Provably Efficient Long-Horizon Exploration in Monte Carlo Tree Search through State Occupancy Regularization

Liam Schramm [1] **Abdeslam Boularias** [1]

Abstract

Monte Carlo tree search (MCTS) has been successful in a variety of domains, but faces challenges with long-horizon exploration when compared to sampling-based motion planning algorithms like Rapidly-Exploring Random Trees. To address these limitations of MCTS, we derive a tree search algorithm based on policy optimization with state occupancy measure regularization, which we call *Volume-MCTS*. We show that count-based exploration and sampling-based motion planning can be derived as approximate solutions to this state occupancy measure regularized objective. We test our method on several robot navigation problems, and find that Volume-MCTS outperforms AlphaZero and displays significantly better long-horizon exploration properties.

1. Introduction

In robotics, sampling-based motion planning (SBMP) algorithms are frequently used instead of reinforcement learning (RL) based methods such as Monte Carlo tree search (MCTS) for long-horizon exploration, due to challenges RL methods face in determining what regions may yield high rewards and how to reach them. While SBMP methods are highly efficient at exploration, they may be slow to converge to near-optimal paths, and do not provide a canonical way to either train or use neural networks to guide search (McMahon et al., 2022). Additionally, SBMP methods require much stronger assumptions than MCTS. They solve the problem of finding the shortest path to a goal region while avoiding obstacles, in a setting with continuous time setting with known and deterministic dynamics, while RL has been used in domains as wide-ranging as video games, autonomous driving, theorem proving, penetration testing, and power grid management (Schrittwieser et al., 2020; Osiński et al., 2021; Lample et al., 2022; Schwartz & Kurniawati, 2019; Zhang et al., 2019). We build on the recent work by Grill et al. (2020) to reveal a mathematical connection between MCTS, regularized policy optimization, and SBMP. We then propose a family of MCTS algorithms based on policy optimization with state occupancy measure regularization, with strong exploration guarantees.

The main contributions of this work are the following: **(1)** We show that both the Voronoi bias of SBMP algorithms and the count-based exploration (CBE) method used in reinforcement learning can be derived as solutions to a state occupancy measure regularization objective. **(2)** We prove that in search trees, for any convex loss function of state occupancy measure, $\mathcal{L}(d^\pi)$, $\mathcal{L}$ can be optimized by independently optimizing the policy at each node. This novel finding makes it possible and efficient to use MCTS-style algorithms for arbitrary regularization of the state occupancy measure. Notably, this is true *only* for trees – general Markov Decision Processes do not have this property (Hazan et al., 2019). **(3)** We derive *Volume-MCTS*, a variant of AlphaZero that uses state occupancy regularization to encourage long-horizon exploration without making the stronger assumptions used in SBMP. We find that this method outperforms a range of reinforcement learning and planning algorithms, including AlphaZero and AlphaZero with CBE, on long-horizon exploration problems. **(4)** We prove non-asymptotic high-probability bounds on Volume-MCTS's exploration efficiency. To the best of our knowledge, this is the first bound of this type to be proven for MCTS-family algorithms.

2. Definitions

Let M be a Markov Decision Process (MDP) with continuous state space S, continuous action space A, reward function $R\colon S \to \mathbb{R}$, discount factor γ, and deterministic transition function $\mathcal{T}\colon S \times A \to S$. We assume S is bounded, measurable, and metrizable. Let T be the set of nodes in a search tree. Let N be defined as $||\ T\ ||$. For any node $n \in T$, let subtree(n) be the subtree of n. Let λ be a regularization coefficient that scales as $\mathcal{O}\left(\frac{1}{\sqrt{N}}\right)$.

Node expansion: Let $\mathcal{M}(n)$ be the set of *tree moves*, which are defined as the set of child actions, plus a "stay" action.

[1]Department of Computer Science, Rutgers University, New Brunswick, USA. This work is supported by NSF awards 1846043 and 2132972. Correspondence to: Liam Schramm <lbs105@cs.rutgers.edu>.

Preprint

Let the tree policy $\pi(n, a)$ be the probability of taking move a when at node n. Note that this is a policy over *moves* we can take in the tree, not just actions. This shift from actions to tree moves is necessary as we will solve for the optimal probability with which to traverse the search tree, including stopping to expand a node. This means that we must explicitly include the choice to expand the current node as part of our policy search. π assigns a probability to taking each child action of the current node n, as well as the probability of expanding n. Let $V(n)$ be a value estimate for node n. We leave the precise estimation method for $V(n)$ open-ended. Let $Q^\pi(\text{stay} \mid n)$ be defined as $V(n)$, and let $Q(a \mid s)$ be defined as $Q(a \mid n) = R(n.\text{state}, a) + \gamma E_{a' \sim \pi(\cdot|child(n,a))}[Q(a' \mid child(n, a))]$ for all actions $a \in A(n)$.

State Occupancy Measure: The state occupancy measure is the expected amount of time a policy π will spend in a given state (or equivalently, the probability distribution of a policy's future states). We repurpose the term slightly here, to focus on the density of the *tree policy* future states in space. To do this, we look at the probability distribution of node expansions the tree policy induces on the tree. Let $P(n \mid \pi)$ be the probability that n is reached when traversing the tree. Let $d^\pi(n')$ be the probability of expanding any given node n' in the tree. Let $d^\pi(n' \mid n) = \frac{d^\pi(n')}{P(n|\pi)}$ be the probability of expanding any given node n' in the tree, assuming we start at n and traverse the tree according to π. Similarly, let $d^\pi(n' \mid n, a) = \frac{d^\pi(n')}{P(n|\pi)\pi(a|n)} = \frac{d^\pi(n')}{P(\text{child}(n,a)|\pi)}$ be the probability of expanding any node n' if we additionally condition on taking action a in state s. Let $\psi(T, d^\pi)$ be the state space S density, the estimated density in the state space of samples drawn from the distribution d^π over nodes T. We will see later that different density estimation methods will lead to different behavior, but we will primarily focus on the 1-nearest neighbor density estimator.

Empirical distributions: Let $\hat{\pi}(n, a)$ be the empirical policy, the fraction of times that each action a has been selected from node n. $\hat{d^\pi}$ is the empirical state occupancy, defined as $\frac{1}{N}$ for all nodes. Let $\hat{\psi}$ be the empirical state space density, the estimated density in the state space of nodes in the search tree. Additionally, note that because T is a tree, it induces a partial ordering over its nodes. We say that $n_i > n_j$ if n_i is an ancestor of n_j. Similarly, we can say that an action $a_k > n_j$ if a_k is higher up in the tree than n_j. We include this note because we will frequently need to sum over all the nodes in the subtree of a particular node or action, which we will write as $\sum_{n<n_i}$.

3. Background

Our work seeks to bridge a range of approaches to long-horizon exploration. For this reason we begin with a review of four approaches to exploration that we show are connected: state space regularization, count-based exploration, SBMP, and Monte Carlo tree search. We build on the work by Grill et al. (2020) illuminating the link between MCTS-based algorithms and regularized policy optimization to formalize these connections.

3.1. State Space Regularization

Hazan et al. (2019) propose exploring by maximizing the entropy of a policy's state occupancy measure in the absence of a dense reward signal, and propose an algorithm that is guaranteed to be efficient in the tabular case. Seo et al. (2021) extend this idea by estimating state space entropy with random encoders and using this as an intrinsic reward in model-free RL. Yuan et al. (2022) further extend this method to the general class of Renyi divergences. Although the motivation of state space entropy maximization is very similar to our motivation of state space f-divergence regularization, the algorithms described are intended for a model-free setting, and do not apply to the MCTS setting.

3.2. Count-Based Exploration (CBE)

One of the most successful methods in long-horizon exploration has been CBE (Tang et al., 2017). This family of methods gives an intrinsic reward to states, so that infrequently-visited states receive higher rewards. In this way, it is similar to performing UCB in the state space. Badia et al. (2020b) develop Never Give Up, an adaptive CBE method for Atari games that projects points into a latent space before doing the kernel density estimate. Similar to our method, Machado et al. (2019) propose using the successor representation for CBE and find that this improves exploration in Atari environments. Agent 57 extends CBE to do long-term exploration for MuZero on Atari games, beating the human benchmark on all games (Badia et al., 2020a). Although CBE is also used in this method, it is not applied to the tree itself – the reward bonus depends only on the previous trajectory, and there is no information-sharing between tree nodes. To the best of our knowledge, no examples exist of MCTS-family algorithms that use CBE to share information about explored regions between nodes in different branches, and no previous works have developed a connection between CBE and f-divergence regularization of the state occupancy measure.

3.3. Sampling-based Motion Planning

SBMP algorithms like Rapidly-exploring Random Trees (RRT) are frequently employed in robotics for their effi-

cient exploration. These methods sample random points in the state space and then expand the nearest point in the search tree in order to bias search towards unexplored regions (LaValle et al., 1998; Lavalle, 2006). Since a node is expanded if and only if a point in its Voronoi region is sampled, SBMP algorithms are called *Voronoi biased*, because the probability of expanding a node is proportional to the volume of its Voronoi region. While SBMP algorithms originally focused on feasible paths, recent work has focused on near-optimal planning. RRT* is an asymptotically-optimal variant of RRT for problems where a steering function is available (Karaman & Frazzoli, 2011). SST and AO-RRT are variants of RRT that are asymptotically optimal, even in the absence of steering functions or heuristics (Li et al., 2016; Hauser & Zhou, 2015). PSST uses an RL-trained policy to guide the search while retaining SST's convergence guarantees (Schramm & Boularias, 2022).

3.4. Monte Carlo Tree Search

MCTS is a tree search strategy based on bandit algorithms (Kocsis & Szepesvári, 2006). The most notable extensions to MCTS are the AlphaZero family, which includes AlphaGo, AlphaGo Zero, AlphaZero, MuZero, and Agent57 (Silver et al., 2016; 2017; 2018; Schrittwieser et al., 2020; Badia et al., 2020a). These algorithms proceed in three main steps: selection, expansion, and backpropagation. For each iteration, the algorithm first selects the child action of the current node that maximizes the upper confidence bound. The algorithm selects actions to traverse the tree until it reaches a leaf node, which it then expands. Lastly, it updates the average value of each ancestor of the leaf node by backpropagating the new value estimate up the tree. While MCTS uses random rollouts to get value estimates, AlphaZero instead trains a neural network to estimate the value. AlphaZero also trains a neural network policy π_θ to imitate the empirical policy $\hat{\pi}$. It then uses the policy-weighted upper confidence bound $UCB(s,a) = Q(s,a) + c\pi_\theta(a \mid s)\frac{\sqrt{N}}{N_a}$, where Q is the value calculated by algorithm, N is s's visitation count, and N_a is a's visitation count. The policy focuses the tree towards branches that have been optimal in previous runs, leading to faster convergence.

Standard MCTS only works for MDPs with finite action spaces. AlphaZero-Continuous is a minimal extension of AlphaZero that uses progressive widening and a continuous policy to extend AlphaZero to continuous environments (Moerland et al., 2018). Progressive widening samples new actions so that the number of actions at each node grows over time, typically as $O(\sqrt{N})$. This is a standard approach for MCTS in continuous environments, but lacks the regret bounds of finite-action-space MCTS. Additionally, progressive widening does not use information from observed rewards to trade off exploration and exploitation, making it closer to ϵ-greedy exploration than UCB. Furthermore, common progressive widening schedules lead to very rapid branching, causing the tree to have many short branches that explore the starting region much more than other regions. For this reason, we argue that it is better to explicitly consider node expansion in the objective, allowing the tree to grow as deeply or as broadly as needed. Since standard AlphaZero does not consider continuous state and action spaces, we will primarily focus on AlphaZero-Continuous as a representative of this family of methods.

3.5. MCTS as Regularized Policy Optimization:

It has been shown in Grill et al. (2020) that the $\frac{\log(N)}{\sqrt{N_a}}$ upper confidence bound in MCTS can be derived as a solution to a regularized policy objective. Consider the objective $\mathcal{L}(\pi) = \sum_a Q(s,a)\pi(a \mid s) - \lambda D_f(\pi \mid\mid \pi_0)$. The authors argue that it is possible to either solve this objective directly and sample from the resulting policy, or to approximate it by taking the action that maximizes the objective for the empirical policy. We will refer to these methods as the *direct* and *empirical* decision rules, respectively. The authors show that the empirical decision rule, $\text{argmax}_a \frac{\partial}{\partial \hat{\pi}(a|s)} \sum_a Q(s,a)\hat{\pi}(a \mid s) - \lambda D_f(\hat{\pi} \mid\mid \pi_0)$, yields the $\frac{log(N)}{\sqrt{N_a}}$ upper confidence bound if D_f is set to be the Hellinger divergence, where $f(t) = 2(1-\sqrt{t})$. If D_f is instead selected to be the reverse KL divergence, $f(t) = -\ln(t)$, then AlphaZero's upper bound of $Q(s,a) + c\pi_\theta(a \mid s)\frac{\sqrt{N}}{N_a}$ is recovered instead.

4. RRT and Count-Based Exploration as Regularized Policy Optimization

In this work, we are interested in what the direct and empirical decision rules are if the state space occupancy is regularized instead of the policy. We find that the direct decision rule yields a search algorithm that uses the Voronoi bias seen in SBMP algorithms, while the empirical decision rule results in a CBE reward. Since both of these methods are widely-used tools for learning and planning in long-horizon exploration problems, we hope that this generalized formalism will yield a family of algorithms that performs well at long-horizon exploration.

4.1. Connection to RRT

We use the direct decision rule described in the previous section to derive a search algorithm (Volume-MCTS) from a regularized return objective. Observe that in a search tree in which each node represents a state and each edge represents an action, each node n is reached by a unique sequence of states $n.\text{traj} = (s_0, s_1, \ldots s_K)$, where K is the node's

depth in the search tree and n's state is s_K. Consider a trajectory that begins with the state sequence $n.\text{traj}$, and then follows the policy π_θ after time K. Observe that the expected return of n's trajectory is

$$\mathcal{V}(n) = E\left[\left(\sum_{i=0}^{K-1} \gamma^i R(n.\text{traj}_i) + \gamma^K V_\theta^\pi(n.\text{traj}_K)\right)\right].$$

We propose maximizing the expected return of nodes in the tree, minus a regularization term that rewards covering the state space as evenly as possible. Let $d^\pi(n)$ be the probability of expanding any node n in the search tree, and let ψ be the estimated density of d^π in the state space. Then we seek to maximize the objective $\mathcal{L}(d^\pi) = E_{n\sim d^\pi}[\mathcal{V}(n)] - \lambda D_f(\psi(T, d^\pi) \mid\mid \psi_0)$ where D_f is an f-divergence.

The intuition for this objective is to balance two goals. The first goal is to maximize reward, and the second is to evenly explore the state space. Since this formulation explicitly considers node expansion instead of a fixed progressive widening schedule, we choose a formulation of optimal return that allows us to compare nodes in different parts of the tree, as opposed to simply selecting actions from the same node. Observe that in this formulation, nodes along an optimal path should score equally. Once we solve for the optimal d^π, we can sample nodes from that distribution to expand.

Because f-divergences are convex, $\mathcal{L}(d^\pi)$ has a unique minimizer as long as ψ is linear with respect to d^π. To solve for this minimizer, we must first choose an f-divergence and a density estimation method for ψ. For f, we follow Grill et al. (2020) in first examining the reverse KL divergence, $f(t) = -\ln(t)$. For density, we introduce a generalization of the 1-nearest-neighbor estimator, which we call Partition Density Estimators. We find that this class of estimators allows us to find a closed-form solution for d^π.

Definition 1. ***Partition density estimator*** *Let D be a set of m weighted points in S with points $p_1 \dots p_m$ and weights $w_1 \dots w_m$.*

*Then $\rho(D)$ is a partition density estimator iff, for each $s \in S$ except for a set of measure zero, **(1)** the gradient $\nabla_w \rho(D)(s)$ is non-zero for exactly one weight w_i, and **(2)** $\rho(D)(s) = w_i g(D, s)$ for some function g.*

We call this a partition density estimator because it allows us to partition the space into regions which only depend on one point in the dataset (except for a zero-measure boundary between these partitions). For instance, consider a weighted variant of the 1-nearest neighbor density estimator, where the estimated density at a point is proportional to the weight of the nearest neighbor. This is a partition density estimator because the density at any state s only depends on the location and weight of s's nearest neighbor. Observe that any partition density estimator is linear with respect to the weights. Therefore, convex functions of such density estimators will also be convex with respect to the weights.

Definition 2. ***Associated Volume*** *Let D be a set of m weighted points in S with points $p_1 \dots p_m$ and weights $w_1 \dots w_m$, and let $\rho(D)$ be a partition density estimator. Let μ be a probability measure on S. Then the associated region $\text{Reg}(i)$ is the set of all $s \in S$ for which $\nabla_{w_i}\rho(D)(s)$ is non-zero. The associated volume $\text{Vol}(i)$ of any point p_i is $\mu(\text{Reg}(i))$, the measure of i's associated region.*

Suppose that D has uniform weights, and μ is a uniform probability measure. Then observe that the 1-nearest neighbor density estimator is a partition density estimator, and the associated volume of any point i in the data set is the volume of its Voronoi region.

Proposition 1. *Suppose $f(t) = -\ln t$ and $\psi(T, d^\pi)(s)$ is a partition density estimator, where the associated measure of any node n is $\text{Vol}(n)$. Then $\mathcal{L}$ has a unique optimizer $d^{\pi*}$, such that $d^{\pi*}(n) = \frac{\lambda}{\alpha - \mathcal{V}(n)}\text{Vol}(n)$, where α is a constant that makes $d^{\pi*}$ a proper probability distribution.*

Proof: The unique solution to a convex function of a probability distribution can be found by setting the gradient equal to a constant α and then solving for π, where α normalizes the solution. We find that the density estimator ψ cancels out, meaning the solution is independent of the choice of ψ. Details are provided in Appendix C.2.1.

Although α does not have a closed form solution, it is possible to find upper and lower bounds for it. Since $d^\pi(n)$ monotonically decreases as α increases, it is simple to calculate α numerically using Newton's method. When α is known, we can sample from d^π.

Relation to RRT: Consider the case in which $\mathcal{V}(n) = 0$ for each node n (or equivalently, the limit in which the regularization coefficient λ is large). Then $\alpha = \lambda$, so $d^\pi(n) = \text{Vol}(n)$. If we choose 1-nearest-neighbor as our density estimator, then the probability of expanding any given node n is the volume of n's Voronoi region. This is the same probability of node expansion used in RRT (LaValle et al., 1998). Thus we can see that the $d^{\pi*}(n) = \frac{\lambda}{\alpha - \mathcal{V}(n)}\text{Vol}(n)$ sampling distribution behaves like RRT when λ is large, but behaves more greedily and has a lower probability of sampling suboptimal nodes as λ decreases over time. Unlike RRT, we makes no assumptions about the reward structure, and can apply this sampling distribution to any continuous state- and action-space RL problem, whereas RRT is limited to path-planning problems. This connection to RRT and the well-motivated generalization of the Voronoi bias to RL is the first major contribution of our work.

4.2. Connection to Count-based Exploration

For this approach, we consider the empirical decision rule. We prove the following:

Proposition 2. *Suppose D_f is chosen to be the Hellinger distance, $f(t) = 2(1-\sqrt{t})$, and $\hat{\psi}$ is chosen to be a kernel density estimator, $\hat{\psi}((T,\hat{d^\pi}))(s) = \sum_{i\in T}\hat{d^\pi}(i)k(i.\text{state},s)$. Additionally, suppose ψ_0 is the uniform distribution over the state space. Let $R_{CBE}(n) = \sqrt{\frac{1}{\sum_{i\in T}k(i.\text{state},n.\text{state})}}$, the CBE reward described in Badia et al. (2020b). Then,*

$$\begin{aligned} a &= \text{argmax}_n \frac{\partial}{\partial \hat{d^\pi}(n)} E_{n'\sim\hat{d^\pi}}[\mathcal{V}(N')] - \lambda D_f(\hat{\psi} \mid\mid \psi_0) \\ &\approx \text{argmax}_a\, Q(s,a) + cE_{n'\sim\text{subtree}(a)}\left[R_{CBE}(n')\right]. \end{aligned}$$

Proof:

This derivative simplifies to:

$$\begin{aligned} &\frac{\partial}{\partial \hat{d^\pi}(n)}\mathcal{L}(\hat{d^\pi}) = \mathcal{V}(n) \\ &+ \lambda \int_S k(n.\text{state},s)\sqrt{\frac{\psi_0(s)}{\sum_{i\in T}\hat{d^\pi}(i)k(i.\text{state},s)}}ds \end{aligned}$$

We can approximate the integral by taking a linear approximation of $\sqrt{\frac{\psi_0(s)}{\sum_{i\in T}\hat{d^\pi}(i)k(i.\text{state},s)}}$ about the point $s = n.\text{state}$, where $k(n.\text{state},s)$ is largest. This reduces to:

$$\begin{aligned} \frac{\partial}{\partial \hat{d^\pi}(n)}\mathcal{L}(\hat{d^\pi}) &\approx \mathcal{V}(n) + c\sqrt{\frac{1}{\sum_{i\in T}k(i.\text{state},n.\text{state})}} \\ &= \mathcal{V}(n) + cR_{CBE}(n') \end{aligned}$$

The empirical decision rule is then

$$\begin{aligned} &\text{argmax}_a \frac{\partial}{\partial\pi(a|n)}\mathcal{L}(\hat{d^\pi}) \\ &= \text{argmax}_a\, Q(s,a) + cE_{n'\sim\text{subtree}(a)}\left[R_{CBE}(n')\right] \end{aligned}$$

The full derivation is in Appendix C.2.3.

Observe that $Q(s,a)$ is the empirical average of future rewards calculated by MCTS, and $E_{n'\sim\text{subtree}(a)}\left[R_{CBE}(n')\right]$ is the empirical average of future exploration rewards calculated by MCTS. In other words, $E_{n'\sim\text{subtree}(a)}\left[R_{CBE}(n')\right]$ is the value for CBE rewards.

5. Volume-MCTS Algorithm

In section 4.1, we proved $d^{\pi*}(n) = \frac{\lambda}{\alpha-\mathcal{V}(n)}\,\text{Vol}(n)$ is the optimal solution to the objective $\mathcal{L}(\pi)$, but this does not show how to calculate $d^{\pi*}$ efficiently. This formulation also does not make the connection between this algorithm and traditional MCTS obvious. To address this, we first prove that it is possible to sample from $d^{\pi*}$ without explicitly solving for it by instead solving for the optimal tree policy π at each node. This is non-trivial to show, as convex functions of the state occupancy measure are not necessarily convex with respect to the policy. For instance, Hazan et al. (2019) show that the entropy of the state occupancy measure is non-convex with respect to the policy, and in fact has local minima. This means that regularization of the state occupancy measure is difficult to solve for in general MDPs, and may not work in combination with standard methods such as policy gradient methods. However, we find that for trees specifically, it is possible to show that solving for the locally optimal policy at each node is sufficient to find $d^{\pi*}$:

Theorem 1. *For any convex loss function $\mathcal{L}(d^\pi)$, $\mathcal{L}$ is convex with respect to $\pi(a \mid n)$ for all nodes n and moves a. Furthermore, $\mathcal{L}$ is minimized if and only if for every node n, $\frac{\partial\mathcal{L}}{\partial\pi(a|n)}$ is constant for all moves a.*

Proof:

Recall that a convex function $\mathcal{L}$ of a probability distribution P is minimized iff $\frac{\partial}{\partial P(n)} = \alpha$ for all n, where α is a constant. Hence, when $\mathcal{L}(d^\pi)$ is minimized, $\frac{\partial}{\partial d^\pi(n)}\mathcal{L}(d^\pi) = \alpha$ for all n. We find that:

$$\frac{\partial}{\partial\pi(a \mid n)}\mathcal{L} = P(n)E_{n'\sim d^\pi(\cdot|n,a)}\left[\frac{\partial f}{\partial d^\pi(n')}\right]$$

This condition implies that $\frac{\partial}{\partial d^\pi(n)}\mathcal{L}(d^\pi) = \alpha$ for all n iff $\frac{\partial}{\partial\pi(a|n)}\mathcal{L}$ is constant for all a at every node n. Hence, the minimization problem for d^π is solved iff the convex loss with respect to the policy at each node is minimized.

The full proof is provided in Appendix C.2.2.

This makes it possible to solve for arbitrary convex losses of d^π by solving for the optimal π at each node n. We use this theorem to derive a closed-form expression for the optimal policy at each node.

Theorem 2. *Suppose $f(t) = -\ln t$ and $\psi(T,d^\pi)(s)$ is a partition density estimator. Then $\mathcal{L}(\pi)$ has a unique optimizer π^*, such that $\pi^*(a \mid n) = \frac{\lambda\,\text{SubtreeVol}(a)}{\alpha-\gamma^{d(n)}P(n\text{ reached }|\pi)Q^{\pi^*}(a|n)}$, where $\text{SubtreeVol}(a)$ is the total associated volume of all nodes in the subtree of a, $d(n)$ is the depth of n in the search tree, $P(n\text{ reached} \mid \pi^*)$ is the probability that we reach n when traversing the tree according to π^*, and α is whatever constant normalizes π^*. Additionally, π^* is the unique distribution that induces $d^{\pi*}$ as the state occupancy measure.*

Proof: Details in Appendix C.2.2.

We propose to expand the search tree by traversing until

we sample an action to expand the current node. Unlike RRT, we cannot expand the tree by sampling the state space and expanding the nearest node, because our expansion probability depends on $\mathcal{V}(n)$ as well as the density ψ_0. Instead, we explicitly solve for the value of π that optimizes $\mathcal{L}(\pi)$. To do this, we select a density estimator that makes $\text{SubtreeVol}(a)$ simple to calculate. RRT implementations typically use a data structure called a k-d tree to store their list of visited states and quickly find the approximate nearest neighbor. These trees effectively act as binary search trees for k-dimensional spaces. Each non-leaf k-d node defines a hyperplane that splits the space along one dimension. All states to one side of the hyperplane are stored in the left child node, and all states on the other side are stored in the right child node. Each child node splits the space again and divides the stored states between its children. This repeats until we reach a leaf node, which has only one state in its region. We can add nodes to this structure by traversing the tree until we find a leaf node, dividing its region in two, and giving that node two child leaf nodes to store its original state and the new state. The region covered by a node is called a k-d region. Since k-d regions are always rectangular, their volumes are easy to calculate. The k-d region of a leaf node always contains exactly one state, so the density estimator $\rho_{kd}(\pi)(x) = \frac{\psi(x)}{\text{kd region volume}(x)}$ is a partition density estimator.

Our algorithm is detailed in Algorithm 1. Starting at the root, we calculate the optimal tree policy at the current node, and then sample from the policy to walk down the tree until we select a node n to expand. After we select n, we sample a new action a to add to the tree, execute this action, and find the next state s'. We then add s' to both the search tree and the k-d tree, find the volume of its k-d region, and get a value estimate. We then backpropagate the value estimate up the search tree. Additionally, at each step in the search tree backpropagation, we backpropagate the value up the k-d tree. We also make a slight approximation to the solution derived above. While it is possible to calculate $\pi^*(a \mid n) = \frac{\lambda SubtreeVol(a)}{\alpha - \gamma^{d(n)} P(n \text{ reached } | \pi) Q^{\pi^*}(a|n)}$ using only local information, the entire tree would need to be recalculated every time λ changes, and λ changes every iteration. Instead we use the k-d tree's approximation of Q in place of Q^{π^*}, which does not require us to recalculate the tree. This approximation is preferable to approximating Q using the MCTS method, where $\hat{Q}$ is the average of the node evaluations in a's subtree, because it allows for information sharing between nodes in different subtrees. This means that $Q(s,a)$ can converge to $Q^{\pi^*}(s,a)$, even if a is not sampled. For instance, if a good trajectory beginning at state s is discovered, then all nodes near s will have their values increase, without needing to be expanded first.

5.1. Guarantees

Volume-MCTS's state-space exploration allows us to derive stronger exploration guarantees than are possible for MCTS. Under mild conditions, we provide non-asymptotic, high-probability bounds on the number of expansions needed to reach a given region in state space. We begin by defining δ-controllability, which we use as a weak notion of continuity.

Definition 3. ***δ-controllable****: Let M be an MDP with action space A, bounded and measurable state space S, deterministic transition function $\mathcal{T}(s_i, a_i)$, and discount factor γ. Let d_A be the dimensionality of A. Let τ be a trajectory in M. Let s_i be the i-th state in τ. Let $\mathcal{B}_\delta(s_i)$ be a ball of radius δ about s_i.*

Then τ is δ-controllable iff there exists a constant $\sigma > 0$ such that for each state s_i in τ, there exists a region in action space A_i with measure at least $\sigma\delta^{d_A}$ such that if a state $s'_i \in \mathcal{B}_\delta(s_i)$ and $a'_i \in A_i$, then $\mathcal{T}(s'_i, a'_i) \in \mathcal{B}_\delta(s_{i+1})$.

Intuitively, if we have a point close to a state s_i in the trajectory τ, then we have a lower bounded chance of sampling an action that stays close to τ at the next state. It is a strictly weaker assumption than the notion of δ-robustness defined in Li et al. (2016).

Theorem 3. *Let τ be a δ-controllable trajectory, with states $s_0 ... s_L$. Let d_A be the dimension of the action space. Let $\mathcal{B}_\delta(\tau_i)$ be the δ-ball around s_i, the i-th state in τ.*

Then the probability that $\mathcal{B}_\delta(\tau_i)$ will be reached after N expansions is lower-bounded by $1 - \frac{\Gamma(i, \frac{1}{2}|\mathcal{B}_{\frac{\delta}{5}}|\sigma\delta^{d_A} c(1-\gamma)(\sqrt{N_1} - c(1-\gamma))))}{\Gamma(i)}$, where Γ is the incomplete Gamma function.

Proof: Details in Appendix C.2.4.

Since this bound takes the form of a Gamma distribution, it is easy to conclude the following corollary.

Corollary 1. *With probability > 0.5, $\mathcal{B}_\delta(\tau_i)$ will be reached after $c^2(1-\gamma)^2(\frac{1}{2}i|\mathcal{B}_{\frac{\delta}{5}}|\sigma\delta^{d_A} + 1)^2$ steps.*

This means that any region on a δ-controllable trajectory of length t will be reached after $O(t^2)$ steps with probability > 0.5. While several MCTS variants have known regret bounds, to the best of our knowledge, this is the first high-probability bound on long-horizon exploration speed for an MCTS-family algorithm, and is a contribution of this work.

5.2. Tree Search Algorithm

Our estimator for $V(s)$ is derived from the k-d tree. We search the k-d tree for the node that contains only the state s, find the node halfway up the tree, and use the average value of all states in that k-d region as the value estimator. The intuition is that this makes a good bias-variance tradeoff. Nodes near the root of the k-d tree average the values of

Algorithm 1 Volume Monte Carlo Tree Search

Have: Regularization coefficient λ, KDTree;
Input: Node n with child branches $a_1, \ldots, a_k$
$Q(\text{“stay”}) \leftarrow \text{KDTree\_value}(n.state)$;
for $a_i \in \{a_1, \ldots, a_k\}$ **do**
  $Q(a_i) \leftarrow \text{KDTree\_value}(\mathcal{T}(n.state, a_i))$;
end for
for $a_i \in \{a_1, \ldots, a_k\} \cup \{\text{“stay”}\}$ **do**
  $\pi(a_i \mid n) \leftarrow \lambda \frac{1}{\alpha - \gamma^d P(n \text{ is reached}) Q(a_i)} \text{Vol}(a_i)$;
end for
Sample next move $a \sim \pi(.|n)$;
if $a =$ “stay” **then**
  value $\leftarrow$ Expand(n);
else
  value $\leftarrow$ Search(a);
end if
value $\leftarrow r + \gamma \times value$;
n.value_sum $\leftarrow$ n.value_sum + value;
n.visit_count $\leftarrow$ n.visit_count + 1;
KDBackprop(value, n.state);
return value;

many states from a large region. This means they have low variance, but high bias (because the states impacting the estimate may be far away and have different values). Nodes near the leaves of the k-d tree have few points from a small region, so the average value of their states is high-variance and low-bias. As more states are explored and the k-d tree grows deeper, nodes halfway down the k-d tree will have regions with volumes that go to zero (implying low bias), and contain many states (implying low variance). Based on this, we conjecture, but do not prove, that this method is a consistent estimator of the true value. We can make this estimate in $O(\ln n)$ time because it only requires one query, as long as we keep track of the average value of each k-d node in the tree. The KDBackprop function keeps these k-d tree values up to date after each node expansion. The full KDTree_value and KDBackprop algorithms, as well as discussion of the challenges to proving consistency, are described in Appendix A.

5.3. Expansion, Value Estimation & K-D Tree Backprop

We sample new actions from a policy π_θ, which is represented by a neural network. The value estimates obtained during expansion also utilize a neural network V_θ. After each episode, we train the value function and policy. The value function is trained to predict the value given by the k-d tree. Ideally, we would train the policy to minimize the objective described earlier. However, it is not trivial to find a closed form representation of the state occupancy divergence from a new policy. Instead, we use ordinary $f-$divergence regularization for the policy. This gives us the following loss function,

$$\begin{aligned}\mathcal{L} \quad &= c_V (V_\theta - \hat{V}_{kd})^2 + c_{KL} \lambda KL(\pi_0 \,||\, \pi_\theta) \\ &\quad - c_A \textstyle\sum_{a \in \text{actions}} A(a) \pi_\theta(a),\end{aligned}$$

in which V_θ is the value network, $\hat{V}_{kd}$ is the k-d tree value, π_θ is the policy network, π_0 is the baseline policy (for instance, a unit Gaussian), KL is the KL divergence between the two policies, λ is the regularization coefficient, and $A(a)$ is the advantage of action a. c_V, c_{KL}, and c_A are hyperparamters.

5.4. Extension to Non-deterministic Environments and Action-dependent Rewards

So far, our approach has made two significant assumptions: we assume that the rewards depend only on the state and not the action, and that the dynamics are deterministic. Here, we would like to briefly note that it is possible to remove these assumptions with some small changes. Action-dependent rewards can be accounted for by regularizing the policy. Stochastic dynamics can be handled using a technique like Double Progressive Widening (Bertsimas et al., 2014). Details are provided in Appendix B.

6. Experiments

To assess Volume-MCTS's performance and the impact of state occupancy measure regularization, we focus on robot navigation experiments. These are environments with significant practical interest where exploration is a central concern, and MCTS's exploration has historically been seen as insufficient by the robotics community. We hypothesize that Volume-MCTS's state occupancy regularization term will motivate the planner to evenly explore the state space, resulting in better exploration. To test this, we conduct two sets of experiments. First, we use a 2D maze environment to visually compare exploration behavior of AlphaZero and Volume-MCTS. We perform an ablation study on this environment to test which factors are relevant to the algorithms' success. We then compare Volume-MCTS to an array of state-of-the-art methods on a challenging quadcopter navigation problem to test its performance in complex and realistic environments.

6.1. Maze Environments

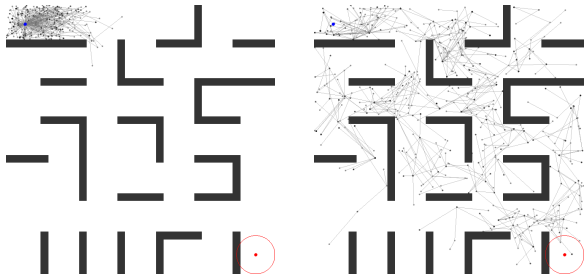
In these experiments, we hope to see how Volume-MCTS's Voronoi bias impacts its exploration behavior, and whether this bias can be matched by other common alterations to the AlphaZero algorithm. To address these questions, we compare Volume-MCTS's performance to three variants of AlphaZero: AlphaZero-Continuous (Moerland et al., 2018), an open-loop variant of AlphaZero-Continuous, and a variant of AlphaZero-Continuous with a CBE reward adapted from

Geometric Maze with Continuous State and Action Spaces (No Training)								
Maze Size	2	3	4	5	6	7	8	9
AlphaZero	36.0 ± 6.0	6.0 ± 4.0	7.0 ± 5.0	0.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.0
AlphaZero w/ CBE	37.0 ± 1.0	38.0 ± 3.0	25.0 ± 4.0	16.0 ± 4.0	6.0 ± 3.0	2.0 ± 2.0	0.0 ± 0.0	0.0 ± 0.0
Volume-MCTS	**49.0 ± 0.0**	**46.0 ± 0.0**	**43.0 ± 1.0**	**38.0 ± 1.0**	**33.0 ± 1.0**	**31.0 ± 1.0**	**22.0 ± 4.0**	**7.0 ± 3.0**
OL AlphaZero	**49.0 ± 0.0**	0.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.0
Geometric Maze with Continuous State and Action Spaces (After Training)								
Maze Size	2	3	4	5	6	7	8	9
AlphaZero	45.0 ± 1.0	42.0 ± 1.0	20.0 ± 6.0	0.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.0
AlphaZero w/ CBE	45.0 ± 1.0	41.0 ± 1.0	32.0 ± 4.0	0.0 ± 0.0	1.0 ± 1.0	0.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.0
Volume-MCTS	44.0 ± 5.0	**47.0 ± 0.0**	**45.0 ± 1.0**	**40.0 ± 1.0**	**38.0 ± 1.0**	**25.0 ± 4.0**	**22.0 ± 4.0**	**26.0 ± 4.0**
OL AlphaZero	**49.0 ± 0.0**	47.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.0

Table 1. Average rewards and standard errors on geometric navigation environments. All methods use 5000 total rollouts per episode.

Maze with Dubins Car Dynamics and Continuous State and Action Spaces (No Training)					
Maze Size	2	3	4	5	6
AlphaZero	46.0±2.0	6.0±4.0	0.0±0.0	0.0±0.0	0.0±0.0
AlphaZero+CBE	29.0±3.0	9.0±4.0	1.0±1.0	1.0±1.0	1.0±1.0
Volume MCTS	43.0±5.0	**42.0±1.0**	**40.0±1.0**	**4.0±3.0**	**4.0±2.0**
OL AlphaZero	**49.0±0.0**	9.0±6.0	0.0±0.0	0.0±0.0	0.0±0.0
Maze with Dubins Car Dynamics and Continuous State and Action Spaces (After Training)					
Maze Size	2	3	4	5	6
AlphaZero	37.0±4.0	8.0±5.0	0.0±0.0	0.0±0.0	0.0±0.0
AlphaZero+CBE	32.0±5.0	30.0±2.0	0.0±0.0	0.0±0.0	0.0±0.0
Volume MCTS	**49.0±0.0**	**46.0±0.0**	**38.0±1.0**	**34.0±1.0**	**26.0±3.0**
OL AlphaZero	**49.0±0.0**	0.0±0.0	0.0±0.0	0.0±0.0	0.0±0.0

Table 2. Average rewards and standard errors on Dubins car environments. All methods use 5000 total rollouts per episode.



(a) AlphaZero (b) Volume-MCTS

Figure 1. Comparison of AlphaZero and Volume-MCTS on the geometric maze environment

Never Give Up (Badia et al., 2020b). We use AlphaZero-Continuous because standard AlphaZero does not work on environments with continuous action spaces. AlphaZero-Continuous is a minimal extension using well-established techniques like progressive widening to generalize AlphaZero to the continuous-action setting. For the sake of brevity, we will refer to AlphaZero-Continuous simply as AlphaZero for all experiments. All environments use $\gamma = .95$ as a discount factor.

The maze environments we test on require long-horizon exploration that makes them inefficient to solve with standard RL methods, but which is well-suited to our method. Each environment is a maze with a continuous state and action space, where the agent must find a goal region opposite the starting location. The maze is N tiles wide and tall, with random walls between these tiles. We test two different sets of dynamics. The first is geometric dynamics, which are described by the update rule $s_{t+1} = s_t + a_t$. The second is the Dubins car dynamics, where the agent maneuvers a car with a fixed turning radius (Lavalle, 2006). These dynamics can be highly challenging, as complex maneuvers may be needed to make sharp turns.

First, we visualize the search trees of AlphaZero and Volume-MCTS in space after 1000 expansions to see how they each perform at reaching novel areas.

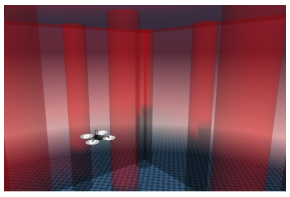
As expected, Volume-MCTS rapidly expands to cover the entire state space (Fig. 1b), while AlphaZero stays very close to the starting location (Fig. 1a). We argue that this is because AlphaZero does not distinguish states based on novelty. Instead, an n-action sequence that ends next to the starting location is counted as being just as novel as one that explores far from the start, as long as both have been expanded the same number of times. Additionally, progressive widening rules cause the tree to branch early – child nodes will never have more branches than their parent. This gives progressive-widening-based approaches a high branching factor and short branches, which can make it difficult to find far-away goals.

We test Volume-MCTS's exploration performance in comparison to the Continuous versions of AlphaZero, AlphaZero with CBE, and Open-loop AlphaZero on geometric mazes of increasing size (Table 1). All methods use 5000 total rollouts for each episode. To evaluate the contributions of the planning algorithm separately from that of the neural network, we first test performance *before* training the value or policy networks. While the average reward for Open-loop AlphaZero exceeds AlphaZero, we find that Volume-MCTS does far better, with the gap among the methods dramatically increasing for large environments with sparse rewards. Second, we compare performance after training. In general, all methods perform better with training than without As with the 'before training' results, the average reward for the AlphaZero methods falls to zero as maze size increases. Here, the differential in average reward for Volume-MCTS versus the other methods is far greater. Instead of falling to zero, the average reward for Volume-MCTS reaches a point where it stops decreasing altogether as the size of the environment increases. Thus, we find that Volume-MCTS performs significantly better than AlphaZero methods at maze navigation tasks. Further, it performs well even where AlphaZero with CBE struggles.

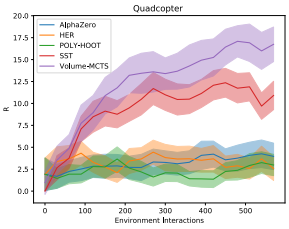
Next, we compare the same methods on a Dubins car maze, an environment with more challenging dynamics (Table 2).

Geometric Maze with Continuous State and Action Spaces (No Training)								
Maze Size	2	3	4	5	6	7	8	9
Volume-MCTS	**49.0 ± 0.0**	**46.0 ± 0.0**	**43.0 ± 1.0**	**38.0 ± 1.0**	33.0 ± 1.0	31.0 ± 1.0	22.0 ± 4.0	7.0 ± 3.0
Volume-MCTS with no Rewards	47.7 ± 0.3	43.9 ± 0.1	40.0 ± 0.8	33.5 ± 4.4	21.9 ± 15.2	29.5 ± 3.2	16.4 ± 13.4	9.1 ± 7.7

Table 3. Comparison of Volume-MCTS with and without reward guidance. Statistically significant improvements are bolded



(a) Quadcopter environment



(b) Reward as a function of total environmental interactions

In the 'before training' results, the average reward for AlphaZero and Open-loop AlphaZero drops precipitously even for relatively small mazes. While AlphaZero with CBE does slightly better, Volume-MCTS results are significantly stronger. However, the differential among methods is much higher after training for 50,000 iterations. The average reward for the AlphaZero methods drops to zero for mazes of 4 by 4 tiles and larger [1]. In contrast, Volume-MCTS experiences a far slower decline in average reward as maze size increases.

6.2. Quadcopter Environment

We turn to the question of how Volume-MCTS performs on realistic problems, when compared to state-of-the-art planning and reinforcement learning methods. To assess this, we test its performance on a challenging quadcopter navigation task. In addition to AlphaZero (Silver et al., 2018), we compare against POLY-HOOT, a recent theoretically-sound MCTS algorithm for continuous action spaces (Mao et al., 2020), Soft-Actor Critic with Hindsight Experience Replay (HER), a state-of-the-art model-free RL method for robotic tasks with sparse rewards (Andrychowicz et al., 2018), and SST, a SBMP algorithm for kinodynamic systems (Li et al., 2016).

Figure 2b shows that Volume-MCTS significantly outperforms HER and the MCTS algorithms. SST also performs well, which we find unsurprising due to its track record in robotics. Perhaps surprisingly, Volume-MCTS performs better than SST when longer search times are used. Additional experiments (included in Appendix D.8) show that SST and Volume-MCTS found the goal approximately the same fraction of the time, but that Volume-MCTS returns shorter paths on average. We believe this is primarily because the RL framing allows us to learn the value function while searching, enabling a better exploration/exploitation tradeoff.

[1] In our results, AlphaZero experienced stability issues. We suspect that this is due to bootstrapping problems common to many reinforcement learning methods. Volume-MCTS proved more stable in this regard because it develops deeper trees, which have a greater distance between the updated value and the target value.

6.3. Ablation Study

To test the importance of using value to guide the search, we perform an ablation study. We test Volume-MCTS on the Maze environment against a variant which treats all rewards as 0. This variant is equivalent to Kinodynamic-RRT, because the probability of expanding a node depends only on that nodes's Voronoi region. Results are shown in Table 3.

We find that Volume-MCTS finds significantly shorter paths than the variant without reward. This indicates that using reward to guide the search allows us to make better exploration/exploitation tradeoffs than is possible for SBMP algorithms, which lack a notion of expected reward to guide the search.

7. Conclusion

We show that two prominent exploration strategies, count-based exploration and Voronoi bias, can be seen as approximate solutions to a policy optimization objective with state occupancy measure regularization. Since both of these methods are established tools for long-horizon exploration in their respective fields, we argue that we can incentivize RL algorithms to explore effectively by minimizing this objective. While state occupancy objectives are typically non-convex in the policy and frequently intractable for RL methods, we show that this objective can be made convex and tractable on search trees. We use this insight to develop Volume-MCTS, a state occupancy-regularized planner that shows strong long-horizon exploration properties. We test our method on an array of robot navigation tasks, and find that Volume-MCTS outperforms methods from model-based RL, model-free RL, and SBMP. While Volume-MCTS is most applicable to deterministic navigation-focused environments, the connection we demonstrate between count-based exploration, Voronoi bias, and state-occupancy measure regularization is more general. We hope that additional work into state occupancy measure regularization will produce a powerful exploration objective that can be used across a wide variety of domains.

References

Andrychowicz, M., Wolski, F., Ray, A., Schneider, J., Fong, R., Welinder, P., McGrew, B., Tobin, J., Abbeel, P., and Zaremba, W. Hindsight experience replay. (arXiv:1707.01495), February 2018. doi: 10.48550/arXiv.1707.01495. URL http://arxiv.org/abs/1707.01495. arXiv:1707.01495 [cs].

Badia, A. P., Piot, B., Kapturowski, S., Sprechmann, P., Vitvitskyi, A., Guo, D., and Blundell, C. Agent57: Outperforming the atari human benchmark, 2020a.

Badia, A. P., Sprechmann, P., Vitvitskyi, A., Guo, D., Piot, B., Kapturowski, S., Tieleman, O., Arjovsky, M., Pritzel, A., Bolt, A., and Blundell, C. Never give up: Learning directed exploration strategies, 2020b.

Bertsimas, D., Griffith, J. D., Gupta, V., Kochenderfer, M. J., Mišić, V. V., and Moss, R. A comparison of monte carlo tree search and mathematical optimization for large scale dynamic resource allocation, 2014.

Grill, J.-B., Altché, F., Tang, Y., Hubert, T., Valko, M., Antonoglou, I., and Munos, R. Monte-carlo tree search as regularized policy optimization. In *Proceedings of the 37th International Conference on Machine Learning*, ICML'20. JMLR.org, 2020.

Hauser, K. and Zhou, Y. Asymptotically optimal planning by feasible kinodynamic planning in state-cost space, 2015.

Hazan, E., Kakade, S., Singh, K., and Van Soest, A. Provably efficient maximum entropy exploration. In Chaudhuri, K. and Salakhutdinov, R. (eds.), *Proceedings of the 36th International Conference on Machine Learning*, volume 97 of *Proceedings of Machine Learning Research*, pp. 2681–2691. PMLR, 09–15 Jun 2019. URL https://proceedings.mlr.press/v97/hazan19a.html.

Karaman, S. and Frazzoli, E. Sampling-based algorithms for optimal motion planning, 2011.

Kleinbort, M., Solovey, K., Littlefield, Z., Bekris, K. E., and Halperin, D. Probabilistic completeness of rrt for geometric and kinodynamic planning with forward propagation. *IEEE Robotics and Automation Letters*, 4(2):i–vii, April 2019. ISSN 2377-3766, 2377-3774. doi: 10.1109/LRA.2018.2888947. URL https://ieeexplore.ieee.org/document/8584061/.

Kocsis, L. and Szepesvári, C. Bandit based monte-carlo planning. In Fürnkranz, J., Scheffer, T., and Spiliopoulou, M. (eds.), *Machine Learning: ECML 2006*, pp. 282–293, Berlin, Heidelberg, 2006. Springer Berlin Heidelberg. ISBN 978-3-540-46056-5.

Lample, G., Lachaux, M.-A., Lavril, T., Martinet, X., Hayat, A., Ebner, G., Rodriguez, A., and Lacroix, T. Hypertree proof search for neural theorem proving, 2022.

Lavalle, S. M. *Planning Algorithms*. Cambridge University Press, 2006. ISBN 0521862051.

LaValle, S. M. et al. Rapidly-exploring random trees: A new tool for path planning. *International Conference on Robotics and Automation*, 1998.

Li, Y., Littlefield, Z., and Bekris, K. E. Asymptotically optimal sampling-based kinodynamic planning. (arXiv:1407.2896), February 2016. doi: 10.48550/arXiv.1407.2896. URL http://arxiv.org/abs/1407.2896. arXiv:1407.2896 [cs].

Machado, M. C., Bellemare, M. G., and Bowling, M. Count-based exploration with the successor representation, 2019.

Mao, W., Zhang, K., Xie, Q., and Başar, T. Poly-hoot: Monte-carlo planning in continuous space mdps with non-asymptotic analysis. (arXiv:2006.04672), December 2020. doi: 10.48550/arXiv.2006.04672. URL http://arxiv.org/abs/2006.04672. arXiv:2006.04672 [cs].

McMahon, T., Sivaramakrishnan, A., Granados, E., and Bekris, K. E. A survey on the integration of machine learning with sampling-based motion planning. *Foundations and Trends® in Robotics*, 9(4):266–327, 2022. doi: 10.1561/2300000063. URL https://doi.org/10.1561%2F2300000063.

Moerland, T. M., Broekens, J., Plaat, A., and Jonker, C. M. A0c: Alpha zero in continuous action space, 2018.

Osiński, B., Miłoś, P., Jakubowski, A., Zięcina, P., Martyniak, M., Galias, C., Breuer, A., Homoceanu, S., and Michalewski, H. Carla real traffic scenarios – novel training ground and benchmark for autonomous driving, 2021.

Schramm, L. and Boularias, A. Learning-guided exploration for efficient sampling-based motion planning in high dimensions. pp. 4429–4435, 05 2022. doi: 10.1109/ICRA46639.2022.9812184.

Schramm, L., Deng, Y., Granados, E., and Boularias, A. Usher: Unbiased sampling for hindsight experience replay. (arXiv:2207.01115), July 2022. doi: 10.48550/arXiv.2207.01115. URL http://arxiv.org/abs/2207.01115. arXiv:2207.01115 [cs].

Schrittwieser, J., Antonoglou, I., Hubert, T., Simonyan, K., Sifre, L., Schmitt, S., Guez, A., Lockhart, E., Hassabis, D., Graepel, T., Lillicrap, T., and Silver,

D. Mastering atari, go, chess and shogi by planning with a learned model. *Nature*, 588(7839):604–609, Dec 2020. ISSN 1476-4687. doi: 10.1038/s41586-020-03051-4. URL https://doi.org/10.1038/s41586-020-03051-4.

Schwartz, J. and Kurniawati, H. Autonomous penetration testing using reinforcement learning, 2019.

Seo, Y., Chen, L., Shin, J., Lee, H., Abbeel, P., and Lee, K. State entropy maximization with random encoders for efficient exploration. In Meila, M. and Zhang, T. (eds.), *Proceedings of the 38th International Conference on Machine Learning*, volume 139 of *Proceedings of Machine Learning Research*, pp. 9443–9454. PMLR, 18–24 Jul 2021. URL https://proceedings.mlr.press/v139/seo21a.html.

Silver, D., Huang, A., Maddison, C. J., Guez, A., Sifre, L., van den Driessche, G., Schrittwieser, J., Antonoglou, I., Panneershelvam, V., Lanctot, M., Dieleman, S., Grewe, D., Nham, J., Kalchbrenner, N., Sutskever, I., Lillicrap, T., Leach, M., Kavukcuoglu, K., Graepel, T., and Hassabis, D. Mastering the game of go with deep neural networks and tree search. *Nature*, 529(7587):484–489, Jan 2016. ISSN 1476-4687. doi: 10.1038/nature16961. URL https://doi.org/10.1038/nature16961.

Silver, D., Schrittwieser, J., Simonyan, K., Antonoglou, I., Huang, A., Guez, A., Hubert, T., Baker, L., Lai, M., Bolton, A., Chen, Y., Lillicrap, T., Hui, F., Sifre, L., van den Driessche, G., Graepel, T., and Hassabis, D. Mastering the game of go without human knowledge. *Nature*, 550(7676):354–359, Oct 2017. ISSN 1476-4687. doi: 10.1038/nature24270. URL https://doi.org/10.1038/nature24270.

Silver, D., Hubert, T., Schrittwieser, J., Antonoglou, I., Lai, M., Guez, A., Lanctot, M., Sifre, L., Kumaran, D., Graepel, T., Lillicrap, T., Simonyan, K., and Hassabis, D. A general reinforcement learning algorithm that masters chess, shogi, and Go through self-play. *Science*, 362(6419):1140–1144, December 2018. ISSN 0036-8075, 1095-9203. doi: 10.1126/science.aar6404. URL https://www.science.org/doi/10.1126/science.aar6404.

Sivaramakrishnan, A., Carver, N. R., Tangirala, S., and Bekris, K. E. Roadmaps with gaps over controllers: Achieving efficiency in planning under dynamics. (arXiv:2310.03239), October 2023. doi: 10.48550/arXiv.2310.03239. URL http://arxiv.org/abs/2310.03239. arXiv:2310.03239 [cs].

Tang, H., Houthooft, R., Foote, D., Stooke, A., Xi Chen, O., Duan, Y., Schulman, J., DeTurck, F., and Abbeel, P. #exploration: A study of count-based exploration for deep reinforcement learning. In Guyon, I., Luxburg, U. V., Bengio, S., Wallach, H., Fergus, R., Vishwanathan, S., and Garnett, R. (eds.), *Advances in Neural Information Processing Systems*, volume 30. Curran Associates, Inc., 2017. URL https://proceedings.neurips.cc/paper_files/paper/2017/file/3a20f62a0af1aa152670bab3c602feed-Paper.pdf.

Yuan, M., Pun, M.-O., and Wang, D. State entropy maximization for exploration acceleration in reinforcement learning. *IEEE Transactions on Artificial Intelligence*, pp. 1–11, 2022. ISSN 2691-4581. doi: 10.1109/TAI.2022.3185180.

Zhang, Z., Zhang, D., and Qiu, R. C. Deep reinforcement learning for power system applications: An overview. *CSEE Journal of Power and Energy Systems*, 6(1):213–225, 2019.

A. Algorithm Details

Here, we present the full pseudocode for the k-d tree algorithms described in the main paper. The KDTree_value algorithm returns an estimate of the state's value in a given region by returning the average value of nodes near that state. KDBackprop updates the average value of nodes in the KDTree.

Algorithm 2 KDTree_value

Have: KDTree;
Input: state;
KDNode ← KDTree.locate(state);
i ← KDNode.depth
while KDNode.depth > i / 2 **do**
  KDNode ← KDNode.parent;
end while
return KDNode.value_sum / KDNode.visit_count

Algorithm 3 KDBackprop

Have: KDTree;
Input: value, state;
KDTreeNode ← KDTree.locate(state);
KDTreeNode.value_sum ← KDTreeNode.value_sum + value;
KDTreeNode.visit_count ← KDTreeNode.visit_count + 1;
while KDTreeNode.has_parent() **do**
  KDTreeNode ← KDTreeNode.parent;
  KDTreeNode.value_sum ← KDTreeNode.value_sum + value;
  KDTreeNode.visit_count ← KDTreeNode.visit_count + 1;
end while

Algorithm 4 Expand

Have: policy network π_θ, value network V_θ;
Input: node n;
$a \sim \pi_\theta(n.\text{state})$;
$s' \leftarrow T(n.\text{state}, a)$;
kd_node ← KDTree.add($V_\theta(s')$)
$\hat{v}$ ←KDTree_value(s')
n.children ← n.children ∪ MCTSNode(a, s', kd_node, $\hat{v}$)
return $\hat{v}$

Earlier, we said that we conjectured that the KDTree value estimate was a consistent estimator. There are two main challenges to proving this. First, the estimator builds estimates from MCTS node values, which are not neither stationary nor independent, which makes traditional bias and variance analysis difficult. Secondly, it is difficult to obtain guarantees on the shape and regularity of the KD regions. It's necessary that, with high probability, the diameter of a KD region goes to zero as the volume does. This is clearly true in practice but tedious to show mathematically. If both of these issues are dealt with or assumed away, the proof outline in the text can be easily formalized.

B. Extension to Non-deterministic Environments and Action-dependent Rewards

To account for action-based rewards, we regularize the policy in addition to the state-action occupancy measure, giving use the loss $\mathcal{L} = E_{n\sim d_\pi}[\mathcal{V}(n) - \lambda D_f(\pi(n)||\pi_0(n))] - \lambda D_f(\phi(d_\pi)||\phi_0)$, where π_0 is the uniform distribution over tree moves. Repeating the derivation of Prop 1 and Thm 2 gives us the tree policy

$$\pi^*(a \mid n) = \frac{\lambda(SubtreeVol(a) + \frac{\gamma^{d(n)} P(n \text{ reached } | \pi)}{1+len(n.children)})}{\alpha - \gamma^{d(n)} P(n \text{ reached } \mid \pi) Q^{\pi^*}(a \mid n)}$$

This is effectively the same method, but with an added offset to each action's volume. We implemented it and found that this regularization was detrimental to long-horizon exploration, so we did not use it in our experiments. However, it is easy to implement when rewards are action-dependent.

In principle, Volume-MCTS could also be extended to non-deterministic environments using double-progressive widening (Bertsimas et al., 2014). In double progressive widening, the tree keeps a list of sampled "next states" for each action, and expands this list over time using progressive widening. This allows the tree to estimate the value under stochastic dynamics. It is straightforward to implement this approach with Volume-MCTS by making two simple changes to the stated algorithm.

1. $\text{Vol}(a) = \sum_i \text{Vol}(s'_i)$, where $s'_i \in next(a)$ is the list of next states from a

2. $Q(a) = \sum_i \frac{\mathcal{V}(s'_i)}{|next(a)|}$

For each of these alterations, we chose not to include them in the main algorithm because they introduce additional complexity to the implementation, and also because they may impact the exploration efficiency. Analysis of the exploration efficiency of these variants is a promising area for future work.

C. Proofs

C.1. Background review

C.1.1. f-DIVERGENCES

Definition 4. *For any probability distributions p and q on $\mathcal{X}$ and function $f\colon \mathbb{R} \to \mathbb{R}$ such that f is a convex function on $\mathbb{R}$ and $f(1) = 0$, the f-divergence D_f between p and q is defined as*

$$D_f(p \mid\mid q) = \int_{\mathcal{X}} q(x) f\left(\frac{p(x)}{q(x)}\right) dx$$

C.1.2. CONVEX FUNCTIONS OF PROBABILITY DISTRIBUTIONS

Convex optimization on probability distributions may be seen as a constrained optimization problem, as the probability distribution is constrained to sum to exactly 1. Under the KKT conditions (which the problems we are considering meet), there is guaranteed to be a unique solution such that $\frac{\partial \mathcal{L}}{\partial d^\pi(n)} = \alpha$ for all n.

C.2. Proofs

C.2.1. UNIQUENESS OF $d^{\pi *}$

Proposition 1. *Let* $\mathcal{V}(n) = E[(\sum_{i=0}^{T-1} \gamma^i R(n.\text{traj}_i)) + \gamma^T V_\theta^\pi(n.\text{traj}_T)]$.
Let $\mathcal{L}(d^\pi) = E_{n\sim d^\pi}[\mathcal{V}(n)] - \lambda D_f(\psi(T, d^\pi) \mid\mid \psi_0)$

Suppose $f(t) = -\ln t$ and $\psi(T, d^\pi)(s)$ is a partition density estimator, where the associated measure of any node n is $\text{Vol}(n)$. *Then $\mathcal{L}$ has a unique optimizer $d^{\pi *}$, such that $d^{\pi *}(n) = \frac{\lambda}{\alpha - \mathcal{V}(n)} \text{Vol}(n)$, where α is a constant that makes $d^{\pi *}$ a proper probability distribution.*

Proof. Observe that $\mathcal{L}(d^\pi)$ is a convex function. Since d^π is a probability distribution, it is known that there exists a unique maximizer for $\mathcal{L}$, which occurs when d^π is normalized and $\frac{\partial}{\partial d^\pi(n)} \mathcal{L}(d^\pi) = \alpha$, where α is a constant. We solve for this value now.

$$
\begin{aligned}
\alpha =& \frac{\partial}{\partial d^\pi(n)} \mathcal{L}(d^\pi) \\
=& \frac{\partial}{\partial d^\pi(n)} E_{n\sim d^\pi}[\mathcal{V}(n)] - \lambda D_f(\psi(T, d^\pi) \mid\mid \psi_0) \\
=& \frac{\partial}{\partial d^\pi(n)} \sum_n \mathcal{V}(n) d^\pi(n) - \lambda \int_S \psi_0(s) f\left(\frac{\psi(T, d^\pi)(s)}{\psi_0(s)}\right) ds \\
=& \mathcal{V}(n) - \lambda \int_S \frac{\partial}{\partial d^\pi(n)} \psi_0(s) f\left(\frac{\psi(T, d^\pi)(s)}{\psi_0(s)}\right) ds \\
=& \mathcal{V}(n) + \lambda \int_S \frac{\partial}{\partial d^\pi(n)} \psi_0(s) \log\left(\frac{\psi(T, d^\pi)(s)}{\psi_0(s)}\right) ds \\
=& \mathcal{V}(n) + \lambda \int_S \psi_0(s) \frac{\partial}{\partial d^\pi(n)} \log\left(\frac{\psi(T, d^\pi)(s)}{\psi_0(s)}\right) ds \\
=& \mathcal{V}(n) + \lambda \int_S \psi_0(s) \frac{\psi_0(s)}{\psi(T, d^\pi)(s)} \frac{\partial}{\partial d^\pi(n)} \frac{\psi(T, d^\pi)(s)}{\psi_0(s)} ds \\
=& \mathcal{V}(n) + \lambda \int_S \frac{\psi_0(s)}{\psi(T, d^\pi)(s)} \frac{\partial}{\partial d^\pi(n)} \psi(T, d^\pi)(s) ds.
\end{aligned}
$$

Observe that by assumption, ψ is a partition density estimator with weights d^π. Therefore, $\frac{\partial}{\partial d^\pi(n)} \psi(T, d^\pi)(s)$ is $\frac{\partial}{\partial d^\pi(n)} d^\pi(n) g(T, s) = g(T, s)$ for some function g if s is in n's associated region and 0 otherwise. Let $\mathbb{1}(n, s)$ be 1 if s is in n's associated region $R(n)$, and 0 otherwise. Then,

$$
\begin{aligned}
\alpha =& \frac{\partial}{\partial d^\pi(n)} \mathcal{L}(d^\pi) \\
=& \mathcal{V}(n) + \lambda \int_S \frac{\psi_0(s)}{\psi(T, d^\pi)(s)} \frac{\partial}{\partial d^\pi(n)} \psi(T, d^\pi)(s) ds \\
=& \mathcal{V}(n) + \lambda \int_S \frac{\psi_0(s)}{\psi(T, d^\pi)(s)} g(T, s) \mathbb{1}(n, s) ds \\
=& \mathcal{V}(n) + \lambda \int_{R(n)} \frac{\psi_0(s)}{\psi(T, d^\pi)(s)} g(T, s) ds \\
=& \mathcal{V}(n) + \lambda \int_{R(n)} \frac{\psi_0(s)}{d^\pi(n) g(T, s)} g(T, s) ds \\
=& \mathcal{V}(n) + \lambda \int_{R(n)} \frac{\psi_0(s)}{d^\pi(n)} ds \\
=& \mathcal{V}(n) + \lambda \frac{1}{d^\pi(n)} \int_{R(n)} \psi_0(s) ds \\
=& \mathcal{V}(n) + \lambda \frac{1}{d^\pi(n)} Vol(n) \\
\alpha - \mathcal{V}(n) =& \frac{\lambda Vol(n)}{d^\pi(n)} \\
d^\pi(n) =& \frac{\lambda Vol(n)}{\alpha - \mathcal{V}(n)}.
\end{aligned}
$$

■

C.2.2. $d^{\pi*}$ CAN BE FOUND BY OPTIMIZING π

We begin by proving that on trees, there is a one-to-one correspondence between d^π and π. This is not true for general MDPs, because actions can be redundant such that multiple actions can lead to the same state. This one-to-one correspondence will

allow us to show that the unique solution to the policy optimization problem is also the unique solution to the state-occupancy measure optimization problem. This will allow us to sample from d^π by calculating π and traversing the tree, which is more efficient than computing and sampling from d^π directly.

Lemma 1. *For any selection of π, π uniquely determines*

$$d^\pi(n) = \pi(\text{stay} \mid n) \prod_{a_i > n} \pi(a_i \mid n_i).$$

Proof. Suppose the algorithm starts at the root node of the tree and selects moves until it selects a "stay" move and stops. Observe that there is exactly one sequence of tree moves that leads to the algorithm stopping at n. When traversing the tree, the algorithm must first select the action that is the ancestor of n at each step. Then it must choose to stop at n. Since these moves are independent, we can write the probability of the algorithm stopping at n as

$$d^\pi(n) = \pi(\text{stay} \mid n) \prod_{a_i > n} \pi(a_i \mid \text{parent}(a_i)).$$

■

Lemma 2. *Exactly one policy π produces a given node distribution $d^\pi(n)$. This policy is*

$$\pi(a \mid n) = \frac{\sum_{n_i < a} d^\pi(n_i)}{\prod_{a_i > n} \pi(a_i \mid \text{parent}(a_i))}.$$

Proof. Consider that the probability of the algorithm reaching a node when traversing the tree $P(n)$. By the same argument used in the previous lemma, this probability is simply the product of all ancestor probabilities, $P(n) = \prod_{a_i > n} \pi(a_i \mid \text{parent}(a_i))$. Also note that some action in the tree a can be taken only if its parent has been reached. Similarly, if action a is taken, then the algorithm is guaranteed to reach a's child. Therefore, $P(\text{child}(a)) = P(a) = \pi(a \mid n)P(n) = \pi(a \mid n) \prod_{a_i > n} \pi(a_i \mid \text{parent}(a_i))$.

Observe that once the algorithm enters a subtree, it has no way to return, so it is guaranteed to stop at one of the nodes in the subtree. Since the algorithm stops in the subtree of n if and only if n is reached, the sum of the stopping probabilities in a subtree is exactly equal to the probability that n is reached. Hence, $P(n) = \sum_{n_i < a} d^\pi(n_i)$. Therefore,

$$\begin{aligned}
\pi(a \mid n) &= \pi(a \mid n) \frac{P(n)}{P(n)} \\
&= \frac{\pi(a \mid n) P(n)}{P(n)} \\
&= \frac{P(a)}{P(n)} \\
&= \frac{P(\text{child}(a))}{P(n)} \\
&= \frac{\sum_{n_i < a} d^\pi(n_i)}{\prod_{a_i > n} \pi(a_i \mid \text{parent}(a_i))}.
\end{aligned}$$

■

Lemma 2 is important because for non-tree environments, multiple policies can produce the same state occupancy. As we proceed, we will use tools from convex optimization to find the optimal policy, which requires us to know that the optimal policy is unique.

Lemma 3.

$$\frac{\partial d^\pi(n')}{\partial \pi(a|n)} = \mathbb{1}(n' < a) \frac{d^\pi(n')}{\pi(a|n)}.$$

Proof. Observe that $\frac{\partial d^\pi(n')}{\partial \pi(a|n)} = 0$ if n' is not a descendent of a, because $d^\pi(n')$ does not depend on $\pi(a \mid n)$. If n' is a descendent of a, then,

$$\begin{aligned}
\frac{\partial}{\partial \pi(a|n)} d^\pi(n') &= \frac{\partial}{\partial \pi(a|n)} (\pi(\text{stay} \,|n') \prod_{\text{action } a' > n'} \pi(a' | \,\text{parent}(a')) \\
&= (\pi(\text{stay} \,|n') \frac{\partial}{\partial \pi(a|n)} \prod_{\text{action } a' > n'} \pi(a' | \,\text{parent}(a')) \\
&= \pi(\text{stay} \,|n') \frac{\partial}{\partial \pi(a|n)} pi(a|n) \prod_{\text{action } a' > n', a' \neq a} \pi(a' | \,\text{parent}(a')) \\
&= \pi(\text{stay} \,|n') \prod_{\text{action } a' > n', a' \neq a} \pi(a' | \,\text{parent}(a')) \\
&= \pi(\text{stay} \,|n') \frac{\prod_{\text{action } a' > n'} \pi(a' | \,\text{parent}(a')}{\pi(a|n)} \\
&= \frac{d^\pi(n')}{\pi(a|n)}.
\end{aligned}$$

■

Lemma 4. *For any function f,*

$$\frac{\partial f}{\partial \pi(a \mid n_t)} = P(n_t) E_{n' \sim d^\pi(\cdot | n_t, a)} \left[\frac{\partial f}{\partial d^\pi(n')} \right].$$

Proof.

$$\frac{\partial f}{\partial \pi(a|n_t)} = \sum_{n'} \frac{\partial f}{\partial d^\pi(n')} \frac{\partial d^\pi(n')}{\partial \pi(a|n_t)}$$

Observe that $\frac{\partial d^\pi(n')}{\partial \pi(a|n_t)} = 0$ if n' is not a descendant of a.

$$\begin{aligned}
\frac{\partial f}{\partial \pi(a|n_t)} &= \sum_{n' < a} \frac{\partial f}{\partial d^\pi(n')} \frac{\partial d^\pi(n')}{\partial \pi(a|n_t)} \\
&= \sum_{n' < a} \frac{\partial f}{\partial d^\pi(n')} \frac{d^\pi(n')}{\pi(a|n_t)} \\
&= \frac{1}{\pi(a|n_t)} \sum_{n' < a} \frac{\partial f}{\partial d^\pi(n')} d^\pi(n') \\
&= \frac{P(n_t)}{\pi(a|n_t) P(n_t)} \sum_{n' < a} \frac{\partial f}{\partial d^\pi(n')} d^\pi(n')
\end{aligned}$$

Observe that $\pi(a|n_t)P(n_t) = P(n_{t+1})$, where n_{t+1} is the child node of a.

$$\begin{aligned}
\frac{\partial f}{\partial \pi(a|n_t)} &= \frac{P(n_t)}{P(n_{t+1})} \sum_{n' < a} \frac{\partial f}{\partial d^\pi(n')} d^\pi(n') \\
&= \frac{P(n_t)}{P(n_{t+1})} \sum_{n' \leq n_{t+1}} \frac{\partial f}{\partial d^\pi(n')} d^\pi(n') \\
&= P(n_t) \sum_{n' \leq n_{t+1}} \frac{\partial f}{\partial d^\pi(n')} d^\pi(n')(P(n_{t+1})) \\
&= P(n_t) \sum_{n' \leq n_{t+1}} \frac{\partial f}{\partial d^\pi(n')} d^\pi(n'|n_{t+1}) \\
&= P(n_t) E_{n' \sim d^\pi(n'|n_{t+1})} \left[\frac{\partial f}{\partial d^\pi(n')} \right] \\
&= P(n_t) E_{n' \sim d^\pi(n'|n_t, a)} \left[\frac{\partial f}{\partial d^\pi(n')} \right].
\end{aligned}$$

■

Theorem 1. *Let T be a tree. Let $\pi(\cdot \mid n)$ be a probability distribution over moves that may be taken from node n, where n is a node in the tree. Let d^π be the probability of stopping at any given node if we traverse the tree by sampling moves from π.*

Then for any convex loss function $\mathcal{L}(d^\pi)$, $\mathcal{L}$ is convex with respect to $\pi(a \mid n)$ for all nodes n and moves a. Furthermore, $\mathcal{L}$ is minimized if and only if for every node n, $\frac{\partial \mathcal{L}}{\partial \pi(a|n)}$ is constant for all moves a.

Proof. The first portion of the proof is easy to establish. Suppose $\mathcal{L}(d^\pi)$ is convex. Recall that $d^\pi(n) = \pi(\text{stay} \mid n) \prod_{a_i > n} \pi(a_i \mid \text{parent}(a_i))$. Observe that $d^\pi(n)$ is linear with respect to the probability of each action a_i above n in the tree. Since a convex function of a linear function is still convex, $\mathcal{L}(\pi(a \mid n))$ is convex.

However, this alone does not establish that optimizing the policy at each node is sufficient to optimize the loss. While $\mathcal{L}$ is convex in $\pi(a \mid n)$ for all a, n, this does not necessarily imply that $\mathcal{L}$ is jointly convex in $\pi(a \mid n)$ and $\pi(a' \mid n')$ for $a \neq a', n \neq n'$. Hazan et al. (2019) provide a counterexample to this, where two distinct policies each induce a uniform distribution over a set of states, but a linear combination of the policies is non-uniform. This implies that the entropy of the state occupancy measure is not convex with respect to the policy. However, this counterexample relies on a directed acyclic graph structure. We aim to show that for trees, convex functions of d^π are minimized if and only if a function is minimized with respect to the policy at every node. While this does not necessarily imply that the loss is convex in π, it does imply that we can minimize convex functions by doing convex optimization of the policy at each node. We prove this by contradiction.

Recall that for a convex loss, d^π is a unique global optimum of $\mathcal{L}(d^\pi)$ if and only if $\frac{\partial \mathcal{L}}{\partial d^\pi(n)} = \alpha$ for all n. Suppose this condition holds for our selection of d^π. Then by Lemma 3, $\frac{\partial \mathcal{L}}{\partial \pi(a|n_t)} = P(n_t) E_{n' \sim d^\pi(\cdot|n_t, a)}[\frac{\partial \mathcal{L}}{\partial d^\pi(n')}] = P(n_t) E_{n' \sim d^\pi(\cdot|n_t, a)}[\alpha] = P(n_t)\alpha$. Observe that $\frac{\partial \mathcal{L}}{\partial \pi(a|n_t)}$ does not depend on a. Therefore $\frac{\partial \mathcal{L}}{\partial \pi(a|n_t)}$ is constant for all moves a.

Suppose that for every node n, $\frac{\partial \mathcal{L}}{\partial \pi(a|n)}$ is constant for all moves a. We use proof by contradiction. Let $d^{\pi *}$ be the optimal value for d^π. Suppose $d^\pi \neq d^{\pi *}$. Then there exists some n where $\frac{\partial \mathcal{L}}{\partial d^\pi(n)} > \alpha$ or $\frac{\partial \mathcal{L}}{\partial d^\pi(n)} < \alpha$. We consider the first case. Observe that since $\mathcal{L}$ is strongly convex with respect to d^π, $\frac{\partial \mathcal{L}}{\partial d^\pi(n)}$ is monotonically increasing with $d^\pi(n)$. Hence, if $\frac{\partial \mathcal{L}}{\partial d^\pi(n)} > \alpha$, $d^\pi(n) > d^{\pi *}(n)$. Since d^π sums to 1, there must also be some $\tilde{n}$ where $d^\pi(\tilde{n}) < d^{\pi *}(\tilde{n})$.

By Lemma 4, $\frac{\partial \mathcal{L}}{\partial \pi(a|n)} = P(n) E[\frac{\partial \mathcal{L}}{\partial d^\pi(n)}]$ for all n. Since $\frac{\partial \mathcal{L}}{\partial d^\pi(n)} > \alpha$, $\frac{\frac{\partial \mathcal{L}}{\partial \pi(\text{stay} \mid n)}}{P(n)} = E_{n' \mid \text{stay}, n}[\frac{\partial \mathcal{L}}{\partial d^\pi(n')}] = \frac{\partial \mathcal{L}}{\partial d^\pi(n)} > \alpha$. Since we assumed that $\frac{\partial \mathcal{L}}{\partial \pi(a|n)}$ was constant for all moves from n, $\frac{\frac{\partial \mathcal{L}}{\partial \pi(a|n)}}{P(n)} > \alpha$ for all moves. Now, consider n's parent action a_1. Since $E_{n' \sim d^\pi(\cdot|n)}[\frac{\partial \mathcal{L}}{\partial d^\pi(n')}] > \alpha$ for each branch of n, $E_{n' \sim d^\pi(\cdot|n)}[\frac{\partial \mathcal{L}}{\partial d^\pi(n')}] > \alpha$ for the whole subtree of a_1. Therefore, $\frac{\frac{\partial \mathcal{L}}{\partial \pi(a_1 \mid \text{parent}(a_1))}}{P(\text{parent}(a_1))} = E_{n' \sim d^\pi(\cdot \mid \text{parent}(a_1))}[\frac{\partial \mathcal{L}}{\partial d^\pi(n')}] > \alpha$. Here, the same reasoning applies as before – the gradient is equal for all moves, so they must all have a gradient > $P(\text{parent}(a_1))\alpha$. We can repeat this argument with induction for

each node in the tree, showing that each ancestor must have a gradient greater than alpha, until we reach the root. Therefore, the gradient of all moves at the root is greater than α.

However, recall that there must be a node $\tilde{n}$ where $d^{\pi}(\tilde{n}) < d^{\pi *}(\tilde{n})$. Since $\frac{\partial \mathcal{L}}{\partial d^{\pi}(\tilde{n})}$ is monotonically increasing with $d^{\pi}(\tilde{n})$, $\frac{\partial \mathcal{L}}{\partial d^{\pi}(\tilde{n})} < \alpha$. We make the same argument as before, showing that $\frac{\frac{\partial \mathcal{L}}{\partial \pi(\tilde{a}|\tilde{n})}}{P(\tilde{n})} < \alpha$ for all moves from $\tilde{n}$. By a symmetrical argument to the previous paragraph, all of $\tilde{n}$'s ancestors must have gradients < α. Therefore the gradient at the root is < α.

However, we already showed that the gradient at the root was > α, so this is a contradiction. Therefore our assumption must be false, and $\frac{\partial \mathcal{L}}{\partial d^{\pi}(n)} = \alpha$ for all nodes. Hence, the induced distribution of π^* is $d^{\pi *}$.

The same contradiction is reached if we assume that there exists some n where $\frac{\partial \mathcal{L}}{\partial d^{\pi}(n)} < \alpha$. ■

Theorem 2. *Suppose $f(t) = -\ln t$ and $\psi(T, d^{\pi})(s)$ is a partition density estimator.*
Then $\mathcal{L}(\pi)$ has a unique optimizer π^*, such that $\pi^*(a \mid n) = \frac{\lambda \operatorname{SubtreeVol}(a)}{\alpha - \gamma^{d(n)} P(n \text{ reached} | \pi) Q^{\pi^*}(a|n)}$, where – $\operatorname{SubtreeVol}(a)$ is the total volume of all nodes in the subtree of a. $d(n)$ is the depth of n in the search tree. $P(n \text{ reached} \mid \pi)$ is the probability that the algorithm reaches n when traversing the tree, or equivalently that the algorithm stops at some node in n's subtree. and α is whatever constant normalizes π^*.

Additionally, π^* is the unique distribution that induces $d^{\pi *}$ as the state occupancy measure.

Proof. As before, $\mathcal{L}$ is convex with respect to $\pi(a \mid n)$, so it has a unique optimizer when $\frac{\partial \mathcal{L}}{\partial \pi(a|n)}$ is equal to a constant for all a. Recall that by Lemma 4, $\frac{\partial \mathcal{L}}{\partial \pi(a|n)} = P(n) E_{n' \sim d^{\pi}(\cdot|n,a)}[\frac{\partial \mathcal{L}}{\partial d^{\pi}(n')}]$. Then

$$
\begin{aligned}
\frac{\partial \mathcal{L}}{\partial \pi(a \mid n)} &= P(n) E_{n' \sim d^{\pi}(\cdot|n,a)} \left[\frac{\partial \mathcal{L}}{\partial d^{\pi}(n')} \right] \\
&= P(n) E_{n' \sim d^{\pi}(\cdot|n,a)} \left[\mathcal{V}(n') + \lambda \frac{\operatorname{Vol}(n')}{d^{\pi}(n')} \right] \\
&= P(n) E_{n' \sim d^{\pi}(\cdot|n,a)}[\mathcal{V}(n')] + \lambda P(n) E_{n' \sim d^{\pi}(\cdot|n,a)} \left[\frac{\operatorname{Vol}(n')}{d^{\pi}(n')} \right] \\
&= P(n) E_{n' \sim d^{\pi}(\cdot|n,a)}[\mathcal{V}(n')] + \lambda P(n) \sum_{n' < a} \frac{d^{\pi}(n')}{P(n)\pi(a \mid n)} \left[\frac{\operatorname{Vol}(n')}{d^{\pi}(n')} \right] \\
&= P(n) E_{n' \sim d^{\pi}(\cdot|n,a)}[\mathcal{V}(n')] + \lambda \sum_{n' < a} \frac{\operatorname{Vol}(n')}{\pi(a \mid n)} \\
&= P(n) \left(\gamma^d Q^{\pi}(n, a) + \sum_{i=0}^{d} \gamma^i R(n_i, a_i) \right) + \lambda \frac{\operatorname{SubtreeVol}(a)}{\pi(a \mid n)} \\
&= P(n) \left(\sum_{i=0}^{d} \gamma^i R(n_i, a_i) \right) + P(n)(\gamma^d Q^{\pi}(n, a)) + \lambda \frac{\operatorname{SubtreeVol}(a)}{\pi(a \mid n)}
\end{aligned}
$$

Observe that since $P(n)(\sum_{i=0}^{d} \gamma^i R(n_i, a_i))$ is constant for all moves, we may absorb it into α

$$
\begin{aligned}
\alpha &= P(n)\gamma^d Q^{\pi}(n, a) + \lambda \frac{\operatorname{SubtreeVol}(a)}{\pi(a \mid n)} \\
\alpha - P(n)\gamma^d Q^{\pi}(n, a) &= \lambda \frac{\operatorname{SubtreeVol}(a)}{\pi(a \mid n)} \\
\pi(a \mid n) &= \lambda \frac{\operatorname{SubtreeVol}(a)}{\alpha - P(n)\gamma^d Q^{\pi}(n, a)}
\end{aligned}
$$

By Theorem 2, $\pi(a \mid n) = \lambda \frac{\operatorname{SubtreeVol}(a)}{\alpha - P(n)\gamma^d Q^{\pi}(n,a)}$ is the unique optimizer of $\mathcal{L}$. ■

C.2.3. Connection to Count-Based Exploration

Proposition 2. *Suppose D_f is chosen to be the Hellinger distance, $f(t) = 2(1-\sqrt{t})$, and $\hat{\psi}$ is chosen to be kernel density estimator, $\hat{\psi}((T, \hat{d^\pi}))(s) = \sum_{i \in T} \hat{d^\pi}(i) k(i.\text{state}, s)$. Additionally, suppose ψ_0 is the uniform distribution over the state space. Let $R_{CBE}(n) = \sqrt{\frac{1}{\sum_{i \in T} k(i.\text{state}, n.\text{state})}}$, the count-based exploration reward described in (Badia et al., 2020b). Then,*

$$\begin{aligned} a &= \text{argmax}_n \frac{\partial}{\partial \hat{d^\pi}(n)} E_{n' \sim \hat{d^\pi}}[\mathcal{V}(N')] - \lambda D_f(\psi_0 \mid\mid \hat{\psi}) \\ &\approx \text{argmax}_a \, Q(s,a) + c E_{n' \sim \text{subtree}(a)} \left[R_{CBE}(n') \right] \end{aligned}$$

Proof. We aim to show that applying the empirical decision rule to the policy optimization problem with Hellinger squared distance regularization over the state spaces yields a count-based exploration reward. The squared Hellinger distance is an f-divergence, where $f(t) = 2(1-\sqrt{t})$. Observe that the derivative of f, $\frac{df}{dt}$, is $\frac{df}{dt}(t) = -\frac{1}{\sqrt{t}}$.

$$\begin{aligned} \frac{\partial}{\partial \hat{d^\pi}(n)} \mathcal{L}(\hat{d^\pi}) &= \frac{\partial}{\partial \hat{d^\pi}(n)} E_{n' \sim \hat{d^\pi}}[\mathcal{V}(n')] - \lambda D_f(\psi_0 \mid\mid \hat{\psi}) \\ &= \frac{\partial}{\partial \hat{d^\pi}(n)} \sum_{n'} \hat{d^\pi}(n') \mathcal{V}(n') - \lambda \int_S \psi_0(s) f\left(\frac{\hat{\psi}(s)}{\psi_0(s)} \right) ds \\ &= \mathcal{V}(n) - \lambda \int_S \psi_0(s) \frac{\partial}{\partial \hat{d^\pi}(n)} f\left(\frac{\hat{\psi}(s)}{\psi_0(s)} \right) ds \\ &= \mathcal{V}(n) - \lambda \int_S \psi_0(s) \frac{df}{dt}\left(\frac{\hat{\psi}(s)}{\psi_0(s)} \right) \frac{\partial}{\partial \hat{d^\pi}(n)} \frac{\hat{\psi}(s)}{\psi_0(s)} ds \\ &= \mathcal{V}(n) - \lambda \int_S \psi_0(s) \frac{df}{dt}\left(\frac{\hat{\psi}(s)}{\psi_0(s)} \right) \frac{\partial}{\partial \hat{d^\pi}(n)} \frac{\sum_{i \in T} \hat{d^\pi}(i) k(i.\text{state}, s)}{\psi_0(s)} ds \\ &= \mathcal{V}(n) - \lambda \int_S \psi_0(s) \frac{df}{dt}\left(\frac{\hat{\psi}(s)}{\psi_0(s)} \right) \frac{k(n.\text{state}, s)}{\psi_0(s)} ds \\ &= \mathcal{V}(n) - \lambda \int_S \psi_0(s) \frac{df}{dt}\left(\frac{\sum_{i \in T} \hat{d^\pi}(i) k(i.\text{state}, s)}{\psi_0(s)} \right) \frac{k(n.\text{state}, s)}{\psi_0(s)} ds \\ &= \mathcal{V}(n) - \lambda \int_S \frac{df}{dt}\left(\frac{\sum_{i \in T} \hat{d^\pi}(i) k(i.\text{state}, s)}{\psi_0(s)} \right) k(n.\text{state}, s) ds \end{aligned}$$

Recall that $f(t) = 2(1-\sqrt{t})$ and $\frac{df}{dt}(t) = -\frac{1}{\sqrt{t}}$.

$$\begin{aligned} \frac{\partial}{\partial \hat{d^\pi}(n)} \mathcal{L}(\hat{d^\pi}) &= \mathcal{V}(n) - \lambda \int_S \frac{df}{dt}\left(\frac{\sum_{i \in T} \hat{d^\pi}(i) k(i.\text{state}, s)}{\psi_0(s)} \right) k(n.\text{state}, s) ds \\ &= \mathcal{V}(n) + \lambda \int_S \sqrt{\frac{\psi_0(s)}{\sum_{i \in T} \hat{d^\pi}(i) k(i.\text{state}, s)}} k(n.\text{state}, s) ds \end{aligned}$$

Now, we observe two properties of kernels assumed for kernel regression. First, kernels are window functions: a kernel $k(x,y)$ is maximal when $x = y$ and decreases rapidly as $\mid x - y \mid$ becomes large. Additionally $\int_{\mathcal{X}} k(x,y) dx = 1$ for $x, y \in \mathcal{X}$. This means that when integrating another function with the kernel, almost all the contribution comes from near the center. This means that for a continuous function f, we may approximate the integral $\int_{\mathcal{X}} f(x) k(x,y) dx$ by linearly approximating f about $x = y$ where the kernel is maximized. Hence $\int_{\mathcal{X}} f(x) k(x,y) dx \approx \int_{\mathcal{X}} [f(y) + (x-y) \cdot$

$\nabla f(y)]k(x,y)dx$. And second, we assume that the kernel is an even function, so integrating $\int_{\mathcal{X}}(x-y)k(x,y)dx = 0$. Hence, $\int_{\mathcal{X}} f(y) + (x-y)\cdot\nabla f(y)]k(x,y)dx = \int_{\mathcal{X}} f(y)k(x,y)dx + \int_{\mathcal{X}}(x-y)\cdot\nabla f(y)k(x,y)dx = f(y) + 0 = f(y)$. Applying this to the derivation from above, we find the following:

$$\begin{aligned}\frac{\partial}{\partial \hat{d^\pi}(n)}\mathcal{L}(\hat{d^\pi}) &= \mathcal{V}(n) + \lambda\int_S k(n.\text{state}, s)\sqrt{\frac{\psi_0(s)}{\sum_{i\in T}\hat{d^\pi}(i)k(i.\text{state}, s)}}ds \\ &\approx \mathcal{V}(n) + \lambda\sqrt{\frac{\psi_0(n.\text{state})}{\sum_{i\in T}\hat{d^\pi}(i)k(i.\text{state}, n.\text{state})}}\end{aligned}$$

$\hat{d^\pi}$ was defined to be $\frac{1}{N}$ for all nodes. λ was assumed to be $\frac{c}{\sqrt{N}}$ for some constant c. Combining constants and simplifying, this yields

$$\begin{aligned}\frac{\partial}{\partial \hat{d^\pi}(n)}\mathcal{L}(\hat{d^\pi}) &\approx \mathcal{V}(n) + \lambda\sqrt{\frac{\psi_0(n.\text{state})}{\sum_{i\in T}\hat{d^\pi}(i)k(i.\text{state}, n.\text{state})}} \\ &= \mathcal{V}(n) + c\frac{1}{\sqrt{N}}\sqrt{\frac{\psi_0(n.\text{state})}{\sum_{i\in T}\frac{1}{N}k(i.\text{state}, n.\text{state})}} \\ &= \mathcal{V}(n) + c\sqrt{\frac{\psi_0(n.\text{state})}{\sum_{i\in T}N\frac{1}{N}k(i.\text{state}, n.\text{state})}} \\ &= \mathcal{V}(n) + c\sqrt{\frac{\psi_0(n.\text{state})}{\sum_{i\in T}k(i.\text{state}, n.\text{state})}}\end{aligned}$$

Observing that ψ_0 is a uniform distribution, we see that $\psi_0(s)$ is constant for all s. We can absord this constant into c, which gives us the expression

$$\frac{\partial}{\partial \hat{d^\pi}(n)}\mathcal{L}(\hat{d^\pi}) \approx \mathcal{V}(n) + c\sqrt{\frac{1}{\sum_{i\in T}k(i.\text{state}, n.\text{state})}}$$

Observe that this is the same exploration reward bonus used in Never Give Up (Badia et al., 2020b).

Applying Lemma 4, we can see this expressed in a more traditional reward based form.

$$
\begin{aligned}
\text{action}(n) &= \text{argmax}_a \frac{\partial f}{\partial \hat{\pi}(a \mid n_t)} \\
&= \text{argmax}_a \, P(n) E_{n' \sim \hat{d^\pi}(\cdot|n,a)} \left[\frac{\partial}{\partial \hat{d^\pi}(n)} \right] \\
&= \text{argmax}_a \, E_{n' \sim \hat{d^\pi}(\cdot|n,a)} \left[\mathcal{V}(n) + 1 \sqrt{\frac{1}{\sum_{i \in T} k(i.\,\text{state}, n.\,\text{state})}} \right] \\
&= \text{argmax}_a \, E_{n' \sim \hat{d^\pi}(\cdot|n,a)} [\mathcal{V}(n)] + E_{n' \sim \hat{d^\pi}(\cdot|n,a)} \left[c \sqrt{\frac{1}{\sum_{i \in T} k(i.\,\text{state}, n.\,\text{state})}} \right] \\
&= \text{argmax}_a \, \hat{Q^\pi}(s,a) + c E_{n' \sim \hat{d^\pi}(\cdot|n,a)} \left[\sqrt{\frac{1}{\sum_{i \in T} k(i.\,\text{state}, n.\,\text{state})}} \right]
\end{aligned}
$$

Observe that $\hat{Q^\pi}(s,a)$ is the empirical average of future rewards – this is exactly the Q value calculated by traditional MCTS. $E_{n' \sim \hat{d^\pi}(\cdot|n,a)} \left[\sqrt{\frac{1}{\sum_{i \in T} k(i.\,\text{state}, n.\,\text{state})}} \right]$ is the empirical average of future values of $\sqrt{\frac{1}{\sum_{i \in T} k(i.\,\text{state}, n.\,\text{state})}}$. In other words, it is an additional term calculated the same way the value is calculated, but treating $\sqrt{\frac{1}{\sum_{i \in T} k(i.\,\text{state}, n.\,\text{state})}}$ as a supplemental reward. This is equivalent to adding $\sqrt{\frac{1}{\sum_{i \in T} k(i.\,\text{state}, n.\,\text{state})}}$ to the reward function, which is what count-based exploration rewards do. Therefore, this approximation is equivalent to a count-based exploration reward.

■

C.2.4. Exploration efficiency

Sampling-based motion algorithms frequently come with guarantees of exploration efficiency in addition to optimality. Reinforcement learning algorithms, on the other hand, rarely enjoy these kinds of guarantees outside of simple cases such as bandits. This is especially true in continuous domains. Recent work has established regret bounds for both MCTS and continuous-space generalizations (Note: early logarithmic regret bounds for MCTS are now thought to be incorrect, as they do not account for the value distribution being non-stationary). However, even these methods do not show that they efficiently explore the state space – instead they show a bound on the regret as a function of γ, with the regret growing asymptotically as $\gamma \to 1$. We are able to provide polynomial bounds on the rate at which Volume-MCTS explores the state space, which do not depend on γ. This is a significant advantage for long-horizon problems where γ may be very close to 1.

As a note, we derive these bounds for the idealized version of Volume-MCTS which uses 1-nearest neighbor as its density estimator, which we will call Voronoi-Volume-MCTS. This version of the algorithm is slightly different than the form presented in the main body of the paper, which uses a KD-tree as its density estimator. We choose to analyze this version instead, because the probability of expanding a node is proportional to the volume of its Voronoi region, rather than the volume of the KD-region. While KD-region volumes are much easier to calculate in practice, Voronoi regions are more mathematically tractable. This is primarily a artifact of the analysis rather than a meaningful feature of the math. Research in SBMP algorithms nearly always uses Voronoi regions for analysis, and approximates these regions using KD-trees in implementation. The difference is rarely relevant in practice.

We will begin by defining the following term

Definition 5. ***δ-controllable****: Let M be an MDP with action space A, bounded state space S, and deterministic transition function $\mathcal{T}(s_i, a_i)$. Let d_A be the dimensionality of A. Let τ be a trajectory in M. Let s_i be the i-th state in the trajectory τ. Let $\mathcal{B}_\delta(s_i)$ be a ball of radius δ about s_i.*

Then τ is δ-controllable iff there exists a constant $\sigma > 0$ such that for each state s_i in τ, there exists a region in action space A_i with measure at least $\sigma \delta^{d_A}$ such that if a state $s'_i \in \mathcal{B}_\delta(s_i)$ and $a'_i \in A_i$, then $\mathcal{T}(s'_i, a'_i) \in \mathcal{B}_\delta(s_{i+1})$.

Intuitively, if we have a point close to a trajectory τ, then we have a lower bounded chance of sampling an action that stays close to τ at the next state. This condition is strictly weaker than the assumptions used for asymptotically optimal

motion planners. Stable Sparse RRT takes a set of assumptions that together necessitate that every point in the δ-ball of s_i is reachable from every point in the δ-ball of s_{i-1}. By contrast, we only assume that the is a lower-bounded chance of reaching somewhere in the δ-ball of s_i, without specifying where that may be or what shape it may have.

Our strategy for this proof is as follows. First, we lower bound the probability of selecting a point near a state s_t in the trajectory. Then, we lower bound the probability of reaching a state near s_{t+1}, given that a state near s_t was reached. Finally, we sum over these bounds and use them to establish a bound of reaching an arbitrary region in a given amount of time.

Recall that the probability of expanding a node Voronoi-Volume-MCTS is given by $d^\pi(n) = \frac{\lambda}{\alpha - \mathcal{V}(n)} \operatorname{Vol}(n)$, where

- $\mathcal{V}(n)$ is the value of the node n
- $\operatorname{Vol}(n)$ is the measure of n's Voronoi region
- $\lambda = \frac{c}{\sqrt{N}}$
- c is a constant
- N is the current iteration number, and
- α is whatever constant normalizes $d^{\pi *}(n)$, so it sums to 1.

We aim to lower-bound the sum of all $d^\pi(n)$ near s_t, assuming that at least one node is near s_t. We begin by bounding the sum of their Voronoi regions.

Lemma 5. *Let $s \in S$ be such that $\mathcal{B}_\delta(s) \subset S$. Suppose that there exists a tree node n with $n.\text{state} \in \mathcal{B}_{\frac{2\delta}{5}}(s)$. Let $s' \in S$ be an arbitrary state in S. Let s_{near} denote the nearest neighbor of s' among all tree nodes.*

Suppose the state space S is bounded. Further suppose, without loss of generality, that the volume of the full state space S is 1.

Then the union of the Voronoi regions of all nodes in $\mathcal{B}_\delta(s)$ has a volume of at least $|\mathcal{B}_{\frac{\delta}{5}}|$

Proof. Our proof closely follows the proof given by Kleinbort et al. (2019) for their Lemma 4.

Case 1: Suppose all nodes in the tree are in $\mathcal{B}_\delta(s)$, then the union of their Voronoi regions is S. Then it is trivial that the union of their Voronoi regions has measure $> |\mathcal{B}_{\frac{\delta}{5}}|$.

Case 2: Suppose there is a tree node with $z \notin \mathcal{B}_\delta(s)$. We show that if $s' \in B_{\frac{\delta}{5}}(s)$ then $s_{near} \in \mathcal{B}_\delta(s)$. Observe that $n.\text{state} \in \mathcal{B}_{\frac{2\delta}{5}}(s)$, so by the triangle inequality, $||s' - n.\text{state}|| \leq \frac{3\delta}{5}$. Since $z \notin \mathcal{B}_\delta(s)$, $||s_{near} - z|| \geq \frac{4\delta}{5}$. Therefore, $||s' - n.\text{state}|| < ||s' - z||$. Hence, z is not the nearest neighbor of s'. Since z was chosen arbitrarily from nodes outside of $\mathcal{B}_\delta(s)$, this holds for all nodes outside of $\mathcal{B}_\delta(s)$. It follows that s' is in the Voronoi region of some node within $\mathcal{B}_\delta(s)$. Since s' was again chosen arbitrarily from points in $B_{\frac{\delta}{5}}(s)$, the union of the Voronoi regions of nodes node in $\mathcal{B}_\delta(s)$ includes all points in $B_{\frac{\delta}{5}}(s)$. Observe that the volume of this region is $|\mathcal{B}_{\frac{\delta}{5}}|$.

Therefore, the union of the Voronoi regions of all nodes in $\mathcal{B}_\delta(s)$ has a volume of at least $|\mathcal{B}_{\frac{\delta}{5}}|$. ■

Next, we must show bounds on α. Together with Lemma 5, this will give us a lower bound on the probability of sampling a node in a given region once that region has been reached.

Lemma 6. *1:*

$$\alpha \geq max_n(\mathcal{V}(n) + \lambda \operatorname{Vol}(n))$$

and 2:

$$\alpha \leq max_n(\mathcal{V}(n)) + \lambda$$

Proof. 1: $\alpha = \mathcal{V}(n) + \lambda \frac{\operatorname{Vol}(n)}{d^\pi(n)}$. Since $d^\pi(n) \leq 1$, $\alpha = \mathcal{V}(n) + \lambda \operatorname{Vol}(n)$ for all n. Hence, $\alpha \geq max_n(\mathcal{V}(n) + \lambda \operatorname{Vol}(n))$.

2: $\sum_n d^\pi(n) = 1$. Therefore, $1 = \sum_n \frac{\lambda \operatorname{Vol}(n)}{\alpha - \mathcal{V}(n)} \geq \sum_n \frac{\lambda \operatorname{Vol}(n)}{\alpha - max_{n'}(\mathcal{V}(n'))}$. Hence, $\alpha - max_{n'}(\mathcal{V}(n')) \geq \sum_n \lambda \operatorname{Vol}(n) = \lambda$. Therefore, $\alpha \geq max_n(\mathcal{V}(n)) + \lambda$. ■

From this result, it is easy to derive the following bound on d^π:

Lemma 7. *Suppose, without loss of generality, that* $0 \leq R \leq 1$. *Then,*

$d^\pi(n) \geq \frac{c(1-\gamma)}{\sqrt{N}+c(1-\gamma)} \operatorname{Vol}(n)$

Proof. $0 \leq R \leq 1$, so $0 \leq \mathcal{V}(n) \leq \frac{1}{1-\gamma}$ for all n. Hence,

$$\begin{aligned}
d^\pi(n) &= \frac{\lambda \operatorname{Vol}(n)}{\alpha - \mathcal{V}(n)} \\
&\geq \frac{\lambda \operatorname{Vol}(n)}{max_n(\mathcal{V}(n)) + \lambda - \mathcal{V}(n)} \\
&\geq \frac{\lambda \operatorname{Vol}(n)}{\frac{1}{1-\gamma} + \lambda - \mathcal{V}(n)} \\
&\geq \frac{\lambda \operatorname{Vol}(n)}{\frac{1}{1-\gamma} + \lambda - 0} \\
&\geq \frac{\lambda \operatorname{Vol}(n)}{\frac{1}{1-\gamma} + \lambda} \\
&\geq \frac{\lambda \operatorname{Vol}(n)}{\frac{1}{1-\gamma} + \lambda} \\
&\geq \operatorname{Vol}(n) \frac{c}{\sqrt{N}(\frac{1}{1-\gamma} + \frac{c}{\sqrt{N}})} \\
&\geq \operatorname{Vol}(n) \frac{c}{\frac{\sqrt{N}}{1-\gamma} + c} \\
&\geq \operatorname{Vol}(n) \frac{c(1-\gamma)}{\sqrt{N} + c(1-\gamma)}
\end{aligned}$$

■

Now that we have established bounds on $d^\pi(n)$, we can establish lower bounds on the probability of sampling a node near s once $\mathcal{B}_\delta(s)$ has been reached.

Corollary 2. *If the ball* $\mathcal{B}_\delta(s)$ *has been reached, then the probability of expanding a node in* $\mathcal{B}_\delta(s)$ *at time* N *is at least* $|\mathcal{B}_{\frac{\delta}{5}}| \frac{c(1-\gamma)}{\sqrt{N}+c(1-\gamma)}$.

Corollary 3. *For* $N \geq c^2(1-\gamma)^2$, $d^\pi(n) \geq \frac{c(1-\gamma)}{2\sqrt{N}} \operatorname{Vol}(n)$ *for all nodes* n.
For $N \leq c^2(1-\gamma)^2$, $d^\pi(n) \geq \frac{1}{2} \operatorname{Vol}(n)$ *for all nodes* n.

Proof. Suppose $N \geq c^2(1-\gamma)^2$. Recall that $d^\pi(n) \geq \operatorname{Vol}(n) \frac{1}{\frac{\sqrt{N}}{c(1-\gamma)}+1}$. Then $1 \leq \frac{\sqrt{N}}{c(1-\gamma)}$. Hence, $\operatorname{Vol}(n) \frac{1}{\frac{\sqrt{N}}{c(1-\gamma)}+1} \geq \operatorname{Vol}(n) \frac{1}{\frac{\sqrt{N}}{c(1-\gamma)}+\frac{\sqrt{N}}{c(1-\gamma)}} = \operatorname{Vol}(n) \frac{1}{2\frac{\sqrt{N}}{c(1-\gamma)}} = \operatorname{Vol}(n) \frac{c(1-\gamma)}{2\sqrt{N}}$

Suppose $N \leq c^2(1-\gamma)^2$. Recall that $d^\pi(n) \geq \operatorname{Vol}(n) \frac{1}{\frac{\sqrt{N}}{c(1-\gamma)}+1}$. Then $1 \geq \frac{\sqrt{N}}{c(1-\gamma)}$. Hence, $\operatorname{Vol}(n) \frac{1}{\frac{\sqrt{N}}{c(1-\gamma)}+1} \geq \operatorname{Vol}(n) \frac{1}{1+1} = \frac{1}{2} \operatorname{Vol}(n)$ ■

Corollary 4. $d^\pi(n) \geq \frac{1}{2} \min(1, \frac{c(1-\gamma)}{\sqrt{N}}) \operatorname{Vol}(n)$ *for all nodes* n.

Now, we can provide a lower bound on the probability that a state in a trajectory will be reached after a given expansion.

Our goal is to take a δ-controllable trajectory and cover each state in a δ-ball. Then we will find a lower bound on the probability of reaching each of these balls in sequence. This will provide us with a high-probability bound on the time it will take to reach the last ball in the sequence.

Lemma 8. *Let M be an MDP with action space A, bounded state space S, and deterministic transition function $\mathcal{T}(s_i, a_i)$. Let d_A be the dimensionality of A. Let τ be a δ-controllable trajectory. Let $\mathcal{B}_\delta(\tau_i)$ be the δ-ball around τ_i, the i-th state in τ. Then $\mathcal{B}_\delta(\tau_{i+1})$ will be reached by time N_{i+1} with probability* $\exp\left(-|\mathcal{B}_{\frac{\delta}{5}}|c(1-\gamma)\sigma\delta^{d_A}(2\sqrt{N_{i+1}}-2\sqrt{N_i})\right)$

Proof. $\mathcal{B}_\delta(\tau_{i+1})$ will be reached if we expand a node in $\mathcal{B}_\delta(\tau_i)$ and then sample an action that takes us to $\mathcal{B}_\delta(\tau_{i+1})$. At timestep $N \geq c^2(1-\gamma)^2$, a node in $\mathcal{B}_\delta(s)$ has a probability of at least $|\mathcal{B}_{\frac{\delta}{5}}|\frac{1}{2}\min\left(1, \frac{c(1-\gamma)}{\sqrt{N}}\right)$ of being expanded. Since τ is a δ-controllable trajectory, we have a probability of at least $\sigma\delta^{d_A}$ of sampling an action that takes the agent to a point in $\mathcal{B}_\delta(\tau_{i+1})$ if a node in $\mathcal{B}_\delta(\tau_i)$ is sampled. Hence, we have a probability of at least

$$|\mathcal{B}_{\frac{\delta}{5}}|\sigma\delta^{d_A}\frac{1}{2}\min\left(1, \frac{c(1-\gamma)}{\sqrt{N}}\right)$$

of reaching the next ball in the sequence at each step. Observe that these samples are drawn independently, so the chance of failing many times in a row is the product of the probability of failure at each time. Then, the probability of failing to reach $\mathcal{B}_\delta(\tau_{i+1})$ by time N_{i+1} is less than or equal to

$$\prod_{t=N_i}^{N_{i+1}} 1 - |\mathcal{B}_{\frac{\delta}{5}}|\sigma\delta^{d_A}\frac{1}{2}\min\left(1, \frac{c(1-\gamma)}{\sqrt{t}}\right)$$

We then have

$$\begin{aligned}
P(\mathcal{B}_\delta(\tau_{i+1}) \text{ not reached} \mid \mathcal{B}_\delta(\tau_i) \text{ reached}) &\leq \prod_{t=N_i}^{N_{i+1}} 1 - |\mathcal{B}_{\frac{\delta}{5}}|\sigma\delta^{d_A}\frac{1}{2}\min\left(1, \frac{c(1-\gamma)}{\sqrt{t}}\right) \\
&\leq \exp\left(\sum_{t=N_i}^{N_{i+1}} \ln\left(1 - |\mathcal{B}_{\frac{\delta}{5}}|\sigma\delta^{d_A}\frac{1}{2}\min\left(1, \frac{c(1-\gamma)}{\sqrt{t}}\right)\right)\right) \\
&\leq \exp\left(-\sum_{t=N_i}^{N_{i+1}} |\mathcal{B}_{\frac{\delta}{5}}|\sigma\delta^{d_A}\frac{1}{2}\min\left(1, \frac{c(1-\gamma)}{\sqrt{t}}\right)\right) \\
&\leq \exp\left(-|\mathcal{B}_{\frac{\delta}{5}}|\sigma\delta^{d_A}\sum_{t=N_i}^{N_{i+1}} \frac{1}{2}\min\left(1, \frac{c(1-\gamma)}{\sqrt{t}}\right)\right)
\end{aligned}$$

This summation is difficult to analyze, so we will instead bound it with an integral that is more tractable.

Observe that $\frac{c(1-\gamma)}{2\sqrt{t}}$ is non-increasing in t. Therefore we can apply the bound

$$\sum_{t=N_i}^{N_{i+1}} \min\left(1, \frac{c(1-\gamma)}{\sqrt{t}}\right) \geq \int_{N_i}^{N_{i+1}} \min\left(1, \frac{c(1-\gamma)}{\sqrt{t}}\right) dt$$

We now attempt to simplify the right hand side. The min operation here produces 3 cases.

1. $N_i < N_{i+1} < c^2(1-\gamma)^2$
2. $N_i < c^2(1-\gamma)^2 < N_{i+1}$
3. $c^2(1-\gamma)^2 < N_i < N_{i+1}$

In the first case, $\int_{N_i}^{N_{i+1}} \min\left(1, \frac{c(1-\gamma)}{\sqrt{t}}\right) dt = \int_{N_i}^{N_{i+1}} 1dt = N_{i+1} - N_i$.
In the second case, we must first split the integral into portions covering $t < c^2(1-\gamma)^2$ and $t \geq c^2(1-\gamma)^2$

$$\begin{aligned}\int_{N_i}^{N_{i+1}} \min\left(1, \frac{c(1-\gamma)}{\sqrt{t}}\right) dt &= \int_{N_i}^{c^2(1-\gamma)^2} 1dt + \int_{c^2(1-\gamma)^2}^{N_{i+1}} \frac{c(1-\gamma)}{\sqrt{t}})dt \\ &= c^2(1-\gamma)^2 - N_i + c(1-\gamma)(\sqrt{N_{i+1}} - \sqrt{c^2(1-\gamma)^2}) \\ &= c^2(1-\gamma)^2 - N_i + c(1-\gamma)\sqrt{N_{i+1}} - c^2(1-\gamma)^2 \\ &= c(1-\gamma)\sqrt{N_{i+1}} - N_i\end{aligned}$$

In the third case, $\int_{N_i}^{N_{i+1}} \min\left(1, \frac{c(1-\gamma)}{\sqrt{t}}\right) dt = \int_{N_i}^{N_{i+1}} \frac{c(1-\gamma)}{\sqrt{t}} dt = c(1-\gamma)(\sqrt{N_{i+1}} - \sqrt{N_i})$.
We can simplify this solution to

$$\min\left(N_{i+1}, c(1-\gamma)\sqrt{N_{i+1}}\right) - \min(N_i, c(1-\gamma)\sqrt{N_i}) = \min\left(t, c(1-\gamma)\sqrt{t}\right) |_{N_i}^{N_{i+1}}$$

Hence,

$$\begin{aligned}\sum_{t=N_i}^{N_{i+1}} \min\left(1, \frac{c(1-\gamma)}{\sqrt{t}}\right) &\geq \int_{N_i}^{N_{i+1}} \min\left(1, \frac{c(1-\gamma)}{\sqrt{t}}\right) dt \\ &= \min\left(t, c(1-\gamma)\sqrt{t}\right) |_{N_i}^{N_{i+1}}\end{aligned}$$

We now see that

$$\begin{aligned}P(\mathcal{B}_\delta(\tau_{i+1}) \text{ not reached} \mid \mathcal{B}_\delta(\tau_i) \text{ reached}) &\leq \exp\left(-|\mathcal{B}_{\frac{\delta}{5}}|\sigma\delta^{d_A} \sum_{t=N_i}^{N_{i+1}} \frac{1}{2} \min\left(1, \frac{c(1-\gamma)}{\sqrt{t}}\right)\right) \\ &\leq \exp\left(-\frac{1}{2}|\mathcal{B}_{\frac{\delta}{5}}|\sigma\delta^{d_A} \min\left(t, c(1-\gamma)\sqrt{t}\right) |_{N_i}^{N_{i+1}}\right)\end{aligned}$$

■

Now that we have a bound on the probability of reaching the next node within a fixed time frame, we will use this to find a bound on the probability of traversing a sequence of points. In other words, we need an upper bound on $P(\mathcal{B}_\delta(\tau_{i+1})$ not reached by time $N)$. To achieve this, we will use the law of total probability, defining

$$\begin{aligned}&P(\mathcal{B}_\delta(\tau_{i+1}) \text{ reached by time } N) \\ &= \sum_{t=0}^{N_{i+1}} P(\mathcal{B}_\delta(\tau_{i+1}) \text{ reached by time } N | \mathcal{B}_\delta(\tau_i) \text{ reached at time } t) P(\mathcal{B}_\delta(\tau_i) \text{ reached at time } t)\end{aligned}$$

However, we do not know the exact probability $P(\mathcal{B}_\delta(\tau_i)$ reached at time $t)$. Instead, we will show that if we have a lower bound $LB_i(t)$ on the probability that $\mathcal{B}_\delta(\tau_i)$ has been reached by time t such that $\lim_{t \to \inf 0_y} LB_i(t) = 1$, then we can define M_i to be the PDF of this upper bound: $M_i(t) = LB_i(t) - LB_i(t-1)$. We show that $\sum_{t=0}^{N_{i+1}} P(\mathcal{B}_\delta(\tau_{i+1})$ reached by time $t) M_i(t)$ is then a lower bound on $P(\mathcal{B}_\delta(\tau_{i+1})$ reached by time $N)$. This is true as long as $P(\mathcal{B}_\delta(\tau_{i+1})$ reached by time $t)$ is monotonically decreasing in t.

Lemma 9. *Let $A(x)$ with $x \in [0, \infty)$ be a function that integrates to 1. Let $B(x)$ with $x \in [0, \infty)$ be a function that integrates to 1. Suppose that for any k, $\int_0^k B(x)dx \leq \int_0^k A(x)dx$. Then if a function $F(x)$ is non-negative and non-decreasing, then $\int_0^\infty B(x)F(x)dx \geq \int_0^\infty A(x)F(x)dx$. Similarly, if F is non-negative and non-increasing, then $\int_0^k B(x)F(x)dx \leq \int_0^k A(x)F(x)dx$.*

Proof. Observe that $\int_0^\infty B(x)F(x)dx \geq \int_0^\infty A(x)F(x)dx$ iff $\int_0^\infty (A(x) - B(x))F(x)dx \leq 0$.

First, recall that for any k, $\int_0^k B(x)dx \leq \int_0^k A(x)dx$. Then,

$$\begin{aligned}
\int_0^k B(x)dx &\leq \int_0^k A(x)dx \\
1 - \int_0^k B(x)dx &\geq 1 - \int_0^k A(x)dx \\
\int_k^\infty B(x)dx &\geq \int_k^\infty A(x)dx \\
0 &\geq \int_k^\infty (A(x) - B(x))dx
\end{aligned}$$

Note that because $F(x)$ is non-negative and non-decreasing, $F'(x) \geq 0$ for all x.

We now introduce an additional integration variable. This will allow us to rearrange to the integral and make the proof easier. Observe that $F(x) - F(0) = \int_0^x F'(y)dy$ Then,

$$\begin{aligned}
\int_0^\infty (A(x) - B(x))F(x)dx &= \int_0^\infty (A(x) - B(x))(\int_0^x F'(y)dy - F(0))dx \\
&= \int_0^\infty (A(x) - B(x)) \int_0^x F'(y)dydx - F(0) \int_0^\infty (A(x) - B(x))dx \\
&= \int_0^\infty (A(x) - B(x)) \int_0^x F'(y)dydx - F(0) \int_0^\infty A(x)dx + F(0) \int_0^\infty B(x)dx \\
&= \int_0^\infty (A(x) - B(x)) \int_0^x F'(y)dydx - F(0)(1) + F(0)(1) \\
&= \int_0^\infty (A(x) - B(x)) \int_0^x F'(y)dydx \\
&= \int_0^\infty \int_0^x (A(x) - B(x))F'(y)dydx
\end{aligned}$$

By Fubini's theorem, we can then rearrange the integrals as follows

$$\begin{aligned}
\int_0^\infty (A(x) - B(x))F(x)dx &= \int_0^\infty \int_0^x (A(x) - B(x))F'(y)dydx \\
&= \int_0^\infty \int_y^\infty (A(x) - B(x))F'(y)dxdy \\
&= \int_0^\infty F'(y) \int_y^\infty (A(x) - B(x))dxdy
\end{aligned}$$

Observe that $\int_y^\infty (A(x) - B(x))dx \leq 0$, and $F'(y) \geq 0$ for all y. Therefore, $F'(y) \int_y^\infty (A(x) - B(x))dx \leq 0$ for all y. It then follows that $F'(y) \int_y^\infty (A(x) - B(x))dx \leq 0$. By our earlier observation, this implies that $\int_0^\infty B(x)F(x)dx \geq \int_0^\infty A(x)F(x)dx$. ■

Theorem 4. *Let τ be a δ-controllable trajectory, with states $s_0 ... s_L$. Let d_A be the dimension of the action space. Let $\mathcal{B}_\delta(\tau_i)$ be the δ-ball around τ_i, the i-th state in τ.*

Then the probability that $\mathcal{B}_\delta(\tau_i)$ will be reached after N expansions is lower-bounded by $1 - \frac{\Gamma(i, \frac{1}{2}|\mathcal{B}_{\frac{\delta}{5}}|\sigma\delta^{d_A}c(1-\gamma)(\sqrt{N_1}-\sqrt{t_0})))}{\Gamma(i)}$

Proof. Note that the first state in the trajectory is the starting state s_0, which is reached by the first time step. Assuming that $\mathcal{B}_\delta(\tau_{i-1})$ was reached at time N_{i-1}, $\mathcal{B}_\delta(\tau_i)$ will be reached by time N_i with probability $1 -$

$\exp\left(-|\mathcal{B}_{\frac{\delta}{5}}|\sigma\delta^{d_A}\min(t, c(1-\gamma)\sqrt{t})\,|_{N_i}^{N_{i+1}}\right)$. We can integrate over the time that $\mathcal{B}_\delta(\tau_{i-1})$ was reached to find a tight bound $LB_i(N_i)$ on the probability of reaching $\mathcal{B}_\delta(\tau_i)$ by time N_i.

We aim to find a closed-form expression $LB_i(t)$ for all i, t. We do this by proof by induction. We show that there exists $LB_i(N_i)$ such that

1. $P(\mathcal{B}_\delta(\tau_i)$ reached by $N_i) \geq LB_i(N_i)$

2. For $N_i < t_0$, $LB_i(N_i) = 0$

3. $\lim_{N_i \to \infty} LB_i(N_i) = 1$

Let $t_0 = c^2(1-\gamma)^2$.

Since the first region is reached when the problem begins, it is clear that $P(\mathcal{B}_\delta(\tau_1)$ reached by $t) = 1$ for all $t \geq 1$. However, we will find that it is more convenient to use the lower bound for the probability $LB_0(t) = 0$ when $t < t_0$ and $LB_0(t) = 1$ when $t \geq t_0$. This trivially gives a lower bound for $P(\mathcal{B}_\delta(\tau_1)$ reached by $N_1) \geq LB_1(N_1)$, where $LB_1(N_1) = 0$ for $N_1 < t_0$ and $LB_1(N_1) = 1 - \exp\left(-\frac{1}{2}|\mathcal{B}_{\frac{\delta}{5}}|\sigma\delta^{d_A}\min(t, c(1-\gamma)\sqrt{t})\,|_{t_0}^{N_1}\right) = 1 - \exp\left(-|\mathcal{B}_{\frac{\delta}{5}}|\sigma\delta^{d_A}c(1-\gamma)(\sqrt{N_1} - c(1-\gamma))\right)$ for $N_1 \geq t_0$.

Let $C = \frac{1}{2}|\mathcal{B}_{\frac{\delta}{5}}|\sigma\delta^{d_A}c(1-\gamma)$. Let $LB_i(N_i) = 0$ for $N_i < t_0$. Then $P(\mathcal{B}_\delta(\tau_i)$ reached by $N_i) \geq LB_i(N_i)$ for $N_i < t_0$.. For the rest of the derivation, we work under the assumption that $N_i \geq t_0$.

Then

$$
\begin{aligned}
P(\mathcal{B}_\delta(\tau_i) \text{ reached by } N_i) &= \sum_{N_{i-1}=t_0}^{N_i} P(\mathcal{B}_\delta(\tau_i) \text{ reached by } N_i \mid \mathcal{B}_\delta(\tau_{i-1}) \text{ reached at } N_{i-1}) P(\mathcal{B}_\delta(\tau_{i-1}) \text{ reached at } N_{i-1}) \\
&\geq \sum_{N_{i-1}=t_0}^{N_i} \left(1 - \exp\left(-C(\sqrt{N_i} - \sqrt{N_{i-1}})\right)\right) P(\mathcal{B}_\delta(\tau_{i-1}) \text{ reached at } N_{i-1}) \\
&\geq \sum_{N_{i-1}=t_0}^{N_i} (1 - \exp\left(-C(\sqrt{N_i} - \sqrt{N_{i-1}})\right)) P(\mathcal{B}_\delta(\tau_{i-1}) \text{ reached at } N_{i-1}) \\
&\geq \sum_{N_{i-1}=t_0}^{\infty} \max\left(0, 1 - \exp\left(-C(\sqrt{N_i} - \sqrt{N_{i-1}})\right)\right) P(\mathcal{B}_\delta(\tau_{i-1}) \text{ reached at } N_{i-1}) \\
&\geq \int_{t_0}^{\infty} \max\left(0, 1 - \exp\left(-C(\sqrt{N_i} - \sqrt{N_{i-1}})\right)\right) P(\mathcal{B}_\delta(\tau_{i-1}) \text{ reached at } \lfloor N_{i-1} \rfloor) dN_{i-1}
\end{aligned}
$$

Observe that $\max\left(0, 1 - \exp\left(-C(\sqrt{N_i} - \sqrt{N_{i-1}})\right)\right)$ is non-negative and non-decreasing in N_{i-1}. Additionally, $LB_{i-1} \leq P(\mathcal{B}_\delta(\tau_{i-1})$ reached by $N_{i-1})$. This means we can apply Lemma 9, using $\frac{dLB_{i-1}}{dN_{i-1}}(N_{i-1})$ to bound

$P(\mathcal{B}_\delta(\tau_{i-1})$ reached at $\lfloor N_{i-1} \rfloor)$.

$$
\begin{aligned}
P(\mathcal{B}_\delta(\tau_i) \text{ reached by } N_i) &\geq \int_{t_0}^{\infty} \max\left(0, 1 - \exp\left(-C(\sqrt{N_i} - \sqrt{N_{i-1}})\right)\right) P(\mathcal{B}_\delta(\tau_{i-1}) \text{ reached at } \lfloor N_{i-1} \rfloor) dN_{i-1} \\
&\geq \int_{t_0}^{\infty} \max\left(0, 1 - \exp\left(-C(\sqrt{N_i} - \sqrt{N_{i-1}})\right)\right) \frac{dLB_{i-1}}{dN_{i-1}}(N_{i-1}) dN_{i-1} \\
&\geq \int_{t_0}^{N_i} \max\left(0, 1 - \exp\left(-C(\sqrt{N_i} - \sqrt{N_{i-1}})\right)\right) \frac{dLB_{i-1}}{dN_{i-1}}(N_{i-1}) dN_{i-1} \\
&\geq \int_{t_0}^{N_i} \left(1 - \exp\left(-C(\sqrt{N_i} - \sqrt{N_{i-1}})\right)\right) \frac{dLB_{i-1}}{dN_{i-1}}(N_{i-1}) dN_{i-1} \\
&\geq \int_{t_0}^{N_i} \left(1 - \exp\left(-C\sqrt{N_i}\right) \exp\left(C\sqrt{N_{i-1}}\right)\right) \frac{dLB_{i-1}}{dN_{i-1}}(N_{i-1}) dN_{i-1}
\end{aligned}
$$

We have then shown that $LB_i(N_i) = 1 - \exp\left(-C\sqrt{N_i}\right) \int_{t_0}^{N_i} \exp\left(C\sqrt{N_{i-1}}\right) \frac{dLB_{i-1}}{dN_{i-1}}(N_{i-1}) dN_{i-1}$ is a lower bound on $P(\mathcal{B}_\delta(\tau_i)$ reached by $N_i)$ if $LB_{i-1}(N_{i-1})$ is a lower bound on $P(\mathcal{B}_\delta(\tau_{i-1})$ reached by $N_{i-1})$. However, we now see that $\frac{dLB_i}{dN_i}(N_i)$ is the more immediately useful term, because it is what appears in our bound. If we find $\frac{dLB_i}{dN_i}(N_i)$ for all N_i, we can use this recurrence relation to calculate a closed-form bound. With this in mind, we now solve for $\frac{dLB_i}{dN_i}$.

$$
\begin{aligned}
\frac{dLB_i}{dN_i} &= \frac{d}{dN_i} \int_{t_0}^{N_i} \left(1 - \exp\left(-C\sqrt{N_i}\right) \exp\left(C\sqrt{N_{i-1}}\right)\right) \frac{dLB_{i-1}}{dN_{i-1}}(N_{i-1}) dN_{i-1} \\
&= \frac{dLB_{i-1}}{dN_{i-1}}(N_i) - \frac{d}{dN_i} \int_{t_0}^{N_i} \exp\left(-C\sqrt{N_i}\right) \exp\left(C\sqrt{N_{i-1}}\right) \frac{dLB_{i-1}}{dN_{i-1}}(N_{i-1}) dN_{i-1} \\
&= \frac{d}{dN_i} \int_{t_0}^{N_i} \frac{dLB_{i-1}}{dN_{i-1}}(N_{i-1}) dN_{i-1} - \frac{d}{dN_i} \exp\left(-C\sqrt{N_i}\right) \int_{t_0}^{N_i} \exp\left(C\sqrt{N_{i-1}}\right) \frac{dLB_{i-1}}{dN_{i-1}}(N_{i-1}) dN_{i-1} \\
&= \frac{dLB_{i-1}}{dN_{i-1}}(N_i) - \frac{C \exp\left(-C\sqrt{N_i}\right)}{2\sqrt{N_i}} \int_{t_0}^{N_i} \exp\left(C\sqrt{N_{i-1}}\right) \frac{dLB_{i-1}}{dN_{i-1}}(N_{i-1}) dN_{i-1} \\
&\quad - \exp\left(-C\sqrt{N_i}\right) \frac{d}{dN_i} \int_{t_0}^{N_i} \exp\left(C\sqrt{N_{i-1}}\right) \frac{dLB_{i-1}}{dN_{i-1}}(N_{i-1}) dN_{i-1} \\
&= \frac{dLB_{i-1}}{dN_{i-1}}(N_i) - \frac{C \exp\left(-C\sqrt{N_i}\right)}{2\sqrt{N_i}} \int_{t_0}^{N_i} \exp\left(C\sqrt{N_{i-1}}\right) \frac{dLB_{i-1}}{dN_{i-1}}(N_{i-1}) dN_{i-1} \\
&\quad - \exp\left(-C\sqrt{N_i}\right) \exp\left(C\sqrt{N_i}\right) \frac{dLB_{i-1}}{dN_{i-1}}(N_i) \\
&= \frac{dLB_{i-1}}{dN_{i-1}}(N_i) - \frac{C \exp\left(-C\sqrt{N_i}\right)}{2\sqrt{N_i}} \int_{t_0}^{N_i} \exp\left(C\sqrt{N_{i-1}}\right) \frac{dLB_{i-1}}{dN_{i-1}}(N_{i-1}) dN_{i-1} - \frac{dLB_{i-1}}{dN_{i-1}}(N_{i-1}) \\
&= -\frac{C \exp\left(-C\sqrt{N_i}\right)}{2\sqrt{N_i}} \int_{t_0}^{N_i} \exp\left(C\sqrt{N_{i-1}}\right) \frac{dLB_{i-1}}{dN_{i-1}}(N_{i-1}) dN_{i-1}
\end{aligned}
$$

We now show by induction that the general solution to this is $\frac{dLB_i}{dN_i} = 0$ for $N_i < t_0$ and $\frac{dLB_i}{dN_i} = \frac{C^i}{2(i-1)!} \exp(-C\sqrt{N_i}) \frac{(\sqrt{N_i} - \sqrt{t_0})^{i-1}}{\sqrt{N_i}}$ for $N_i \geq t_0$, for all $i \geq 1$.

Base Case: Let $i = 1$ Recall that $LB_0(N_0) = 1$ for all $N_0 > t_0$, and $LB_1(N_1) = 1 - \exp\left(-C(\sqrt{N_1} - c(1-\gamma))\right)$.

$$\begin{aligned}\frac{dLB_1}{dN_1} &= \frac{C\exp\left(-C(\sqrt{N_1} - c(1-\gamma))\right)}{2\sqrt{N_1}} \\ &= \frac{C}{2}N_1^{-\frac{1}{2}}\exp\left(-C(\sqrt{N_1} - c(1-\gamma))\right) \\ &= -\frac{C^1}{21!}\frac{(\sqrt{N_1} - \sqrt{t_0})^0}{\sqrt{N_i}}\exp\left(-C(\sqrt{N_1} - c(1-\gamma))\right)\end{aligned}$$

Thus, $\frac{dLB_i}{dN_i} = \frac{C^i}{2(i-1)!}\exp(-C\sqrt{N_i})(N_i)^{\frac{i-2}{2}}$ for $i = 1$.

Inductive case: By the inductive hypothesis,

$$\begin{aligned}&\frac{C\exp(-C\sqrt{N_{i+1}})}{2\sqrt{N_{i+1}}}\int_{t_0}^{N_{i+1}}\frac{dLB_i}{dN_i}\exp(C\sqrt{N_i})dN_i \\ &= \frac{C\exp(-C\sqrt{N_{i+1}})}{2\sqrt{N_{i+1}}}\int_{t_0}^{N_{i+1}}\frac{C^i}{2(i-1)!}\exp(-C\sqrt{N_i})\frac{(\sqrt{N_i} - \sqrt{t_0})^{i-1}}{\sqrt{N_i}}\exp(C\sqrt{N_i}))dN_i \\ &= \frac{C\exp(-C\sqrt{N_{i+1}})}{2\sqrt{N_{i+1}}}\int_{t_0}^{N_{i+1}}\frac{C^i}{2(i-1)!}\frac{(\sqrt{N_i} - \sqrt{t_0})^{i-1}}{\sqrt{N_i}}dN_i \\ &= \frac{C\exp(-C\sqrt{N_{i+1}})}{2\sqrt{N_{i+1}}}(\frac{C^i}{2(i-1)!(\frac{i}{2})}(\sqrt{N_i} - \sqrt{t_0})^i\ |_{t_0}^{N_{i+1}}) \\ &= \frac{C\exp(-C\sqrt{N_{i+1}})}{2\sqrt{N_{i+1}}}(\frac{C^i}{i!}(\sqrt{N_i} - \sqrt{t_0})^i\ |_{t_0}^{N_{i+1}}) \\ &= \frac{C\exp(-C\sqrt{N_{i+1}})}{2\sqrt{N_{i+1}}}\frac{C^i}{i!}((\sqrt{N_i} - \sqrt{t_0})^i - 0^{\frac{i}{2}})\end{aligned}$$

Observe that $i \geq 1$, so $0^{\frac{i}{2}} = 0$

$$\begin{aligned}\frac{C\exp(-C\sqrt{N_{i+1}})}{2\sqrt{N_{i+1}}}\int_{t_0}^{N_{i+1}}\frac{dLB_i}{dN_i}\exp(C\sqrt{N_i})dN_i &= \frac{C\exp(-C\sqrt{N_{i+1}})}{2\sqrt{N_{i+1}}}\frac{C^i}{i!}(\sqrt{N_i} - \sqrt{t_0})^i \\ &= \frac{C^{i+1}}{2i!}\frac{\exp(-C\sqrt{N_{i+1}})}{\sqrt{N_{i+1}}}(\sqrt{N_i} - \sqrt{t_0})^i \\ &= \frac{C^{i+1}}{2i!}\exp(-C\sqrt{N_{i+1}})\frac{(\sqrt{N_{i+1}} - \sqrt{t_0})^i}{\sqrt{N_{i+1}}} \\ &= \frac{C^{i+1}}{2((i+1)-1)!}\exp(-C\sqrt{N_{i+1}})\frac{(\sqrt{N_{i+1}} - \sqrt{t_0})^i}{\sqrt{N_{i+1}}}\end{aligned}$$

Thus, the solution holds for the inductive case. Hence, $\frac{C^i}{2(i-1)!}\exp(-C\sqrt{N_i})\frac{(\sqrt{N_i} - \sqrt{t_0})^{i-1}}{\sqrt{N_i}}$ is the solution for all $i \geq 1$.

It follows that

$$\begin{aligned} LB_i(N_i) &= \int_{t_0}^{N_i} \frac{dLB_i}{dT} dT \\ &= \int_{t_0}^{N_i} \frac{C^i}{2(i-1)!} \exp(-C\sqrt{T}) \frac{(\sqrt{T}-\sqrt{t_0})^{i-1}}{\sqrt{T}} dT \\ &= \frac{C^i}{2(i-1)!} \int_{t_0}^{N_i} \exp(-C\sqrt{T}) \frac{(\sqrt{T}-\sqrt{t_0})^{i-1}}{\sqrt{T}} dT \end{aligned}$$

Here, we can make an interesting observation – this integral is in fact an incomplete Gamma function. Simplifying, we find that

$$\begin{aligned} LB_i(N_i) &= \frac{C^i}{2(i-1)!}(-2C^{-i})\Gamma(i, C(\sqrt{T}-\sqrt{t_0}))\ |_0^{N_i} \\ &= \frac{1}{(i-1)!}(-1)\Gamma(i, C(\sqrt{T}-\sqrt{t_0}))\ |_0^{N_i} \\ &= \frac{1}{(i-1)!}(\Gamma(i) - \Gamma(i, C(\sqrt{N_1}-\sqrt{t_0}))) \\ &= 1 - \frac{\Gamma(i, C(\sqrt{N_1}-\sqrt{t_0})))}{(i-1)!} \\ &= 1 - \frac{\Gamma(i, C(\sqrt{N_1}-\sqrt{t_0})))}{\Gamma(i)} \end{aligned}$$

Hence, $P(\mathcal{B}_\delta(\tau_i) \text{ reached by } N_i) \geq 1 - \frac{\Gamma(i, C(\sqrt{N_1}-\sqrt{t_0})))}{\Gamma(i)}$

■

Observe that the form given is a Gamma distribution over the variable $(\sqrt{N_1} - \sqrt{t_0})$ with shape i and rate C.

D. Experimental Details

D.1. Hardware

All experiments were performed on an Alienware-Aurora-R9 with an 8-core Intel i7-9700 CPU. Since tree operations were the performance bottleneck, we did not use a graphics card for training.

D.2. Hyperparameters

AlphaZero and Volume-MCTS:
For all AlphaZero variants, we set $\lambda = \frac{1}{(1-\gamma)\sqrt{N}}$ (equivalent to setting the exploration coefficient c to $\frac{1}{1-\gamma}$ for AlphaZero). We chose this value due to an insight from our efficient exploration proof. The bound on time needed to reach new states depends on $\sqrt{N} - c(1-\gamma)$. Setting $c = \frac{1}{1-\gamma}$ makes $c(1-\gamma) = 1$, which minimizes the bound on exploration time.

For the loss coefficients, we used $c_V = 1$
$c_{KL} = 10$
$c_A = 1$.

All neural nets use MLPs with ReLU activations and 3 hidden layers of 256 each. Training uses the Adam optimizer with the following hyperparameters

Learning rate = 0.001
β_1, β_2 = [0.9, 0.99]
Weight decay = 0
ϵ = 1e-07
amsgrad: False
These hyperparameters were standard for the implementation our code was based on, and we did not change them.

We did not otherwise do any extensive hyperparameter search. As much as possible, we used the hyperparameter settings from the existing implementation we were comparing to. These included the following hyperparameter values

SST:
Selection radius = 0.3
Witness radius = 0.16

POLY-HOOT:
HOO depth limit = 10
α = 2.5
ξ = 10
η = 0.5

HER:
Replay k = 4
Polyak averaging = 0.95
Entropy regularization = 0.01
Batch Size = 256
Batches per episode = 40

D.3. Setting seeds

For all experiments, we repeat these experiments with three random seeds. We report the average and two-standard deviation confidence interval.

D.4. Data Collection

For all of the Maze environments, we performed 3 training runs, and gathered 10 samples from each. We report the mean and 95% confidence interval for each method.

We found that the Quadcopter environment was significantly higher-variance, so we used more evaluations. Each method was run 60 times. For all the planning methods, this time was spend purely on search instead of learning. We found that planning was much more efficient per environmental interaction than learning, at least on the scale we evaluated for. For each HER run, we initialized a new neural net, trained it for the stated number of environmental interactions, and then evaluated it once.

D.5. Algorithm details

Beyond the algorithm described in the paper, there are a few problem-specific adaptations we make to the algorithms we study in order to improve convergence on the navigation environments. Firstly, we assume that there exists a "stay still" action in the action space that allows the agent to stay in the same state. This is important for two reasons. First, Volume-MCTS and Open-Loop AlphaZero are both open-loop algorithms – they plan out a sequence of actions, and then follow that sequence without replanning at future steps. If they run out of actions in that sequence before the episode ends, the agent takes the "stay still" action until the episode ends. Since the agent always has access to this action, we also lower bound the value estimate for every state as $\frac{1}{1-\gamma}R(s)$, as the agent can always achieve this reward by just repeatedly selecting the "stay still" action.

Data collection: AlphaZero only uses the root node of the tree for training. This works for closed-loop algorithms, because

they run a search at each step of the episode, so they will perform searches with root nodes in every explored location. However, for open-loop algorithms, the root node is always the state that the agent starts the episode in, which may be a much more limited distribution. Therefore, we must use the entire search tree as learning data if we wish to train on data from the whole space. We use data from every node that has at least 1 action to train.

Action selection: MCTS also typically selects the action that has been explored the most times to be executed by the agent. *MCTS as Regularized Policy Optimization* instead chooses to calculate the optimal policy exactly and then samples from it to take actions. For both methods, this is preferable to selecting the action with the highest value, because it encourages exploration. However, as a open-loop algorithm, Volume-MCTS selects an action only after it has completed all its search for the entire episode. The whole search tree can be built and stored for training before actions are executed. It never benefits from selecting suboptimal or exploratory actions for execution, because selecting these actions never leads to different data than it would get by taking the optimal action.

Instead, both Volume-MCTS and Open-Loop AlphaZero keep track of the maximum actual earned reward of each branch, and always select the branch with the highest maximum value at the end of the episode.

D.6. Implementation

Our implementation draws on several existing codebases: an implementation of AlphaZero-Continuous by Moerland et al. (2018), the pyOptimalMotionPlanning package developed by Kris Hauser (https://github.com/krishauser/pyOptimalMotionPlanning). For HER, we draw on Tianhong Dai's implementation of HER (https://github.com/TianhongDai/hindsight-experience-replay) and the implementation from the authors of USHER (https://github.com/schrammlb2/USHER_Implementation) (Schramm et al., 2022). Our POLY-HOOT implementation uses the author's implementation (https://github.com/xizeroplus/POLY-HOOT) (Mao et al., 2020).

D.7. Environment Details

D.7.1. MAZE

In this environment, the agent must navigate a maze to reach a goal. Episodes are 50 steps long. The reward function is 1 in the goal region, and 0 at all other states. If an agent reaches the goal before the end of the episode, the episode ends and the agent receives a reward of 1 for each remaining time step left in the episode.

We tested two sets of dynamics on the maze environment. Geometric dynamics are simple; the state space and action spaces are both 2-dimensional, and $s_{t+1} = s_t + v_{max} a_t$, where s_{t+1} is the next state, s_t is the current state, a_t is the action, and $v_{\max}$ is the maximum speed allowed by the environment. If this movement would cause the agent to collide with a wall, instead the agent does not move ($s_{t+1} = s_t$).

Dubins car dynamics are slightly more complicated. The state space has three dimensions: two position coordinates and one rotation coordinate. The action space is two-dimensional. The agent selects a forward/backward speed and a turning angle, which is bounded to give the agent a minimum turning radius. The dynamics are as follows: Let x, y be the car's x and y coordinates. Let θ be the car's rotation coordinate. Let v_{max} be the car's maximum speed and ϕ_{max} be the car's maximum steering angle. Let a_0 be the first dimension of the action, controlling the car's speed. Let a_1 be the second dimension of the action, controlling the car's steering.

Then the next state described by the variables x, y, θ is found by numerically integrating the differential equation

$$\frac{dx}{dt}(t) = a_0 \cos\theta(t) \frac{dy}{dt}(t) = a_0 \sin\theta(t) \frac{d\theta}{dt}(t) = a_1$$

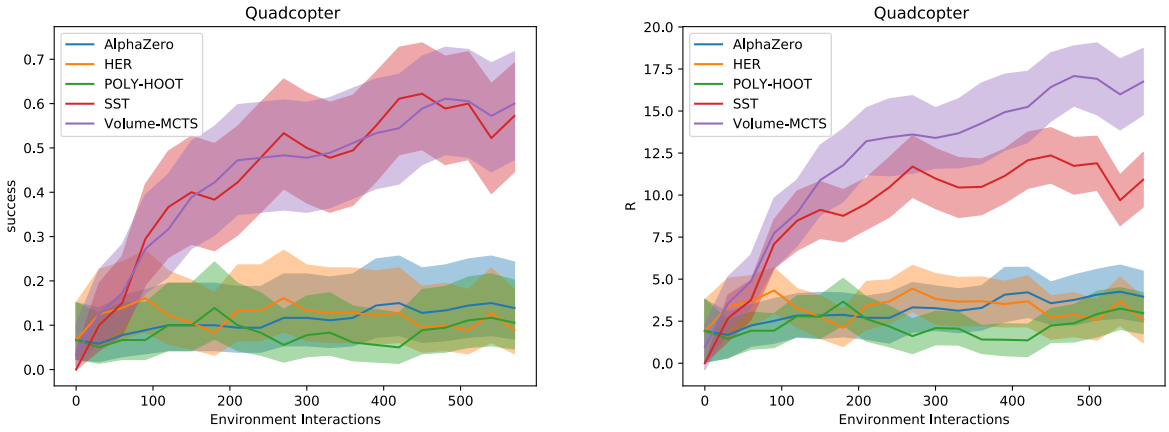
from time t to time $t+1$. The $x(t+1), y(t+1), \theta(t+1)$ found at the end of this numerican integration is the next state.

D.7.2. QUADCOPTER

The Quadcopter environment is taken from Sivaramakrishnan et al. (2023). In this environment, the agent must navigate a quadcopter around a series of pillars to reach a goal. Episodes are 30 steps long. The reward function is 1 in the goal region, and 0 at all other states. If an agent reaches the goal before the end of the episode, the episode ends and the agent receives a reward of 1 for each remaining time step left in the episode. The dynamics of this environment are given by a Mujoco simulation.

D.8. Additional Experiments

In the experiments section, we reported that Volume-MCTS outperformed SST on reward, but not success rate. Here we provide addition details on this finding.



(a) Success rate as a function of total environmental interactions (b) Reward as a function of total environmental interactions

Figure 3. Reward and success rate on Quadcopter environment

Here, success is defined as reaching the goal within the 30-step episode. The goal state is treated as a s SST and Volume-MCTS reach the goal roughly the same fraction of the time. However,