ExPoSe: Combining State-Based Exploration with Gradient-Based Online Search

Dixant Mittal
National University of Singapore
Singapore
dixant@comp.nus.edu.sg

Siddharth Aravindan
National University of Singapore
Singapore
siddharth.aravindan@comp.nus.edu.sg

Wee Sun Lee
National University of Singapore
Singapore
leews@comp.nus.edu.sg

ABSTRACT

Online tree-based search algorithms iteratively simulate trajectories and update action-values for a set of states stored in a tree structure. It works reasonably well in practice but fails to effectively utilise the information gathered from similar states. Depending upon the smoothness of the action-value function, one approach to overcoming this issue is through online learning, where information is interpolated among similar states; Policy Gradient Search provides a practical algorithm to achieve this. However, Policy Gradient Search lacks an explicit exploration mechanism, which is a key feature of tree-based online search algorithms. In this paper, we propose an efficient and effective online search algorithm called Exploratory Policy Gradient Search (ExPoSe), which leverages information sharing among states by updating the search policy parameters directly, while incorporating a well-defined exploration mechanism during the online search process. We evaluate ExPoSe on a range of decision-making problems, including Atari games, Sokoban, and Hamiltonian cycle search in sparse graphs. The results demonstrate that ExPoSe consistently outperforms other popular online search algorithms across all domains. The ExPoSe source code is available at *https://github.com/dixantmittal/ExPoSe*.

KEYWORDS

Exploration; Online Tree Search; Reinforcement Learning

ACM Reference Format:
Dixant Mittal, Siddharth Aravindan, and Wee Sun Lee. 2023. ExPoSe: Combining State-Based Exploration with Gradient-Based Online Search. In *Proc. of the 22nd International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2023), London, United Kingdom, May 29 – June 2, 2023*, IFAAMAS, 14 pages.

1 INTRODUCTION

Online search algorithms have shown good performance on complex search problems such as Chess [36] and Go [35]. Monte Carlo Tree Search (MCTS) is a popular online tree search algorithm that builds a tree structure to maintain a Monte Carlo average of Q-values by simulating different scenarios from the input state. Some variants of MCTS, like Upper Confidence bound applied to Trees (UCT) [19], use a best-first search approach to explore regions of the search tree that could potentially return better rewards. UCT achieves this by adding an exploration bonus to the Q-values of the state-action nodes based on their visitation counts while selecting an action during the simulation phase. This encourages exploration by occasionally prompting it to choose a less-visited child node. Predictor-UCT (PUCT) [29] is a variant of UCT used in AlphaGo [35] that incorporates prior domain knowledge by scaling the exploration bonus of an action with its prior probability, which is predicted by a learnt policy network. UCT-type tree search methods have a well-defined exploration-exploitation trade-off mechanism that is easy to implement and perform well in practice.

Unfortunately, tree search methods have a significant limitation: they do not fully exploit the information gained from each simulation. Specifically, the information obtained from a rollout trajectory is only used to update the action-value estimates of the nodes in the path traversed by the agent, and the rest of the nodes in the tree are ignored. As a result, the search algorithm cannot share the knowledge gained from the expanded leaf nodes to similar states. This problem becomes more pronounced when the search budget is limited, severely limiting the performance of tree search methods.

Online learning methods provide a way to distribute search information obtained from a trajectory to the entire state space. They directly update the parameters of the function being learned, and the information gained during each update is reflected across the state space due to the generalisation capabilities of the learned function approximator. Online gradient descent is a common online learning method, and policy gradient can be used to produce the online gradient from a trajectory. A search algorithm can be developed by iteratively simulating a trajectory from the input state using the current policy and applying policy gradient updates to improve the policy after each trajectory. Policy Gradient Search (PGS) [1] is one such example and performs well for many combinatorial problems [1, 7]. However, a key issue with PGS is the inherent lack of a proper exploration mechanism.

We propose a novel online search algorithm called **Ex**ploratory **Po**licy Gradient **Se**arch (ExPoSe) that leverages the benefits of both tree search and online learning methods. ExPoSe utilises a tree structure to store state statistics, such as visitation counts and value estimates, allowing it to define a robust exploration mechanism and exploit the valuable information from the subtree at each node in the search tree. Moreover, it incorporates online learning to efficiently distribute the information obtained from a new trajectory across the entire state space.

ExPoSe iteratively simulates trajectories from the input state using a simulation policy, and maintains a visitation count for each encountered state. To achieve exploration, the simulation policy modifies the learned policy using a state-based exploration term that is inversely proportional to the visitation count of the state. The exploration term is added to the unnormalised log probabilities of actions predicted by the learned policy network, thereby enabling a bonus-based exploration mechanism. Moreover, ExPoSe propagates the information obtained from the simulated trajectory by

Proc. of the 22nd International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2023), A. Ricci, W. Yeoh, N. Agmon, B. An (eds.), May 29 – June 2, 2023, London, United Kingdom.

directly updating the learnt policy parameters using policy gradient methods.

After conducting experiments, we observe that ExPoSe outperforms PGS when exploration is incorporated into the policy gradient search. However, there are still two issues that need to be addressed to further improve the agent's performance. Thefi rst issue is related to the fact that the trajectory is generated using a simulation policy, which differs from the target policy being learned due to the addition of an exploration term. This difference can lead to incorrect gradient computation if we use policy gradient methods, such as REINFORCE [41]. To address this issue, we use importance sampling to correct the off-policy evaluation and obtain an unbiased estimate of the gradients. The second issue is that policy gradient methods, like REINFORCE, suffer from high variance due to the stochastic nature of the policy. To reduce the variance of the policy gradients while being unbiased, we use a baseline term. The value estimate of each state in the trajectory can serve as an effective baseline and can be used to construct an advantage while computing the policy gradients. This technique generally works well in practice. Fortunately, the tree structure used in ExPoSe enables us to efficiently compute the value estimates of a state from its subtree by using the Bellman equation.

In our empirical study, we investigate the efficacy of incorporating these key ideas from tree-based and gradient-based search methods on a wide range of decision-making problems, including popular goal-based problems like Sokoban, Hamiltonian cycle search in sparse graphs, and grid navigation, as well as image-based games in the Atari 2600 test suite. Wefi nd that ExPoSe consistently outperforms existing online search algorithms on all the domains, demonstrating its ability to leverage the strengths of tree-based and gradient-based search methods to enhance the agent's performance. These results indicate the potential of ExPoSe as a general-purpose online search algorithm for a variety of complex decision-making problems.

2 BACKGROUND

2.1 Markov Decision Process

Many real-world tasks can be formulated as Markov decision processes (MDPs), where the agent interacts with the environment by choosing a sequence of actions to reach the goal. In MDPs, the agent observes a state described by a collection of variables that provide relevant details for action selection. Given a state, the agent chooses an action determined by a policy π, which is a function that maps every state s in the state space to an action a and receives a feedback reward r from the environment. A value function $V^{\pi}(s)$ represents the expected sum of all rewards the agent accrues by following a policy π when starting from state s. Similarly, a Q-function (or action-value function) $Q^{\pi}(s,a)$ represents the expected sum of rewards obtained when the agent chooses the action a in the state s but follows the policy π in the subsequent states until the episode terminates. To solve the MDP, the agent has tofi nd a policy π^*, which maximises the value function $V^{\pi}(s)$ for all states s, i.e.,

$$\pi^* = \arg\max_{\pi} V^{\pi}(s)$$

In general,fi nding the optimal policy for all the states is a challenging problem in an MDP with a large state space. Therefore, agents solving such MDPs are often assisted by online search methods, such as UCT [19] and branch-and-bound [21] search. These online search methods are designed to help the agent determine an approximately optimal action for a given state.

2.2 Monte Carlo Tree Search

Monte Carlo Tree Search (MCTS) is an online tree-based search algorithm that improves the policy by iteratively simulating trajectories and expanding the search tree towards a more promising search space. Each tree node represents a state, and the root node represents the input state for which the agent has to output an action. The children of any node in the tree represent the states that are reachable from that node by taking a single action represented by the corresponding branch. In every iteration, MCTS performs the following operations in order: *selection*, *initialisation* and *backup*. This process is repeated until termination, determined by either afi xed number of search iterations or a maximum search time allocated for each step. At the end of the search process, the agent chooses the action corresponding to the highest action-value at the root node.

The selection phase in MCTS isfl exible and has been modified to create different variants, such as UCT [19] or PUCT [29]. These variants usually differ in their exploration mechanism and have achieved good performances in different problems.

2.2.1 *Selection.*

During the selection phase, the agent uses a tree simulation policy to traverse a path from the root node to a leaf node in the search tree. The tree simulation policy uses various node statistics for a given node, such as a Monte-Carlo average of Q-values or visitation counts, to select an action. Tree simulation policies may also include a bonus term favouring actions that are likely to have the highest Q-value but are selected less frequently. UCT, for example, uses the UCB1 [19] formulation to add an exploration bonus $U(s,a)$ to the estimated Q-values $Q(s,a)$. More precisely, $U(s,a) \propto \sqrt{\frac{\log N(s)}{N(s,a)}}$, where $N(s)$ is the number of times the agent has visited the node s, while $N(s,a)$ is the number of times the agent has chosen the action a at node s. On the other hand, PUCT uses a policy π_θ, which is learnt offline, to scale the exploration bonus added to the Q-value corresponding to action branch a by $\pi_\theta(a|s)$, i.e. $U(s,a) \propto \frac{\pi_\theta(a|s)}{1+N(s,a)}$. The agent uses the tree simulation policy until it reaches a leaf node.

2.2.2 *Expansion.*

The leaf node encountered at the end of the simulation phase is expanded by adding its children to the search tree. These new nodes are initialised with a prior value estimate given by an offline learnt value function approximator. After adding these nodes to the tree, a rollout trajectory is played using a rollout policy from the leaf node to get an unbiased estimate of its value. Usually, the rollout policy is trained offline on a large data set to get a better estimate of the node's value. However, some variants, like AlphaGo[35], may use a faster but less accurate policy to perform the rollout.

2.2.3 *Backup.*

After completing the rollout from the leaf node, the rollout value is

propagated upwards to update the Q-value estimates along the path to the root. The common variants of MCTS maintain a Monte-Carlo average of the Q-values for each node and can trivially incorporate the new rollout value into the average. Some variants may also maintain an empirical variance of the observed rollout values, which could be used to compute the upper confidence bound during the simulation phase.

2.3 Policy Gradient Search

Policy Gradient Search (PGS) is an online gradient-based search algorithm that iteratively improves a simulation policy, parameterised by θ, using policy gradient methods. Each iteration starts from the root and uses a simulation policy to simulate a trajectory. We define an objective function $J_\theta(\tau)$ that measures the policy's performance based on the rewards obtained from the trajectory τ i.e.:

$$J_\theta(\tau) = \mathbb{E}_{\tau\sim\pi_\theta(\tau)} r(\tau)$$

where $r(\tau)$ is the sum of rewards obtained for the trajectory τ. Given this objective function, we can compute the gradient of the policy parameters using policy gradient methods, such as REINFORCE [41], and update the policy parameters using an optimiser such as stochastic gradient descent. The parameter update is given by:

$$\theta \leftarrow \theta + \alpha \frac{1}{N} \sum_{i=1}^{N} \left[\sum_{t=1}^{T} \nabla_\theta \log \pi_\theta(a_{i,t}|s_{i,t}) \sum_{t'=1}^{T} r(s_{i,t'}, a_{i,t'}) \right]$$

where

- π_θ: parameterised policy with parameters θ,
- $\sum_{t'=1}^{T} r(s_{i,t'}, a_{i,t'})$: sum of rewards for trajectory τ_i.
- $s_{i,t}$ and $a_{i,t}$ are the state and the action respectively at timestep t in trajectory τ_i.

Policy Gradient Search maintains the values observed from simulations at the root node and uses PUCT-type formulation to select thefi rst action a_{root} at root node s_{root} i.e.

$$a_{root} = \arg\max_a \left[Q(s_{root}, a) + c\pi(a|s_{root}) \frac{\sqrt{N(s_{root})}}{1 + N(s_{root}, a)} \right]$$

However, the subsequent actions are sampled from the simulation policy only.

Neural Networks are commonly used to approximate policy in gradient-based search. However, due to the high computational cost of backpropagation, PGS calculates gradients for only the policy head parameters. Additionally, Anthony et al. [1] recommend freezing earlier layers to reduce the FLOPS required for the backward pass, resulting in the parameter optimisation step being computationally insignificant compared to other search operations.

3 EXPLORATORY POLICY GRADIENT SEARCH

Both tree-based and gradient-based online search algorithms improve the agent's performance when combined with offline learnt prior policy and value functions. We identify and empirically verify that the key properties that improve the performance of these methods are complementary to each other. These properties are:

(1) Interpolation of the updated search information across the state space helps in adapting the search policy for similar states.
(2) A well-defined exploration mechanism that can efficiently balance the exploration-exploitation trade-off is important for the online search.

Along the same lines, we propose an efficient and effective search method named **Ex**ploratory **Po**licy Gradient **Se**arch (ExPoSe). ExPoSe iteratively simulates trajectories from the input state using a simulation policy and directly updates the parameters of the search policy using policy gradient methods, effectively interpolating the information obtained from the simulated trajectories across the state space. Moreover, it maintains a tree structure that stores state statistics, like visitation counts and the value estimate of a state, of all the states encountered during the simulation. ExPoSe uses a visitation counts-based exploration incentive that encourages the simulation policy to discover new states during the search. Additionally, the tree structure allows a better estimation of the state values using the Bellman equation, which can be used to reduce the variance in policy gradient methods. We discuss ExPoSe in detail in the following sections.

3.1 Policy Gradients

The overall structure of ExPoSe is similar to PGS (Section 2.3). ExPoSe iteratively simulates trajectories from the input state using a simulation policy, $\phi_{sim}(s,a)$, which is a combination of a parameterised policy network $\phi_\theta(s,a)$ and an exploration term (defined in Section 3.2). After each simulation, the information obtained from the trajectory τ is used to optimise the objective function $J_\theta(\tau)$ by computing gradients for the parameters θ of the policy network $\phi_\theta(s,a)$ using REINFORCE i.e.

$$\nabla_\theta J(\tau) = \mathbb{E}_{\tau\sim\pi_\theta(\tau)} \left[\sum_{t=1}^{T} \nabla_\theta \log \pi_\theta(a_t|s_t) \sum_{t'=1}^{T} r(s_{t'}, a_{t'}) \right] \quad (1)$$

where

- $J_\theta(\tau)$: objective function to compute policy gradients.
- τ: sequence of state action pairs i.e. $\tau = (s_1, a_1, ..., s_T, a_T)$.
- $r(s_{t'}, a_{t'})$: rewards obtained at timestep t.
- π_θ: normalised probability i.e. $\pi_\theta(a|s) = \frac{\exp(\phi_\theta(s,a))}{\sum_a \exp(\phi_\theta(s,a))}$

Since we have a limited search budget, we set $N = 1$ and use a single trajectory τ to approximate the expectation. Hence, we can simplify the equation

$$\nabla_\theta J(\tau) = \left[\sum_{t=1}^{T} \nabla_\theta \log \pi_\theta(a_t|s_t) \sum_{t'=1}^{T} r(s_{t'}, a_{t'}) \right] \quad (2)$$

Correspondingly, the policy network parameters θ can be updated using the gradient ascent algorithm.

$$\theta \leftarrow \theta + \alpha \left[\sum_{t=1}^{T} \nabla_\theta \log \pi_\theta(a_t|s_t) \sum_{t'=1}^{T} r(s_{t'}, a_{t'}) \right] \quad (3)$$

where α is the tunable learning rate.

We repeat this process of iterating through simulation and policy optimisation until we exhaust the search budget. Since we optimise

the policy parameters directly using REINFORCE, the updated information is reflected on the policy determined by $\phi_\theta(s,a)$ for every state s in the state space.

3.2 State-based Exploration

Policy Gradient Search's major drawback is the inherent lack of a well-defined exploration mechanism. Larma et.al. [20] suggests a simple way to induce exploration in the policy by adding entropy regularisation in equation 3. However, entropy regularisation is state-agnostic and induces exploration by increasing the randomness in the policy throughout the state space, which may not be desirable. We suggest and empirically verify that a better way to balance the exploration-exploitation trade-off is to use a principled approach based on state statistics like visitation counts. We take inspiration from UCB1 [19] and propose to augment the simulation policy used in ExPoSe with a state-based exploration term $E(s,a)$, for a state s and corresponding action a, which is inversely proportional to the visitation count $N(s,a)$, i.e. $E(s,a) = \frac{c}{1+N(s,a)}$, where c is a tunable hyperparameter. ExPoSe maintains the visitation counts $N(s,a)$ of all the state-action pairs encountered during the simulations. The simulation policy at any state is derived by combining the logits $\phi_\theta(s,a)$ predicted by the parameterised policy network and the exploration term $E(s,a)$ as follows:

$$\begin{aligned}\phi_{sim}(s,a) &= \phi_\theta(s,a) + E(s,a) \\ &= \phi_\theta(s,a) + \frac{c}{1+N(s,a)}\end{aligned}$$

We apply the softmax operation over the logits $\phi_{sim}(s,a)$ to get the normalised simulation policy distribution $\pi_{sim}(s,a)$ i.e.

$$\pi_{sim}(a|s) = \frac{\exp(\phi_{sim}(s,a))}{\sum_a \exp(\phi_{sim}(s,a))}$$

We sample actions from this normalised simulation policy distribution to simulate a trajectory. The exploration term encourages less frequently selected actions to be sampled with higher probability, which leads to the exploration of new state space. Furthermore, the exploration term for a certain state-action pair decreases as its visitation count increases, allowing for an efficient transfer from exploration to exploitation.

3.3 Importance Sampling

If the trajectory τ is generated using the exploration-induced simulation policy $\pi_{sim}(a|s)$, the policy gradient computed in equation 2 is biased because it is being computed off-policy, i.e. the policy used to generate the trajectory is different from the parameterised policy used to compute policy gradients. However, we canfi x this issue and compute an asymptotically unbiased estimate of the gradients using importance sampling. Let us assume that the simulation policy is the behaviour policy, i.e. the policy that generates the data, and the policy network is the target policy, i.e. the policy which will be optimised. We can compute the policy gradients for the target policy by reweighing the objective function, $J_\theta(\tau)$, with the likelihood of the trajectory under the behaviour policy and the target policy. The importance sampling ratio w is computed as:

$$w = \frac{\mathrm{P}_\theta(\tau)}{\mathrm{P}_{sim}(\tau)} = \prod_{t=1}^{T} \frac{\pi_\theta(a_t|s_t)}{\pi_{sim}(a_t|s_t)}$$

We can re-write equation 2 along with the importance weights as follows:

$$\nabla_\theta J(\tau) = \left[\sum_{t=1}^{T} \prod_{j=1}^{T} \frac{\pi_\theta(a_j|s_j)}{\pi_{sim}(a_j|s_j)} \nabla_\theta \log \pi_\theta(a_t|s_t) \sum_{t'=1}^{T} r(s_{t'}, a_{t'}) \right] \quad (4)$$

We empirically verify that importance sampling helps in improving the agent's performance.

3.4 Tree-based Value Approximation

We use Monte-Carlo sampling to compute the total reward used in REINFORCE's objective function in equation 1. However, even if we start from the same state, the sampled trajectories can lead to different rewards due to stochasticity in the policy. Consequently, the variance in estimating the total reward is high and can adversely affect policy optimisation.

Previous works[23, 24, 37, 41] have used causality and the Markov property of these problems to discover tricks to reduce the variance of the policy gradient while keeping the estimation unbiased. Firstly, we can replace the total rewards in equation 4 with the sum of rewards obtained from state at timestep t onwards as the action at timestep t cannot affect the rewards obtained prior to timestep t. Similarly, the importance sampling ratio at timestep t is not affected by future actions. Hence, we can rewrite equation 4 as follows:

$$\begin{aligned}\nabla_\theta J(\tau) &= \left[\sum_{t=1}^{T} \prod_{j=1}^{t} \frac{\pi_\theta(a_j|s_j)}{\pi_{sim}(a_j|s_j)} \nabla_\theta \log \pi_\theta(a_t|s_t) \sum_{t'=t}^{T} r(s_{t'}, a_{t'}) \right] \\ &= \left[\sum_{t=1}^{T} w_t \nabla_\theta \log \pi_\theta(a_t|s_t)\ \hat{Q}_t \right]\end{aligned} \quad (5)$$

where $w_t = \prod_{j=1}^{t} \frac{\pi_\theta(a_j|s_j)}{\pi_{sim}(a_j|s_j)}$ and $\hat{Q}_t = \sum_{t'=t}^{T} r(s_{t'}, a_{t'})$ be the observed Q-value at timestep t.

Secondly, we can subtract a baseline term from the observed Q-values [24]. A popular baseline is the value predictions of the states observed in a trajectory. If we subtract the value of a state from the observed Q-value, we get the advantage value of taking an action over its expected value. The advantage value has lower variance while being unbiased and works well in practice [24]. We can rewrite the equation 5 as follows:

$$\nabla_\theta J_\theta(\tau) = \left[\sum_{t=1}^{T} w_t \nabla_\theta \log \pi_\theta(a_t|s_t) \Big(\hat{Q}_t - V(s_t) \Big) \right] \quad (6)$$

On the other hand, a tree structure can store state statistics that allow us to efficiently maintain an estimate of the values of the observed states by backing up the observed value from a trajectory. The tree structure also enables us to recursively apply the Bellman equation on all the states in the trajectory as follows:

$$V_{tree}(s) = \max_a \left[r(s,a) + \gamma \sum_{s'} P(s'|s,a) V_{tree}(s') \right]$$

where

- s' is the next state observed after taking action a in state s.
- $V(s)$ is the value of state s.
- $r(s,a)$ is the reward obtained after taking action a in state s.

Finally, we can write the combined policy gradient update as follows:

$$\theta \leftarrow \theta + \alpha \left[\sum_{t=1}^{T} w_t \nabla_\theta \log \pi_\theta(a_t|s_t) \Big(\hat{Q}_t - V_{tree}(s_t) \Big) \right] \tag{7}$$

In policy gradient-based search methods, it is a common practice to freeze all the parameters except the parameters of the last network layer to speed up the search and avoid slowdowns due to excessive gradient computation [1]. We follow the same process in ExPoSe and calculate the gradients for the parameters of the last network layer only.

4 EXPERIMENTS

4.1 Experiment Setup

For our experiments, we set a maximum number of search iterations per step for each method to limit the search budget. All the online search methods use the same policy and value function approximators for each problem to enable comparison of policy improvement due to the strengths of each search method. We measure the success rate to evaluate agent performance in goal-based planning problems, i.e., the fraction of instances where the agent reaches the goal state. In contrast, we use Human Normalised Score (HNS) and Baseline Normalised Score (BNS) to evaluate agent performance on Atari games.

Each online search method has its set of adjustable hyperparameters. We use 10% of the testing data as the validation set and the remaining 90% as the holdout test set to report thefi nal evaluation score. We select the best set of hyperparameters using grid search on a log-linear scale with the validation set. Finally, we use the best set of hyperparameters for each method to evaluate the agent's performance on the holdout test set.

4.2 Baselines

We use popular tree-based and gradient-based online search algorithms, PUCT and PGS, as primary baselines for comparison. Below are the specific implementation details for each method:

- **PUCT**: We implement the standard version of PUCT as described in Section 2.2.
- **PGS**: We implement the standard version of PGS as described in Section 2.3. Additionally, we add entropy and L2 regularisation to the policy to prevent it from converging to a deterministic policy. Entropy regularisation also induces exploration by increasing the randomness in the policy [20].
- **ExPoSe**: We implement ExPoSe as described in Section 3. We also add entropy and L2 regularisation as described for PGS above.

4.3 Domains

We evaluate ExPoSe and other online search methods on two types of decision-making problems:

- A set of goal-based planning problems that includes Sokoban, Hamiltonian cycle search in sparse graphs and 2D grid navigation.
- Atari 2600 benchmarking suite, which is a set of image-based games.

We describe each domain in brief as follows:

4.3.1 *Sokoban.*

Sokoban is a classic puzzle game where an agent must move boxes to designated goal positions without hitting the walls. Because actions have irreversible consequences, Sokoban poses a complex planning problem. Sokoban has been widely used for experimentation in recent research [14, 16].

To ensure standardised comparison across methods, we use the Boxoban dataset [15] as described in [14]. The dataset consists of Sokoban instances from three difficulty levels: hard, medium, and unfiltered. Rather than using a reinforcement learning algorithm, such as A3C [24], to train both the policy and value function approximators (as done in [14]), we use Expert Iteration (ExIt) [2] to train the policy and value function due to its more stable training regime. The ExIt algorithm iteratively executes two processes: (1) using an expert policy to collect better data and (2) using the collected data to train the policy and value function. We use PUCT with a high number of search iterations combined with a partially trained policy and value function as the expert policy. We use training samples from the unfiltered and medium problem sets to collect expert data. Rather than starting from random weights, we pre-train the policy and value function approximators on data collected using A^* search. To compare different search methods, we focus on the Boxoban hard test set, as almost all test instances in the unfiltered and medium test sets are solvable by all search methods.

4.3.2 *Hamiltonian Cycle.*

A Hamiltonian cycle is a path in a graph that visits each node exactly once and returns to its starting node. Finding a Hamiltonian cycle in a sparse graph is a challenging planning problem because choosing the wrong action at the beginning can have long-term consequences, leading to the algorithm getting stuck with no node to visit next. Therefore, it is crucial to anticipate the future effects of actions and create a plan that can ultimately complete a Hamiltonian cycle.

To generate expert data for this problem, we randomly permute the nodes, connect them to create a cycle, and add random edges to create a sparse graph with a known Hamiltonian cycle. We create a training dataset of 10,000 graphs with 50 nodes and 50% sparsity. To make the problem harder for the test set, we generate graphs with 50 nodes and 10% sparsity.

4.3.3 *Grid Navigation.*

Grid navigation is a task that requires an agent to navigate a 2D grid to reach a goal position while avoiding obstacles. The agent has access to information about its current and goal positions, as well as an environment map that indicates the locations of obstacles. The main challenge is tofi nd a path from the current position to the goal that may require taking detours due to obstacles blocking the direct path.

We conduct experiments on a 2D grid of size 50×50. To create a training dataset, we generate 100,000 random maps where obstacles occupy each cell with a probability of 0.25. During map generation, we randomly select the agent's starting and goal positions using rejection sampling such that the shortest distance between the starting and goal positions is more than 100 units, and their Manhattan distance is more than 50 units. This ensures that

Table 1: Comparison of test performance on Sokoban (Boxoban hard set) measured in success rate (i.e. % of test instances solved) for different search methods ["#iterations" stands for number of search iterations]

Search Method	#iterations=10	#iterations=50	#iterations=100
Prior Policy	60.39 ± 0.9	–	–
PUCT (AlphaGo)	91.92 ± 0.5	94.62 ± 0.4	95.25 ± 0.4
PGS (with entropy)	91.26 ± 0.5	94.85 ± 0.4	95.58 ± 0.4
ExPoSe	**94.92** ± 0.4	**97.16** ± 0.3	**97.39** ± 0.3

Table 2: Comparison of test performance on Hamiltonian cycle search (nodes = 50, sparsity = 10%) measured in success rate (i.e. % of test instances solved) for different search methods ["#iterations" stands for number of search iterations]

Search Method	#iterations=10	#iterations=50	#iterations=100
Prior Policy	01.31 ± 0.2	–	–
PUCT (AlphaGo)	68.76 ± 0.7	79.67 ± 0.6	83.53 ± 0.6
PGS (with entropy)	70.62 ± 0.7	91.47 ± 0.4	95.27 ± 0.3
ExPoSe	**71.27** ± 0.7	**93.40** ± 0.4	**97.09** ± 0.3

the generated levels include a detour from the direct path. Additionally, we create a test set of 5,000 instances with even more challenging and denser maps by increasing the probability of an obstacle occupying a cell to 0.50. The shortest path to the goal in these instances involves significant detours. Since the state space is small, any shortest path algorithm can act as an expert policy to generate the training data. We use this toy problem primarily to visualise the state space explored by different search methods (refer to Figure 3 in the appendix).

4.3.4 Atari 2600 Benchmarking Suite.
Atari 2600 is a suite of image-based games commonly used to evaluate agents' policies [24, 25]. In this suite, the agent receives an image as an observation from the simulator, and the goal is to score as high as possible before the episode ends. To represent the state, popular methods stack a sequence of images.

We use a set of 50 Atari 2600 games (as mentioned in [5]) to evaluate the online search methods. To do this, we use a prior policy and value functions for each game, trained through A3C [24], and provided by PFRL [12]. For evaluation, we apply Atari wrappers and agents from PFRL, and each online search method uses the same prior policy and value networks. To measure the agents' performance, we use Human Normalised Score (HNS) (Eq. 8) and Baseline Normalised Score (BNS) (Eq. 9), averaged over all the games.

$$HNS = \frac{S_\pi - S_R}{S_H - S_R} \tag{8}$$

$$BNS = \frac{S_\pi - S_R}{S_B - S_R} \tag{9}$$

where S_π, S_R, S_H, S_B represents the score achieved by the agent when following the online search policy π, a random policy, the human expert policy and the baseline policy, respectively.

We limit each search method to a maximum of 10 iterations during each prediction step. To speed up evaluation, we simulate a trajectory of 20 steps and use the value predicted by the value network at the last step as a bootstrap. In Table 4, we report both the mean and median scores achieved by the agent. The appendix provides a detailed list of scores achieved by the agent on each of the 50 games in the Atari test suite.

4.4 Results

We present our experimental results in Table 1, Table 2, Table 3, and Table 4 for Sokoban, Hamiltonian cycle search, grid navigation, and Atari 2600 games respectively. Wefi nd that ExPoSe consistently outperforms all baselines across a diverse set of testing domains. We also observe that all online search methods improve the performance of the prior policy in goal-based problems, and ExPoSe outperforms PUCT and PGS given the same search budget. Moreover, ExPoSe exhibits a higher margin of improvement over baselines when we allow a small search budget. Our results also indicate that ExPoSe performs better than PGS in 38 out of 50 Atari games, with better scores on HNS and BNS averaged over all the games (see Table 4).

4.4.1 Does information sharing among states help in online search?
Methods that directly update the information into the parameters of the policy, i.e. PGS and ExPoSe, outperform PUCT, which does not share information across the search tree. Thus, we can empirically conclude that information sharing across the states helps improve the policy obtained using an online search.

4.4.2 Does state-based exploration help improve the performance of gradient-based online search?
ExPoSe, which combines state-based exploration with gradient-based online search, outperforms PGS with entropy regularisation across all domains. This further supports the importance of a well-defined state-based exploration mechanism in gradient-based online search methods.

4.5 Ablation Studies

We conduct an ablation study on Sokoban to evaluate the contribution of the improvements described in Section 3.3 and 3.4. To

Table 3: Comparison of test performance on grid navigation (hard set) measured in success rate (i.e. % of test instances solved) for different methods ["#iterations" stands for number of search iterations]

Search Method	#iterations=10	#iterations=50	#iterations=100
Prior Policy	81.18 ± 0.6	–	–
PUCT (AlphaGo)	92.38 ± 0.4	93.44 ± 0.4	94.02 ± 0.4
PGS (with entropy)	94.67 ± 0.3	97.78 ± 0.2	98.91 ± 0.2
ExPoSe	**99.38** ± 0.1	**99.93** ± 0.1	**99.96** ± 0.1

Table 4: Comparison of test performance on a set of 50 Atari game measured in Human Normalised Score (HNS) and Baseline Normalised Score (BNS) averaged over all the games. We use prior policy as baseline to compute Baseline Normalised Score.

Search Method	Human Normalised Score Mean(Median)	Baseline Normalised Score Mean(Median)
Prior Policy	3.32 (0.63)	1.00 (1.00)
PGS (with entropy)	6.79 (0.86)	1.22 (1.27)
ExPoSe	**7.15 (0.96)**	**1.32 (1.38)**

perform the comparison, we modify the original ExPoSe implementation and create the following implementations:

- **ExPoSe without Importance Sampling**: We remove the importance sampling ratio term while computing the policy gradients. The corresponding parameter update is given by:

$$\theta \leftarrow \theta + \alpha \left[\sum_{t=1}^{T} \nabla_\theta \log \pi_\theta(a_t|s_t) \Big(\hat{Q}_t - V_{tree}(s_t) \Big) \right]$$

- **ExPoSe without Value Baseline**: We exclude any baseline term while computing the policy gradients. The corresponding parameter update is given by:

$$\theta \leftarrow \theta + \alpha \left[\sum_{t=1}^{T} w_t \nabla_\theta \log \pi_\theta(a_t|s_t)\, \hat{Q}_t \right]$$

- **ExPoSe with Value Network Baseline**: We replace the tree-based value estimation term with an offline learnt value function as the baseline while computing the policy gradients. The corresponding parameter update is then given by:

$$\theta \leftarrow \theta + \alpha \left[\sum_{t=1}^{T} w_t \nabla_\theta \log \pi_\theta(a_t|s_t) \Big(\hat{Q}_t - V_{net}(s_t) \Big) \right]$$

We present the results of our ablation study in Table 5, which allows us to address the following questions:

4.5.1 Does importance sampling help improve the computation of the policy gradients?
In Table 5, we compare the performance of the original *ExPoSe* with that of *ExPoSe without importance sampling*. The results demonstrate that the original *ExPoSe* consistently outperforms the *ExPoSe without importance sampling*, providing empirical evidence for the effectiveness of importance sampling in computing policy gradients.

4.5.2 Does using state values as a baseline help reduce the variance and improve the agent's performance?
Table 5 shows that the *ExPoSe* variant without a value baseline consistently performs worse than the original *ExPoSe* and the modified version with a value network baseline. These results empirically demonstrate that incorporating a value baseline term in policy gradient computation helps reduce variance and improve agent performance.

4.5.3 Does tree-based value estimation perform better than an offline learnt value network?
Table 5 shows that the *ExPoSe* algorithm with tree-based value estimation consistently outperforms the *ExPoSe with value network baseline* by a small margin. These results suggest that incorporating a tree-based value estimation mechanism can help improve the agent's performance when compared to using an offline-learned value network as a baseline.

5 RELATED WORKS

Decision-making problems with smaller state spaces can be solved exactly using the value iteration algorithm [37]. However, applying value iteration on problems with massive search space, such as chess and go, is intractable. Instead, we can use a parameterised policy or value function for these larger decision-making problems. These parameterised models can be learnt using reinforcement learning algorithms like Q-learning or policy gradients [37]. More recent works have tried to take advantage of neural networks as function approximators [17, 24, 25, 31, 32] and have reported great success on problems like atari, OpenAI gym [8], or Mujoco simulator [40].

Alternatively, we can use an online search to select an action for the agent's current state. Methods like UCT [19] iteratively expand a search tree using a black-box simulator and output an action. These methods can massively improve their performance when coupled with well-defined heuristics [9]. The importance of exploration in decision making problems has also been studied

Table 5: Comparison of test performance on Sokoban for the ablation study to analyse the contribution of each improvement described in ExPoSe. We modify the original ExPoSe implementation to create other methods for comparison ["#iterations" stands for number of search iterations].

Search Method	#iterations=10	#iterations=50	#iterations=100
ExPoSe	94.92 ± 0.4	97.16 ± 0.3	97.39 ± 0.3
ExPoSe *without Importance Sampling*	94.00 ± 0.4	96.30 ± 0.3	97.16 ± 0.3
ExPoSe *without Value Baseline*	93.14 ± 0.5	95.61 ± 0.4	96.41 ± 0.3
ExPoSe *with Value Network Baseline*	94.36 ± 0.4	96.41 ± 0.3	96.70 ± 0.3

for both reinforcement learning algorithms [3, 4, 11, 26] as well as online search algorithms [19, 29].

Many recent works have tried to combine these two decoupled approaches to leverage their capabilities [2, 6, 35, 39]. For example, AlphaGo [35] demonstrated that by combining an online tree search method with an offline learnt policy and value function, an agent could beat the world champion in the game of Go, which was seen as a complex challenge for an agent. Following it, AlphaZero [36] showed that combining online tree search with learnt policy and value function can beat any other agent in chess, shogi and go without prior human knowledge by following a simple training regime defined in [36].

Within this research direction, some proposed works focus on learning the environment model and using it for simulation [28, 34]. For example, Predictron [34] learns an abstract environment model to simulate a trajectory and accumulate internal rewards. On the other hand, Value Iteration Network [38] and Gated Path Planning Network [22] learn an environment model in the context of a planning algorithm by embedding the algorithmic structure of the value iteration algorithm in the neural network architecture. Furthermore, ATreeQN [10] learns an environment model in the context of a tree structure and uses it to predict the Q-values. These algorithms jointly optimise the environment dynamics with the policy or value function approximator. Further, Guez et al. [14] argue that a recurrent neural network could exhibit the properties of a planning algorithm without specifying an algorithmic structure in the network architecture.

Alternatively, MCTSnets [16] tries to learn a parameterised policy in the context of a planning algorithm. It mimics the structure of the MCTS algorithm and learns to guide the search using parameterised memory embeddings stored in a tree structure. Similarly, Pascanu et al. [27] also learn to plan using a neural network model, but it uses an unstructured memory representation. MuZero [30] learns the environment model and uses it with an online tree search algorithm like UCT.

In an alternative research direction, some online search methods try to adapt a parameterised policy or value function learnt offline on a large dataset by simulating trajectories from the input state and using model-free RL methods to optimise the parameters of the policy or the value function [1, 7, 33]. For example, Policy Gradient Search [1] follows a Monte-Carlo Search framework while iteratively adapting the simulation policy using policy gradients. Further, entropy regularisation can be used while computing the gradients to prevent the issues of early commitment and initialisation bias [20]. Policy gradient methods can also be used to improve the rollout policy, which can further improve the performance of Monte-Carlo Tree Search [13].

6 CONCLUSION

In this paper, we identify and empirically analyse the key reasons behind the success of tree-based and gradient-based online search methods. Firstly, wefi nd that information sharing across the state space during the online search helps in improving the agent's performance, and Policy Gradient Search provides a practical algorithm to achieve this by directly updating the parameters of the search policy. Secondly, we determine that an explicit exploration mechanism is essential for efficiently balancing the exploration-exploitation trade-off and enabling the online search to escape from a local optimum. While tree-based methods have a well-defined exploration mechanism by design, Policy Gradient Search relies on entropy regularisation to induce exploration by increasing randomness in the policy.

To address these issues, we propose an efficient and effective online search method called Exploratory Policy Gradient Search (ExPoSe), which combines gradient-based policy improvement with state-based exploration. ExPoSe iteratively simulates trajectories using a simulation policy that incorporates an exploration term depending on the state-action visitation count, and updates the parameters of the prior policy using REINFORCE. Furthermore, we discuss the issue of optimising the prior policy using the trajectories generated by the exploration-induced simulation policy naively, which could limit the agent's performance gain due to the off-policy data generation. ExPoSe resolves this problem by using importance sampling to obtain an unbiased estimate of the policy gradients. Additionally, we highlight the benefits of using a tree structure to maintain the value estimates of the states encountered during the online search, which can further reduce the variance of the policy gradients.

We conduct experiments on a diverse set of decision-making problems, including goal-based planning problems like Sokoban, Hamiltonian cycle search in sparse graphs and grid navigation, and image-based games such as Atari 2600. Our experimental results demonstrate that ExPoSe outperforms other online search methods consistently across all test domains. However, ExPoSe, like many other search algorithms, requires access to the environment simulator for the search, which may be infeasible for some problems. Integrating a learned world model into the algorithm could be a potential solution to this limitation, and future work could explore its effectiveness in improving the applicability and robustness of ExPoSe.

ACKNOWLEDGEMENT

This research is supported by the National Research Foundation Singapore and DSO National Laboratories under the AI Singapore Programme (Award Number: AISG2-RP-2020-016).

We would like to acknowledge the usage of ChatGPT for helping in correcting the grammar in this paper.

REFERENCES

[1] Thomas Anthony, Robert Nishihara, Philipp Moritz, Tim Salimans, and John Schulman. 2019. Policy gradient search: Online planning and expert iteration without search trees. *arXiv preprint arXiv:1904.03646* (2019).

[2] Thomas Anthony, Zheng Tian, and David Barber. 2017. Thinking fast and slow with deep learning and tree search. In *Proceedings of the 31st International Conference on Neural Information Processing Systems*. 5366–5376.

[3] Siddharth Aravindan and Wee Sun Lee. 2021. State-Aware Variational Thompson Sampling for Deep Q-Networks. In *Proceedings of the 20th International Conference on Autonomous Agents and MultiAgent Systems*. 124–132.

[4] Peter Auer, Nicolo Cesa-Bianchi, and Paul Fischer. 2002. Finite-time analysis of the multiarmed bandit problem. *Machine learning* 47, 2 (2002), 235–256.

[5] Adrià Puigdomènech Badia, Bilal Piot, Steven Kapturowski, Pablo Sprechmann, Alex Vitvitskyi, Zhaohan Daniel Guo, and Charles Blundell. 2020. Agent57: Outperforming the atari human benchmark. In *International Conference on Machine Learning*. PMLR, 507–517.

[6] Jonathan Baxter, Andrew Tridgell, and Lex Weaver. 1998. KnightCap: A Chess Programm That Learns by Combining TD (lambda) with Game-Tree Search. In *Proceedings of the Fifteenth International Conference on Machine Learning*. 28–36.

[7] Irwan Bello, Hieu Pham, Quoc V Le, Mohammad Norouzi, and Samy Bengio. 2016. Neural combinatorial optimization with reinforcement learning. *arXiv preprint arXiv:1611.09940* (2016).

[8] Greg Brockman, Vicki Cheung, Ludwig Pettersson, Jonas Schneider, John Schulman, Jie Tang, and Wojciech Zaremba. 2016. Openai gym. *arXiv preprint arXiv:1606.01540* (2016).

[9] Rémi Coulom. 2007. Computing "elo ratings" of move patterns in the game of go. *ICGA journal* 30, 4 (2007), 198–208.

[10] Gregory Farquhar, Tim Rocktäschel, Maximilian Igl, and Shimon Whiteson. 2018. TreeQN and ATreeC: Differentiable Tree-Structured Models for Deep Reinforcement Learning. In *International Conference on Learning Representations*.

[11] Meire Fortunato, Mohammad Gheshlaghi Azar, Bilal Piot, Jacob Menick, Matteo Hessel, Ian Osband, Alex Graves, Volodymyr Mnih, Remi Munos, Demis Hassabis, et al. 2018. Noisy Networks For Exploration. In *International Conference on Learning Representations*.

[12] Yasuhiro Fujita, Prabhat Nagarajan, Toshiki Kataoka, and Takahiro Ishikawa. 2021. ChainerRL: A Deep Reinforcement Learning Library. *Journal of Machine Learning Research* 22, 77 (2021), 1–14. http://jmlr.org/papers/v22/20-376.html

[13] Tobias Graf and Marco Platzner. 2015. Adaptive playouts in monte-carlo tree search with policy-gradient reinforcement learning. In *Advances in Computer Games*. Springer, 1–11.

[14] Arthur Guez, Mehdi Mirza, Karol Gregor, Rishabh Kabra, Sébastien Racanière, Théophane Weber, David Raposo, Adam Santoro, Laurent Orseau, Tom Eccles, et al. 2019. An investigation of model-free planning. In *International Conference on Machine Learning*. PMLR, 2464–2473.

[15] Arthur Guez, Mehdi Mirza, Karol Gregor, Rishabh Kabra, Sebastien Racaniere, Theophane Weber, David Raposo, Adam Santoro, Laurent Orseau, Tom Eccles, Greg Wayne, David Silver, Timothy Lillicrap, and Victor Valdes. 2018. An investigation of Model-free planning: boxoban levels. https://github.com/deepmind/boxoban-levels/.

[16] Arthur Guez, Théophane Weber, Ioannis Antonoglou, Karen Simonyan, Oriol Vinyals, Daan Wierstra, Rémi Munos, and David Silver. 2018. Learning to search with MCTSnets. In *International Conference on Machine Learning*. PMLR, 1822–1831.

[17] Matteo Hessel, Joseph Modayil, Hado Van Hasselt, Tom Schaul, Georg Ostrovski, Will Dabney, Dan Horgan, Bilal Piot, Mohammad Azar, and David Silver. 2018. Rainbow: Combining improvements in deep reinforcement learning. In *Thirty-second AAAI conference on artificial intelligence*.

[18] Yujiao Hu, Yuan Yao, and Wee Sun Lee. 2020. A reinforcement learning approach for optimizing multiple traveling salesman problems over graphs. *Knowledge-Based Systems* 204 (2020), 106244.

[19] Levente Kocsis and Csaba Szepesvári. 2006. Bandit based monte-carlo planning. In *European conference on machine learning*. Springer, 282–293.

[20] Mikel Landajuela Larma, Brenden K Petersen, Soo K Kim, Claudio P Santiago, Ruben Glatt, T Nathan Mundhenk, Jacob F Pettit, and Daniel M Faissol. 2021. Improving exploration in policy gradient search: Application to symbolic optimization. *arXiv preprint arXiv:2107.09158* (2021).

[21] Eugene L Lawler and David E Wood. 1966. Branch-and-bound methods: A survey. *Operations research* 14, 4 (1966), 699–719.

[22] Lisa Lee, Emilio Parisotto, Devendra Singh Chaplot, Eric Xing, and Ruslan Salakhutdinov. 2018. Gated path planning networks. In *International Conference on Machine Learning*. PMLR, 2947–2955.

[23] Sergey Levine and Vladlen Koltun. 2013. Guided policy search. In *International conference on machine learning*. PMLR, 1–9.

[24] Volodymyr Mnih, Adria Puigdomenech Badia, Mehdi Mirza, Alex Graves, Timothy Lillicrap, Tim Harley, David Silver, and Koray Kavukcuoglu. 2016. Asynchronous methods for deep reinforcement learning. In *International conference on machine learning*. PMLR, 1928–1937.

[25] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin A. Riedmiller. 2013. Playing Atari with Deep Reinforcement Learning. *ArXiv* abs/1312.5602 (2013).

[26] Ian Osband, Charles Blundell, Alexander Pritzel, and Benjamin Van Roy. 2016. Deep exploration via bootstrapped DQN. *Advances in neural information processing systems* 29 (2016), 4026–4034.

[27] Razvan Pascanu, Yujia Li, Oriol Vinyals, Nicolas Heess, Lars Buesing, Sebastien Racanière, David Reichert, Théophane Weber, Daan Wierstra, and Peter Battaglia. 2017. Learning model-based planning from scratch. *arXiv preprint arXiv:1707.06170* (2017).

[28] Sébastien Racanière, Théophane Weber, David P Reichert, Lars Buesing, Arthur Guez, Danilo Rezende, Adria Puigdomenech Badia, Oriol Vinyals, Nicolas Heess, Yujia Li, et al. 2017. Imagination-augmented agents for deep reinforcement learning. In *Proceedings of the 31st International Conference on Neural Information Processing Systems*. 5694–5705.

[29] Christopher D Rosin. 2011. Multi-armed bandits with episode context. *Annals of Mathematics and Artificial Intelligence* 61, 3 (2011), 203–230.

[30] Julian Schrittwieser, Ioannis Antonoglou, Thomas Hubert, Karen Simonyan, Laurent Sifre, Simon Schmitt, Arthur Guez, Edward Lockhart, Demis Hassabis, Thore Graepel, et al. 2020. Mastering atari, go, chess and shogi by planning with a learned model. *Nature* 588, 7839 (2020), 604–609.

[31] John Schulman, Sergey Levine, Pieter Abbeel, Michael Jordan, and Philipp Moritz. 2015. Trust region policy optimization. In *International conference on machine learning*. PMLR, 1889–1897.

[32] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. 2017. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347* (2017).

[33] David Silver. 2009. *Reinforcement Learning and Simulation-Based Search*. Ph.D. Dissertation. Ph. D. thesis, University of Alberta.

[34] David Silver, Hado Hasselt, Matteo Hessel, Tom Schaul, Arthur Guez, Tim Harley, Gabriel Dulac-Arnold, David Reichert, Neil Rabinowitz, Andre Barreto, et al. 2017. The predictron: End-to-end learning and planning. In *International Conference on Machine Learning*. PMLR, 3191–3199.

[35] David Silver, Aja Huang, Chris J Maddison, Arthur Guez, Laurent Sifre, George Van Den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, Marc Lanctot, et al. 2016. Mastering the game of Go with deep neural networks and tree search. *nature* 529, 7587 (2016), 484–489.

[36] David Silver, Julian Schrittwieser, Karen Simonyan, Ioannis Antonoglou, Aja Huang, Arthur Guez, Thomas Hubert, Lucas Baker, Matthew Lai, Adrian Bolton, et al. 2017. Mastering the game of go without human knowledge. *nature* 550, 7676 (2017), 354–359.

[37] Richard S Sutton and Andrew G Barto. 2018. *Reinforcement learning: An introduction*. MIT press.

[38] Aviv Tamar, Yi Wu, Garrett Thomas, Sergey Levine, and Pieter Abbeel. 2016. Value iteration networks. In *Proceedings of the 30th International Conference on Neural Information Processing Systems*. 2154–2162.

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

[40] Emanuel Todorov, Tom Erez, and Yuval Tassa. 2012. MuJoCo: A physics engine for model-based control. In *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*. 5026–5033. https://doi.org/10.1109/IROS.2012.6386109

[41] Ronald J Williams. 1992. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine learning* 8, 3 (1992), 229–256.