Monte Carlo Tree Search With Iteratively Refining State Abstractions

Samuel Sokota
Carnegie Mellon University
ssokota@andrew.cmu.edu

Caleb Ho
Independent Researcher
caleb.yh.ho@gmail.com

Zaheen Ahmad
University of Alberta
zfahmad@ualberta.ca

J. Zico Kolter
Carnegie Mellon University
zkolter@cs.cmu.edu

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, Hex, 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 advantages. To support this claim, we present a series of experimental examples in which abstraction refining outperforms progressive widening, given equal simulation budgets.

1 Introduction

In control problems, an agent makes sequential decisions toward the end of accumulating a large amount of reward [21]. 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 [6, 10, 15]. Algorithms powered by MCTS, like AlphaZero and MuZero, yield strong performance in games with complex value function landscapes, like Go, chess, Hex, and shogi [2, 20, 19]. 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, MCTS may be relegated to building shallow search trees. 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.

35th Conference on Neural Information Processing Systems (NeurIPS 2021).

One way to address this issue is by using progressive widening [9]. Progressive widening works by alternating between adding new children and selecting among existing children such that, asympototically, the search tree becomes both infinitely deep and infinitely wide. Progressive widening is also advantageous because it includes hyperparameters that control its propensity to add new children (as opposed to selecting among existing children). 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 performing well in both settings with important stochasticity and in settings with unimportant (or non-existent) stochasticity.

However, progressive widening's decision rule for expansion is the same for every transition. In settings with non-uniform stochasticity, we argue that this property is undesirable, as it 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. 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.

In addition to proposing the abstraction refining algorithm, our contributions are twofold. First, we present both a proof that, when used for policy evaluation, abstraction refining converges almost surely in finite MDPs. Second, we make an empirical case that, provided a good notion of state similarity, abstraction refining can outperform progressive widening in settings with stochasticity of varying importance, given an equal simulation budget. To make this case, we present a series of domains in which i) stochasticity is sometimes, but not always, relevant to performing well on the task and ii) the naive notion of distance between states is reflective of behavioral similarity (though we also include some settings in which we use a learned notion of distance). In these domains, we find that abstraction refining can outperform progressive widening.

2 Background

We consider a Markov decision process (MDP) setting $\langle \mathcal{S}, \mathcal{A}, \mathcal{T}, \mathcal{R}, \gamma \rangle$, 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 $\langle \mathcal{S}, \mathcal{T}, \mathcal{R}, \gamma \rangle$, with state space $\mathcal{S}$, transition function $\mathcal{T}$, reward function $\mathcal{R}$, and discount factor γ. 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. For the purposes of this work, 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 the number of times the action has been selected at that state $N(s, a)$ and the average of the value assessments of that action $Q(s, a)$. 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) *backpropagates* the evaluation from the new child to the root.

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 the UCT algorithm [15], or its variants, like PUCT [18]. Although it is less common, MCTS can also be used as a policy evaluation algorithm. In this case, the selection policy is set to the target policy.

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.

Evaluation Whenever MCTS adds a new state to the tree, it evaluates the state. 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.

Backpropagation 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 $Q(s_k, a_k)$ using a weighted average between the existing value $Q(s_k, a_k)$ and the leaf evaluation summed with the accumulated reward $r_{k+1} + \cdots + r_T$. Additionally each visit counter $N(s_k, a_k)$ gets incremented by one.

Selecting an Action To Execute After the search is over, 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 node.

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 most obvious 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 deep 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 [9] 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 $N(s, a)$ 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 $N(s, a)$ 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$[s, a] < k \cdot N(s, a)^{\alpha}$ 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.

Informally, the hyperparameter α can be thought of as a propensity to select among existing children, rather than adding new children. 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 time 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 flexibility. In domains in which stochasticity is 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, it cannot discriminate between stochasticity that matters and stochasticity can safely be ignored. This means that, if some stochasticity

[1] Note that we call progressive widening here is generally 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.

is important but some is unimportant, the best thing that progressive widening can do requires either placing too much emphasis on unimportant stochasticity, or too little on important stochasticity.

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 expansion rate purely based on visit-count, as progressive widening does, abstraction refining determines whether or not to expand using state similarity. At each selection 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: when stochasticity is unimportant, abstraction refining can cluster states together but when it is important, abstraction refining can cluster states separately. 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.

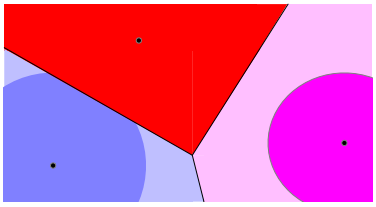

Figure 1: A visual depiction of an example low dimensional state space. There are three states already in the tree, represented by the black dots. The Voronoi diagram is shown using the colors blue, red and purple. The intersection of the Voronoi cells with each child's epsilon ball (shaded darker),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

A low dimensional example is shown in Figure 1.

3.1 Consistency

Used for policy evaluation in a finite MDP, abstraction refining is asymptotically consistent.

Proposition 1. *Let $\langle \mathcal{S}, \mathcal{R}, \mathcal{T} \rangle$ 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 $V_n(s_d) = \mathcal{R}(s_d) + \sum_{s \in \mathcal{S}_{d+1}} \frac{N_{s,n}}{n} V_{N_{s,n}}(s)$ where $N_{s,n}$ denotes the number of times that state s has been visited by the nth iteration. By Lemma 1, we have that $N_{s,n}/n$ converges to $\mathcal{T}(s \mid s_d)$ almost surely. Note that by Lemma 2, 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, and Lemma 3, we conclude that $V_n(s_d)$ converges to $\mathcal{R}(s_d) + \sum_{s \in \mathcal{S}_{d+1}} \mathcal{T}(s \mid s_d) v(s) = v(s_d)$ almost surely. □

Proofs for the lemmas below can be found in the appendix.

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 almost surely.*

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

Given Proposition 1, the extension to the discounted infinite horizon is relatively straightforward.

Corollary 1. *Abstraction refining also converges almost surely in a policy evaluation setting 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 over a finite horizon 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. □

It is important to note that convergence in the policy evaluation case does not guarantee convergence in combination with tree search algorithms [16]. In contrast to abstraction refining, there do exist variants of progressive widening for which such guarantees have been proven [5]. In other words, the theoretical justification for abstraction refining presented here is less strong than the existing theoretical justification for progressive widening.

3.2 Runtime Discussion

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 [8]. As a result, the most important analysis regards non-parallelizable runtime. We discuss abstraction refining's runtime for both serial and parallelized cases.

Abstraction refining performs two potentially expensive computations. The first of these is sampling states $s' \sim \mathcal{T}(s, a)$, which could be costly if the simulator is slow, or if sampling is being executed from a large learned model. Abstraction refining must perform this sampling at each state it visits during its selection process. In contrast, progressive widening only needs to perform sampling once each selection process. Thus, in settings where sampling states is a bottleneck, abstraction refining could add a significant amount of additional overhead on top of progressive widening. However, if sampling can be parallelized, this cost can be amortized by sampling a large number of next states in parallel the first time a state-action pair is visited.

The second potentially expensive computation is computing the nearest neighbor. This operation takes linear time serially (in the number of children) and logarithmic time when fully parallelized, and

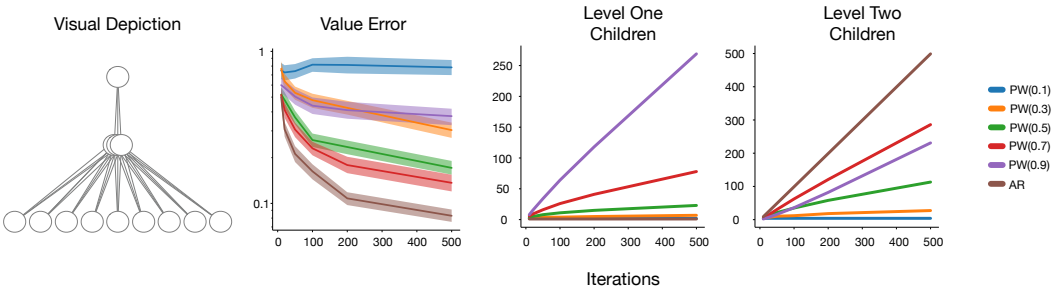

Figure 2: A comparison between progressive widening and abstraction refining on a toy MRP designed to illustrate the drawback of progressive widening's global hyperparameters. (far left) A visualization of the MRP; (middle left) value error, shown on a log scale, as a function of iterations; (middle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

must be performed at each step of the selection process. In contrast, progressive widening requires no such operation. Therefore, for systems in which a nearest neighbor computation would be the bottleneck for MCTS, abstraction refining would be substantially more expensive per iteration than progressive widening. On the other hand, for systems in which this would not be the case, perhaps such as those that use large networks to perform evaluations, abstraction refining may offer a cost per iteration competitive with that of progressive widening.

In our experiments, we compare abstraction refining and progressive widening given equal numbers of search iterations. It is worth bearing in mind, given the discussion above, that this is not always a fair comparison.

4 Experiments

We present four sets of experiments in the main body of the paper. The first two investigate the prediction (policy evaluation) problem; the second two investigate the control problem. A fifth set of experiments, as well as complementary experiments to those in the main body of the paper, are included in the appendix. In all plots, bands depict estimates of 95% confidence intervals computed using bootstrapping, except those in the pendulum plots in the appendix, which were computed using the central limit theorem.

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 behavior for a state selector is to add as few level one children as possible and as many level two children as possible.

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

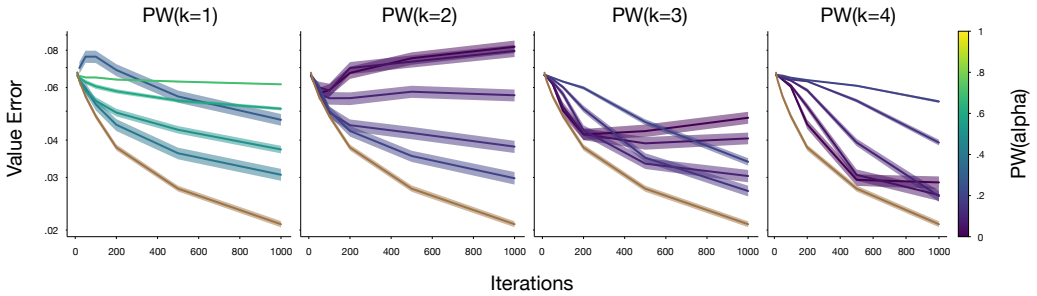

Figure 3: Abstraction refining (brown), with a fixed set of hyperparameters, compared against progressive widening with various hyperparameter values on policy evaluation in a continuous variant of blackjack. The value error (y-axis) is shown on a log scale.

$\epsilon_n = n^{-0.1}$. As a result, it can spend its entire quota on level two states (see brown in middle and far right plots; Figure 2). These distinctions manifest directly in prediction error, as shown in Figure 2, middle left plot, where abstraction refining outperformed progressive widening over each α value.

We also ran these experiments using other values of k for progressive widening and found qualitatively similar results. For these plots and a specification of the MRP, see the appendix.

Policy Evaluation in Blackjack The previous section illustrated that, in a policy evaluation setting designed adversarially for progressive widening, abstraction refining can offer superior per iteration performance. 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. We chose a policy described by *Wizard of Odds Consulting*. Details for the game, the policy, and plots with additional hyperparameters can be found in the appendix.

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. 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, but expand slowly width-wise, leading to repeated evaluations of states that do not accurately reflect the 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 the iteration horizon for which to optimize.

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. As a result, 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

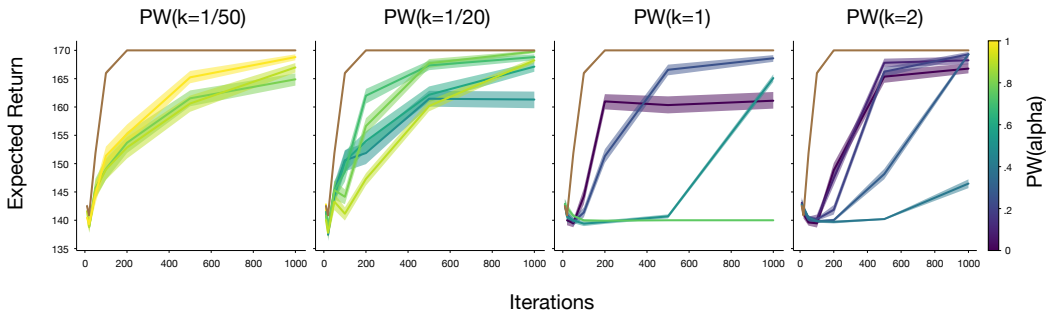

Figure 4: Abstraction refining (brown), with a fixed set of hyperparameters, compared against progressive widening with various hyperparameter values on control in a variant of the trap problem.

problem, which was first introduced in the original progressive widening paper [9] 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 specification of the problem, hyperparameter settings, and plots for additional hyperparameter values can be found in the appendix.

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.

In the results, we see that a single hyperparameter configuration of abstraction refining is able to outperform progressivew widening across a wide range of hyperparameters. One reason for abstraction refinement's strong performance here is likely that it is able to discriminate between having fallen into the trap and not having fallen into the trap. On the other hand, progressive widening has no mechanism for making this distinction, which may lead to it being overly confident about the safety of large jumps. In the appendix, we show additional results for abstraction refining with a distance function that does not allow for such a distinction. We find that, given this distance function, abstraction refining is unable to reproduce the level of performance shown in Figure 4.

Opponent Exploitation in Five by Five Go For our fourth 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 Black's policy is fixed, White can view the game as a stochastic MDP in which the transition function is dictated by Black'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 [17] AlphaZero and Go implementations for this experiment. Discussion of these design choices and the experimental setup can be found in the appendix.

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. It seems 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.

Control in Pendulum An additional set of experiments performed on a variant of pendulum in which the agent experiences actuator error are included in the appendix.

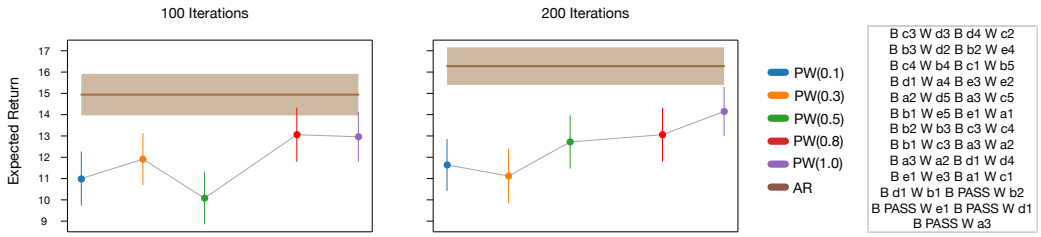

Figure 5: (left) Abstraction refining (brown), compared against progressive widening with $k = 1$ and various α values for opponent exploitation in five by five Go; (right) an example game in which the exploiter wins by the maximum margin (25).

5 Closing Discussion

On The Distance Function An important facet of abstraction refining that, heretofore, has gone largely uncommented upon, 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 our first three experiments, would perform poorly in many cases. And while our 5x5 Go and pendulum experiments (the latter of which are included in the appendix) 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 for constructing this distance metric. Perhaps the most natural notion of state similarity for abstraction refining is a bisimulation metric [12, 11]. 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 [24, 7], doing so in large MDPs with arbitrary transition distributions remains a challenging problem.

Related Work There are a number of works related to this one in that they 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 for action selection in a single-decision setting [1]. Yee et al. propose incorporating kernel regression into MCTS for continuous action spaces [23]. Most recently, Kim et al. show how Voronoi diagrams can be applied to extend MCTS to a continuous action space [14]. Our work can be viewed as following in the spirit of that of Kim et al., but focused on states (and evaluations) rather than actions (and control).

There has also been work leveraging state similarity for MCTS. Specifically, Xiao et al. propose using kernel regression to improve state evaluations for MCTS [22]. Experimentally, they show that using an intermediate layer of AlphaZero's policy-value network in 19x19 Go to compute similarity performs well in practice.

Outside of progressive widening [9], there also additional work on performing state abstractions during search [13]. Specifically, Hostetler et al. investigate two mechanisms for adapatively refining state abstractions. The first, called random refinement, randomly divides existing state abstractions. The second, called decision tree-based refinement, uses a decision tree over feature space of the successor states to determine state abstractions, and adds new splits to the tree for refinement.

Finally, there is also additional work motivated by the deficiencies of MCTS in settings with large branching factors arising from stochasticity [3, 4]. Anthony proposes performing additional policy gradient updates to the network to improve the policy during decision time, rather than maintaining a tabular tree of visited nodes. Results suggest that this approach, called policy gradient search, can achieve performance comparable or superior to that of MCTS in Hex.

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 almost surely when used for policy evaluation in finite MDPs and, through a series of experiments, show evidence suggesting that abstraction refining can offer advantages over progressive widening in settings with stochasticity of varying importance, given an equal number of simulations.

Acknowledgments and Disclosure of Funding

We thank Ryan D'Orazio, Alexander Robey and Yiding Jiang for providing helpful feedback for this work.

This work was supported by funding from the Bosch Center for Artificial Intelligence.

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] T. W. Anthony. *Expert iteration*. PhD thesis, UCL (University College London), 2021.

[5] D. Auger, A. Couëtoux, and O. Teytaud. Continuous upper confidence trees with polynomial exploration – consistency. In H. Blockeel, K. Kersting, S. Nijssen, and F. Železný, editors, *Machine Learning and Knowledge Discovery in Databases*, pages 194–209, Berlin, Heidelberg, 2013. Springer Berlin Heidelberg. ISBN 978-3-642-40988-2.

[6] 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.

[7] P. S. Castro. Scalable methods for computing state similarity in deterministic markov decision processes. In *AAAI*, 2020.

[8] 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.

[9] 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`.

[10] 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.

[11] 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`.

[12] N. Ferns, P. Panangaden, and D. Precup. Metrics for finite Markov decision processes. *AAAI*, 2012.

[13] J. Hostetler, A. Fern, and T. Dietterich. Sample-based tree search with fixed and adaptive state abstractions. *J. Artif. Int. Res.*, 60(1):717–777, Sept. 2017. ISSN 1076-9757.

[14] 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`.

[15] 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.

[16] L. Kocsis, C. Szepesvári, and J. Willemson. Improved monte-carlo search.

[17] 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.

[18] C. D. Rosin. Multi-armed bandits with episode context. *Ann. Math. Artif. Intell.*, 61(3):203–230, 2011. URL http://dblp.uni-trier.de/db/journals/amai/amai61.html#Rosin11.

[19] 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. *Nature*, 588 7839:604–609, 2020.

[20] D. Silver, T. Hubert, J. Schrittwieser, I. Antonoglou, M. Lai, A. Guez, M. Lanctot, L. Sifre, D. Kumaran, T. Graepel, T. P. Lillicrap, K. Simonyan, and D. Hassabis. Mastering chess and shogi by self-play with a general reinforcement learning algorithm. *ArXiv*, abs/1712.01815, 2017.

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

[22] 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.

[23] 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.

[24] A. Zhang, R. T. McAllister, R. Calandra, Y. Gal, and S. Levine. Learning invariant representations for reinforcement learning without reconstruction. In *International Conference on Learning Representations*, 2021. URL https://openreview.net/forum?id=-2FCwDKRREu.

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 discussions on convergence, runtime and distance function.
   (c) Did you discuss any potential negative societal impacts of your work? [No]
   (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 appendix
   (b) Did you include complete proofs of all theoretical results? [Yes] See appendix
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 the codebases used for the experiments in the main body of the paper in the supplementary material and included descriptions of our experiments in the appendix. 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, which would be difficult to replicate. We also did not include all of the code used to generate the experiments in the appendix.
   (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 for the experiments in the main body of the paper in the supplementary material. We included plots for a variety of hyperparameter settings in the main paper and in the appendix.
   (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? [N/A]
   (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]

A Appendix

A.1 Proofs

First, we show that, after a finite amount of time, abstraction refining stops performing abstractions.

Lemma 0. Say we are running abstraction refining from state s_d. Let T_s be the first iteration after which s, a next state with non-zero transition probability, has been added to the tree and has ϵ-ball of radius less than δ, where δ is the minimum over pairwise distances among next states. Then T_s is finite almost surely.

Proof. Let m be the smallest index such that $\epsilon_m < \delta$. Consider that after a state has been selected m or more times, it can no longer abstract other states because its ϵ-ball cannot contain them. Therefore, s cannot be abstracted more than $m(|\mathcal{S}| - 1)$ times. Thus, if it is sampled at least $m|\mathcal{S}|$ times, it must both have been added to the tree and been selected at least m times (and therefore have ϵ-ball of radius smaller than δ). So it suffices to show that s is sampled more than $m|\mathcal{S}|$ times almost surely.

Let $E_{s,t}$ be the event in which state s is sampled at iteration t. Then $\sum_{t=1}^{\infty} p(E_{s,t}) = \sum_{t=1}^{\infty} \mathcal{T}(s \mid s_d) = \infty$ because $\mathcal{T}(s \mid s_d) > 0$. Then, by the second Borel-Cantelli lemma, s must be sampled an infinite number of times almost surely. □

From the lemma above, we now show that abstraction refining's empirical visitation distribution converges to the underlying transition probabilities.

Lemma 1. Again assume we are running abstraction refining from state s_d and that s is a successor state. Then the sequence $N_{s,n}/n$ converges almost surely to $\mathcal{T}(s \mid s_d)$.

Proof. Let $I_{s,t}$ be the event that s is selected on iteration t. Then observe

$$\begin{aligned} N_{s,n}/n &= \sum_{t=1}^{n} I_{s,t}/n \\ &= \sum_{t=1}^{T_s} I_{s,t}/n + \sum_{t=T_s}^{n} I_{s,t}/n \\ &= \sum_{t=1}^{T_s} I_{s,t}/n + \frac{n - T_s}{n} \sum_{t=T_s}^{n} I_{s,t}/(n - T_s). \end{aligned}$$

As $n \to \infty$ the left term must go to zero, almost surely, because T_s is finite almost surely, from Lemma 0. The right factor of the right term must converge to $\mathcal{T}(s \mid s_d)$ be the strong law of large numbers. The left factor of the right term must converge to one, as T_s is finite almost surely. Thus $N_{s,n}/n \to \mathcal{T}(s \mid s_d)$ almost surely. □

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

Proof. This follows from the previous lemma and the fact that n diverges. □

Lastly, we show that re-indexing a sequence doesn't break almost sure convergence, so long as the re-indexing is monotonically increasing and onto. **Lemma 3.** Let X_n be a sequence converging to x almost surely. Let $f \colon \mathbb{N} \to \mathbb{N}$ be a monotonically increasing, onto function. Then $Y_n = X_{f(n)}$ converges to x almost surely.

Proof. Recall the following definition of almost sure convergence

$$\lim_{n \to \infty} \mathcal{P}(\{\omega : \sup_{k \geq n} |X_k(\omega) - x| > \epsilon)\}) = 0.$$

By virtue of almost sure convergence holding for X_n, there exists some $\mathcal{N}$ such that for all $n > \mathcal{N}(\epsilon_1, \epsilon_2)$,

$$\mathcal{P}(\{\omega : \sup_{k \geq n} |X_k(\omega) - x| > \epsilon_1)\}) < \epsilon_2.$$

Define $g(n) = \min f^{-1}(n)$. We claim that $g \circ \mathcal{N}$ satisfies the desired property for Y_n. It suffices to show that the sets over which the supremums range are the same. Observe

$$\begin{aligned}\{f(k) : k \geq g \circ \mathcal{N}(\epsilon_1, \epsilon_2)\} &= \{f(k) : f(k) > f \circ g \circ \mathcal{N}(\epsilon_1, \epsilon_2)\} \\ &= \{k : k > f \circ g \circ \mathcal{N}(\epsilon_1, \epsilon_2)\} \\ &= \{\ell : \ell > \mathcal{N}(\epsilon_1, \epsilon_2)\},\end{aligned}$$

where the first equality follows from the monotonicity of f, the second follows from that f is onto, and the last from the fact that $f \circ g^{-1}$ is the identity. □

B Experiments

B.1 Illustrating Progressive Widening's Weakness

B.1.1 Experimental Setup

We performed the experiment over a randomly generated distribution of MRPs of the same structure. In each MRP the state space was 30 dimensional, the level one states were generated using a random normal distribution with mean 0 and variance 0.01. Given a level one state, the level two state was generated by adding the level one state to a sample from a Gaussian mixture model. The mixture model had 10 components. The categorical mixture distribution was sampled from a dirichlet distribution with each α parameter set to one. The mean of each mixture was generated from a multivariate normal distribution with mean zero and identity variance. The covariance matrices were generated by first generating matrices of the appropriate dimensions distributed componentwise with mean zero and variance 0.1 and then multiplying these matrices with their own transposes. The agent's value function for level one states was set to an average of the values of five randomly chosen (but fixed) level two states. The agent's value function for level two states, and also the true value function, was set to a randomly initialized network with an input layer, relu activations and an output layer. For abstraction refining, we used $\epsilon = n^{-0.1}$ and Euclidean distance.

B.1.2 Additional Plots

Testing Other Value Functions Figure 6 suggests that using different classes of value functions do not appear to have an effect on the qualitative takeaway of the experiment.

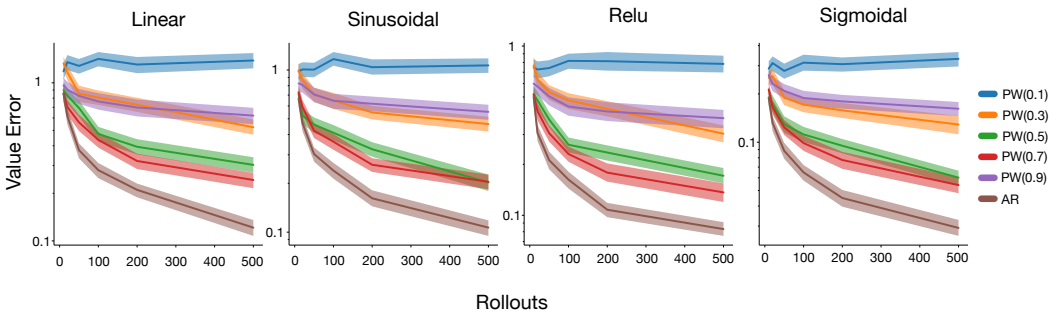

Figure 6: The same toy MRP experiment repeated with different value function classes.

Testing Number of Children For Other Progressive Widening Hyperparameters Figures 7 and 8 show the number of level one and level two children for different hyperparameter values for progressive widening. The ones that appear to trade-off best between selecting few level one children and many level two children at 500th iteration are $k = 10, \alpha = 0.3$ and $k = 20, \alpha = 0.1$.

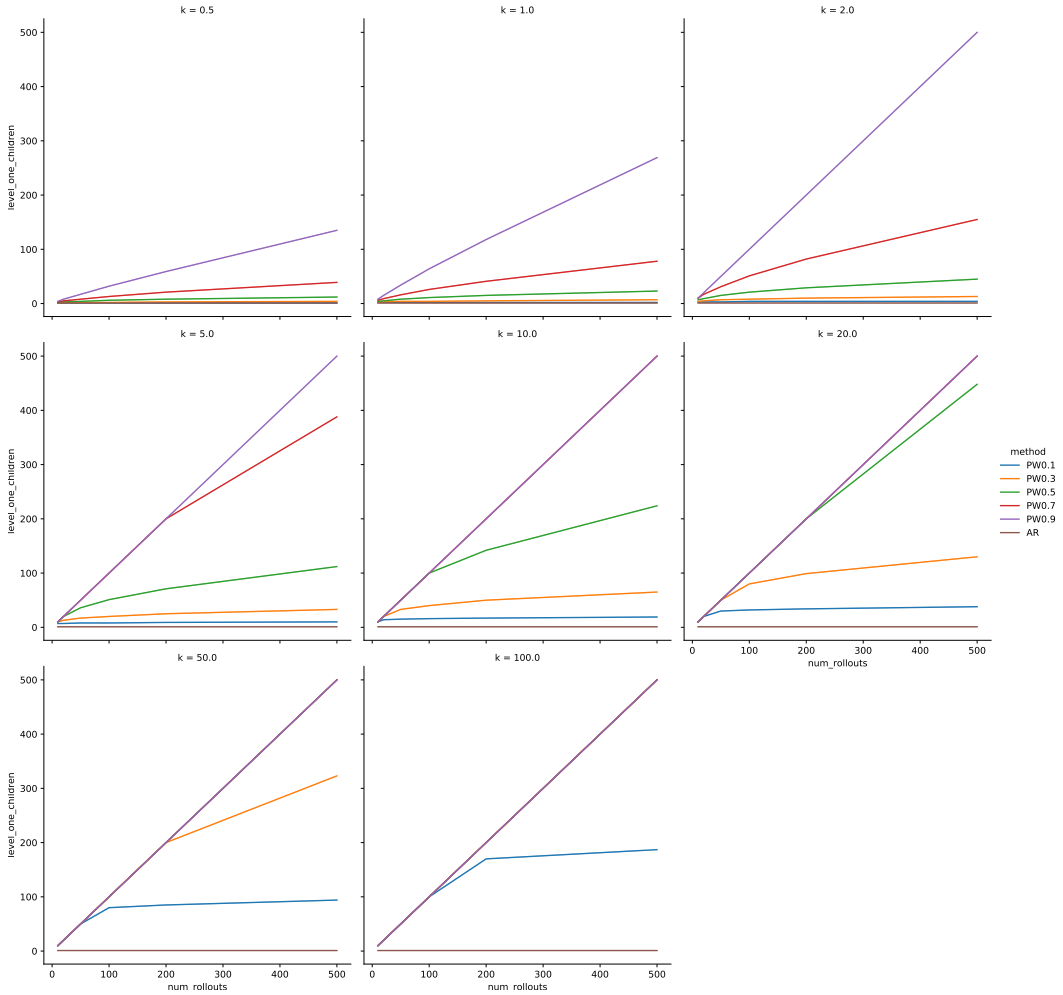

Figure 7: The number of level one children for different progressive widening hyperparameter values.

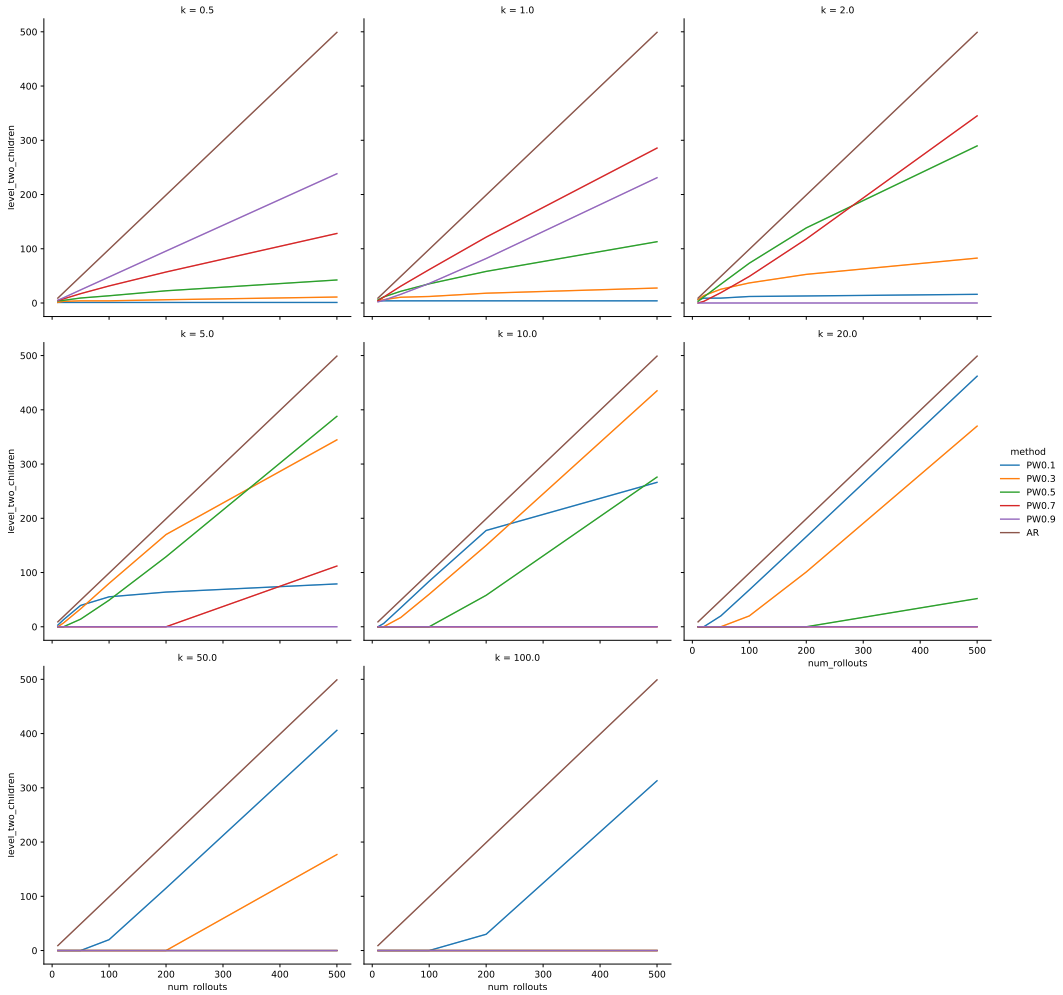

Figure 8: The number of level two children for different progressive widening hyperparameter values.

Testing Value Error For Other Progressive Widening Hyperparameters Figure 2 shows the value error for each of these hyperparameter settings. The narrative appears similar to the phenomenon observed in the blackjack experiments—for $k > 1$ small values of α can perform competitively with abstraction refining over a particular horizon length (e.g. $\alpha = 0.1, k = \{5, 10, 20\}$), but perform poorly thereafter. This phenomenon can be observed qualitatively by examining the number of children added (shown in Figures 8 and 7). The number of level two children for small α values increases quickly at first, but eventually reaches an inflection and thereafter increases relatively slowly. This inflection point arises as a result of the inflection point in the function $\min(n, kn^{\alpha})$, which governs the number of children added as a function of the number of visits for progressive widening with $k > 1$.

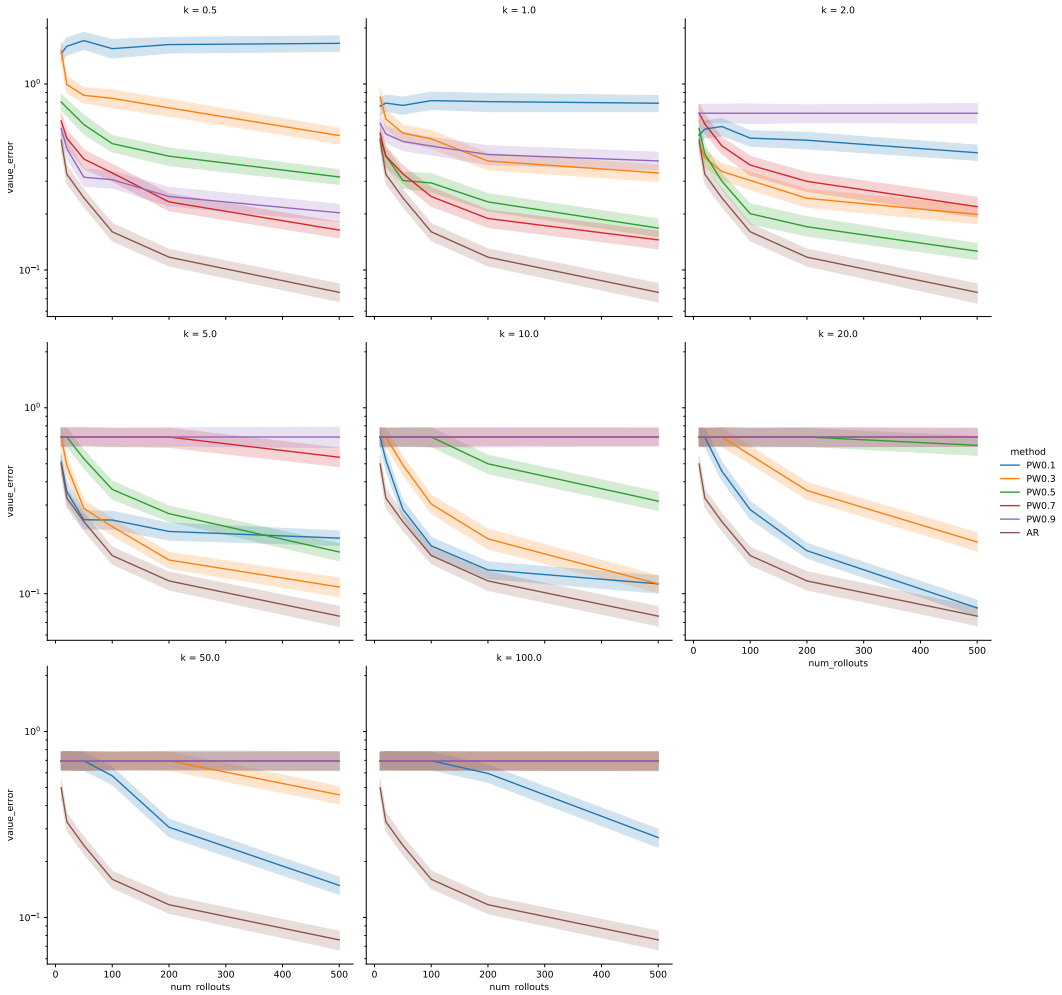

Figure 9: The value error for different progressive widening hyperparameter values.

B.2 Policy Evaluation in Blackjack

B.2.1 Experimental Setup

In our continuous variant of blackjack, there are two actions: hit and stand. When a player hits, it is dealt a card from the distribution Uniform$([1, 11])$. This distribution does not change as a function of previously dealt cards. At the beginning of the game, the dealer is dealt one card, which is observable to the player. The player then decides whether to hit or stand. If the player hits and the sum of its newly dealt card and its previously dealt cards do not exceed 21, the player goes again. If the sum exceeds 21, the player loses. If the player stands, it is the dealer's turn. The dealer's policy is fixed—if its sum is less than 17, it hits; if its sum exceeds 17 it stands. If the player stands and the dealer exceeds 21, the player wins. If neither the player nor the dealer busts, the larger total wins the game. The player receives a reward of $+1$ for winning the game and -1 for losing the game.

We performed policy evaluation on a policy taken from *Wizard of Odds Consulting* (adapted slightly to accommodate non-integer card values). The policy is described below:

- If the dealer's card is less than 3.5:
  - If the player's total is less than 12.5:
    - Hit
  - Else:
    - Stand

- If the dealer's card is greater than 3.5 but less than 6.5:
  - If the player's total is less than 11.5:
    - Hit
  - Else:
    - Stand
- If the dealer's card is greater than 6.5:
  - If the player's total is less than 16.5:
    - Hit
  - Else:
    - Stand

To acquire a(n approximate) ground-truth value for the policy, we averaged over 10 million rollouts. For the evaluation function, we used the function $* \mapsto 0$ (i.e., each state was given an evaluation of zero). We used Euclidean distance for abstraction refining, with a state representation $(\text{dealer's total}, \text{player's total}) \in \mathbb{R}^2$.

B.2.2 Additional Plots

As shown in Figure 10, $k = 1/2$ does not appear to improve progressive widening's performance.

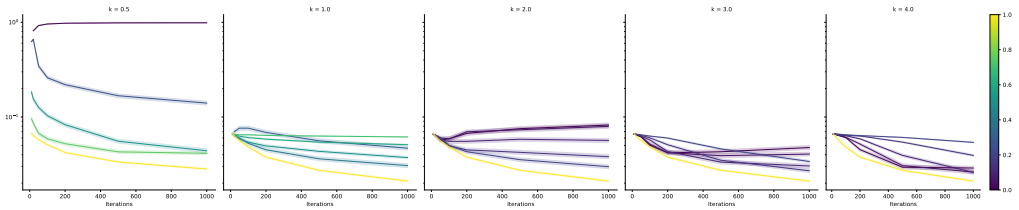

Figure 10: The blackjack plot from the main text, with an additional subplot (far left) for $k = 1/2$ for progressive widening. Abstraction refining is shown in yellow in each plot.

We also include results abstraction refining with different hyperparameter configurations in Figure 11. In the plot, (k, α) corresponds to $\epsilon_n = n^{-\alpha}$.

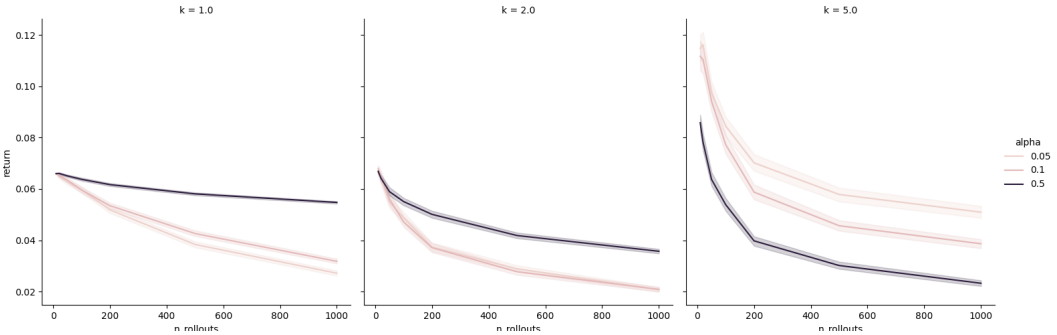

Figure 11: Performance over a range of hyperparameters for abstraction refining in continuous blackjack.

B.3 Control in Trap

B.3.1 Experimental Setup

In the trap environment, an agent attempts to navigate from a platform 70 units tall to a platform that is 100 units tall. The platforms are separated by a gap, where the ground is 0 units tall. The agent begins 1 unit away from the edge of the 70-unit tall platform. The gap is 0.7 units wide. The

agent's actions allow it to leap forward. It can attempt to leap any of $\{0, 0.25, 0.5, 0.75, 1.0\}$ units forward; however, its leaping abilities are imprecise, so the amount that it actually leaps differs from the amount that it attempted to leap by a sample from the distribution Uniform$(-0.01, +0.01)$. There agent is allowed two actions. After each action, the agent is given a reward equal to the height of its platform. The optimal policy is for the agent to leap 0.75 on the first time step (moving it close to the edge) and 1.0 on the second time step (leaping to the taller platform). If the agent leaps 1.0 on the first time step, it risks falling into the trap; if it leaps 0.5 or less on the first time step, it is unable to reach the taller platform. In our experiments, with used UCT as our action selector (with $c = 100$) and random rollouts as our evaluator. We used Euclidean distance for abstraction refining over Cartesian coordinate state representation $(x, y) \in \mathbb{R}^2$.

B.3.2 Additional Plots

Figure 12 shows the performance of progressive widening over a larger set of k values. Each hyperparameter setting for progressive widening appears to require at least 1000 iterations to perform optimally.

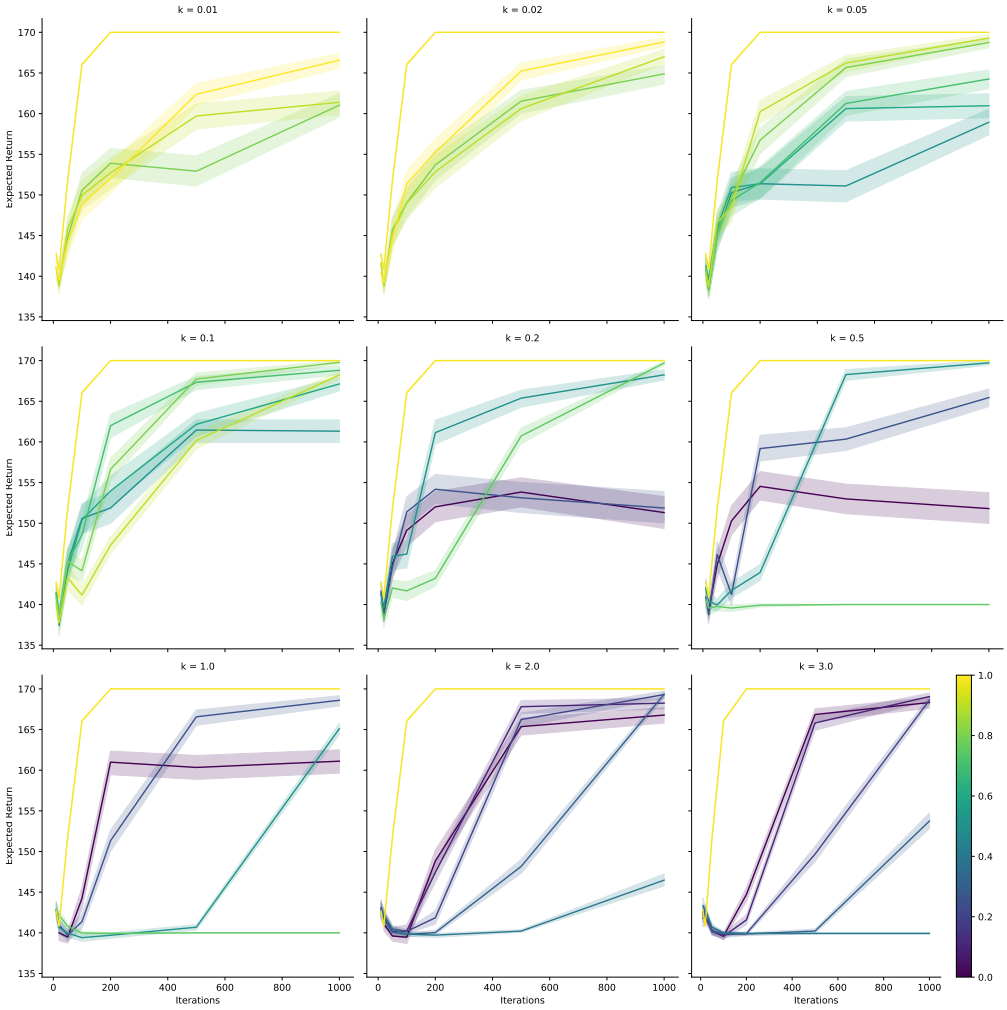

Figure 12: The traps plot from the main text, with additional subplots (far left) for other k values for progressive widening. Abstraction refining is shown in yellow in each plot.

Figure 13 shows the performance of abstraction refining for various hyperparameter configurations. In the plot, (k, α) corresponds to $\epsilon_n = kn^{-\alpha}$.

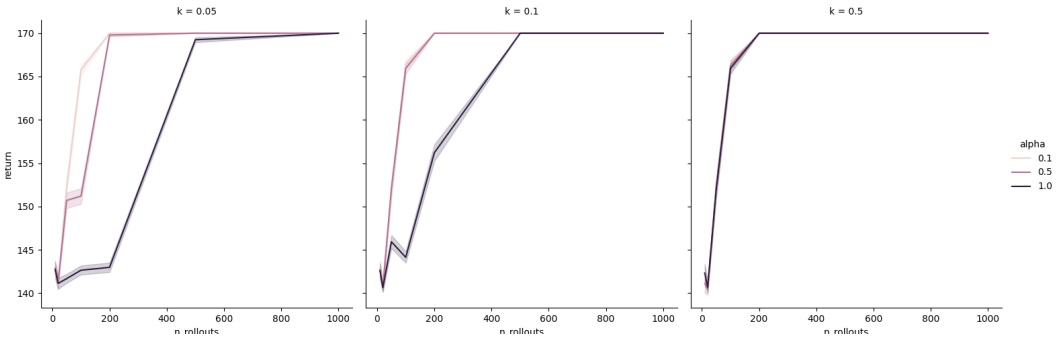

Figure 13: Performance over a range of hyperparameters for abstraction refining in trap.

Figure 14 shows results for abstraction with a distance metric that only observes horizontal distance. The hyperparameters used in the main body of the paper perform poorly. Other hyperparameters avoid failure, but are unable to match the performance of abstraction refining with the distance metric metric used in the main body of the paper.

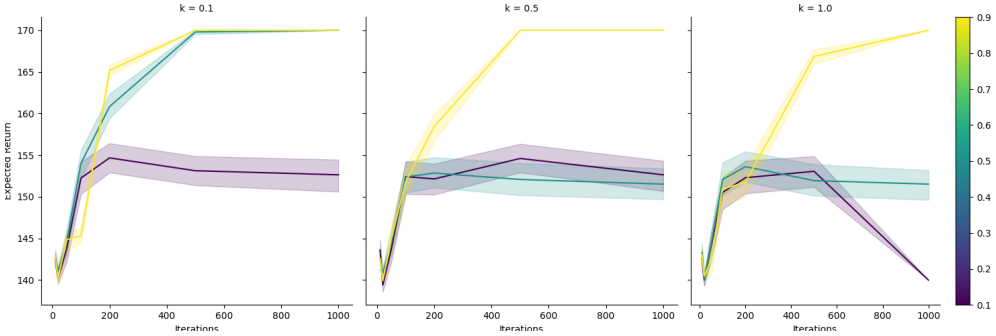

Figure 14: Results for abstraction refining with a bad distance metric.

In Figures 15 and 16, we show results for abstraction refining and progressive widening for trap stratified by the outcome of the first large jump taken by the agent. For progressive widening we observe that the agent performs worse if the first big jump sampled does not land in the trap. This makes sense because it would lead the agent to believe that taking big jumps is safe and because it reuses the first sample for multiple iterations before taking an additional sample. In contrast, abstraction refining samples at every search iteration and the first big jump in which the agent falls into the trap is guaranteed to be added to the tree because the distance between the plateau and the trap is sufficiently large.

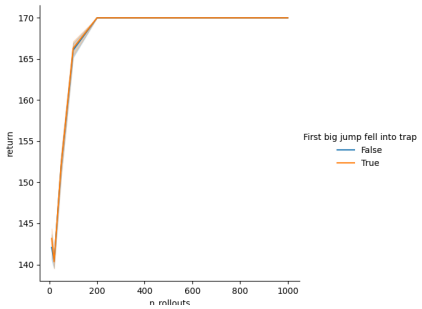

Figure 15: Performance of abstraction refining as function of first big jump outcome.

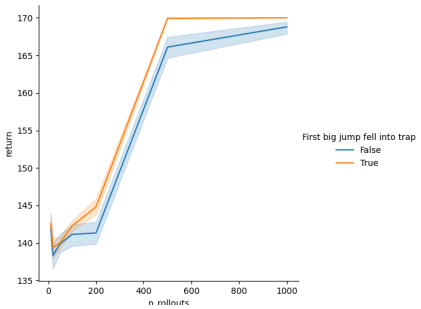

Figure 16: Performance of progressive widening as function of first big jump outcome.

B.4 Opponent Exploitation in Five by Five Go

B.4.1 Experimental Setup

We used OpenSpiel's [17] implementation of AlphaZero and of 5x5 Go. We adjusted the implementation of Go to provide the margin of victory as the reward, instead of the win-loss value. To create our networks, we trained AlphaZero in self play for about $200,000$ games. Some training plots are shown in Figure 17. The top left subplot shows the training loss as a function of the number of training steps. As expected, the loss generally decreases as training goes on. The top right subplot shows the performance of AlphaZero against MCTS (with endgame solving and random rollouts) for various numbers of simulation budgets, where AlphaZero has equal chances of playing as Black and White. The bottom right subplot shows the percentage of outcomes each player wins over the course of training. Around the steps in the late 30s, Black discovered a strong strategy that causes it to win (have margin of victory greater than zero) in all of the evaluation games. However, White appears to learn to counter this strategy later in training.

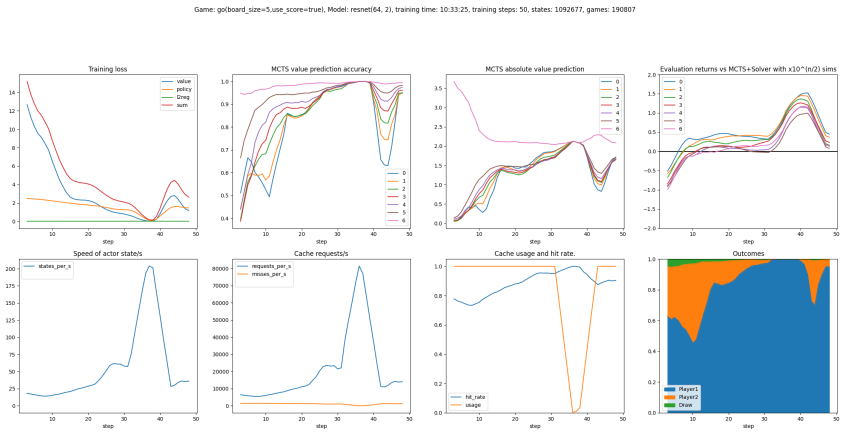

Figure 17: Training plots for AlphaZero in self-play.

We performed training on a single GPU and five CPUs using ten asynchronous actors. For our network, we used a resnet with two resblocks, meaning that the policy network had seven layers and the value network had eight layers, five of which were shared among the two.

For our experiments, we had Black act according to the AlphaZero policy network from the the training run above. We had White perform AlphaZero search. However, rather than perform search as AlphaZero typically does, we ran MCTS assuming that Black's policy was a known and fixed transition function, thereby allowing us to apply progressive widening and abstraction refining. Our preliminary results suggested that all of the stochasticity was important—i.e., that vanilla MCTS (an edge case of both progressive widening and abstraction refining) performed best. Thus, to make the search problem more interesting, we added a small amount of noise (sampled from Uniform$(-0.1, +0.1)$ independently for each component) to the latent state transitions, thereby requiring abstraction refining and progressive widening to deviate from vanilla MCTS in order to perform deep searches. For abstraction refining, we used $\epsilon_n = 200n^{-0.1}$ and Manhattan distance, which we chose because of the high dimensionality (1600) of the latent state representation.

B.5 Pendulum Experiments

For our pendulum experiments experiment, we used the OpenAI gym pendulum environment, with discretized actions (-2, -1, 0, 1, 2) and stochasticity in the form of actuator noise. We used two types of actuator noise comes. First, the agent has a "fat finger" in the sense that, with probability a small probability, a random action is selected instead of the intended action. We used 0.1 for this probability in our experiments. Second, uniform mean zero noise is added to the continuous representation of the selected action. We used $1e-3$ for the magnitude of this noise. To help keep the variance of the outcomes manageable, we imposed a maximum horizon length of 10.

We performed three variants of this experiment. In this first variant, we trained a single DQN network to maximize the average reward of our modified pendulum environment, without imposing a finite horizon. Subsequently, we ran a modified version of AlphaZero on top of the DQN network, where we used a softmax distribution over the Q-values for the policy and the maximum Q-value as the value. We compare the performance of this modified version of AlphaZero for different α settings of progressive widening ($k = 1$) and for $\epsilon_n = 0.1n^{-0.1}$ for abstraction refining. We used the Q-values for the representation for abstraction refining. The results of this experiments are shown in Figure 18.

The results are shown in the Figure 18. The x-axis shows the number of search iterations (values are measured at 50, 100, and 200). The y-axis shows the average reward. Evaluating algorithm performance was quite costly in the sense that we required a very large number of games to achieve a high level of certainty. (The lines below are each averages over 100,000 games.) The bands depict estimates of 95% confidence intervals computed using the standard error of the return outcomes.

In these results, abstraction refining appears to show the strongest performance. Among the hyper-parameters settings for progressive widening, performance appears to monotonically increase with decreasing values of α. This may be because the value function for the infinite horizon version of the game is a poor proxy for the value function for the finite horizon for the game.

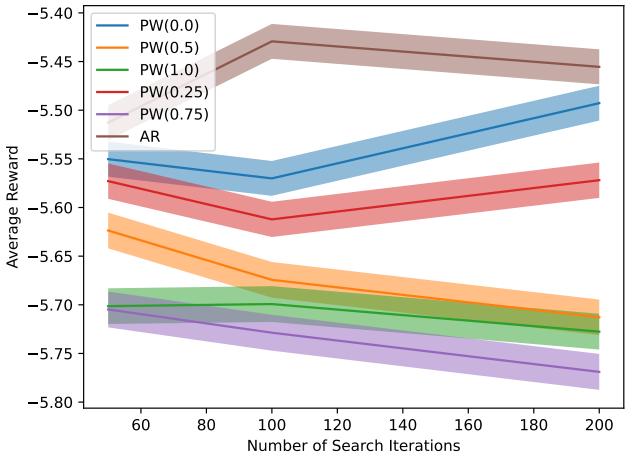

Figure 18: Pendulum results using DQN trained for infinite horizon.

For the second version of the pendulum experiment, we repeated the experiment above, but used a version of DQN trained specifically to maximize the return for the finite horizon variant of the game. Note that this involved modifying the architecture so that the agent would be aware of the time step.

We show the results in Figure 19. The lines depicted in the figure are again averages over 100,000 games. The bands are again estimates of 95% confidence intervals computed using the standard error. The results show a large improvement in performance for all algorithms and hyperparameters, suggesting, perhaps unsurprisingly, that pretraining a network for a finite horizon offers much stronger performance. The difference between the performance of the algorithms is much smaller than it was in the previous experiment. It appears to be the case that abstraction refining outperforms progressive widening. It also appears to be the case that an intermediate value of α, such as 0.75, performs best for progressive widening. However, there is a substantial amount of uncertainty with respect to these observations.

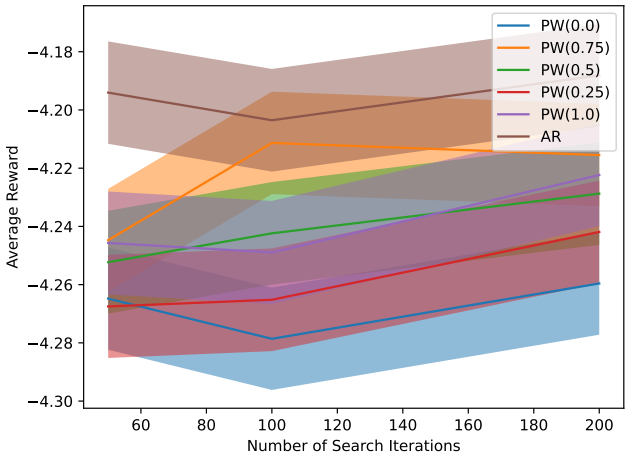

Figure 19: Pendumulum results using DQN trained for finite horizon.

For our last version of the experiment, we investigated the performance of each algorithm using AlphaZero at both training time and testing time. For each hyperparameter setting, and for each number of search iterations, we trained 100 distinct AlphaZero networks. We evaluated each of these networks over 1000 games. The results are shown in Figure 20.

The performance of the AlphaZero-at-training-time configurations fall in between those of the two previous experiments, outperforming the DQN pretrained for infinite horizon but falling short of DQN trained for finite horizon. This may not necessarily reflect the maximal performance of AlphaZero, but rather the fact that we were unable to perform rigorous tuning due to the number of runs required to generate a reliable signal of performance. Among the takeaways, perhaps the most striking is that vanilla AlphaZero (i.e., progressive widening $\alpha = 1.0$) is significantly outperformed by all other hyperparameter configurations. This may reflect the fact that using Monte Carlo returns (as AlphaZero does) is not an ineffective means of learning an accurate value function in a setting with large return variance. Another takeaway is that, across all three search budgets, the best hyperparameter setting for progressive widening outperforms abstraction refining. One possible explanation for this phenomenon is that, early on in training, when the representation is not reflective of state similarity, performing abstraction has a negative effect on training progress. However, confirming this explanation would require additional experiments beyond those presented here.

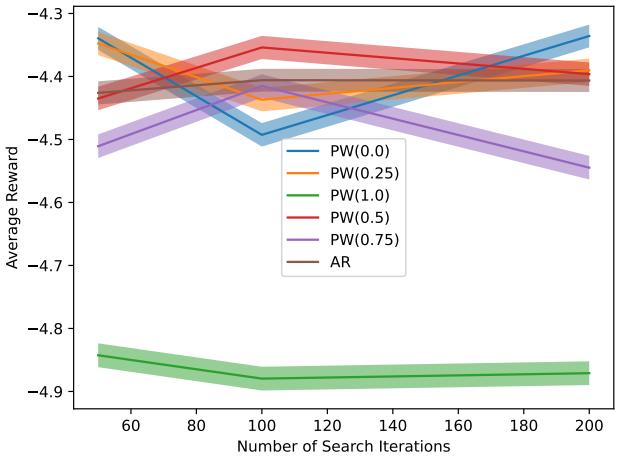

Figure 20: Pendulum results using AlphaZero during training.