Monte Carlo Tree Search With Iteratively Refining State Abstractions

Anonymous Author(s)
Affiliation
Address
email

Abstract

Decision-time planning is the process of constructing a transient, local policy with the intent of using it to make the immediate decision. Monte Carlo tree search (MCTS), which has been leveraged to great success in Go, chess, shogi, Atari, and other settings, is perhaps the most celebrated decision-time planning algorithm. Unfortunately, in its original form, MCTS can degenerate to one-step search in domains with stochasticity. Progressive widening is one way to ameliorate this issue, but we argue that it possesses undesirable properties for some settings. In this work, we present a method, called abstraction refining, for extending MCTS to stochastic environments which, unlike progressive widening, leverages the geometry of the state space. We argue that leveraging the geometry of the space can offer significant advantages. To support this claim, we present a series of experimental examples in which abstraction refining outperforms progressive widening.

1 Introduction

In control problems, an agent makes sequential decisions toward the end of accumulating a large amount of reward [17]. Many reinforcement learning algorithms approach this task by computing a policy, which, given a state, dictates the agent's behavior; others operate under the paradigm of decision-time planning, in which, after the agent has arrived at a state, it spends additional computation constructing or revising its policy for the immediate decision(s). The decision-time planning paradigm can be advantageous because it allows for an additional step of policy improvement that is not available to agents acting according to a predetermined policy.

Of the numerous ways to perform decision-time planning, Monte Carlo tree search (MCTS) is among the most influential [4, 8, 12]. Algorithms powered by MCTS, like AlphaZero and MuZero, are dominant in games with complex value function landscapes, like Go, chess, and shogi [2, 16, 15]. MuZero even shows strong performance on Atari games, where model-free algorithms had previously been the most performant.

Unfortunately, in its original form, MCTS can degenerate to a one-step search when the transition function has an infinite support. Even if the support is finite but non-trivial, the performance of MCTS can suffer. However, despite the importance of stochasticity in real-world problems, relatively little attention has been paid to this issue, perhaps as a result of the fact that many popular benchmarks are deterministic or close-to-deterministic.

One way to address this issue is by using progressive widening [7]. Progressive widening works by alternating between expanding the width of the tree (sampling new states to add to the tree) and the depth of the tree (sampling among states already in the tree), such that, asymptotically, the tree will both be infinitely deep and infinitely wide. It thereby avoids vanilla MCTS's potential for degeneracy. Additionally, progressive widening includes hyperparameters that control its propensity to expand the

Submitted to 35th Conference on Neural Information Processing Systems (NeurIPS 2021). Do not distribute.

width (as opposed to the depth) of its trees. These hyperparameters can interpolate between width-wise expansion only (vanilla MCTS) and depth-wise expansion only (transition determinization). As a result, with proper tuning, progressive widening is capable of peforming well in both settings with important stochasticity and and in settings with unimportant (or non-existent) stochasticity.

However, progressive widening's hyparameters are global, meaning its expansion decision rule must be the same for every transition. In settings with non-uniform stochasticity, this property is undesirable and requires progressive widening to compromise between focusing on unimportant stochasticity and ignoring important stochasticity.

In this work, we propose a new method, called abstraction refining, for extending MCTS to stochastic settings. Abstraction refining uses random and iteratively refining state abstractions defined by the geometry of the state space. Intuitively, if a newly sampled state is similar to a state that is already in the tree, it is discarded in favor of the pre-established state; otherwise, the newly sampled state is added to the tree. Because the criteria for similarity becomes more strict the more often an abstraction is used, abstraction refining guarantees that, in the limit, the search tree will grow both infinitely wide and infinitely deep. For our contributions, we present both a proof that, when used for policy evaluation, abstraction refining converges almost surely in finite MDPs and a series of experimental examples in which abstraction refining is able to outperform progressive widening.

2 Background

We consider a Markov decision process (MDP) setting, with state space $\mathcal{S}$, action space $\mathcal{A}$, transition function $\mathcal{T}$, reward function $\mathcal{R}$ and discount factor γ. The objective is to determine a policy π that achieves a large expected cumulative reward $\mathbb{E}_\pi\left[\sum_{t=0}^{\infty}\gamma^t\mathcal{R}(s_t,a_t)\right]$. Rather than explicitly computing a function π, decision-time planning algorithms' policies are defined implicitly by their planning procedures. These planning procedures are given generative access to (i.e., the ability to sample from) the transition function $\mathcal{T}$.

We also consider a Markov reward process (MRP) setting, with state space $\mathcal{S}$, transition function $\mathcal{T}$ and reward function $\mathcal{R}$. An MRP is like an MDP, except that the agent is a passive observer, rather than a participator. Equivalently, one could also think of an MRP as an MDP in which the policy has been fixed and baked into the transition and reward functions. The goal of an agent in an MRP is to evaluate the expected return from state—i.e., to perform policy evaluation.

2.1 Monte Carlo Tree Search

MCTS works by iteratively building a look-ahead tree of states. For each action a at a state s, the algorithm keeps track of $N(s,a)$, the number of times the action has been selected at that state, and $Q(s,a)$, the average of the value assessments of that action. At each iteration, the agent (i) *selects* a leaf node to expand; (ii) *expands* the tree by adding a child node of the leaf node; (iii) *evaluates* the newly added state, and (iv) *updates* the nodes on the path from the new state to the root according to the evaluation.

Selection In the selection phase, starting from the root node, which corresponds to the state s_0 currently occupied by the agent, MCTS traverses the tree until a leaf node is reached. The tree traversal is dictated by a *selection policy*. The most commonly used selection policies are based on variants of the UCT algorithm [12] which selects the action at each state that maximizes $Q(s,a)+c\cdot\sqrt{\frac{2\log N(s)}{N(s,a)}}$, where $N(s)$ is the total number of times a state is visited and c is a hyperparameter that controls the amount of exploration. Although it is less common, MCTS can also be used as a policy evaluation algorithm. In this case, the selection policy is set to be the policy being evaluated.

Expansion Once a leaf node is reached, the tree is expanded by adding a child of the leaf node to the tree. If the leaf node is a terminal state, no children are added and the algorithm proceeds to evaluation.

Evaluation Whenever MCTS adds a new state to the tree, the state is evaluated. Traditionally, MCTS practitioners performed evaluation by generating a rollout to the end of the game using a

fast-to-evaluate simulation policy. Modern implementations often use a parameterized value function in lieu of simulation rollouts to evaluate a state.

Update After MCTS evaluates a state, it updates the nodes along the path from the root to the evaluated state. Denote the path from the root to the leaf using $s_0, a_0, r_0, \ldots, s_T$. Then for each pair (s_k, a_k), MCTS updates $Q(s_k, a_k) \leftarrow$ weighted_average $\Big(Q(s_k, a_k), v(s_T) + \sum_{t=k}^{T-1} \mathcal{R}(s_t, a_t)\Big)$, where the the weights are $N(s_k, a_k)$ and one, respectively. Additionally each visit counter $N(s_k, a_k)$ gets incremented by one.

Selecting an Action To Execute After the search is over, in a control setting, MCTS selects an action for the agent to take. Common ways of doing this include selecting the action at the root node with the highest average Q-value, the action with highest lower confidence value or the action with the highest visit count. When used in an evaluation context, MCTS returns the average value of the root note—i.e. $\frac{1}{n} \sum_a N(s_0, a) Q(s_0, a)$.

What about Stochasticity? The description of MCTS above assumes that the environment is determinstic. In games with stochasticity, the selection process has an additional complication. Specifically, after an action a is selected at a state s, there is no uniquely determinable next state s', but rather, a distribution over next states $\mathcal{T}(s, a)$. The natural way to extend MCTS to this setting is to sample $s' \sim \mathcal{T}(s, a)$. If there is already a child node for s', then that node is selected and the selection process continues. Otherwise, if there is no child node for s', one is added to the tree and evaluated. The consequence of this modification is that, for a fixed number of iterations, the search tree cannot be as deep as it would be in a deterministic setting. In the most extreme case, where there are an infinite number of possible next states, this modification can degenerate MCTS into one-step search (because it may be the case that no state is sampled twice). This means that the performance of MCTS in games in which search trees are crucial for good performance could be ruined by arbitrarily small (but infinite support) perturbations to state representations.

2.2 Progressive Widening

Progressive widening [7] is one way to augment MCTS in such a way that it can build deep search trees in arbitrarily stochastic domains.[1] The basic idea behind progressive widening is to alternate between sampling new next states and sampling among the next states that are already in the tree. If the state-action pair (s, a) has been tried a large number of times relative to the number of successor states in the tree, progressive widening adds a new state. On the other hand, if there are a large number of successor states relative to the number of times (s, a) has been tried, progressive widening samples among the successor states already in the tree. Whether or not to sample is determined by the boolean num_children$[s, a] < k \cdot$ num_visits$[s, a]^\alpha$ where $\alpha \in [0, 1]$ and $k \in \mathbb{R}_+$ are hyperparameters. Simplified pseudocode for the sampling step of progressive widening is shown in Algorithm 1.

Intuitively, the hyperparameter α can be thought of as a propensity to sample new states (i.e., add width) rather than sample among states existing in the tree (i.e., add depth). When $k = 1$, $\alpha = 0$, progressive widening reduces to transition determinization—the first time that a state-action pair is visited, a successor state is sampled and added to the tree; thereafter, the same successor state is selected every the state-action pair is visited. When $k = 1$, $\alpha = 1$, progressive widening reduces to vanilla MCTS—every time a state action pair is encountered, a new successor state is sampled and added to the tree.

If k and α can be properly tuned, progressive widening offers significant flexibility. In domains in which stochasticity is very important, α can be set to one or close to one. In domains in which stochasticity can be ignored, α can be set to zero or close to zero. Otherwise, an intermediate value often works well.

However, there is a downside to progressive widening. In particular, these hyperparameters are determined globally. This means that, if stochasticity is important in part of the MDP but unimportant

[1] Note that what we have been describing as progressive widening is in some literature referred to as double progressive widening. Double progressive widening uses progressive widening over both the action space and the state space. For simplicity, this work restricts its attention to small action spaces, and therefore uses the term progressive widening to refer to the state space behavior.

another part of the MDP, the best thing that progressive widening can do requires either placing too much emphasis on unimportant stochasticity, or too little on important stochasticity, or both.

Algorithm 1 Progressive Widening

```
procedure SAMPLE(s, a)
    N ← num_visits[s, a]
    if num_children[s, a] < kN^α then
        s' ~ T(s, a)
        if s' ∉ children[s, a] then
            children[s, a].add(s')
        return s'
    else
        return sample(children[s, a])
```

Algorithm 2 Abstraction Refining

```
procedure SAMPLE(s, a)
    s' ~ T(s, a)
    s'' ← nearest_neighbor(s', children[s, a])
    if d(s', s'') < ε_num_visits[s,a,s''] then
        return s''
    else
        children[s, a].add(s')
        return s'
```

3 Abstraction Refining

In this work, we introduce an alternative to progressive widening that we call abstraction refining. Abstraction refining is motivated by the deficiency of progressive widening described above: in settings with heterogeneous stochasticity, progressive widening is forced to make an undesirable compromise. Rather than defining a global expansion function, as progressive widening does, abstraction refining determines whether or not to expand using local information via state abstractions—collections of states that are grouped together and considered as a single state [13]. At each step, abstraction refining samples a new state and checks if it is similar to an existing child. If it is, abstraction refining abstracts them together and selects the similar, existing child; if it is not, abstraction refining adds the new state to the tree. In doing so, in principle, abstraction refining avoids the weakness of progressive widening: in areas of the MDP in which stochasticity is unimportant, abstraction refining can cluster states together and in areas of the MDP in which stochasticity is important, abstraction refining can cluster states individually. Simplified pseudocode for abstraction refining is given in Algorithm 2.

Each state abstraction is defined by two principles. The first principle is that, if a state is to be abstracted, it should be abstracted to its nearest neighbor among states already in the tree. Equivalently, this principle states that, when abstractions take place, they should use a mapping induced by the Voronoi diagram over the children in the tree. (A Voronoi diagram is a partition over a space based on cluster centers. The number of blocks (or cells) in the partition is equal to the number of cluster centers and each block is associated with a cluster center. Each point in the space belongs to block associated with the nearest cluster.)

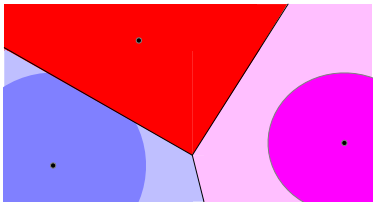

Figure 1: A visual depiction of an example low dimensional state space. There are three states already in the tree, represented by the black dots. The Voronoi diagram is shown using the colors blue, red and purple. The intersection of the Voronoi cells with each child's epsilon ball, which corresponds to states that would be abstracted if they were sampled, is shown with darker shading. The lighter shading shows the states that would be added to the tree if they were sampled.

The second idea is that, if abstraction is used many times, its notion of similarity should become more precise. This idea is motivated by controlling the amount of error that is introduced by a state abstraction. A necessary condition for asymptotic correctness is that state abstractions become arbitrarily small in the limit. Toward that end, abstraction refining makes use of a sequence $\{\epsilon_n\}_{n\in\mathbb{N}}$ that is strictly decreasing ($\epsilon_n > \epsilon_m$ for $n < m$) and going to zero in the limit ($\lim_{n\to\infty} \epsilon_n = 0$). A state that has been selected n times previously can only abstract new states within a distance of ϵ_n.

Let $\mathcal{R} = \{R_{s''} : s'' \in \text{children}[s, a]\}$ be the Voronoi diagram induced by the children of the state action pair. Let $\mathcal{B} = \{B_{s''} = B_{s''}(\epsilon_{\text{num_visits}[s,a,s'']}) : s'' \in \text{children}[s, a]\}$ be the set of epsilon balls centered at the children of the state action pair, where the radius of each ball is determined by the sequence $\{\epsilon_n\}$ and the number of visits. Then the set of points that gets abstracted to s'' is exactly

$R_{s''} \cap B_{s''}$; the set of points that get abstracted is $\cup_{s''} R_{s''} \cap B_{s''}$; and the set of points which would be added to the tree is $\cap_{s''} R^c_{s''} \cup B^c_{s''}$. A low dimensional example is shown in Figure 1.

3.1 Consistency

What can we say about abstraction refining's theoretical properties? Abstraction refining is not a policy optimization algorithm, so it does not make sense to consider whether it converges to an optimal policy. Instead, abstraction is more like a policy evaluation algorithm. Thus, we could ask: at what rate does abstraction refining converge to the correct value? However, this rate is heavily dependent on the regularity of the transition function and the evaluation function, making it unlikely to apply in practice. Thus, instead, we simply show that abstraction refining is asymptotically consistent.

Proposition 1. *Let $G = (\mathcal{S}, \mathcal{R}, \mathcal{T})$ be a finite Markov reward process with horizon h and bounded reward. Then for any $s \in \mathcal{S}$, abstraction refining's assessment of state s converges to $v(s)$, almost surely, where $v(s) = \mathbb{E}\left[\sum_{t=d}^{h-1} \mathcal{R}(S_t) \mid S_d = s\right]$, given a bounded evaluation function.*

Proof. We proceed by induction. First, consider that for any state s_{h-1}, abstraction refining converges surely after one iteration because $v(s_{h-1}) = \mathcal{R}(s_{h-1})$. Now consider some state s_d at arbitrary depth d. The value assessed by abstraction refining after n iterations is $V_n(s_d) = \sum_{s \in \mathcal{S}_{d+1}} \frac{N_{s,n}}{n} V_{N_{s,n}}(s)$ where $N_{s,n}$ denotes the number of times that state s has been visited by the nth iteration. Note that, by Lemma 1, we have that $N_{s,n}/n$ converges to $\mathcal{T}(s \mid s_d)$ almost surely. Note that by Lemmas 2 and 3, and by inductive hypothesis, $V_{N_{s,n}}(s)$ converges to $v(s)$ almost surely. Then invoking the fact that the term-wise product of sequences of almost surely convergent random variables converges almost surely to the product of the limits, and the fact that the term-wise sum of sequences of almost surely convergent random variables converges almost surely to the sum of the limits, we conclude that $V_n(s_d)$ converges to $\sum_{s \in \mathcal{S}_{d+1}} \mathcal{T}(s \mid s_d) v(s) = v(s_d)$ almost surely. □

Proofs for the lemmas below can be found in the supplementary material.

Lemma 1. *For a fixed s, the sequence $N_{s,n}/n$ converges almost surely to $\mathcal{T}(s \mid s_d)$.*

Lemma 2. *For a fixed s with $\mathcal{T}(s \mid s_d) > 0$, the sequence $N_{s,n}$ diverges.*

Lemma 3. *Let X_n be a sequence converging to x almost surely. Let $f : \mathbb{N} \to \mathbb{N}$ be a monotonically increasing, onto function. Then $Y_n = X_{f(n)}$ converges to x almost surely.*

Given Proposition 1, it is relatively straightforward to show the same property holds in a discounted infinite horizon case.

Corollary 1. *Abstraction refining also converges almost surely for infinite horizon finite Markov reward processes, with rewards discounted by $\gamma \in [0, 1)$.*

Proof. (Sketch) Fix $\epsilon > 0$. Select h such $\sum_{t=h}^{\infty} \gamma^t m < \epsilon/4$, where m is a bound on the reward and evaluation function. Then the amount of error that can be accrued from states beyond time h is bounded by $\epsilon/2$, surely. Additionally, by Proposition 1, the amount of error that can be accrued is bounded by $\epsilon/2$ for all sufficiently large n, almost everywhere. Therefore, the aggregate error is bounded by ϵ, almost everywhere for all sufficiently large n. □

3.2 Runtime Analysis

In a decision-time planning paradigm, which is where MCTS is most often used, planning time is scarce. Thus, the runtime of algorithmic extensions to MCTS is important. However, high performance MCTS implementations almost ubiquitously use large amounts of parallelization [6]. As a result, the most important analysis regards non-parallizable runtime.

Abstraction refining performs two potentially expensive computations. First, it samples a state $s \sim \mathcal{T}(s, a)$, which could be costly if the simulator is slow, or if sampling is being executed from a large learned model. (Note that progressive widening would also suffer from this issue.) However sampling is unavoidable the first time that a state action pair is visited and is perfectly parallelizable. Therefore, an implementation could completely amortize the cost of sampling by sampling a sufficiently large number of next states, such that no more would have to be generated

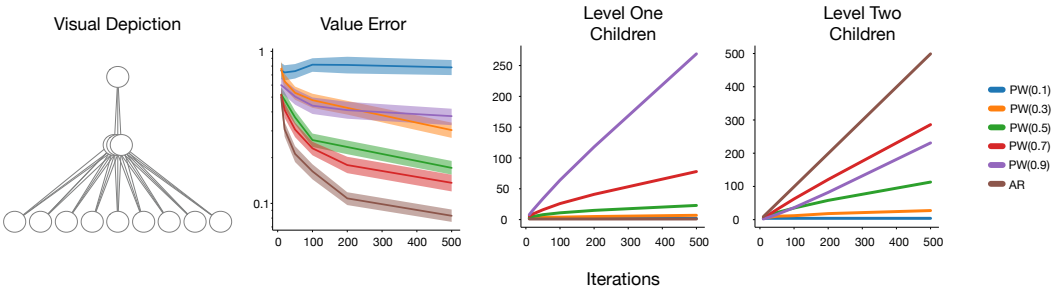

Figure 2: A comparison between progressive widening and abstraction refining on a toy MRP designed to illustrate the drawback of progressive widening's global hyperparameters. (far left) A visualization of the MRP; (middle left) value error, shown on a log scale, as a function of iterations; (middle right) number of nodes at depth one of the tree as a function of the number of iterations; (far right) number of at depth two of the tree as a function of the number of iterations.

during the search, the first time a state action pair is visited. Moreover, modern high performance implementations of MCTS, whose runtime is dominated by expensive function evaluators, typically only do on the order of 1000 iterations or fewer, making this parallelization practical in many cases.

The second potentially expensive computation is computing the nearest neighbor. (Note that progressive widening shares some, but not all, of this cost—it must check whether the sampled state is already in the tree, but does need to compute a distance.) For serial computation, the cost of computing a distance between two states embedded in $\mathbb{R}^d$ is $O(d)$ for most reasonable distance functions and must be done for each child, yielding an aggregate cost of $O(d\,\text{num_children}[s, a])$. However, the cost of doing this across children can be completed parallelized and the cost of computing the distance for any individual child can be parallelized to a runtime of $O(\log d)$. Selecting the smallest distance requires an additional $O(\text{num_children}[s, a])$ cost, but it can also be parallelized logarithmically. Thus, the aggregate cost of a maximally parallelized system is $O(\log d + \log \text{num_children}[s, a])$, which is modest compared to the cost of modern state evaluators. Comparatively, the analogous cost of progressive widening for a maximally parallelized system is $O(\log \text{num_children}[s, a])$.

4 Experiments

4.1 Illustrating Progressive Widening's Weakness

For our first experiment, we seek to confirm the claim made in previous parts of the paper—that progressive widening is forced to make an undesirable compromise in situations with heterogeneous stochasticity. Toward that end, we construct a toy Markov reward process, depicted visually in Figure 2 (far left). In the MRP, the agent begins in the single state at the top. The first transition moves the agent to one of an infinite number of very similar states (level one), shown in the middle. At these states, the reward function is zero and the transition function is nearly identical. The second transition moves the agent to one of an infinite number of dissimilar states (level two), shown at the bottom. The agent is given a bad value function for the level one states and a good value function for the level two states, reflecting a situation in which searching deeply is important. To evaluate the expected return in the fewest number of iterations, the optimal thing for a state selector to do is to add as few level one children as possible and as many level two children as possible.

Unfortunately, this problem is difficult for progressive widening to address. For $k = 1$, setting α to a small value restricts the number of level one children but also restricts the number of level two children (see blue and orange in middle right and far right plots; Figure 2). But setting α to a large value causes progressive widening to repeatedly add a large number of level one children, preventing focus on level two children (see purple in middle and far right plots; Figure 2). Intermediate values of α (see red and greed in middle and far right plots; Figure 2) offer the best compromise but, nevertheless, add too many level one children and too few level two children.

In contrast, abstraction refining is perfectly suited to address this problem—it can abstract the very similar level one states together while treating the dissimilar level two states separately using $\epsilon_n = n^{-0.1}$. As a result, it can spend its entire quota on level two states (see brown in middle and

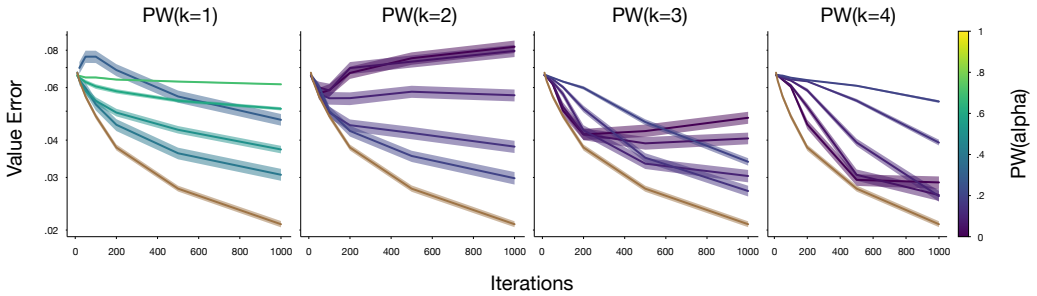

Figure 3: Abstraction refining (brown), with a fixed set of hyperparameters, compared against progressive widening with various hyperparameter values on policy evaluation in a continuous variant of blackjack. The value error (y-axis) is shown on a log scale.

far right plots; Figure 2). These distinctions manifest directly in policy evaluation error, as shown in Figure 2, middle left plot, where abstraction refining is able to outperform progressive widening over each α value.

We also ran these experiments using other values of k and found qualitatively similar results. For these plots, a complete specification of the MRP, and code for these experiments, etc., see the supplementary material.

4.2 Policy Evaluation in Blackjack

The previous section illustrated that, in a policy evaluation setting designed adversarially for progressive widening, abstraction refining can offer significant benefits. To confirm that the qualitative takeaway of the previous section was not solely a result of the adversarial design, this section investigates policy evaluation on a less contrived setting—a variant of blackjack with continuous card values. For our policy, we chose a strategy described by *Wizard of Odds Consulting*. The details of the game, the evaluation policy, the code used to generate these results, as well as a plot for $k = 1/2$, which did not perform competitively, can be found in the supplementary material.

The results are shown in Figure 3. Abstraction refining with $\epsilon_n = 2n^{-0.1}$ is shown in brown in each subplot. The k value of progressive widening increases from one to four from left to right. Darker colors correspond to smaller lighter values of alpha. An effort was made to prune noncompetitive α values from the graph to reduce visual clutter.

There are a number of observations to make. First, notice the relationship between k and α for progressive widening. As k becomes the larger, the most competitive values of α become smaller, reflecting the fact that k also plays a role in progressive widening's propensity to expand its width. Note that, in general, the role k plays is subtle—for $k > 1$, the number of children at a particular grows as $\min(n, kn^{\alpha})$. This means that, for the first $k^{1/(1-\alpha)}$ times a node is visited, it always adds a child but, thereafter, only adds a new child a much smaller proportion of the time.

Second, notice the instability that can result from small α values. This reflects the reality that small α values can build deep trees quickly, which can be advantageous for getting high quality evaluations, but expand slowly width-wise, leading to repeated evaluations of states that do not fully reflect the true distribution.

Lastly, notice that while some settings of progressive widening perform competitively with abstraction refining for some iteration numbers, there is no setting that performs competitively across all iterations. In other words, progressive widening does not perform competitively with abstraction refining on an *anytime* basis. While this is not necessarily problematic in a strictly policy evaluation context, it is an undesirable for a subcomponent of MCTS, used for control, for two reasons: first, because MCTS uses previous evaluations to inform its selection decisions, performing inaccurate evaluations at some iteration horizon could cause it to misallocate its budget to the wrong actions; second, because MCTS visits different actions and states vastly different numbers of times in a fashion that is not initially predictable, there is no good way to determine which iteration horizon to optimize for.

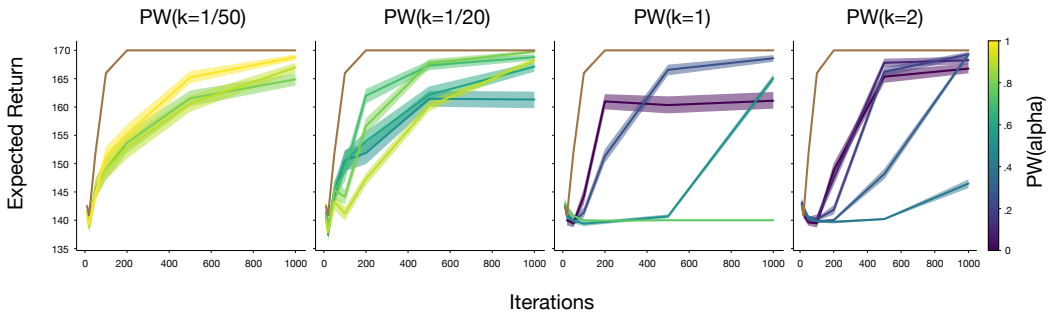

Figure 4: Abstraction refining (brown), with a fixed set of hyperparameters, compared against progressive widening with various hyperparameter values on control in a variant of the trap problem.

4.3 Control in Trap

Thus far, we have focused on comparing state selectors in a policy evaluation setting. While such comparisons make sense, as state selectors serve an evaluatory purpose, MCTS is rarely used for policy evaluation. Therefore, the utility of a state selector is closely linked to its ability to increase performance on control tasks. For our first control experiment, we use a variant of the trap problem, which was first introduced in the original progressive widening paper [7] to demonstrate the advantages of progressive widening compared to vanilla MCTS. The basic idea of problem is that an agent must move from one platform to another without falling into the trap. Stochasticity comes in the form of actuator error—the amount the agent intends to move differs from the amount the agent actually moves by stochastic noise. A complete specification of the problem, hyperparameter settings, plots for additional k values, and code for this experiment can be found in the supplementary material.

The results are shown in Figure 4. Abstraction refining with $\epsilon_n = 0.1n^{-0.1}$ is shown in brown in each subplot. Again, the k value of progressive widening is shown increasing from left to right, the α value is depicted by hue, with darker values meaning smaller αs, and the plots were pruned of non-competitive results to result visual clutter.

First, focus on the results for $k = 1$. Note that there is an interesting trade-off between small α which are able to search deeply quickly but ultimately level off, and large α, which take longer to show improvement, but level off less than the smaller α do. Next, note that, interestingly, unlike in the blackjack experiment, the most competitive values of k are fractional. However, no hyperparameter setting for progressive widening performs comparably to abstraction refining, which plays optimally within 200 iterations. In contrast, the progressive widening requires 1000 iterations to achieve the optimal expected return.

4.4 Opponent Exploitation in Five by Five Go

For our last experiment, we use AlphaZero to compare abstraction refining and progressive widening in opponent exploitation in five by five Go. In our setup, the first moving player (black), is controlled by an AlphaZero policy network. The second moving player (white), performs search using the latent state representation of the AlphaZero network as its state representation of the system. Because the black player's policy is fixed, the white player can view the game as a stochastic MDP in which the transition function is dictated by the black player's policy. We also add a small amount of noise to the observation function to increase the difficulty of the search problem. We quantify performance using the amount of territory controlled by a player at the end of a game (i.e., the margin of victory), rather than the win-loss value. We modified OpenSpiel's [14] AlphaZero and Go implementations for this experiment. Discussion of these design choices, as well as a more complete description of the experimental setup, and the code that was used to produce these results, can be found in the supplementary material.

Results are shown in Figure 5, where $k = 1$ for progressive widening. While the results are somewhat uncertain, there appears to be upward trends in α, suggests that larger values of α perform better, and in the number rollouts, suggesting that larger rollout budgets perform better, as expected. Lastly,

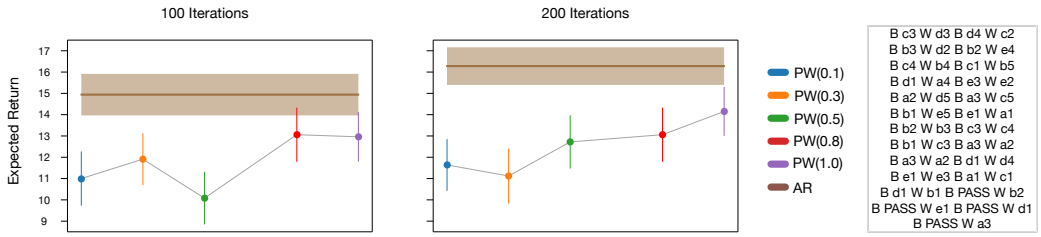

Figure 5: (left) Abstraction refining (brown), compared against progressive widening with $k = 1$ and various α values for opponent exploitation in five by five Go; (right) an example game in which the exploiter wins by the maximum margin (25).

it appears to be the case that abstraction refining is able to outperform progressive widening—possibly because it is able to abstract together board states that are behaviorally identical but differing superficially by the stochastic noise. In contrast, progressive widening is forced to treat these states separately. Unfortunately, the variance on the outcome of these experiments was large and each run required a significant amount of computation to perform, so the takeaways are less certain than those of the other experiments.

5 Closing Discussion

On The Distance Function An important facet of abstraction refining that, heretofore, has gone largely uncommented on, is that it requires a notion of distance between states in order to compute the nearest neighbor. It is readily apparent that using Euclidean distance over the canonical state representations, as we did in some our experiments, would perform poorly in many cases. And while our AlphaZero experiments offer some initial work suggesting that using abstraction refining over learned state embedding can be successful, it would be better to have a more principled way of constructing this distance metric. Perhaps the most natural notion of state similarity for abstraction refining is a bisimulation metric [10, 9]. Informally, under a bisimulation metric, two states are similar if, for all actions, (i) the states yield similar rewards and (ii) the states have similar transition distributions (defined recursively in terms of the bisimulation metric)—in short, if they are behaviorally similar. However, while there has been recent progress in learning bisimulation metrics [20, 5], existing methods that are scalable are on policy and remain limited to either deterministic MDPs or those with unimodal stochasticity.

Related Work While existing work relating to the state selection for MCTS is limited to progressive widening, there are a number of other works that are related in less direct ways. Anthony et al., similarly motivated by performing search in settings with large branching factors, proposes using a policy gradient, in place of MCTS, for expert iteration [3]. There are also a number of works that use a notion of distance between actions to limit the branching factor and improve action selection. Ahmad et al. show how to leverage Delaunay triangulation, a concept related to Voronoi diagrams, for action selection in a single-decision setting [1]. Yee et al. propose incorporating kernel regression into MCTS for continuous action spaces [19]. Most recently, Kim et al. show how Voronoi diagrams can be applied to extend MCTS to a continuous action space [11]. There has also been work leveraging state similarity for MCTS. Specifically, Xiao et al. show how kernel regression can be used to improve state evaluations for AlphaZero, using an intermediate layer of the network, as we do, to compute state similarity [18].

Summarizing Remarks This work introduces abstraction refining, a new method for performing state selection for MCTS in stochastic environments. We prove that abstraction refining converges in finite MRPs, offer analysis suggesting that its runtime can be competitive with that of progressive widening, and, through a series of experiments, show evidence suggesting that abstraction refining can offer advantages over progressive widening in settings with heterogeneous stochasticity.

References

[1] Z. F. Ahmad, R. Holte, and M. Bowling. Action selection for hammer shots in curling. In *IJCAI*, 2016.

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

[3] T. Anthony, R. Nishihara, P. Moritz, T. Salimans, and J. Schulman. Policy gradient search: Online planning and expert iteration without search trees, 2019.

[4] C. Browne, E. Powley, D. Whitehouse, S. Lucas, P. I. Cowling, S. Tavener, D. Perez, S. Samothrakis, S. Colton, and et al. A survey of monte carlo tree search methods. *IEEE TRANSACTIONS ON COMPUTATIONAL INTELLIGENCE AND AI*, 2012.

[5] P. S. Castro. Scalable methods for computing state similarity in deterministic markov decision processes. *CoRR*, abs/1911.09291, 2019. URL `http://arxiv.org/abs/1911.09291`.

[6] G. M. J. B. Chaslot, M. H. M. Winands, and H. J. van den Herik. Parallel monte-carlo tree search. In H. J. van den Herik, X. Xu, Z. Ma, and M. H. M. Winands, editors, *Computers and Games*, pages 60–71, Berlin, Heidelberg, 2008. Springer Berlin Heidelberg. ISBN 978-3-540-87608-3.

[7] A. Couëtoux, J.-B. Hoock, N. Sokolovska, O. Teytaud, and N. Bonnard. Continuous upper confidence trees. In *Proceedings of the 5th International Conference on Learning and Intelligent Optimization*, LION'05, page 433–445, Berlin, Heidelberg, 2011. Springer-Verlag. ISBN 9783642255656. doi: 10.1007/978-3-642-25566-3_32. URL `https://doi.org/10.1007/978-3-642-25566-3_32`.

[8] R. Coulom. Efficient selectivity and backup operators in monte-carlo tree search. In H. J. van den Herik, P. Ciancarini, and H. H. L. M. J. Donkers, editors, *Computers and Games*, pages 72–83, Berlin, Heidelberg, 2007. Springer Berlin Heidelberg. ISBN 978-3-540-75538-8.

[9] N. Ferns, P. Panangaden, and D. Precup. Bisimulation metrics for continuous markov decision processes. *SIAM J. Comput.*, 40(6):1662–1714, Dec. 2011. ISSN 0097-5397. doi: 10.1137/10080484X. URL `https://doi.org/10.1137/10080484X`.

[10] N. Ferns, P. Panangaden, and D. Precup. Metrics for finite markov decision processes. *CoRR*, abs/1207.4114, 2012. URL `http://arxiv.org/abs/1207.4114`.

[11] B. Kim, K. Lee, S. Lim, L. Kaelbling, and T. Lozano-Perez. Monte carlo tree search in continuous spaces using voronoi optimistic optimization with regret bounds. *Proceedings of the AAAI Conference on Artificial Intelligence*, 34(06):9916–9924, Apr. 2020. doi: 10.1609/aaai.v34i06.6546. URL `https://ojs.aaai.org/index.php/AAAI/article/view/6546`.

[12] L. Kocsis and C. Szepesvári. Bandit based monte-carlo planning. In *Proceedings of the 17th European Conference on Machine Learning*, ECML'06, pages 282–293, Berlin, Heidelberg, 2006. Springer-Verlag. ISBN 3-540-45375-X, 978-3-540-45375-8. doi: 10.1007/11871842_29. URL `http://dx.doi.org/10.1007/11871842_29`.

[13] G. Konidaris. On the necessity of abstraction. *Current Opinion in Behavioral Sciences*, 29:1–7, 2019. ISSN 2352-1546. doi: https://doi.org/10.1016/j.cobeha.2018.11.005. URL `https://www.sciencedirect.com/science/article/pii/S2352154618302080`. Artificial Intelligence.

[14] M. Lanctot, E. Lockhart, J.-B. Lespiau, V. Zambaldi, S. Upadhyay, J. Pérolat, S. Srinivasan, F. Timbers, K. Tuyls, S. Omidshafiei, D. Hennes, D. Morrill, P. Muller, T. Ewalds, R. Faulkner, J. Kramár, B. D. Vylder, B. Saeta, J. Bradbury, D. Ding, S. Borgeaud, M. Lai, J. Schrittwieser, T. Anthony, E. Hughes, I. Danihelka, and J. Ryan-Davis. Openspiel: A framework for reinforcement learning in games, 2019.

[15] J. Schrittwieser, I. Antonoglou, T. Hubert, K. Simonyan, L. Sifre, S. Schmitt, A. Guez, E. Lockhart, D. Hassabis, T. Graepel, T. P. Lillicrap, and D. Silver. Mastering atari, go, chess and shogi by planning with a learned model. *CoRR*, abs/1911.08265, 2019. URL `http://arxiv.org/abs/1911.08265`.

[16] D. Silver, T. Hubert, J. Schrittwieser, I. Antonoglou, M. Lai, A. Guez, M. Lanctot, et al. Mastering chess and shogi by self-play with a general reinforcement learning algorithm. *CoRR*, abs/1712.01815, 2017. URL `http://arxiv.org/abs/1712.01815`.

[17] R. S. Sutton and A. G. Barto. *Reinforcement Learning: An Introduction*, chapter 8.9. A Bradford Book, USA, 2018. ISBN 0262039249, 9780262039246.

[18] C. Xiao, J. Mei, and M. Müller. Memory-augmented monte carlo tree search. *Proceedings of the AAAI Conference on Artificial Intelligence*, 32(1), Apr. 2018. URL `https://ojs.aaai.org/index.php/AAAI/article/view/11531`.

[19] T. Yee, V. Lisy, and M. Bowling. Monte carlo tree search in continuous action spaces with execution uncertainty. In *Proceedings of the Twenty-Fifth International Joint Conference on Artificial Intelligence*, IJCAI'16, page 690–696. AAAI Press, 2016. ISBN 9781577357704.

[20] A. Zhang, R. McAllister, R. Calandra, Y. Gal, and S. Levine. Learning invariant representations for reinforcement learning without reconstruction, 2021.

Checklist

1. For all authors...
   (a) Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope? [Yes]
   (b) Did you describe the limitations of your work? [Yes] See e.g. On The Distance Function
   (c) Did you discuss any potential negative societal impacts of your work? [No] We do not believe there are any immediate potential negative societal impacts.
   (d) Have you read the ethics review guidelines and ensured that your paper conforms to them? [Yes]
2. If you are including theoretical results...
   (a) Did you state the full set of assumptions of all theoretical results? [Yes] See supplementary material
   (b) Did you include complete proofs of all theoretical results? [Yes] See supplementary material
3. If you ran experiments...
   (a) Did you include the code, data, and instructions needed to reproduce the main experimental results (either in the supplemental material or as a URL)? [No] We included each of our codebases as well as descriptions of our experiments in the supplementary material. We did not record the specific seeds or include pretrained models. The AlphaZero results also depend on the number of actors and the number and quality of devices used, which would be difficult to replicate.
   (b) Did you specify all the training details (e.g., data splits, hyperparameters, how they were chosen)? [No] We included a substantial amount of detail about the training process, as well as each of the codebases we used in the supplementary material. We included plots for a variety of hyperparameter settings of the baseline in the main paper and supplementary material.
   (c) Did you report error bars (e.g., with respect to the random seed after running experiments multiple times)? [Yes] See plots
   (d) Did you include the total amount of compute and the type of resources used (e.g., type of GPUs, internal cluster, or cloud provider)? [No] We did not keep track of the total amount of compute used. We used an internal cluster.
4. If you are using existing assets (e.g., code, data, models) or curating/releasing new assets...
   (a) If your work uses existing assets, did you cite the creators? [Yes] OpenSpiel
   (b) Did you mention the license of the assets? [No] Can be found on OpenSpiel Github.
   (c) Did you include any new assets either in the supplemental material or as a URL? [Yes] See supplementary material
   (d) Did you discuss whether and how consent was obtained from people whose data you're using/curating? [N/A]
   (e) Did you discuss whether the data you are using/curating contains personally identifiable information or offensive content? [N/A]

5. If you used crowdsourcing or conducted research with human subjects...
   (a) Did you include the full text of instructions given to participants and screenshots, if applicable? [N/A]
   (b) Did you describe any potential participant risks, with links to Institutional Review Board (IRB) approvals, if applicable? [N/A]
   (c) Did you include the estimated hourly wage paid to participants and the total amount spent on participant compensation? [N/A]