Scalable Online Planning via Reinforcement Learning Fine-Tuning

Anonymous Author(s)
Affiliation
Address
email

Abstract

Lookahead search has been a critical component of recent AI successes, such as in the games of chess, go, and poker. However, the search methods used in these games, and in many other settings, are tabular. Tabular search methods do not scale well with the size of the search space, and this problem is exacerbated by stochasticity and partial observability. In this work we replace tabular search with online model-based fine-tuning of a policy neural network via reinforcement learning, and show that this approach outperforms state-of-the-art search algorithms in benchmark settings. In particular, we use our search algorithm to achieve a new state-of-the-art result in self-play Hanabi, and show the generality of our algorithm by also showing that it outperforms tabular search in the Atari game Ms. Pacman.

1 Introduction

Lookahead search has been a key component of successful AI systems in sequential decision-making problems. For example, in order to achieve superhuman performance in go, chess and shogi, AlphaZero leveraged Monte Carlo tree search (MCTS) [38]. MuZero extended this even further to Atari games, again using MCTS [33]. Without MCTS, AlphaZero performs below a top human level, and more generally no superhuman Go bot has yet been developed that does not use some form of MCTS. Similarly, search algorithms were a critical component of AI successes in backgammon [45], chess [10], poker [28, 7, 8], and Hanabi [22]. However, even though different search algorithms were used in each domain, *all* of them were *tabular* search algorithms, i.e., a distinct policy was computed for each state encountered during search, without any function approximation to generalize between similar states.

While tabular search has achieved great success, particularly in perfect-information deterministic environments, its applicability is clearly limited. For example, in the popular partially observable stochastic AI benchmark game Hanabi [5], one-step lookahead search involves a search over about 20 possible next states. However, searching over all two-step joint policies would require a search over 20^{20} states, which is clearly intractable for tabular search. Additionally, unlike perfect-information deterministic games where it is only necessary to search over a tiny fraction of the next several moves, partial observability and stochasticity make it impossible to limit the search to a tiny subset of all possible states. Fortunately, many of these states are extremely similar, so a search algorithm can in theory benefit by generalizing between similar states. This is the motivation for our non-tabular search algorithm.

In this paper we take inspiration from related research in continuous control environments that use non-tabular planning algorithms improve performance at inference time [47, 26, 1]. These methods leverage finite-horizon model-based rollouts. Specifically, we replace tabular search with fine-tuning of the policy network at inference time. We show that with this approach we are able to achieve state-of-the-art performance in Hanabi. Specifically, our method is able to search multiple moves ahead

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

and discover joint deviations, which in general is intractable using tabular search. We also show the generality of our approach by showing that it outperforms Monte Carlo tree search in deterministic and stochastic versions of the Atari game Ms. Pacman.

2 Background

2.1 MDPs, POMDPs, and Dec-POMDPs

We consider Markov decision processes (MDPs) [42], where the agent observes the state $s_t \in S$ at time t, performs an action according to their policy $a_t \sim \pi(s_t)$ and receive reward $r_t = r(s_t, a_t)$. The environment transit to the next state following transition probability $s_{t+1} \sim P(s_{t+1}|s_t, a_t)$.

POMDPs [39] extend MDPs to the partially setting where the agent cannot observe the true underlying state but instead receive information through a observation function $o_t^i = \Omega^i(s_t)$. Due to the partial observability, the agent policy π often needs to take into account the entire action-observation history (AOH) denoted as $\tau'_t = \{o_0, a_0, r_0, \dots o_t\}$ with $\tau_t = \{s_0, a_0, r_0, \dots, s_t\}$ representing the true underlying trajectory.

Dec-POMDPs [30, 5] extend POMDPs to the cooperative multi-agent setting. At each time step t, each agent i receives their individual observation from the state o_t^i and selects action a_t^i. We denote the individual policy for each player as π_i and joint policy as π. We use $\tau_t^i = \{o_0^i, \mathbf{a_0}, r_0, \dots o_t^i\}$ to represent the individual AOH from the perspective of agent i.

2.2 Tabular Search in Dec-POMDPs

SPARTA [22] is a tabular search algorithm previously applied in Dec-POMDPs that is proven to never hurt performance and in practice greatly improves performance. Specifically, SPARTA assumes that all agents in a game agree on a joint blueprint policy π_{bp} and that one or more agents may choose to deviate from π_{bp} at test time. We assume a two player setting in the following discussion. There are two versions of SPARTA:

In **single-agent SPARTA**, only one agent may deviate from the blueprint. We denote the search agent as agent i and the other agent as $-i$. Agent i populates their belief over states with $B(\tau_0^i) = \{\tau_0 | \Omega^i(\tau_0) = \tau_0^i\}$ and updates the belief at each step based on agent $-i$'s move, common knowledge revealed through hinting, and the new observations. At action selection time, agent i compute a Monte Carlo estimation of

$$Q(\tau_t^i, a_t) = E_{\tau_t \sim B(\tau_t^i), \tau_T \sim P(\tau_T|\tau_t, a_t, \pi_{bp})} R_t(\tau_T),$$

where $R_t(\tau_T) = \sum_{t'=t}^{T} r_{t'}$ is the forward looking return from t to termination step T. Agent i will pick $\arg\max_{a_t} Q(\tau_t^i, a_t)$ instead of the blueprint action a_{bp} if $\max_{a_t} Q(\tau_t^i, a_t) - Q(\tau_t^i, a_{bp}) > \epsilon$.

In **multi-agent SPARTA**, both agents are allowed to search at test time. However, since agent i's belief over states depends on agent $-i$'s policy, it is essential that agent i replicate the search policy of agent $-i$. Since agent i does not know the private state of agent $-i$, agent i must therefore compute agent $-i$'s policy for *every* possible AOH τ_t^i agent i may have seen, a process referred to as *range-search*. Once a correct belief is computed, agent $-i$ runs their own search. The range-search can be prohibitively expensive to compute in many cases because players might have millions of potential AOH. For example, Hanabi might have up to about 10 million AOHs. To mitigate this problem, the authors introduce a way to execute multi-agent search only when the range of AOHs is small enough that it is computationally feasible to do so, and the algorithm uses single-agent search otherwise. Even so, SPARTA multi-agent search is limited to *one-ply* search – each agent calculates the expected value of only the next action. The computational complexity grows exponentially with each additional ply. In contrast, our non-tabular multi-agent search technique makes it possible to conduct 2-ply, and deeper, search.

2.3 Reinforcement Learning

Our goal is to learn a stationary policy $\pi(a|s)$ such that the expected non-discounted return $J(\pi) = \mathbb{E}_{s_0 \sim \rho(s)} \mathbb{E}_\pi \sum_{t=0}^{\infty} r(s_t, a_t)$ is maximized, where $\rho(s)$ is the initial state distribution. For any policy π we define its value $V_\pi(s_t) = \mathbb{E}_\pi(\sum_{l=0}^{\infty} r(s_{t+l}, a_{t+l})|s_t)$, its Q-value $Q_\pi(s_t, a_t) =$

$\mathbb{E}_\pi(\sum_{l=0}^{\infty} r(s_{t+l}, a_{t+l})|s_t, a_t)$ and its advantage $A_\pi(s_t, a_t) = Q_\pi(s_t, a_t) - V_\pi(s_t)$. In reinforcement learning (RL), the transition probability p and the reward r are unknown, and the policy is learned by sampling trajectories $\tau = ((s_0, a_0), ..., (s_T, a_T))$ from the environment.

In Q-learning, we use the sampled trajectories to learn the Q-function of the optimal policy, given by the Bellman equation: $Q(s, a) = \mathbb{E}_{s'}(r(s, a) + \gamma \max_{a'} Q(s', a'))$. The learned policy is greedy with respect to Q: $\pi(a|s) = \arg\max_a Q(s, a)$. In DQN, the Q-value is approximated with a neural network Q_θ that is trained by taking gradients of the mean squared Bellman error:

$$\nabla_\theta \hat{\mathbb{E}}(Q_\theta(s, a) - (r(s, a) + \gamma \max_{a'} Q_{\theta'}(s', a')))^2 \tag{1}$$

where θ' is a fixed copy of θ and $\hat{\mathbb{E}}$ denotes the empirical average over a finite number of batches.

In policy gradient (PG), we directly learn a parametrized policy π_θ by performing gradient descent using an estimator of J:

$$\nabla_\theta \hat{\mathbb{E}}(\log \pi_\theta(a|s) \hat{A}(s, a)) \tag{2}$$

where $\hat{A}$ is an estimate of the advantage of π_θ.

2.4 Decision-Time Planning

To make better decisions at test time, we can specialize the learned policy, which we refer to as the **blueprint** policy (also known as the prior policy), to the current state by using it as prior in an online planning algorithm.

One of the most popular planning algorithms is **Monte Carlo tree search (MCTS)**, which has famously enabled superhuman performance in go, chess, and many other settings [36, 37, 38, 33]. MCTS builds a tree of potential future states starting from the current state. Each node s keeps track of its action-state visitation counts $N(s, a)$ and an estimate of its Q-values $Q(s, a)$, refined each time the node is visited. The prior policy is used to select the next action to take during the tree traversal.

MCTS don't scale well in environments with large branching factors caused by a high amount of stochasticity or partial observability. For example, MCTS becomes very expensive as the size of the action space increases because it needs to build an explicit tree to keep statistics about state-action pairs, and nodes need to be visited several times for the method to be effective. Furthermore, MCTS can in the worst case expand very deeply a branch that it misidentifies to be optimal due to a lack of exploration – exacerbated in high dimensions– which leads to a worst-case sample complexity worse than uniform sampling [29]. On the other side, MC methods need a prohibitively large amount of samples to be accurate in high dimensions.

3 Reinforcement Learning Fine-Tuning

To alleviate the inefficiency of tabular search methods like MCTS in large state and action spaces, we propose to formulate online planning as a RL problem that should be quickly solvable given a sufficiently optimal blueprint policy. We do so by using RL as a multiple-step policy improvement operator [12]. More specifically, we bias the initial state distribution towards the current state s^* and we reduce the horizon of the problem, leading to the following truncated objective:

$$\max_{\tilde{\pi}} (\mathbb{E}_{s_0 \sim \rho_{s^*}(s)} \mathbb{E}_{a_{0:H-1}, s_{1:H} \sim \tilde{\pi}} \sum_{t=0}^{H-1} (r(s_t, a_t)) + \mathbb{E}_{a_{H:\infty}, s_{H+1:\infty} \sim \pi} \sum_{t=H}^{\infty} (r(s_t, a_t))) \tag{3}$$

where $\rho_{s^*}(s) = \mathbb{1}(s = s^*)$ and π is the blueprint policy. If π is optimal, then $\pi' = \pi$ is a solution to the problem.

The blueprint π is either directly parameterized by θ or is greedy with respect to a Q-value parameterized by θ. In both cases, we improve π for the next H steps by following the gradient of the truncated objective with respect to θ. We present the two approaches in more detail in the following subsections.

3.1 Policy Fine-Tuning

In policy fine-tuning, we use any actor-critic algorithm to train a blueprint policy and a value network. For our experiments in the Atari environment, we use PPO [35]. At action selection time, to solve objective (3) we use the blueprint policy to initialize the online policy and the blueprint value to truncate our objective:

$$\max_{\tilde{\theta}}(\mathbb{E}_{s_0\sim\rho_{s^*}(s)}\mathbb{E}_{a_{0:H-1},s_{1:H}\sim\pi_{\tilde{\theta}}}\sum_{t=0}^{H-1}(r(s_t,a_t))+V_\phi(s_H)) \tag{4}$$

If V_ϕ is a perfect estimator of V_{π_θ}, then we exactly optimize the truncated objective 3.

The fine-tuned policy $\pi_{\tilde{\theta}}$ is obtained by performing N gradient steps with the PPO objective. The policy improvement step via policy gradient is presented in Algorithm 1.

Algorithm 1: Policy Gradient Improvement: We use a standard PPO update to fine-tune the blueprint policy π_θ and value V_ϕ from some state s_t.

Input : current state s_t, number of updates N, global policy parameter θ, global value parameter ϕ, horizon H, number of rollouts M

Output : updated parameter $\tilde{\theta}$

Init:

$\theta_0 = \theta$, $\phi_0 = \phi$

for $i \leftarrow 1$ **to** N **do**

Collect M trajectories of H time steps starting from s_t using $\pi_{\theta_{i-1}}$.

Compute the generalized advantage estimate using:

$$\begin{aligned}\delta_t &= r_t + \gamma V_{\phi_{i-1}}(s_{t+1}) - V_{\phi_{i-1}}(s_t)\forall t \in [0, H-2]\\ \delta_{H-1} &= r_t + \gamma V_\phi(s_H) - V_{\phi_{i-1}}(s_t)\end{aligned} \tag{5}$$

$(\phi_i, \theta_i) \leftarrow PPO(\phi_{i-1}, \theta_{i-1})$

return θ_N

3.2 Q-value Fine-Tuning

In Q-value fine-tuning, we train a Q-network using any offline RL algorithm. At action selection time, to solve objective (3) we use the blueprint Q-value to truncate our objective:

$$\max_{\tilde{\pi}}(\mathbb{E}_{s_0\sim\rho_{s^*}(s)}\mathbb{E}_{a_{0:H-1},s_{1:H}\sim\tilde{\pi}}\sum_{t=0}^{H-1}(r(s_t,a_t))+\max_a Q_\theta(s_H,a)) \tag{6}$$

The online policy $\tilde{\pi}$ is greedy with respect to the fine-tuned Q-network $Q_{\tilde{\theta}}$ obtained by performing N gradient steps with the mean squared Bellman error. To alleviate instabilities, the transitions used to fine-tune the Q-network are sampled with probability p from the global buffer replay and with probability $1-p$ from the trajectories sampled at action selection time. The Q-value improvement step is presented in Algorithm 1.

3.3 Scaling Multi-Agent Search in Dec-POMDPs with RL Search

3.3.1 Scaling Single-Agent RL Search

In a test game, agent i maintains a belief over possible trajectories given their own AOH, $B(\tau_t^i)$. At action selection time, agent i performs Q-value fine-tuning (Algorithm 2) of the blueprint policy π_{bp} on the current belief $B(\tau_t^i)$. This produces a new policy π^*. Next, we evaluate both π^* and π_{bp} on 10K trajectories sampled from the belief. If the expected value of π^* is at least ϵ higher than the expected value of π_{bp}, the agent plays according to π^* for the next H moves and search again at the

Algorithm 2: Q-Value Improvement We use a standard Bellman residual update to fine-tune the blueprint Q function Q_θ from some state s_t.

Input : current state s_t, number of updates N, global Q-network parameter θ, horizon H, number of rollouts M, batch size B

Output : updated parameter $\tilde{\theta}$

Init:

$\theta_0 = \theta$

Collect M trajectories of H time steps starting from s_t using an ϵ-greedy policy wrt Q_θ.

For each trajectory, if the environment is not terminated, replace r_{t+H-1} with $r_{t+H-1} + \max_a Q_\theta(s_{t+H}, a)$

for $i \leftarrow 1$ **to** N **do**

  Sample B transitions with probability p from the global buffer and probability $1-p$ from the M collected trajectories.

  $\theta_i \leftarrow \nabla_{\theta_{i-1}} \hat{\mathbb{E}}(Q_{\theta_{i-1}}(s,a) - (r(s,a) + \gamma \max_{a'} Q_{\theta'}(s',a')))^2$

return θ_N

$H+1$th move. Otherwise, the agent sticks to playing according to π_{bp} for the current move and searches again on their next turn.

3.3.2 Scaling Multi-Agent RL Search

A critical limitation of single-agent SPARTA, multi-agent SPARTA, and single-agent RL search is that the searching agent assumes the other agent will follow the blueprint policy on all future turns. It is thus impossible for those methods to find *joint* deviations $(a_t^{i*}, a_{t_1}^{-i*})$ where it is beneficial for agent i deviate to action a_t^{i*} if and only if the other agent $-i$ deviates to a_{t+1}^{-i*}, an action that will not be selected under blueprint. Searching for joint deviation with general tabular methods is computationally infeasible in large Dec-POMDPs such as Hanabi because the effective branching factor is $\prod_i(|B(\tau_t^i)| \times |A^i|)$, which is about 20^{20} in Hanabi.

RL search enables searching for the next H moves for all agents jointly conditioning on the common knowledge. The observations in a Dec-POMDP can be factorized into private observation and public observations o_t^{pub}. For example, in Hanabi the private observations are the teammates hands and the public observation are the played cards, discarded cards, hints given, and previous actions. The public observation is common knowledge among all players. We can define the common knowledge public belief as $B^{pub}(\tau_t^{pub}) = \{\tau_t | \Omega^{pub}(\tau_t) = \tau_t^{pub}\}$ where $\tau_t^{pub} = \{o_0^{pub}, a_1, r_1, \ldots, o_t^{pub}\}$ and Ω^{pub} is the public observation function. We can then draw $\tau \sim B^{pub}(\tau_t^{pub})$ repeatedly. The new policy is obtained via Q-value fine-tuning, in the same way as in single-agent RL search. Since the search procedure uses only public information, it can in principle be replicated by every player independently. However, to simplify the engineering challenges of our research, we conduct search just once and share the solution with all the players. If the newly trained policy is better than the blueprint by at least ϵ, every player will use it for their next L moves.

4 Experiments

4.1 Hanabi Experiments

Hanabi is a 2-5 player partially observable fully cooperative card game and a popular Dec-POMDP benchmark. A detailed description of the rules of the game and an explanation of its challenges can be found in [5]. The deck in Hanabi consists of 5 color suites and each suite contains 10 cards with three 1s, two 2s, two 3s, two 4s and one 5. At the beginning of the game each player draws 4 or 5 cards. The main twist of the game is that players cannot see their own hand but instead observe all other players' hands. All players work together to play cards from 1 to 5 in order for each suite. At each turn, the active player may choose to give a hint to another player at the cost of a hint token. However, the players start with only 8 available hint token, and must discard a card in order to gain a new hint token.

In recent years, there has been tremendous progress in developing increasingly strong agents for Hanabi [5, 13, 22, 18, 19, 20] to the point where the latest agents are approachingn near-perfect scores. As a result, it is difficult to distinguish performance differences between different techniques. For this reason, we additionally evaluate our technique on a harder version of Hanabi in which the maximum number of hint tokens is 2 rather than 8.

We test both single-agent RL search and multi-agent joint RL search in 2-player Hanabi. Blueprint policies are trained with Q-learning and the same learning method is used for test-time fine-tuning. We compare our methods against SPARTA [22], the state-of-the-art search technique that has been applied in this domain.

Blueprint Training. We train blueprint policy using independent Q-learning with a distributed learning setting. The entire process consists of two modules that run simultaneously. The first is a simulation module that runs M copies of the Hanabi game distributed across N parallel running workers. The simulation module collects AOH τ_T at the end of each game and writes those data into a centralized prioritized replay buffer [32, 17]. Meanwhile, a training module samples from the replay buffer to train the policy and send a new copy of the policy to the simulation module every K gradient steps. We set $M = 6400$, $N = 80$ and $K = 10$, the first two of which are chosen to make the replay buffer write speed of the simulation module roughly the same as the replay buffer read speed of the training module. We use the canonical techniques such as double Q-learning [46] and dueling DQN [48] for better performance. We use the same Public-LSTM network architecture as in [20] to simplify the belief update procedure. For simplicity, we use independent Q-learning [44]. We train the blueprint policy for 2 million gradient steps/batches and each batch contains 128 AOHs τ_i. Our blueprint gets 24.23 points on average, which is comparable to the strongest blueprint (24.29) in [20] which is trained with a value decomposition network.

Results. We present our results in Table 1. For single-agent RL search we set the search depth $L = 3$, the number of gradient steps $G = 5000$, and the deviation threshold $\epsilon = 0.05$. For multi-agent RL search we set $L = 1$, $G = 1000$, and $\epsilon = 0.035$. For comparison we also run both the single-agent and multi-agent version of SPARTA on our blueprint. Multi-agent SPARTA can be extremely expensive to run depending on the maximum size allowed for range-search, so we pick the hyper-parameter to make it consume roughly the same amount of time per game as multi-agent RL search. Each cell in Table 1 is evaluated on 2000 games. The numbers on top shows the average score and standard error of the mean, and the number at the bottom of each cell is the percentage of winning games (perfect score games). In normal Hanabi, single-agent RL search beats single-agent SPARTA and multi-agent RL search beats multi-agent SPARTA. However, the scores are close enough that the differences are difficult to distinguish. This is likely because the blueprint in normal Hanabi is already extremely strong, so the benefit of additional search is diminished. On the harder 2-hint variant, RL search, and especially multi-agent RL search, significantly outperforms SPARTA. This indicates that searching for joint deviations is helpful, and RL search is a scalable way to find them.

Whether or not RL search is preferable to SPARTA depends in part on the computational budget. Single-agent SPARTA need only search over about 20 possible actions, so it takes 4 seconds to make a move using 5 CPU cores and 1 GPU. For comparison, single-agent RL search would take 69 seconds per move when searching one move ahead with 20 CPU cores and 2 GPUs. However, RL search scales much more effectively. The branching factor in Hanabi is around 500 and the game involves a high amount of stochasticity due to new cards being drawn, so searching even 3 moves ahead is essentially intractable for single-agent SPARTA. In contrast, single-agent RL search that looks 3 moves ahead takes only 88 seconds per move. Similarly, searching two moves ahead (one for each player) in multi-agent SPARTA would mean searching over about 20^{20} joint policies, which would be intractable. In contrast, multi-agent RL search is able to do this while taking only 180 seconds per move.

4.2 Atari

We demonstrate the generality of our approach by comparing policy gradient fine-tuning to MCTS in the Atari [6] Ms. Pacman environment.

Blueprint Training. We train a PPO [35] agent until convergence, for a total of 10M steps. In the deterministic version of the environment, the PPO agent achieves a return of 3950 at convergence

Variant	Blueprint	SPARTA (Single)	SPARTA (Multi)	RL Search (Single)	RL Search (Multi)
Normal	24.23 $\pm$ 0.04 63.20%	24.57 $\pm$ 0.03 73.90%	24.61 +/- 0.02 75.46%	24.59 $\pm$ 0.02 75.05%	**24.62 $\pm$ 0.03** **75.93%**
2 Hints	22.99 $\pm$ 0.04 17.50%	23.60 $\pm$ 0.03 25.85%	23.67 +/- 0.03 26.87%	23.61 $\pm$ 0.03 27.85%	**23.76 $\pm$ 0.04** **31.01%**

Table 1: **Performance on Hanabi**. Each cell is averaged over 2000 games. The number in the upper half of the cell is the average score $\pm$ standard error of mean (s.e.m.) and the number in the lower half is the percentage of winning games where agents score 25 points.

while in the stochastic version of the environment – via sticky action[25]–, the PPO agent achieves a return of 2390.

MCTS. At every testing time step, we build a tree starting from the current state. We use the blueprint policy to guide the action selection during the tree traversal and we use the value network every time we reach a node never seen before or we reach the depth limit of the tree. In our experiments we use a depth limit of 100. We can significantly improve the performance of MCTS by allowing for an additional hyperparameter to balance policy prior and visitation counts in the second term:

$$\arg\max_a Q(s,a) + c \cdot \pi_\theta(a|x)^\beta \cdot \frac{\sqrt{\sum_{a'} N(s,a')}}{1+N(s,a)} \tag{7}$$

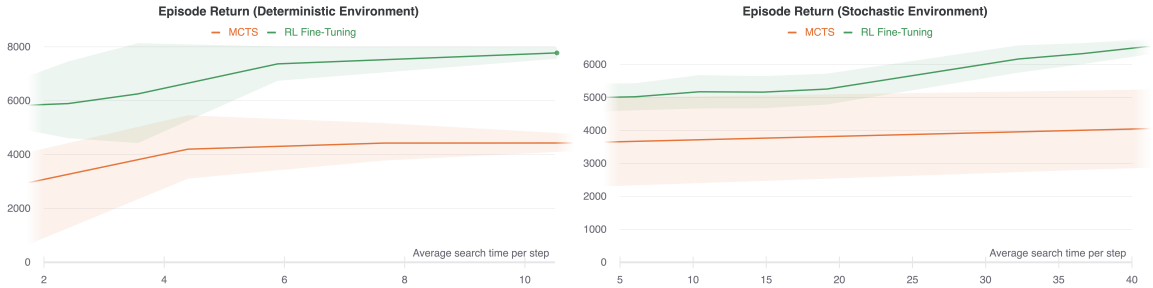
We obtain the best performance with $c = 5$ and $\beta = 0.1$

Policy Gradient Fine-Tuning. Our method achieves a small average time budget by amortizing the search time across multiple steps. In Ms. Pacman, we solve a finite-horizon problem of horizon 30 and we only need to replan every 30 steps (Algorithm 2). We have also tried to amortize MCTS across multiple steps, where we update the tree only after 30 steps. In this setting however, the episode return is worse than what is achieved by the blueprint, emphasizing the need to replan at every timestep, which is not necessary when performing RL search. With RL search, we need an average of 621 samples and 1.2 seconds per step to achieve a return of 8080 which is more than 2 times the return achieved by our asymptotic PPO policy. The total number of additional samples needed is 502,000, which is less than 5% of the samples needed for the blueprint PPO policy to converge. To optimize an infinite-horizon problem rather than a finite-horizon problem, we can use the refined value instead of the blueprint for the last step of each trajectory. The problem is still simplified due to the biased initial state distribution. In the Pacman environment, this setting also yields a maximal return around 8000.

Results. Both MCTS and policy gradient fine-tuning are multi-step improvement operators that optimize objective (3) using a value estimate to truncate the objective. Therefore we expect both methods to achieve the same asymptotic performance. We compare both methods with a finite time budget of the order of 10s. Figure 1 shows the return achieved by the agent when performing either MCTS or RL search at action selection time, versus the average search time budget. We see that RL search consistently outperforms MCTS, contrasting with recent work showing that policy gradient was worse than MCTS for planning in the game Hex [3]. The difference of performance with this previous work might be due to the fact that they are using vanilla PG while we are using PPO.

5 Related Work

Fine-tuning in supervised learning. Instead of searching for the optimal action sequence in the action space – like MCTS do– we are effectively searching in the parameter space of the neural network. We do that by following the gradient of the truncated objective. Since the truncated objective and the biased data distribution are close to the objective and the data distribution the blueprint neural network was trained on, the performance of our method should reflect the success of neural network fine-tuning in supervised learning [43, 31].



(a) Deterministic

(b) Stochastic

Figure 1: **MCTS vs RL Fine-Tuning**: when the average time budget is on the order of 1-10 seconds, RL fine-tuning consistently outperforms MCTS, both in the deterministic and the stochastic versions of Ms. Pacman. The shaded area represent the min/max range across 5 seeds. The curves are smoothed with an exponential moving average.

Fine-tuning in reinforcement learning. Replacing action search with neural network parameter search is reminiscent of recent work in continuous control performing cross-entropy method in the parameter space of the neural network [47]. They argue that the improvement over searching in the action space is due to the smoothness of the neural network loss landscape [24]. We take this idea further by making full use of the parameter space. While they are using scalar information – the return of each rollout –, we use richer information – the gradient of the return with respect to θ. By doing so, we argue that we lose less information about the blueprint than other online planning methods, which enables us to stay efficient in high dimensions.

Policy gradient fine-tuning is related to recent work on iterative amortized policy optimization [26]. Considering policy optimization in the lens of variational inference [23], they argue that amortizing the policy into a neural network leads to the same suboptimal gap than amortizing the posterior in VAEs and propose to inject gradient information into the policy network's input at action selection time to close the gap. While their policy network meta-learns how to use the gradient, we use a fixed update rule. Meta-learning the update rule is a promising future direction that could be stacked on top of our method. However, we suspect learning the update rule at blueprint training time to be intractable in complex environments such as Hanabi.

A large range of previous work has proposed to combine RL with tabular search [40]. In Expert Iteration [2, 33], a policy network is used to amortize an expert tree search algorithm using imitation learning. Rather than amortizing the traversal policy, Hamrick et al. propose to train a Q-network via model-free RL to amortize the Q-values returned by MCTS [16]. Our method differs from this line of work because we don't use any tree search for maximum scalability.

Finally, there has been previous work investigating some variants of RL search. Policy gradient search [3] performs vanilla policy gradient to fine-tune a rollout policy and selects the action by performing MC rollouts with this policy. They perform worse than MCTS in the game Hex. Springenberg et al. [40] compare a variant of policy gradient fine-tuning to tree search in continuous control environments and show that PG fine-tuning performs almost as well as tree search, but not equally well. By contrast to these rather negative results, our positive results show that RL search outperforms MCTS in Ms. Pacman and achieves state-of-the-art performance in Hanabi. Additionally, prior work has looked to generalize between states during MCTS, though without fine-tuning of a pre-trained network [49].

6 Discussion

Despite the incredible success of neural network function approximation in reinforcement learning environments [27, 15, 4], search algorithms have largely remained tabular, as can be seen for example in AI algorithms for go [36, 37, 38], poker [28, 7, 8, 9], and Atari [33, 34, 21]. Tremendous research has gone into adding heuristics that enables tabular search to perform better in these domains, especially in perfect-information deterministic environments [14, 11]. However, we argue

that in order to scale to larger, more realistic environments, especially those involving stochasticity and partial observability, it is necessary to move past purely tabular search algorithms.

The jump from tabular search to RL search comes at a steep upfront cost. However, we show in this paper that in sufficiently complex environments, including the established AI benchmark Hanabi, RL search enables state-of-the-art performance and scales much more effectively. We show this also holds in the Atari Ms. Pacman environment, and we expect it to be true in a wide variety of other domains as well. As computational costs continue to decrease, it is likely that the relative performance of RL search will improve even further. To quote Richard Sutton's *The Bitter Lesson* [41], "One thing that should be learned from the bitter lesson is the great power of general purpose methods, of methods that continue to scale with increased computation even as the available computation becomes very great. The two methods that seem to scale arbitrarily in this way are search and learning."

While a lot of methods combining RL with tabular search have been developed, we have shown that we can achieve better results by completely amortizing online planning into a neural network via RL fine-tuning. Most importantly, we have shown that RL search can effectively search complex domains where tabular search is intractable. This enables us to achieve state-of-the-art results in the AI benchmark Hanabi, which features both stochasticity and partial observability, both of which are prevalent in the real world.

Limitations: While we achieve good results in discrete environments, our algorithms are currently focused on discrete environments. A promising direction for future work would be an investigation of similar techniques in continuous control domains and a comparison to existing techniques (mentioned in the main text) in those domains. A potential limitation of our method compared to gradient-free methods is that we might be constrained by the local minimum of a too weak blueprint. In other words, we may be unable to discover drastically new solutions to the control problem that an optimal global method solving the problem from scratch would find.

Broader impact: RL search is a decision-time planning method aimed at improving search in very high-dimensional spaces. There has been a rich history of previous work in this area, where the goal has been to develop more efficient search methods. This work is in line with previous work and should not directly cause broader harm to society. We are not using any dataset or tool that will perpetuate or enforce bias. Nor does our method make judgements on our behalf that could be misaligned with our values.

References

[1] B. Amos and D. Yarats. The differentiable cross-entropy method. In *International Conference on Machine Learning*, pages 291–302. PMLR, 2020.

[2] T. Anthony, Z. Tian, and D. Barber. Thinking fast and slow with deep learning and tree search. *arXiv preprint arXiv:1705.08439*, 2017.

[3] T. Anthony, R. Nishihara, P. Moritz, T. Salimans, and J. Schulman. Policy gradient search: Online planning and expert iteration without search trees. *arXiv preprint arXiv:1904.03646*, 2019.

[4] A. P. Badia, B. Piot, S. Kapturowski, P. Sprechmann, A. Vitvitskyi, Z. D. Guo, and C. Blundell. Agent57: Outperforming the atari human benchmark. In *International Conference on Machine Learning*, pages 507–517. PMLR, 2020.

[5] N. Bard, J. N. Foerster, S. Chandar, N. Burch, M. Lanctot, H. F. Song, E. Parisotto, V. Dumoulin, S. Moitra, E. Hughes, et al. The hanabi challenge: A new frontier for ai research. *Artificial Intelligence*, 280:103216, 2020.

[6] M. G. Bellemare, Y. Naddaf, J. Veness, and M. Bowling. The arcade learning environment: An evaluation platform for general agents. *Journal of Artificial Intelligence Research*, 47:253–279, 2013.

[7] N. Brown and T. Sandholm. Superhuman AI for heads-up no-limit poker: Libratus beats top professionals. *Science*, page eaao1733, 2017.

[8] N. Brown and T. Sandholm. Superhuman AI for multiplayer poker. *Science*, page eaay2400, 2019.

[9] N. Brown, A. Bakhtin, A. Lerer, and Q. Gong. Combining deep reinforcement learning and search for imperfect-information games. *arXiv preprint arXiv:2007.13544*, 2020.

[10] M. Campbell, A. J. Hoane Jr, and F.-h. Hsu. Deep Blue. *Artificial intelligence*, 134(1-2):57–83, 2002.

[11] P. I. Cowling, E. J. Powley, and D. Whitehouse. Information set monte carlo tree search. *IEEE Transactions on Computational Intelligence and AI in Games*, 4(2):120–143, 2012.

[12] Y. Efroni, G. Dalal, B. Scherrer, and S. Mannor. Multiple-step greedy policies in approximate and online reinforcement learning. In *NeurIPS*, 2018.

[13] J. Foerster, F. Song, E. Hughes, N. Burch, I. Dunning, S. Whiteson, M. Botvinick, and M. Bowling. Bayesian action decoder for deep multi-agent reinforcement learning. In *International Conference on Machine Learning*, pages 1942–1951, 2019.

[14] J.-B. Grill, F. Altché, Y. Tang, T. Hubert, M. Valko, I. Antonoglou, and R. Munos. Monte-carlo tree search as regularized policy optimization. In *International Conference on Machine Learning*, pages 3769–3778. PMLR, 2020.

[15] T. Haarnoja, A. Zhou, P. Abbeel, and S. Levine. Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor. In *International Conference on Machine Learning*, pages 1861–1870. PMLR, 2018.

[16] J. B. Hamrick, V. Bapst, A. Sanchez-Gonzalez, T. Pfaff, T. Weber, L. Buesing, and P. W. Battaglia. Combining q-learning and search with amortized value estimates. *arXiv preprint arXiv:1912.02807*, 2019.

[17] D. Horgan, J. Quan, D. Budden, G. Barth-Maron, M. Hessel, H. van Hasselt, and D. Silver. Distributed prioritized experience replay. In *6th International Conference on Learning Representations, ICLR 2018, Vancouver, BC, Canada, April 30 - May 3, 2018, Conference Track Proceedings*. OpenReview.net, 2018. URL `https://openreview.net/forum?id=H1Dy---0Z`.

[18] H. Hu and J. N. Foerster. Simplified action decoder for deep multi-agent reinforcement learning. In *International Conference on Learning Representations*, 2019.

[19] H. Hu, A. Lerer, A. Peysakhovich, and J. Foerster. other-play for zero-shot coordination. In *International Conference on Machine Learning*, pages 4399–4410. PMLR, 2020.

[20] H. Hu, A. Lerer, N. Brown, and J. N. Foerster. Learned belief search: Efficiently improving policies in partially observable settings, 2021. URL `https://openreview.net/forum?id=xP37gkVKa_0`.

[21] T. Hubert, J. Schrittwieser, I. Antonoglou, M. Barekatain, S. Schmitt, and D. Silver. Learning and planning in complex action spaces. *arXiv preprint arXiv:2104.06303*, 2021.

[22] A. Lerer, H. Hu, J. N. Foerster, and N. Brown. Improving policies via search in cooperative partially observable games. In *The Thirty-Fourth AAAI Conference on Artificial Intelligence, AAAI 2020, The Thirty-Second Innovative Applications of Artificial Intelligence Conference, IAAI 2020, The Tenth AAAI Symposium on Educational Advances in Artificial Intelligence, EAAI 2020, New York, NY, USA, February 7-12, 2020*, pages 7187–7194. AAAI Press, 2020. URL `https://aaai.org/ojs/index.php/AAAI/article/view/6208`.

[23] S. Levine. Reinforcement learning and control as probabilistic inference: Tutorial and review. *arXiv preprint arXiv:1805.00909*, 2018.

[24] H. Li, Z. Xu, G. Taylor, C. Studer, and T. Goldstein. Visualizing the loss landscape of neural nets. *arXiv preprint arXiv:1712.09913*, 2017.

[25] M. C. Machado, M. G. Bellemare, E. Talvitie, J. Veness, M. Hausknecht, and M. Bowling. Revisiting the arcade learning environment: Evaluation protocols and open problems for general agents. *Journal of Artificial Intelligence Research*, 61:523–562, 2018.

[26] J. Marino, A. Piché, A. D. Ialongo, and Y. Yue. Iterative amortized policy optimization. *arXiv preprint arXiv:2010.10670*, 2020.

[27] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski, et al. Human-level control through deep reinforcement learning. *Nature*, 518(7540):529–533, 2015.

[28] M. Moravčík, M. Schmid, N. Burch, V. Lisỳ, D. Morrill, N. Bard, T. Davis, K. Waugh, M. Johanson, and M. Bowling. Deepstack: Expert-level artificial intelligence in heads-up no-limit poker. *Science*, 356(6337):508–513, 2017.

[29] R. Munos. From bandits to monte-carlo tree search: The optimistic principle applied to optimization and planning. 2014.

[30] F. A. Oliehoek, M. T. Spaan, and N. Vlassis. Optimal and approximate q-value functions for decentralized pomdps. *Journal of Artificial Intelligence Research*, 32:289–353, 2008.

[31] M. Oquab, L. Bottou, I. Laptev, and J. Sivic. Learning and transferring mid-level image representations using convolutional neural networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1717–1724, 2014.

[32] T. Schaul, J. Quan, I. Antonoglou, and D. Silver. Prioritized experience replay. In Y. Bengio and Y. LeCun, editors, *4th International Conference on Learning Representations, ICLR 2016, San Juan, Puerto Rico, May 2-4, 2016, Conference Track Proceedings*, 2016. URL `http://arxiv.org/abs/1511.05952`.

[33] J. Schrittwieser, I. Antonoglou, T. Hubert, K. Simonyan, L. Sifre, S. Schmitt, A. Guez, E. Lockhart, D. Hassabis, T. Graepel, et al. Mastering atari, go, chess and shogi by planning with a learned model. *arXiv preprint arXiv:1911.08265*, 2019.

[34] J. Schrittwieser, T. Hubert, A. Mandhane, M. Barekatain, I. Antonoglou, and D. Silver. Online and offline reinforcement learning by planning with a learned model. *arXiv preprint arXiv:2104.06294*, 2021.

[35] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*, 2017.

[36] D. Silver, A. Huang, C. J. Maddison, A. Guez, L. Sifre, G. Van Den Driessche, J. Schrittwieser, I. Antonoglou, V. Panneershelvam, M. Lanctot, et al. Mastering the game of go with deep neural networks and tree search. *Nature*, 529(7587):484, 2016.

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

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

[39] E. J. Sondik. The optimal control of partially observable markov decision processes. *PhD the sis, Stanford University*, 1971.

[40] J. T. Springenberg, N. Heess, D. Mankowitz, J. Merel, A. Byravan, A. Abdolmaleki, J. Kay, J. Degrave, J. Schrittwieser, Y. Tassa, et al. Local search for policy iteration in continuous control. *arXiv preprint arXiv:2010.05545*, 2020.

[41] R. Sutton. The bitter lesson. `http://www.incompleteideas.net/IncIdeas/BitterLesson.html`. Accessed: 2021-05-27.

[42] R. S. Sutton and A. G. Barto. *Reinforcement learning: An introduction*. MIT press, 2018.

[43] C. Tan, F. Sun, T. Kong, W. Zhang, C. Yang, and C. Liu. A survey on deep transfer learning. In *International conference on artificial neural networks*, pages 270–279. Springer, 2018.

[44] M. Tan. Multi-agent reinforcement learning: Independent vs. cooperative agents. In *Proceedings of the tenth international conference on machine learning*, pages 330–337, 1993.

[45] G. Tesauro. TD-Gammon, a self-teaching backgammon program, achieves master-level play. *Neural computation*, 6(2):215–219, 1994.

[46] H. van Hasselt, A. Guez, and D. Silver. Deep reinforcement learning with double q-learning. In D. Schuurmans and M. P. Wellman, editors, *Proceedings of the Thirtieth AAAI Conference on Artificial Intelligence, February 12-17, 2016, Phoenix, Arizona, USA*, pages 2094–2100. AAAI Press, 2016. URL `http://www.aaai.org/ocs/index.php/AAAI/AAAI16/paper/view/12389`.

[47] T. Wang and J. Ba. Exploring model-based planning with policy networks. *arXiv preprint arXiv:1906.08649*, 2019.

[48] Z. Wang, T. Schaul, M. Hessel, H. van Hasselt, M. Lanctot, and N. de Freitas. Dueling network architectures for deep reinforcement learning. In M. Balcan and K. Q. Weinberger, editors, *Proceedings of the 33nd International Conference on Machine Learning, ICML 2016, New York City, NY, USA, June 19-24, 2016*, volume 48 of *JMLR Workshop and Conference Proceedings*, pages 1995–2003. JMLR.org, 2016. URL `http://proceedings.mlr.press/v48/wangf16.html`.

[49] T. Yee, V. Lisỳ, and M. H. Bowling. Monte carlo tree search in continuous action spaces with execution uncertainty. In *IJCAI*, pages 690–697, 2016.

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 Section 6**
   (c) Did you discuss any potential negative societal impacts of your work? [Yes] **See Section 6**
   (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? [N/A]
   (b) Did you include complete proofs of all theoretical results? [N/A]
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)? [Yes] **see supplemental material**
   (b) Did you specify all the training details (e.g., data splits, hyperparameters, how they were chosen)? [Yes]
   (c) Did you report error bars (e.g., with respect to the random seed after running experiments multiple times)? [Yes]
   (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)? [Yes] 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]
   (b) Did you mention the license of the assets? [N/A]
   (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] **We do not use human subjects data.**
   (e) Did you discuss whether the data you are using/curating contains personally identifiable information or offensive content? [N/A] **We do not use human subjects data.**
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]