda, \alpha, \beta\rangle$ and a pdf of the form:

$$f(\mu, \tau|\mu_0, \lambda, \alpha, \beta) = \frac{\beta^\alpha\sqrt{\lambda}}{\Gamma(\alpha)\sqrt{2\pi}}\tau^{\alpha-\frac{1}{2}}e^{-\beta\tau}e^{-\frac{\lambda\tau(\mu-\mu_0)^2}{2}},$$

where $\Gamma(\alpha)$ is the Gamma function. `DNG-MCTS` then uses the following equations in the backup procedure, given the backed-up value r:

$$\begin{aligned}
\alpha &\leftarrow \alpha + 0.5 \\
\beta &\leftarrow \beta + (\lambda(r-\mu_0)^2/(\lambda+1))/2 \\
\mu_0 &\leftarrow (\lambda\mu_0 + r)/(\lambda+1) \\
\lambda &\leftarrow \lambda + 1.
\end{aligned} \tag{8}$$

More specifically, since the evaluated maze environment is deterministic, the Q value at each node is distributed as a single Gaussian, hence we don't need the Dirichlet distribution for sampling the weight of each Gaussian, as described in the original `DNG-MCTS` paper. The implementation of `DNG-MCTS` is then almost identical to `N-MCTS`, except the following changes: a node's value is sampled from the NormalGamma distribution given it's current $\langle\mu_0, \lambda, \alpha, \beta\rangle$ values, and a backup for updating these values is performed using equations 8. In addition, for any un-visited node, its value is approximated using the NN, instead of the simulation using a rollout policy done in the original `DNG-MCTS` algorithm. Following the suggestion in Bai et al. (2013), for each node in `DNG-MCTS`, α is initialized to 1, and μ is initialized to 0 (we also tried initializing μ from the NN and found out it performed worse). We performed a hyper-parameter tuning and found that initializing $\beta = 100$ and $\lambda = 0.001$ performed best in the maze setting.

The second method we compared against is Wasserstein MCTS (`W-MCTS`), suggested by Dam et al. (2023). `W-MCTS` models value distributions as Gaussians similar to our work, and propagates the uncertainty of the estimate of value nodes

across the tree using a backup operator that computes value nodes as Wasserstein barycenters of children action-value nodes. Both an Optimistic Selection (similar to the UCT formula by replacing exploration term by the standard deviation) and Thompson sampling are proposed as action-selection strategies. We found that using Thompson sampling for action selection in `W-MCTS` outperforms using Optimistic Selection, therefore we compare it to our `TSTS` algorithm. For both algorithms (`W-MCTS` and `TSTS`) we use the deterministic variant (see Section 4). In practice, the implementation of `W-MCTS` with Thompson sampling is directly derived from our `TSTS` implementation by only replacing the backup method. To set the p parameter we performed a hyper-parameter tuning and found that $p = 1$ worked best in the maze setting.

We evaluated both methods on the test dataset of the ProcGen maze environment, using the same neural network used in our method.

Our results show (Figure 14) that `DNG-MCTS` (with the neural network value backup) is comparable to standard `N-MCTS` (though much worse when search budget is small). Since `DNG-MCTS` is a stochastic algorithm, we compare it to `TSTS` and observe that it performs worse. With GT uncertainty, `TSTS` significantly outperforms `DNG-MCTS`.

Additionally, Figure 15 shows that `W-MCTS` is comparable (but slightly worse on most search iterations) to `TSTS` without GT uncertainty, but significantly outperformed by `TSTS` with GT uncertainty for all search iterations. We hypothesize that our Max-Backup, which was naturally derived for this task, is the reason for this result.
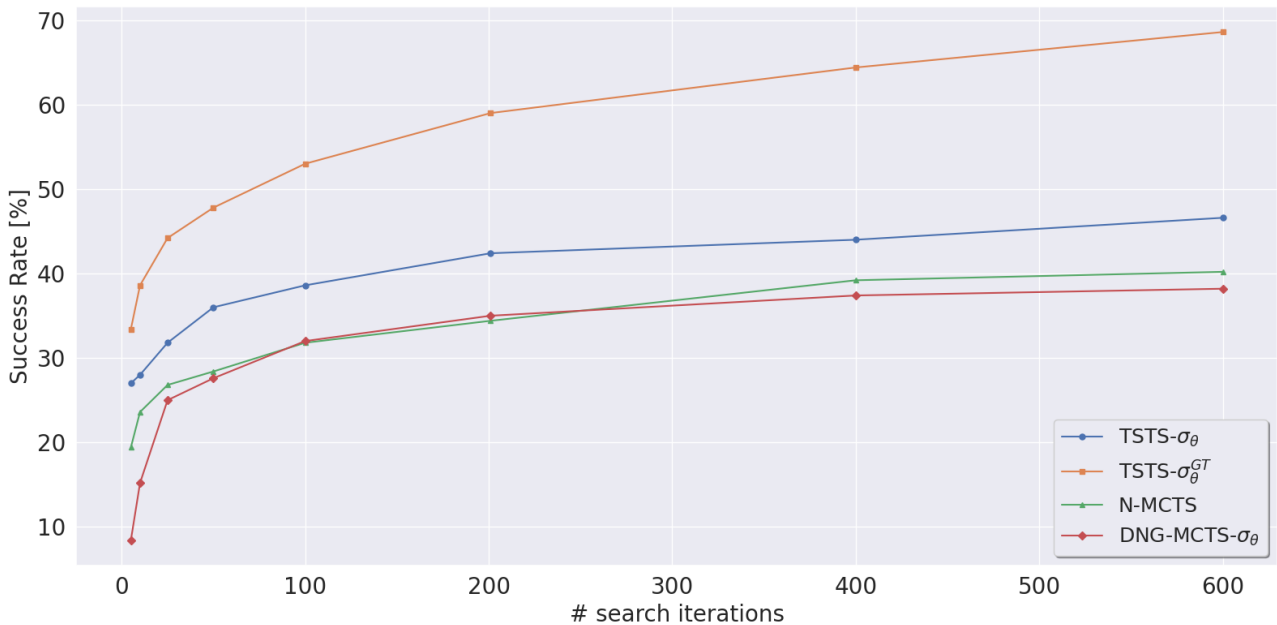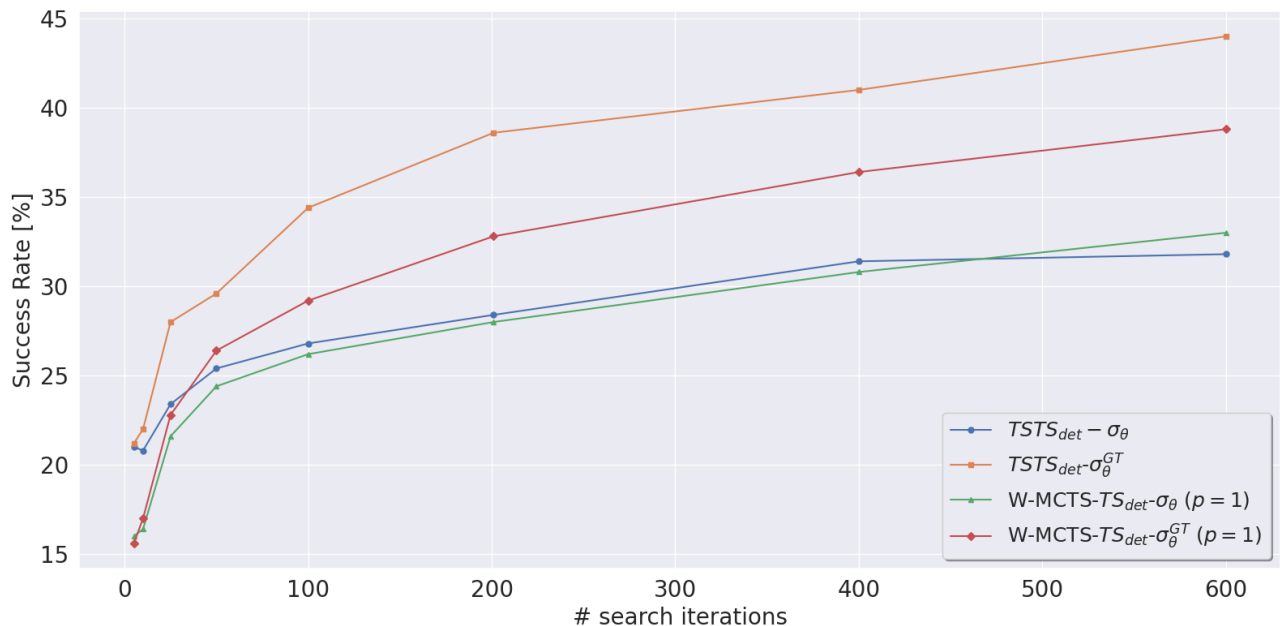

Figure 14. `TSTS` vs. `DNG-MCTS`

Figure 15. TSTS vs. W-MCTS

F. Tree Search Analysis

In this section we emphasize the potential of our suggested BTS algorithm compared to N-MCTS. Specifically, we show how BTS exploits the uncertainty of the NN (ground truth in this example) to perform a more informed exploration of the search tree, allowing it to explore the tree to a significant depth, eventually leading to finding a rewarded state.

Figure 16 shows an example maze, where the cheese (depicted in yellow) is 11 steps away from the mouse (depicted in gray). Figure 17 shows the predicted $Q(s, a)$ values from the root output by the NN. We see that the NN predictions are pretty accurate in this case, implying that Up is the correct action in this state. Both planners are given a search budget of 25 iterations in this example.

Figures 18 and 19 show the trees explored by N-MCTS and BTS, respectively. N-MCTS follows the UCB formula, causing it to explore the tree in a balanced manner, since there are no significant differences in the $Q(s, a)$ values of the actions from the root. BTS on the other hand, exploits the uncertainty (or more accurately, the certainty) in the predictions to explore the tree to a much more significant depth, by always selecting the correct (Up) action from the root, and similarly at following nodes. In other words, all other actions from the root, other than Up, yield low $Q(s, a)$ values with a very high certainty, causing BTS to only exploit the correct action. This eventually lead to finding a terminal (rewarded) state (at leaf node 17). We observe similar behavior in the Leaper environment as well.
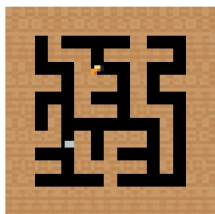

Figure 16. Example maze

Action	Predicted $Q(s, a)$	$Q^{\text{GT}}(s, a)$	$\sigma_\theta^{\text{GT}}(s, a)$
Up	-1.278	-1.0	0.278
Down	-3.03	-3.0	0.03
Right	-2.197	-2.0	0.197
Left	-2.375	-2.0	0.375

Figure 17. Predicted $Q(s, a)$, $Q(s, a)^{GT}$ and $\sigma_\theta^{\text{GT}}(s, a)$ values at the root node
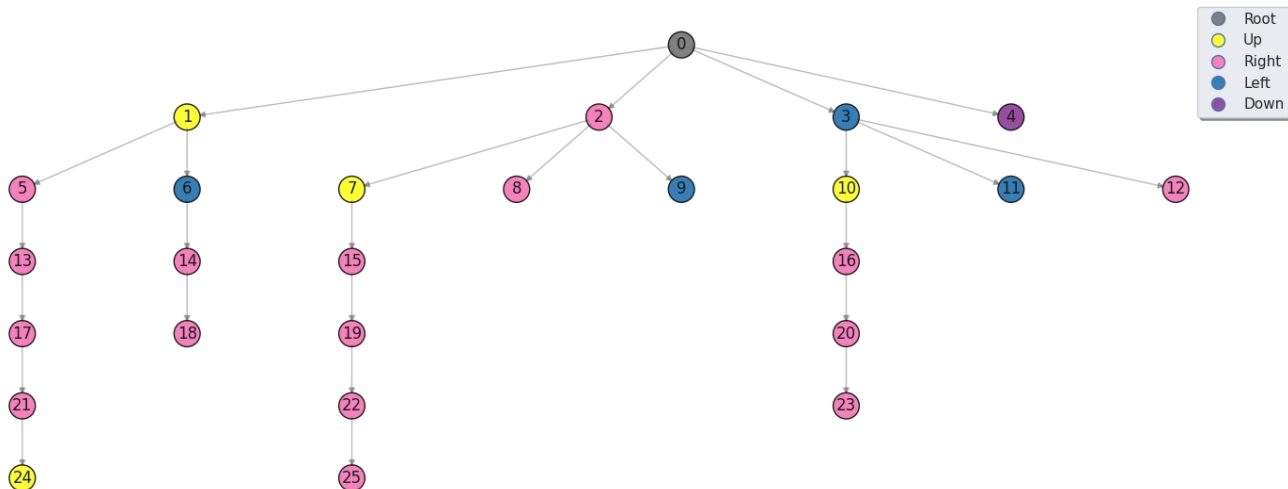
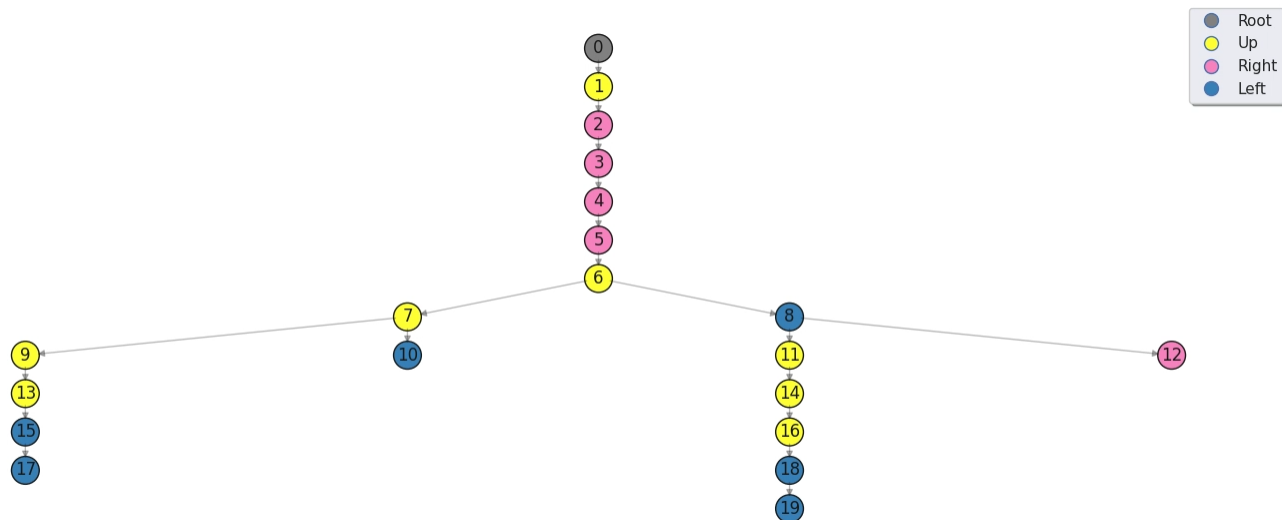*Figure 18.* Tree opened by `N-MCTS` in the Maze environment

Figure 19. Tree opened by `BTS` in the Maze environment

G. Proofs

For our analysis, we will consider an equivalent regret definition, based on a random variable sequence Y_t. Recall that at time t, the observation is O_t, and the information available *after* observing O_t is $\mathcal{F}_{t+1} = \{\mathcal{F}_t, z_t, r(z_t)\}$.

Let $Y_t \in \mathbb{R}^{|\mathcal{Z}_t|}$ be a random variable that is distributed as follows. For each $z \in \mathcal{Z}_t$, let $\{\mathcal{F}_t, z, r(z)\}$ denote the information set of the history $\mathcal{F}_t$, and the observation that results from selecting z at time t. We let $P(Y_t(z)|\mathcal{F}_t) = P(Q_0(s_0, A_{\text{root}}(z))|\mathcal{F}_t, z, r(z))$, that is, for each possible leaf z, $Y_t(z)$ is drawn from the posterior reward *after* observing that leaf. We furthermore assume that $Y_t(z) = f(\mathcal{F}_t, z, r(z), \mathcal{N}_t)$, where f is some deterministic function, and $\mathcal{N}_t$ is a noise sequence that is independent of the past, and of A^*. This is a technical assumption that essentially states that sampling from the posterior is independent of the decision process. Note that in general, Y_t is not necessarily i.i.d., and may depend on the action selection policy. The next proposition shows that we can define the Bayesian regret using Y_t.

Proposition 2. *We have that* $\mathbb{E}[Regret(T)] = \mathbb{E}\left[\sum_{t=1}^{T}[Y_t(z_t^*) - Y_t(z_t)]\right]$.

Proof. Using the tower rule:

$$\mathbb{E}\left[\sum_{t=1}^{T} Q_0(s_0, A_{\text{root}}(z_t))\right] = \sum_{t=1}^{T} \mathbb{E}\left[\mathbb{E}\left[Q_0(s_0, A_{\text{root}}(z_t))|\, \mathcal{F}_{t+1}\right]\right]$$

$$= \sum_{t=1}^{T} \mathbb{E}\left[\mathbb{E}\left[Y_t(z_t)|\, \mathcal{F}_{t+1}\right]\right]$$

$$= \mathbb{E}\left[\sum_{t=1}^{T} [Y_t(z_t)]\right].$$

Let us define a policy $\tilde{\pi}_\tau$ that until time τ selects action z_t according to π, and at time τ selects z_τ^*. Let $\tilde{\mathcal{F}}_{t+1}$ denote the history of following policy $\tilde{\pi}_\tau$ for t steps. By definition, $\tilde{\mathcal{F}}_{t+1} = \{\mathcal{F}_t, z_t^*, r(z_t^*)\}$

$$\mathbb{E}\left[\sum_{t=1}^{T} [Q_0(s_0, A^*)]\right] = \sum_{t=1}^{T} \mathbb{E}\left[Q_0(s_0, A^*)\right]$$

$$= \sum_{t=1}^{T} \mathbb{E}\left[\mathbb{E}\left[Q_0(s_0, A^*)|\, \tilde{\mathcal{F}}_{t+1}\right]\right]$$

$$= \sum_{t=1}^{T} \mathbb{E}\left[\mathbb{E}\left[Q_0(s_0, A_{\text{root}}(z_t^*))|\, \tilde{\mathcal{F}}_{t+1}\right]\right]$$

$$= \sum_{t=1}^{T} \mathbb{E}\left[\mathbb{E}\left[Y_t(z_t^*)|\, \tilde{\mathcal{F}}_{t+1}\right]\right]$$

$$= \mathbb{E}\left[\sum_{t=1}^{T} [Y_t(z_t^*)]\right].$$

$\square$

G.1. Proof of Theorem 1

Proof. Our proof is broken into several parts, similarly to the analysis in (Russo & Van Roy, 2016). We begin with several information theoretic definitions. We then define the information ratio, and derive a general regret bound, and finally bound the information ratio in our case.

The Shannon entropy of a random variable X is

$$\mathcal{H}(X) = \sum_x -P(X = x) \log P(X = x).$$

The mutual information $\mathcal{I}(X; Y)$ between two random variables X, Y satisfies

$$\mathcal{I}(X; Y) = \mathcal{H}(X) - \mathcal{H}(X|Y) = \mathcal{H}(Y) - \mathcal{H}(Y|X) = \mathcal{I}(Y; X).$$

The KL-divergence between two distributions is $D(P||Q) = \int \log\left(\frac{dP}{dQ}\right) dP$. It holds that (Fact 6 in (Russo & Van Roy, 2016))

$$\mathcal{I}(X; Y) = \sum_x P(X = x) D(P(Y|X = x)||P(Y)).$$

Let $P_t(X) = P(X|\mathcal{F}_t)$, and $\mathbb{E}_t[\cdot] = \mathbb{E}[\cdot|\mathcal{F}_t]$. Similarly,

$$\mathcal{H}_t(X) = \sum_x -P_t(X = x) \log P_t(X = x),$$

and $\mathcal{I}_t(X; Y) = \mathcal{H}_t(X) - \mathcal{H}_t(X|Y)$.

As discussed in Section 3.1 of (Russo & Van Roy, 2016), we have that

$$\mathbb{E}\left[\mathcal{I}_t(X;Y)\right] = \mathcal{I}(X;Y|\mathcal{F}_t). \tag{9}$$

Let $Y_{t,a}$ denote the a's component of Y_t. Define the information ratio,

$$\Gamma_t = \frac{\mathbb{E}_t\left[Y_{t,z_t^*} - Y_{t,z_t}\right]^2}{\mathcal{I}_t(z_t^*;(z_t,Y_{t,z_t}))}. \tag{10}$$

The following proposition bounds the regret using a bound on the information ratio.

Proposition 3. *If $\Gamma_t \leq \bar{\Gamma}$ almost surely, we have that*

$$\mathbb{E}\left[Regret(T)\right] \leq \sqrt{\bar{\Gamma}\mathcal{H}(z^*)T}.$$

Proof. We have that,

$$
\begin{aligned}
\mathbb{E}\left[\text{Regret}(T)\right] &= \mathbb{E}\left[\sum_{t=1}^T [Y_t(z_t^*) - Y_t(z_t)]\right] \\
&= \mathbb{E}\left[\sum_{t=1}^T \mathbb{E}_t\left[Y_t(z_t^*) - Y_t(z_t)\right]\right] \\
&= \mathbb{E}\left[\sum_{t=1}^T \sqrt{\Gamma_t \mathcal{I}_t(z_t^*;(z_t,Y_{t,z_t}))}\right] \\
&\leq \sqrt{\bar{\Gamma}}\mathbb{E}\left[\sum_{t=1}^T \sqrt{\mathcal{I}_t(z_t^*;(z_t,Y_{t,z_t}))}\right] \\
&\leq \sqrt{\bar{\Gamma}}\sqrt{T\mathbb{E}\left[\sum_{t=1}^T \mathcal{I}_t(z_t^*;(z_t,Y_{t,z_t}))\right]} \\
&= \sqrt{\bar{\Gamma}T\mathbb{E}\left[\sum_{t=1}^T \mathcal{I}_t(z_t^*;(z_t,Y_{t,z_t}))\right]},
\end{aligned}
$$

where the first equality is by Proposition 2. The second equality is from the tower rule. The third equality is by definition (10). The second inequality is by the Cauchy–Schwarz (CS) inequality, as follows. Define the linear inner product $< u, v > = \mathbb{E}[\sum_t u_t v_t]$. For $u = [1,\ldots,1]$ and $v = [\sqrt{\mathcal{I}_1},\ldots,\sqrt{\mathcal{I}_T}]$ we have $< u, v > = \mathbb{E}\left[\sum_{t=1}^T \sqrt{\mathcal{I}_t}\right]$, $< u, u > = \mathbb{E}\left[\sum_{t=1}^T 1\right] = T$, and $< v, v > = \mathbb{E}\left[\sum_{t=1}^T \mathcal{I}_t\right]$. Then, from CS, $\mathbb{E}\left[\sum_{t=1}^T \sqrt{\mathcal{I}_t}\right] \leq \sqrt{T\mathbb{E}\left[\sum_{t=1}^T \mathcal{I}_t\right]}$.

Next, define $Z_t = (O_t, z_t, Y_{t,z_t})$. We have that

$$\mathbb{E}\left[\mathcal{I}_t(z_t^*;Z_t)\right] = \mathcal{I}(z_t^*;Z_t|Z_1,\ldots,Z_{t-1}).$$

Therefore,

$$
\mathbb{E}\left[\sum_{t=1}^{T}\mathcal{I}_t(z_t^*; z_t, Y_{t,z_t})\right] \leq \mathbb{E}\left[\sum_{t=1}^{T}\mathcal{I}_t(z_t^*; Z_t)\right]
$$

$$
= \mathbb{E}\left[\sum_{t=1}^{T}\mathcal{I}_t(z_t^*; O_t)\right]
$$

$$
\leq \mathbb{E}\left[\sum_{t=1}^{T}\mathcal{I}_t(z^*; O_t)\right]
$$

$$
= \sum_{t=1}^{T}\mathcal{I}(z^*; O_t|O_1,\dots,O_{t-1})
$$

$$
= \mathcal{I}(z^*; (O_1,\dots,O_T))
$$

$$
= \mathcal{H}(z^*) - \mathcal{H}(z^*|O_1,\dots,O_T)
$$

$$
\leq \mathcal{H}(z^*),
$$

where the first inequality is since Z_t contains Y_{t,A_t}, the first equality is by the definition of Y_t, which, given the history, is independent of z_t^*, and the third equality is from the chain rule of mutual information (Fact 5 in (Russo & Van Roy, 2016)). The second inequality is by the data processing inequality, since z_t^* is a deterministic function of z^*. Combining the results above gives the desired result. $\qquad\square$

We proceed, similarly to (Russo & Van Roy, 2016), to derive an equivalent form of the information ratio, which will facilitate further analysis.

Proposition 4. *We have that*

$$
\mathcal{I}_t(z_t^*; (z_t, Y_{t,z_t})) = \sum_{a,a^*} P_t(z_t^* = a^*)P_t(z_t^* = a)\left[D(P_t(Y_{t,a}|z_t^* = a^*)||P_t(Y_{t,a}))\right],
$$

and

$$
\mathbb{E}_t\left[Y_{t,z_t^*} - Y_{t,z_t}\right] = \sum_{a} P_t(z_t^* = a)\left(\mathbb{E}_t\left[Y_t|z_t^* = a\right] - \mathbb{E}_t\left[Y_t\right]\right).
$$

Proof. We have

$$
\mathcal{I}_t(z_t^*; (z_t, Y_{t,z_t})) = \mathcal{I}_t(z_t^*; z_t) + \mathcal{I}_t(z_t^*; Y_{t,z_t}|z_t)
$$

$$
= \mathcal{I}_t(z_t^*; Y_{t,z_t}|z_t)
$$

$$
= \sum_{a} P_t(z_t = a)\mathcal{I}_t(z_t^*; Y_{t,z_t}|z_t = a)
$$

$$
= \sum_{a} P_t(z_t = a)\mathcal{I}_t(z_t^*; Y_{t,a}))
$$

$$
= \sum_{a} P_t(z_t = a)\sum_{a^*} P_t(z_t^* = a^*)D(P_t(Y_{t,a}|z_t^* = a^*)||P_t(Y_{t,a}))
$$

$$
= \sum_{a,a^*} P_t(z_t = a)P_t(z_t^* = a^*)D(P_t(Y_{t,a}|z_t^* = a^*)||P_t(Y_{t,a}))
$$

$$
= \sum_{a,a^*} P_t(z_t^* = a)P_t(z_t^* = a^*)D(P_t(Y_{t,a}|z_t^* = a^*)||P_t(Y_{t,a})),
$$

25

where the last equality uses the probability matching of TS: $P_t(z_t = a) = P_t(z_t^* = a)$. Also,

$$
\begin{aligned}
\mathbb{E}_t\left[Y_{t,z_t^*} - Y_{t,z_t}\right] &= \sum_a P_t(z_t^* = a)\mathbb{E}_t\left[Y_{t,a}|z_t^* = a\right] - \sum_a P_t(z_t = a)\mathbb{E}_t\left[Y_{t,a}|z_t = a\right] \\
&= \sum_a P_t(z_t^* = a)\left(\mathbb{E}_t\left[Y_{t,a}|z_t^* = a\right] - \mathbb{E}_t\left[Y_{t,a}|z_t = a\right]\right), \\
&= \sum_a P_t(z_t^* = a)\left(\mathbb{E}_t\left[Y_{t,a}|z_t^* = a\right] - \mathbb{E}_t\left[Y_{t,a}\right]\right).
\end{aligned}
$$

where the second equality uses the probability matching of TS: $P_t(z_t = a) = P_t(z_t^* = a)$, and the third equality is since given the history, Y_t is independent of z_t. $\qquad\square$

We will use the following lemma.

Lemma 1. *Let P, Q be two distributions of X with support $[-B, B]$, such that P is absolutely continuous with respect to Q. Then,*

$$
\mathbb{E}_P[X] - \mathbb{E}_Q[X] \le B\sqrt{\frac{1}{2}D(P\|Q)}.
$$

Proof. We have

$$
\mathbb{E}_P[X] - \mathbb{E}_Q[X] = \sum_x x(P(x) - Q(x)) \le \sum_x |x| |P(x) - Q(x)| \le B \max_x |P(x) - Q(x)| \le B\sqrt{\frac{1}{2}D(P\|Q)},
$$

where the last inequality is Pinsker's inequality. $\qquad\square$

We are finally ready to bound the information ratio.

Proposition 5. *We have that $\Gamma_t \le \frac{|\mathcal{Z}|R_{max}^2 H^2}{2}$ almost surely.*

Proof. We have

$$
\begin{aligned}
\mathbb{E}_t\left[Y_{t,z_t^*} - Y_{t,z_t}\right]^2 &= \left(\sum_a P_t(z_t^* = a)\left(\mathbb{E}_t\left[Y_{t,a}|z_t^* = a\right] - \mathbb{E}_t\left[Y_{t,a}\right]\right)\right)^2 \\
&\le |\mathcal{Z}| \sum_a P_t(z_t^* = a)^2 \left(\mathbb{E}_t\left[Y_{t,a}|z_t^* = a\right] - \mathbb{E}_t\left[Y_{t,a}\right]\right)^2 \\
&\le |\mathcal{Z}| \sum_{a,a^*} P_t(z_t^* = a)P_t(z_t^* = a^*)\left(\mathbb{E}_t\left[Y_{t,a}|z_t^* = a\right] - \mathbb{E}_t\left[Y_{t,a}\right]\right)^2 \\
&\le \frac{|\mathcal{Z}|R_{max}^2 H^2}{2} \sum_{a,a^*} P_t(z_t^* = a)P_t(z_t^* = a^*)D\left(P_t(Y_{t,a}|z_t^* = a)\|P_t(Y_{t,a})\right) \\
&= \frac{|\mathcal{Z}|R_{max}^2 H^2}{2}\mathcal{I}_t(z_t^*; (z_t, Y_{t,z_t})),
\end{aligned}
$$

where the first equality is by Proposition 4. The first inequality is by CS, as follow. Consider the inner product $<u, v> = \sum_a u(a)v(a)$, and let $u = [1, \ldots, 1]$, and $v(a) = P_t(z_t^* = a)\mathbb{E}_t[Y_t|z_t^* = a] - \mathbb{E}_t[Y_t]$. Then by CS, $(\sum_a u(a)v(a))^2 \le \sum_a u(a)^2 \sum_{a'} v(a')^2 = |\mathcal{Z}| \sum_a \left(P_t(z_t^* = a)\mathbb{E}_t[Y_t|z_t^* = a] - \mathbb{E}_t[Y_t]\right)^2$. The second inequality follows from the following fact: let $C(i, j) > 0$. Then $\sum_{i,j} C(i)C(j)D(i)^2 = \sum_{i,j=i} C(i)C(j)D(i)^2 + \sum_{i,j\ne i} C(i)C(j)D(i)^2 \ge \sum_i C(i)^2 D(i)^2$. The third inequality is by Lemma 1, using that $|Y_{t,a}| \le R_{max}H$, since $|Q(s_0, a)| \le R_{max}H$. The last equality is again from Proposition 4. $\qquad\square$

Plugging Proposition 5 in the bound of Proposition 3 completes the proof. $\qquad\square$

G.2. Proof of Proposition 1

We note that for any leaf $z = (\tilde{s}, \tilde{a})$, there is a unique branch leading to it, which we shall denote $b(z) = \{(s_0, a_0), (s_1, a_1), \ldots, (s_k, a_k), (\tilde{s}, \tilde{a})\}$. We shall denote by $b_t^* = b(z_t^*)$ the optimal branch.

In the remainder of this proof, all probabilities are conditioned on $\mathcal{F}_t$. To simplify the notation, we omit this dependence.

From the sequential structure of the branch, we have that

$$P(b(z) = b_t^*) = P((s_0, a_0) \in b_t^*)P((s_1, a_1) \in b_t^*|(s_0, a_0) \in b_t^*) \cdots P((\tilde{s}, \tilde{a}) \in b_t^*|(s_k, a_k) \in b_t^*).$$

To see this, note that $P((s_k, a_k) \in b_t^*|(s_{k-1}, a_{k-1}) \in b_t^*, (s_{k-2}, a_{k-2}) \in b_t^*) = P((s_k, a_k) \in b_t^*|(s_{k-1}, a_{k-1}) \in b_t^*)$, since if (s_{k-1}, a_{k-1}) belongs to the optimal branch, its predecessor (s_{k-2}, a_{k-2}) must also be on the optimal branch.

Observe that if (s_{k-1}, a_{k-1}) is on the optimal branch, then the successor state s_k must also be on the optimal branch. Therefore,

$$P((s_k, a_k) \in b_t^*|(s_{k-1}, a_{k-1}) \in b_t^*) = P(Q(s_k, a_k) \in \arg\max_a Q(s_k, a)).$$

We therefore have that if $P(Q(s, a))$ in Algorithm 1 corresponds to the true posterior for each s, a, then the forward sampling procedure samples a branch from $P(b(z) = b_t^*)$, and equivalently, samples a leaf from $P(z_t^*)$.

We now show by induction that $P(Q(s, a))$ in Algorithm 1 corresponds to the true posterior, which we shall denote here $P_{true}(Q(s, a))$. For any leaf (s, a), by the independence assumption, $P(Q(s, a))$ is independent of other leaves or nodes in the tree, therefore after each update of the algorithm we have $P(Q(s, a)) = P_{true}(Q(s, a))$. Assume that for some node s' and all actions a', we have that $P(Q(s', a')) = P_{true}(Q(s', a'))$. Let s, a be the state-action leading to s'. By definition, $P_{true}(Q(s, a))$ depends only on the decedents of s, a in the tree. We therefore have that

$$P_{true}\left(Q(s, a)\right) = P\left(r(s, a) + \max_{a'}\{Q(s', a')\}\right) = P\left(Q(s, a)\right).$$