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.

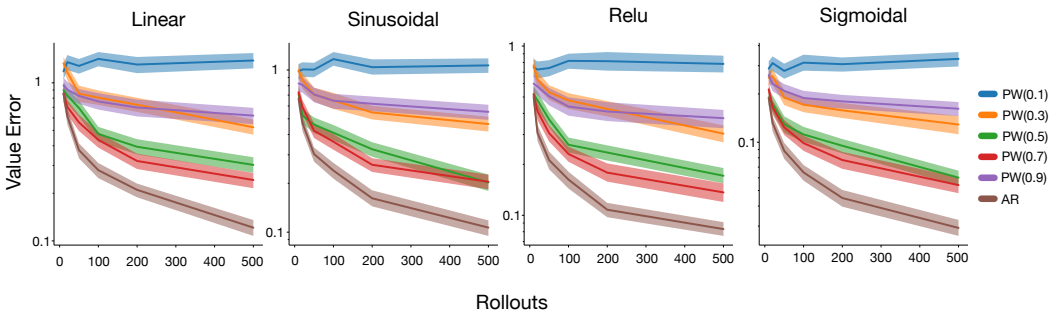

Figure 6: The same toy MRP experiment repeated with different value function classes.

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 $k = 20, \alpha = 0.1$.

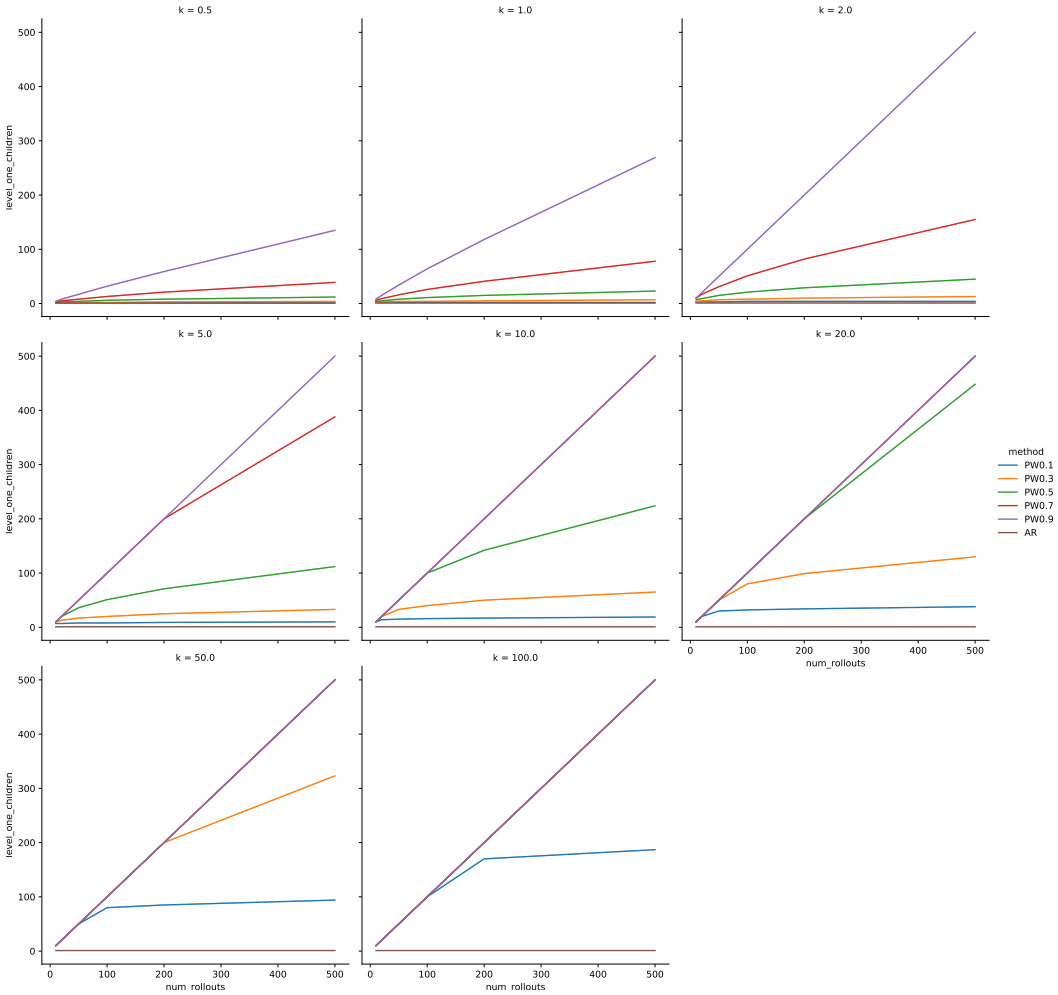

Figure 7: The number of level one children for different progressive widening hyperparameter values.

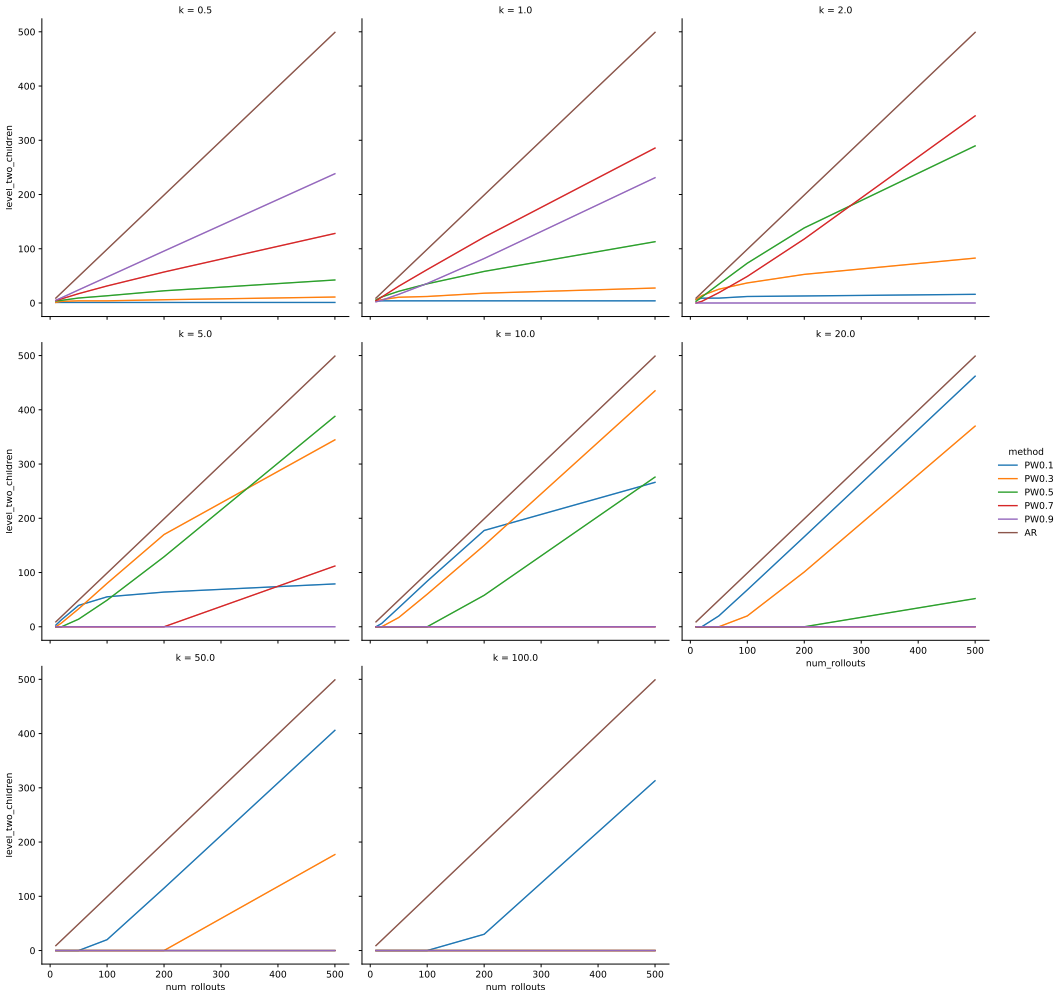

Figure 8: The number of level two children for different progressive widening hyperparameter values.

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 $k > 1$.

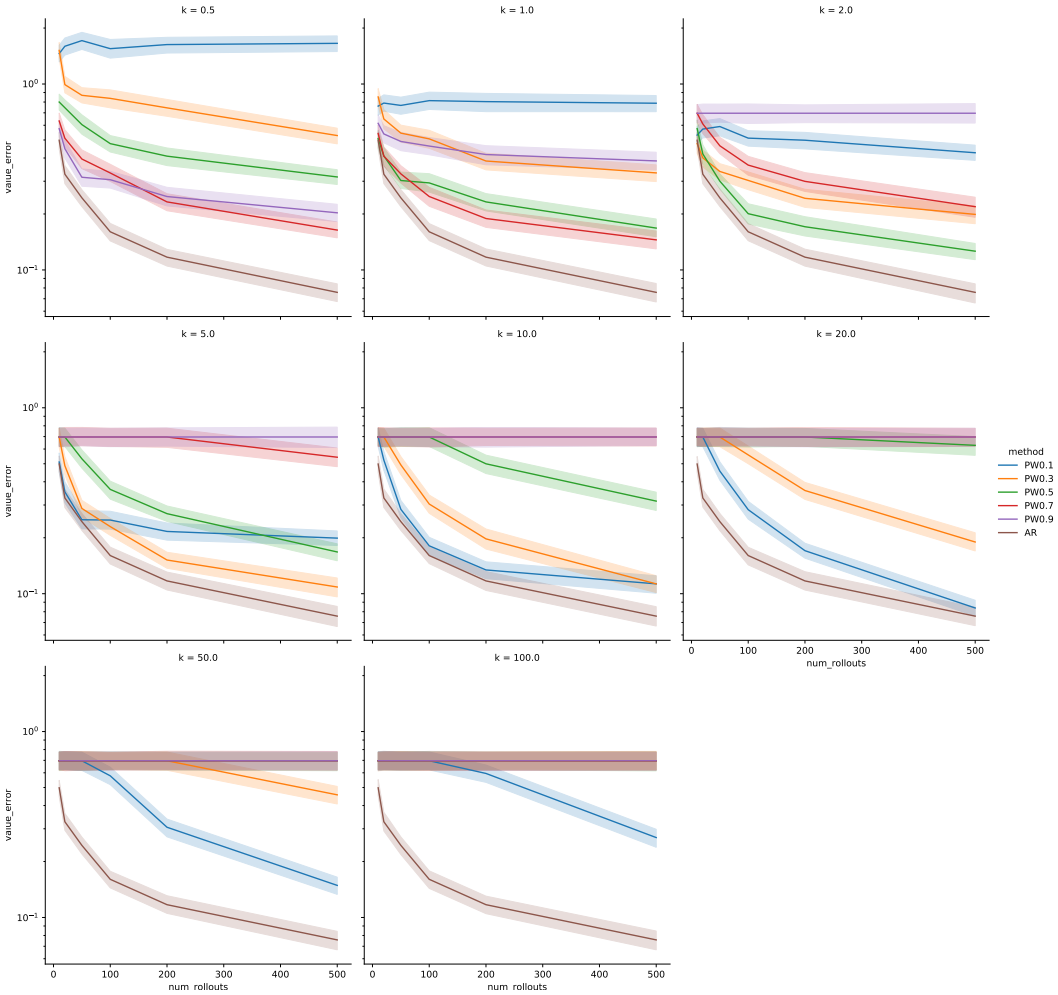

Figure 9: The value error for different progressive widening hyperparameter values.

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.

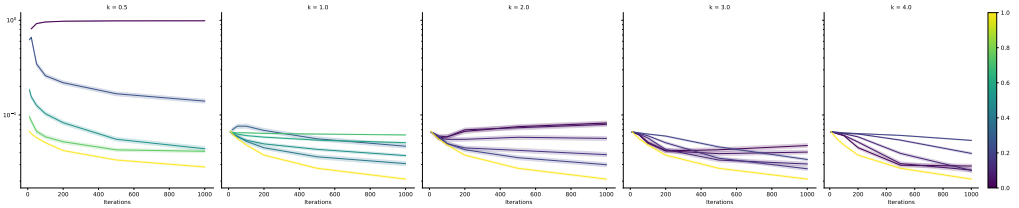

Figure 10: The blackjack plot from the main text, with an additional subplot (far left) for $k = 1/2$ for progressive widening. Abstraction refining is shown in yellow in each plot.

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

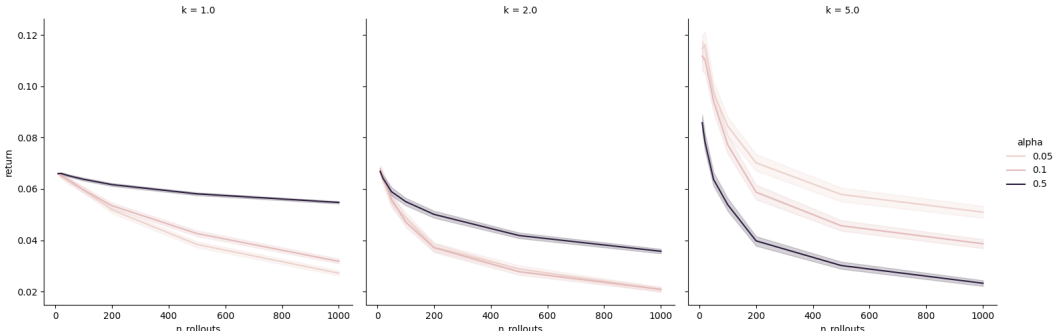

Figure 11: Performance over a range of hyperparameters for abstraction refining in continuous blackjack.

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.

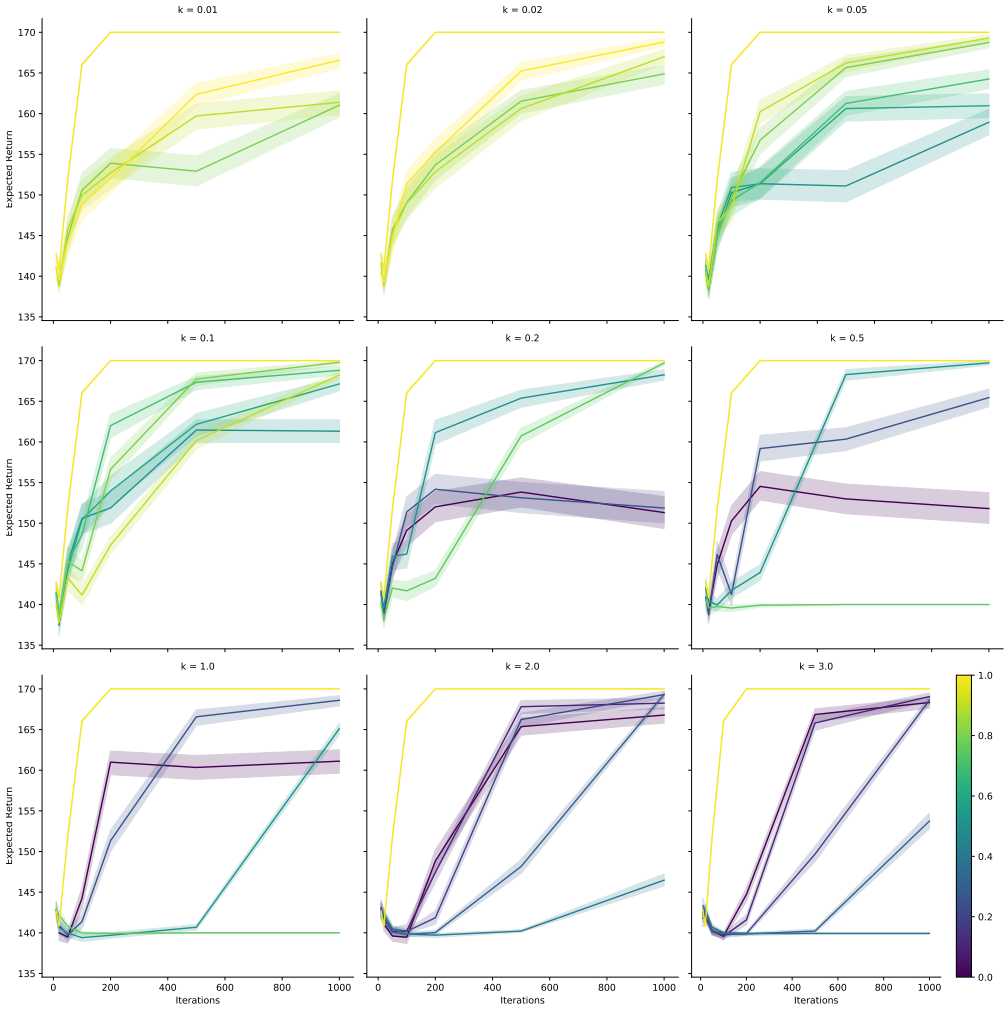

Figure 12: The traps plot from the main text, with additional subplots (far left) for other k values for progressive widening. Abstraction refining is shown in yellow in each plot.

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

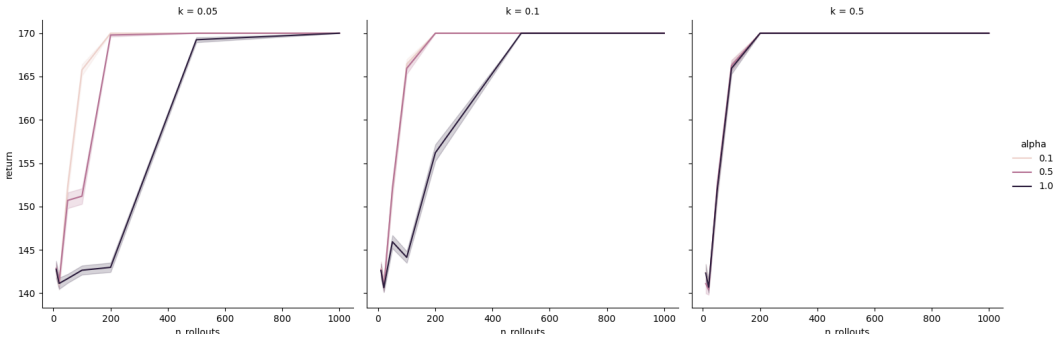

Figure 13: Performance over a range of hyperparameters for abstraction refining in trap.

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.

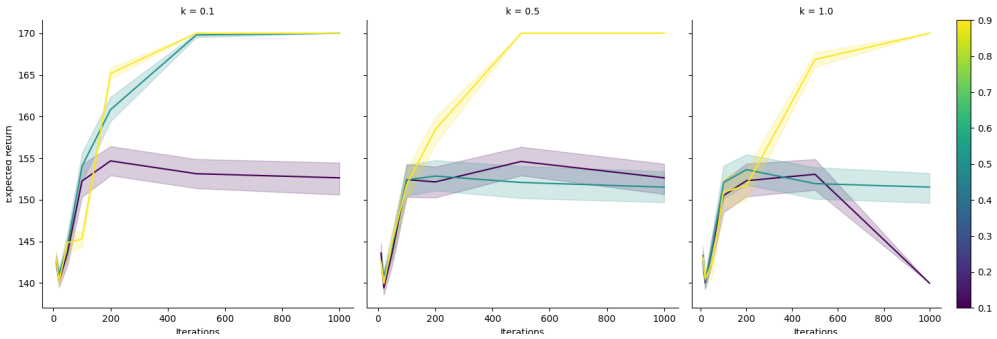

Figure 14: Results for abstraction refining with a bad distance metric.

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.

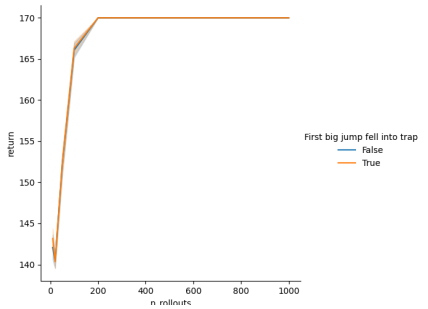

Figure 15: Performance of abstraction refining as function of first big jump outcome.

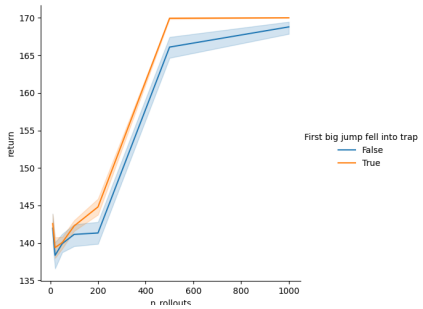

Figure 16: Performance of progressive widening as function of first big jump outcome.

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.

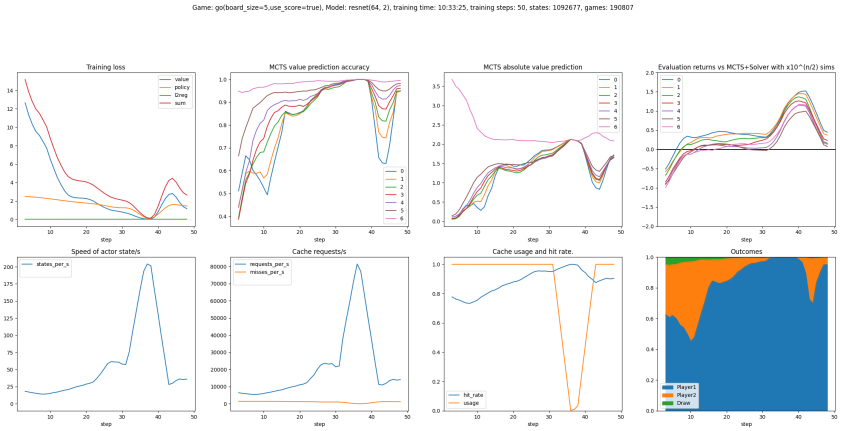

Figure 17: Training plots for AlphaZero in self-play.

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.

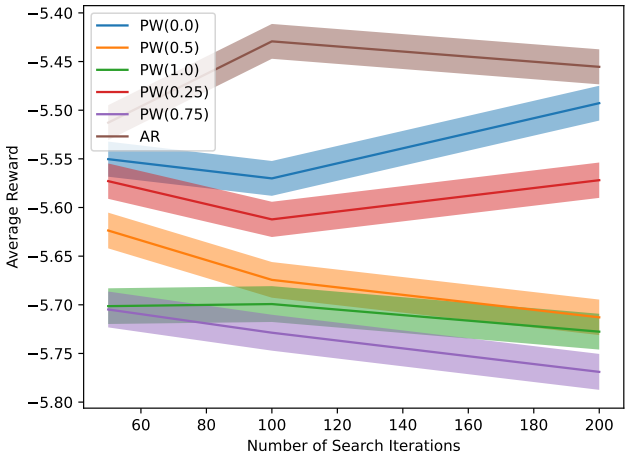

Figure 18: Pendulum results using DQN trained for infinite horizon.

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.

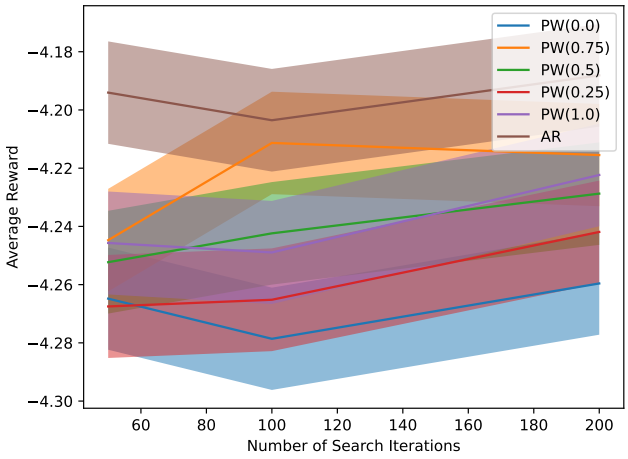

Figure 19: Pendumulum results using DQN trained for finite horizon.

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.

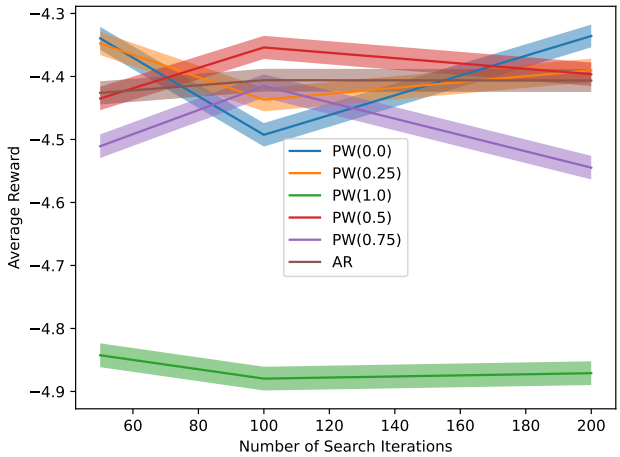

Figure 20: Pendulum results using AlphaZero during training.