Preference learning for guiding the tree search in continuous POMDPs

Anonymous Author(s)
Affiliation
Address
email

Abstract: A robot operating in a partially observable environment must perform sensing actions to achieve a goal, such as clearing the objects in front of a shelf to better localize a target object at the back, and estimate its shape for grasping. A POMDP is a principled framework for enabling robots to perform such information-gathering actions. Unfortunately, while robot manipulation domains involve high-dimensional and continuous observation and action spaces, most POMDP solvers are limited to discrete spaces. Recently, POMCPOW [1] has been proposed for continuous POMDPs, which handles continuity using sampling and progressive widening [2]. However, for robot manipulation problems involving camera observations and multiple objects, POMCPOW is too slow to be practical. We take inspiration from the recent work in learning to guide task and motion planning [3] to propose a framework that learns to guide POMCPOW from past planning experience. Our method uses preference learning [4, 5, 6, 7] that utilizes both success and failure trajectories, where the preference label is given by the results of the tree search. We demonstrate the efficacy of our framework in several continuous partially observable robotics domains, including real-world manipulation, where our framework explicitly reasons about the uncertainty in off-the-shelf segmentation and pose estimation algorithms. In our supplementary video, we show that our algorithm finds the optimal solution while others cannot.

Keywords: POMDP, Online planning, Guided Search, Preference-based learning

1 Introduction

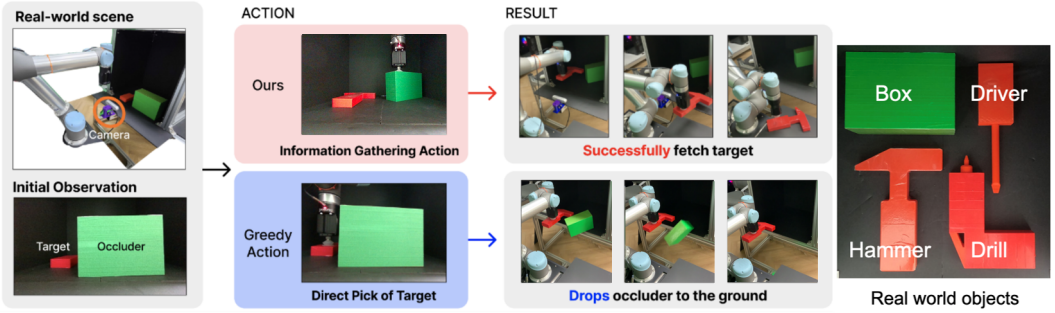

Figure 1: An object fetching task from a tight cabinet. The camera is at a location marked with orange, and the initial observation is shown at the bottom left. The target can be any one of the red objects from the right-most figure, and the occluder is the green box. The occluder prevents accurately estimating the target object's shape and pose. Only by performing information-gathering actions can we safely fetch the target object.

Consider a robot manipulator tasked with retrieving a target object occluded by another object in a tight cabinet. Due to significant occlusion, the exact shape and location of the target object are

Submitted to the 7th Conference on Robot Learning (CoRL 2023). Do not distribute.

unknown, and rashly fetching the object without gathering these pieces of information could make the occluding object fall. Our goal is to enable robots to efficiently compute plans that consider these uncertainties and perform information-gathering actions to robustly operate in unstructured environments. This is a challenging problem that involves partial observability in high dimensional continuous state, action, and observation spaces. Figure 1 demonstrates an example scenario.

Typically, such problems are modeled with POMDPs, but the primary difficulty in using POMDP solvers is their computational complexity. To resolve this, several sampling-based online planning algorithms, such as POMCP [8, 1], have been proposed, which effectively handle large-scale discrete POMDPs by using Monte Carlo Tree Search (MCTS). Instead of computing the solutions for *all* belief states [9, 10, 11, 12, 13, 14, 15], POMCP performs a tree search to compute the solution for a single initial belief state. A recent extension, Partially Observable Monte Carlo Planning with Observation Widening (POMCPOW) [1], extends POMCP to continuous observation and action spaces using progressive widening [2] in both observation and action nodes. With this extension, POMCPOW can readily be applied to robotics problems involving continuous spaces.

However, even with POMCPOW, the computational challenge stemming from continuous and high-dimensional observations, actions, and states is still significant. In [3], the authors propose a method for learning to *guide* planning for challenging long-horizon task and motion planning problems in continuous spaces. The idea is to learn the value function and policy from past planning experience in order to speed up the tree search in a similar fashion to AlphaGo [16]. However, these methods are limited to fully observability environments and cannot handle state uncertainty.

In this work, we extend [3] to a partially observable setup. Like [3], we learn a value function and policy for guiding a tree search. However, unlike in that work, where the value function and policy operate on fully observable states, ours operate on action-observation histories and guides POMCPOW. In this context, the simplest way to train a value function would be to gather all history and future-sum-of-reward pairs from past search trees, and learn a mapping from history to its value. However, this approach would require an unthinkably large amount of data to train an effective value function due to the infinite possible histories containing high dimensional observations. As generating a dataset involves performing a tree search in a POMDP, such large-scale data generation would require a significant amount of time.

We propose an alternative data-efficient technique for learning a value function based on the following two observations. First, a search tree for a POMDP typically consists of a few success histories that led to a goal and a large number of other histories that did not. Second, all we need to efficiently guide a tree search is ranking among the histories specifying which one is more likely to lead to the goal, not their actual values, since the purpose of a value function in a tree search is to determine which node to explore next. Based on these two observations, we propose a value function learning algorithm that learns the ranking among histories.

More concretely, we use preference-based learning. In preference-based *reward* learning [4, 7, 5, 6], you are given two trajectories, and an oracle tells you which one of the two trajectories is more preferred. You then train a reward function that rewards the preferred trajectory more than the other trajectory. We adapt this idea to value function learning, where the preference label comes from a result of a tree search. Given a search tree, we take a history that got to a goal, called a *success history*, and one that did not, called a *failure history*, from a search tree and learn a value function such that it prefers the success history over the failure history. One issue with this simple success-and-fail preference labeling scheme might be that it does not have the notion of optimality. To incorporate that, we create additional data by pairing two successful histories and then labeling the one that is closer to the goal as more preferred. Because this method uses a *pairing* of histories, it can create much larger data from a single search tree than the one that regresses the value from history, shortening the data generation time. See Figure 2 for an illustration of our labeling scheme.

For training a policy, we could, in principle, imitate the actions on success histories from past planning experiences using regression. However, to facilitate efficient exploration during tree search,

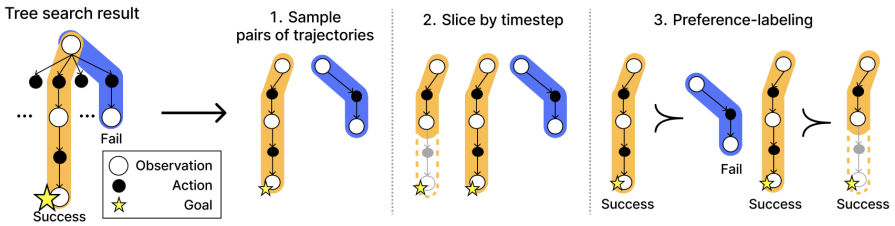

Figure 2: **Preference-dataset generation.** Given the tree search result, we first sample a pair of trajectories, at least one of which is a success trajectory. We then slice a subset of them at a particular time step, as shown on the left-most trajectory in step 2, and then label them with preferences as follows: if the pair consists of success and failure trajectories, then we prefer the success trajectory (left in step 3). If the pair consists only of success trajectories, then we prefer the one that is closer to the goal (right in step 3).

we need a multi-modal policy rather than a single-modal policy [17]. Furthermore, if we imitate success histories, then we are limited to a small number of histories per search tree, which is data inefficient. We instead propose to use an energy-based model, where the energy is based on a Q-function, which effectively defines a multi-modal policy whose probability of action is proportional to its Q-value [17]. We train the Q-function simultaneously with the value function by implementing these two functions as two heads that share the same backbone transformer but with different inputs. Preference-V, the value function trained with preference labels, consumes the sequence of action-observation history up to time step t, while Preference-Q does not see the last observation. Figure 3 shows the architecture of the preference value functions.

The problem with an energy-based function in continuous space is that exact sampling is intractable. One way to get around this is to uniform-randomly sample several actions, evaluate their values, and then choose the one with the highest Q-value. However, while fast, this procedure generates poor-quality actions, especially in high-dimensional spaces. Alternatively, we can use MCMC sampling [18], which, given a sufficient amount of time, can generate actions with high Q-values. However, the inference time is typically too long to be practical. Instead, we propose to use VAE [19] to approximate the Preference-Q-based energy function. At every gradient step, we uniform-randomly sample N number of actions and minimize the Q-function-weighted loss for training the VAE, so that the higher Q-value, the higher the chance of being sampled from the VAE. This approach generates higher-quality samples than uniform random sampling because the VAE sees N number of new actions at every gradient step, and has a much faster inference time than MCMC sampling because the inference is just a simple feed-forward prediction using a VAE.

We call our framework PGP (Preference-Guided POMCPOW). In our experiments, we demonstrate that PGP achieves a superior data efficiency than imitation-guided POMCPOW, and that it is more computationally efficient than unguided POMCPOW. In a real-world robot experiment, we demonstrate a system that integrates the off-the-shelf perception algorithms, POMCPOW, and our learning framework to solve the object fetching problem shown in Figure 1.

2 Related work

Algorithms for POMDPs To combat the curse of history, several online POMDP algorithms have been proposed. Unlike the conventional methods that compute solutions off-line for all possible belief states using full-width backup algorithms such as value iteration [9, 10, 11, 12, 13, 14, 15], online planning algorithms use tree search to compute a solution for a single belief state [8, 20]. In particular, POMCP [8] combines MCTS and unweighted particle filtering to apply a sampling-based online planning algorithm to solve large-scale discrete POMDP. However, for POMDPs with continuous action and observation spaces, POMCP cannot build a search tree for more than one step because, for each new simulation, a new action and observation branch is expanded every time, as the probability of sampling the same action and observation from a continuous space is zero.

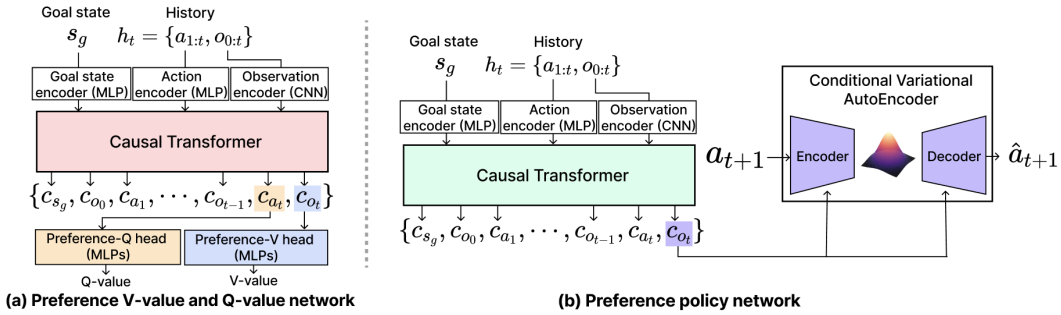

Figure 3: **The network architectures** (a) The value networks receive inputs of different types: goal state s_g, action a_t, and observation o_t. The encoders encode them into vectors of the same dimension. The causal transformer turns these encoded vectors into context vectors, $\{c_{s_g}, c_{a_{1:t}}, c_{o_{0:t}}\}$, where c_{o_t} depends on h_t, while c_{a_t} depends on $h_{t-1}a_t$, without o_t. The preference-Q head receives c_{a_t} as input, and the preference-V head receives c_{o_t}. (b) The policy network has a similar architecture as the value networks, but uses a CVAE [19] that conditions on c_{o_t}.

POMCPOW [1] extends POMCP to continuous spaces by using progressive widening (PW) [2] which samples a new branch only if the number of branches in that node is below $k \cdot M^{\alpha}$, where $k > 0$ and $\alpha \in (0, 1)$ are user-defined hyper-parameter and M is the number of visits to that node. Otherwise, it revisits existing branches in order to improve their value estimates. POMCPOW uses PW both at action and observation nodes and integrates it with UCT [21] to handle the exploration-exploitation trade-offs in continuous spaces. The use of PW in POMCPOW specifies *when* we should sample a new action but does not specify *which* new action we should sample next. Further, it uses a random roll-out to estimate the value of a leaf node, which only gives a very crude estimate. PGP uses a policy for sampling a new action and value function to replace the roll-out in POMCPOW.

Learning to guide tree search The most prominent system for guiding a search is AlphaGo [22, 16, 23, 24]. AlphaGo guides the MCTS using a policy network that recommends actions and a value network that predicts the probability of winning at a given state, both of which are learned offline using self-play. Like AlphaGo, we learn a value function and policy to guide the search. One key difference is that for us, we learn from *planning experience*, while AlphaGo learns from *execution experience*, which is more expensive to obtain. Another key difference is that AlphaGo is limited to discrete action and state spaces, with full state information, whereas we are considering continuous state, observation, and action spaces with partial observability.

The body of work most closely related to our framework includes algorithms for learning to guide task and motion planning based on planning experience [3, 25, 26, 27, 28, 29]. In [28, 29, 26], the authors propose a method for learning a visual heuristic function to guide the discrete task level search. In [25, 27], the authors propose strategies for learning a policy using variants of Generative Adversarial Networks to guide the continuous motion-level search. In [3], the authors integrate both the policy and the value function learned from planning experience with a tree search to solve long-horizon manipulation problems. Like us, they use planning experience to learn to guide the search in continuous state and action spaces. PGP is an extension of these works to a partially observable setup with state uncertainty.

Learning from preference Various works utilize *preference value* from humans to train agents, by ranking between pairs of trajectory segments [7]. For instance, recent Large Language Models (LLMs) are fine-tuned to use human preferences as a reward signal [4]. In the context of reinforcement learning, reward functions are modeled using a deep neural network that learns the trajectory preferences given by a human [5]. Inverse reinforcement learning also learns a reward function from a sequence of ranked demonstrations [6]. In PGP, we are already given a reward function, and the goal is to learn the value function that expresses the preference over different trajectories.

3 Learning the preference value and policy networks for PGP

Solving previous planning problems provides us with a set of search trees. There are largely three steps for using PGP: (1) generating the preference dataset from the search trees, (2) training the value and policy networks, and (3) deploying them to tree search. We provide the pseudocode for step (3) in the appendix. We first describe the data-generation process.

In POMCPOW, we simulate histories to find the solution to the given planning problem. So, solving a problem gives us a search tree containing a set of histories. For each problem, we assume we have a set of goal states, and each history is defined as $h_t = \{o_0, a_1, o_0, \cdots, a_t, o_t\}$ where, a_t and o_t are action and observation at time step t respectively. Suppose we are given a pair of histories $(h^i_{t^i}, h^j_{t^j})$, at least one of which is a success. Let G^i indicate whether the history $h^i_{t^i}$ is part of a success history $h^i_{T^i}$, where $T^i > t_i$, that got to a goal. We label the pair of history, $(h^i_{t^i}, h^j_{t^j})$, as

$$y(h^i_{t^i}, h^j_{t^j}) = \begin{cases} (1,0) & \text{if } G^i = 1 \text{ and } G^j = 0 \\ (0.5, 0.5) & \text{if } G^i = G^j = 1 \text{ and } T^i - t^i = T^j - t^j \\ (0,1) & \text{if } G^i = G^j = 1 \text{ and } T^i - t^i > T^j - t^j \end{cases}$$

In simple terms, if the ith history is part of a success history and the jth history is not, then we favor the ith history. If both are part of success histories and have an equal number of remaining time steps to the goal, they are both equally preferred. However, if both are success histories but the jth history is closer to the goal than the ith history, then we prefer the jth history. This preference labeling scheme operates under the assumption that all actions carry a uniform cost.

Algorithm 1 Inputs: $\tilde{Q}(h_t, a_{t+1}), P_{\text{U}}(a)$

Initialize θ
while θ not converged **do**
  Sample h_t from a set of past search trees
  for $k = 1, ..., K$ **do**
    Sample $a^k_{t+1} \sim P_{\text{U}}(a)$
    Compute $\tilde{Q}(h_t, a^k_{t+1})$
    $\hat{\pi}_{\tilde{Q}}(a^k_{t+1}|h_t) \leftarrow \frac{\exp(\tilde{Q}(h_t, a^k_{t+1}))}{\sum_{j=1}^{K} \exp(\tilde{Q}(h_t, a^j_{t+1}))}$
  $g_\theta \leftarrow \nabla_\theta \left[\sum_{k=1}^{K} \hat{\pi}_{\tilde{Q}}(a^k_{t+1}|h_t) \cdot \mathcal{L}_{\text{ELBO}}(\theta)\right]$
  Update θ with g_θ

We construct the preference dataset, $\mathcal{D}_{pref}$, in which each data point is of the form $(h^i, h^j, y(h^i, h^j))$. This is done by sampling two histories from a search tree, one of which must be a success history, and preference-labeling them. We ensure that $\mathcal{D}_{pref}$ contains an equal number of success-failure and success-success pairs to prevent the significant outnumbering of success-success pairs by success-failure pairs. This happens because the number of success trajectories gathered is greatly outnumbered by failed trajectories (approximately 1/10 in our experiments) especially when using unguided search. Therefore, if we form $\mathcal{D}_{pref}$ with all potential pairs of histories, we have empirically found that it impairs the learning of the preference between two successful histories.

We train Preference-V, denoted $\tilde{V}$, by setting it as a preference predictor, as in [7, 5, 6]. We approximate the probability of preferring history h^i over h^j as $P[h^i \succ h^j] = \frac{\exp(\tilde{V}(h^i))}{\exp(\tilde{V}(h^i)) + \exp(\tilde{V}(h^j))}$ and learn the function by minimizing the parameters of $\tilde{V}$ with respect to

$$\mathop{\mathbb{E}}_{(h^i, h^j, y) \sim \mathcal{D}_{pref}} [-y(h^i, h^j)[0] \cdot \log P[h^i \succ h^j] - y(h^i, h^j)[1] \cdot \log P[h^j \succ h^i]]$$

where $y(h^i, h^j)[i]$ denotes the ith entry in the preference label. We denote the preference-Q function as $\tilde{Q}$, and train $\tilde{V}$ and $\tilde{Q}$ concurrently by designing the neural network to share the same backbone, with separate heads that accept different inputs as shown in Figure 3.

Given $\tilde{Q}$, we would like to use the energy-based policy of the form $\pi_{\tilde{Q}}(a_{t+1}|h_t) = \frac{\exp(\beta \cdot \tilde{Q}(h_t, a_{t+1}))}{\int_{a'} \exp(\beta \cdot \tilde{Q}(h_t, a'))}$ to guide the search, where $\beta \in \mathbb{R}^+$ is a hyper-parameter that controls the how peaked the distribution is. To sample from this distribution, we must resort to either MCMC sampling or uniform-randomly sampling a fixed number of actions, and then selecting the one with the highest $\tilde{Q}$ value. The former approach is slow yet precise, while the latter is quick but less accurate.

Instead, our strategy is to use VAE to imitate $\pi_{\tilde{Q}}(a_{t+1}|h_t)$. Denote $\mathcal{L}_{\text{ELBO}}$ as the evidence of lower bound (ELBO) loss used to train a VAE whose parameters are denoted as θ. To train our VAE-based policy, denoted $\hat{\pi}_\theta$, it would be ideal to minimize $\mathbb{E}_{a\sim\pi_{\tilde{Q}}(a_{t+1}|h_t)}[\mathcal{L}_{\text{ELBO}}(\hat{\pi}_\theta)]$ with respect to θ. However, sampling from $\pi_{\tilde{Q}}$ is expensive, and the training time for $\hat{\pi}_\theta$ would be too long. So, we use importance sampling to train $\hat{\pi}_\theta$ by sampling actions from a uniform distribution, and evaluating the importance weight according to $\pi_{\tilde{Q}}$. Our objective function is $\underset{\theta}{\text{argmin}}\ \mathbb{E}_{a\sim P_{\text{U}}(a)}\left[\frac{\pi_{\tilde{Q}}(a|h_t)}{P_{\text{U}}(a)}\cdot\mathcal{L}_{\text{ELBO}}(\theta)\right]$, where P_U denotes the uniform distribution over the action space. Intuitively, if an action sampled from the uniform distribution has a high $\tilde{Q}$ value, then it would have a high probability in $\hat{\pi}_\theta$ as well, because the ELBO loss for that value would be lower. Unlike raw uniform sampling that does not use VAE, this procedure exposes $\hat{\pi}_\theta$ to a new set of action samples at every gradient step. Algorithm 1 describes the pseudocode for training $\hat{\pi}_\theta$.

4 Experiments

We conduct our experiments in three challenging continuous planning domains under partial observability: finding a path in a 2D light-dark room, object fetching with known object classes, and real-world object fetching with unknown object classes. We now describe the POMDP model for each domain.

2D light-dark room The objective of the robot is to navigate to a goal position with the minimum number of steps in a 2D plane. A state is the (x, y) location of the robot, an action is $(r, \theta) \in (0, 2) \times [0, 2\pi)$ that defines the movement $(\Delta x, \Delta y) = (r\cos\theta, r\sin\theta)$, the transition model is $(x', y') = (x + \Delta x, y + \Delta y)$, an observation $o \in O$ is $(\hat{x}, \hat{y}) \in \mathbb{R}^2$, $(\hat{x}, \hat{y}) = (x, y) + (\eta_1, \eta_2)$ where $\eta_i \sim \mathcal{N}(0, \sigma(x))$, and $\sigma(x) = 0.01 \cdot (4 - x)^2 + \epsilon$, with $\epsilon = 1e^{-5}$. The robot incurs a reward of -1 for every action and a reward of +100 for reaching the goal region. The discount factor γ is set to 1.0, and the maximum planning horizon H is 30. An example problem instance is shown in Figure 4 (a) with the initial robot state shown with a blue dot, the goal region shown with a red circle, and its observations shown in green. A problem instance is defined by an initial state and goal region. The initial state of the robot is sampled from a uniform distribution from the blue box, and the goal region is defined by a circle with a radius of 0.25 whose center is uniform-randomly sampled from the red box.

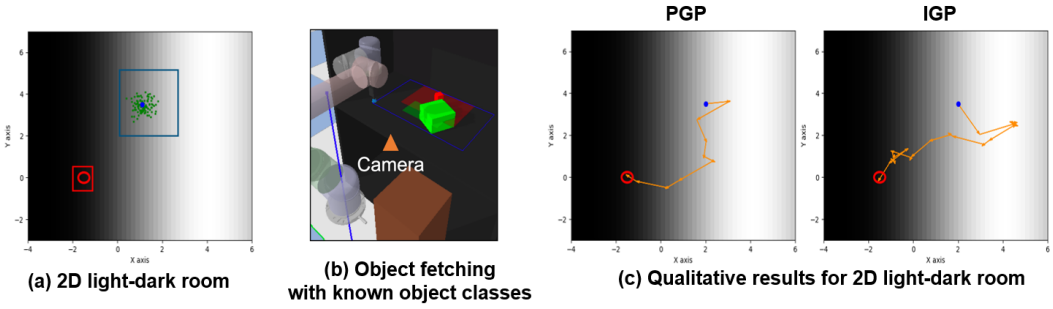

Figure 4: Example problem instances of (a) 2D light-dark room, adapted from [30], (b) Object fetching with known object classes. (c) Example of execution trajectories (orange lines) of PGP and IGP. Note that PGP computes a shorter path.

Object fetching with known object classes Figure 4 (b) shows an example problem instance from this domain. The robot's objective is to fetch the target object (red) completely occluded by the non-target object (green) from the cabinet and place it in the goal region (the brown stand to the robot's right) in a minimum number of steps. The camera is located at the front of the shelf, marked with an orange triangle. A problem instance is defined by the initial poses of the objects, where we first uniform-randomly sample the green box's position from the green transparent box, and then sample the red object location from the region completely occluded by the green box. The orientations are

uniformly sampled from $[0, 2\pi)$. The non-target object pose is observable, but not for the target object.

A state is defined as $(q_{\text{robot}}, q_{\text{target}}, q_{\text{non-target}})$, where q_{robot} is the robot's joint positions and $q_{\text{target}}, q_{\text{non-target}} \in SE(2)$ are the poses of the target and non-target objects, respectively. The action space consists of two action primitives with discrete and continuous parameters, $\texttt{PICK}(\texttt{obj}, \mathcal{T}_{\text{pick}})$ and $\texttt{PLACE}(\texttt{obj}, \mathcal{T}_{\text{place}})$ where $\texttt{obj}$ is the object to manipulate and $\mathcal{T}_{\text{pick}}, \mathcal{T}_{\text{place}} \in SE(2)$ are the poses of the gripper. We ensure that each action is feasible using rejection sampling based on the existence of inverse kinematics solution, collision-free motion, and the contact between the object and the gripper, which we describe in detail in the appendix. The transition model is defined with Bullet physics engine [31]. An observation consists of depth and RGB images, denoted I_{depth} and I_{RGB} respectively, the union of foreground segmentation mask for each object $Seg_{\text{target}} \cup Seg_{\text{non-target}}$, and the indicator whether the robot is holding an object, $g \in \{0, 1\}$. The observation model is defined as the product of two functions $Z(o|s) = \mathcal{N}(\text{CD}(\text{PCD}(o), \text{PCD}(s))|0, \sigma^2) \cdot \mathbb{1}\{g(o) = g(s)\}$, where CD denotes the Chamfer distance, $\text{PCD}(o)$ and $\text{PCD}(s)$ denote the observed and ground-truth point cloud respectively, $g(o)$ and $g(s)$ are the observation and ground-truth object-holding flag respectively, and $\sigma = 0.0085$. The robot gets +100 if the target object is placed in the goal region, -100 if an object is dropped on the ground, or if the non-target object is placed in the goal region, and -1 for all the other actions. The discount factor γ is set to 1.0. The planning depth H is 6.

Real-world object fetching with unknown object classes In this domain, the robot operates in the real world and must estimate both the object classes and poses from camera images. Example problems and objects are shown in Figure 1. We implement off-the-shelf perception algorithms and explicitly reason about the uncertainty in the object classifier and pose estimator. More concretely, we model the initial belief as $P(s|o) = P(q, c|o) = P(q|c, o) \cdot P(c|o)$, where c is the class label and q is SE(3) pose of the object. We use Mask R-CNN [32] with dropout sampling [33] to model $P(c|o)$, and a PointNet [34]-based architecture for the pose estimator which also gives a distribution over the potential poses. The details of the perception system is in the appendix. A state is defined as $(q_{\text{robot}}, q_{\text{target}}, q_{\text{non-target}}, c_{\text{target}}, c_{\text{non-target}})$, where $c_{\text{target}}, c_{\text{non-target}}$ are the class labels of the target and non-target objects. Other quantities in the POMDP are equivalent to the previous domain.

We have two hypotheses. First, for learning from search trees, preference-based learning is more data efficient than imitation learning, and second, guided search is faster than an unguided search. To validate these, we compare PGP and its variant, success-fail-preference-guided POMCPOW (SF-PGP), where the value and policy networks are trained to prefer the success trajectory over a failure trajectory but not necessarily to prefer shorter success trajectories, to following methods. The first is **unguided POMCPOW** (UNGUIDED) which is a standard POMCPOW that samples actions with a uniform random policy and evaluates a leaf node using a rollout. The second is **imitation-guided POMCPOW** (IGP) which uses an imitation policy and MSE-V function for guiding POMCPOW. The imitation policy is trained to imitate actions on success histories, and MSE-V is trained to predict the future sum of rewards of the given trajectory using a mean squared error loss. For the guided search, we replace the rollout with value function evaluation, and the uniform sampling over actions using the trained policy. For fetching domains, we learn the policy for just the place action.

Figure 5 (a) shows the result for the light-dark room domain. In the first two columns, preference-based approaches, PGP and SF-PGP, achieve significantly higher success rates than IGP when they are trained with the same number of tree searches and computes, supporting our hypotheses. With 10 search trees and 100 simulations, PGP and SF-PGP achieve 80% success rate, while IGP and UNGUIDED require 200 simulations to achieve that. With 100 search trees, IGP gets better, but the IGP and SF-PGP are more efficient and achieve higher success rates. The third and fourth columns show that PGP and SF-PGP compute shorter successful trajectories more efficiently compared to UNGUIDED and IGPwhen using 10 and 100 tree searches, although UNGUIDED eventually gets similar optimality as PGP and SF-PGP. Figure 4 (c) shows the comparison of the trajectories computed by PGP and IGP.

Figure 5 (b) shows the result for the fetching domain with known object classes. In the first column, we again can see that PGP and SF-PGP achieve higher success rates than IGP and UNGUIDED when

using the same number of simulations and using 10 tree searches. When using 100 tree searches (second column), we see that IGP improves, but PGP and SF-PGP still outperforms it, demonstrating both computational and data efficiency. Notably, when using just 10 search trees, PGP outperforms SF-PGP, unlike the previous domain. This is because, in the previous domain, the length of the trajectory does not influence success or failure as the planning horizon is long and the robot has ample opportunity to correct its behavior. In contrast, in this domain, shorter trajectories tend to have higher success rates because the robot needs to fetch the target object within 6 steps, otherwise, it is a failure. This indicates that the success-success comparison scheme in PGP is helpful in this domain. In the third and fourth columns, we see that PGP outperforms all other algorithms in terms of optimality. These results again support our hypothesis.

Figure 5 (c) shows the results from the real robot experiments from the scenario shown in Figure 1 for target objects from different classes. From the first column, we see that PGP again outperforms UNGUIDED in all objects in terms of success rate, and outperforms IGP in the hammer case. In two other cases, IGP gets the same success rate as PGP. The second column shows that for success trajectories, PGP finds the most optimal solution among all the baselines. We include a more exhaustive comparison in the simulated version of this domain for all baselines in the appendix.

5 Limitations and future directions

In this work, we propose a preference-based learning algorithm for guiding POMCPOW and show that it is more computationally efficient than UNGUIDED and more data-efficient than IGP in several robotics domains. One key limitation of our framework, as with any POMDP-based planner, is the heavy reliance on the quality of the perception algorithm (more exhaustive limitations are included in the appendix). To focus on planning, we deliberately simplified the perception problem in our real-robot domain by considering a limited set of objects. Even with this simplification, however, we had many cases where we lacked particles that matched the real world, and often had geometrically infeasible object poses and shapes where objects penetrated each other. We used particle re-invigoration and re-planning, which helped, but they incurred additional computation costs. These findings suggest that the learning-to-guide-planning paradigm shows promise in reducing planning time. However, they also highlight the need for significant advancements in representation and uncertainty quantification for perception to improve the initial belief generator.

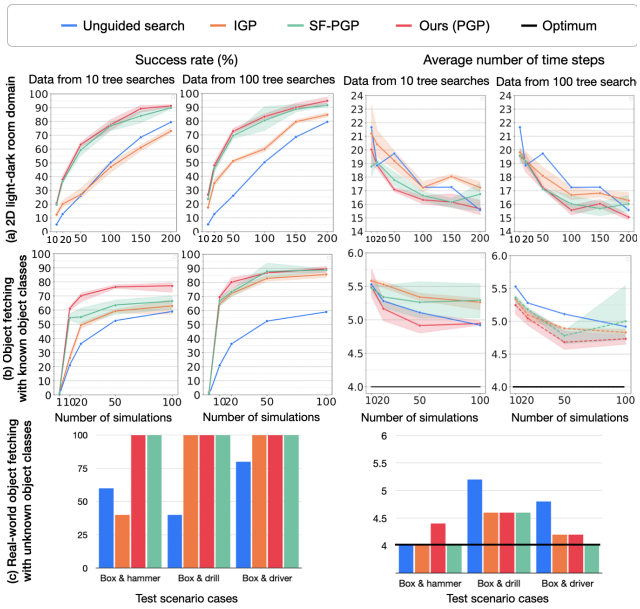

Figure 5: Success rates vs a number of simulations for 10 and 100 tree searches (columns 1 and 2 respectively), and an average number of steps to the goal for successful plans for 10 and 100 tree searches (columns 3 and 4 respectively) for (a) light-dark room domain and (b) object fetching with known object classes. (c) Success rates (column 1) and number of steps to the goal (column 2) in real-world object fetching using 50 simulations and 100 tree searches.

References

[1] Z. Sunberg and M. Kochenderfer. Online algorithms for pomdps with continuous state, action, and observation spaces. In *Proceedings of the International Conference on Automated Planning and Scheduling*, volume 28, pages 259–263, 2018.

[2] A. Couëtoux, J.-B. Hoock, N. Sokolovska, O. Teytaud, and N. Bonnard. Continuous upper confidence trees. In *Learning and Intelligent Optimization: 5th International Conference, LION 5, Rome, Italy, January 17-21, 2011. Selected Papers 5*, pages 433–445. Springer, 2011.

[3] B. Kim, L. Shimanuki, L. Kaelbling, and T. Lozano-Pérez. Representation, learning, and planning algorithms for geometric task and motion planning. In *The International Journal of Robotics Research*, 2022.

[4] OpenAI. Gpt-4 technical report, 2023.

[5] B. Ibarz, J. Leike, T. Pohlen, G. Irving, S. Legg, and D. Amodei. Reward learning from human preferences and demonstrations in atari. *Advances in neural information processing systems*, 31, 2018.

[6] D. Brown, W. Goo, P. Nagarajan, and S. Niekum. Extrapolating beyond suboptimal demonstrations via inverse reinforcement learning from observations. In *International conference on machine learning*, pages 783–792. PMLR, 2019.

[7] P. F. Christiano, J. Leike, T. Brown, M. Martic, S. Legg, and D. Amodei. Deep reinforcement learning from human preferences. *Advances in neural information processing systems*, 30, 2017.

[8] D. Silver and J. Veness. Monte-carlo planning in large pomdps. *Advances in neural information processing systems*, 23, 2010.

[9] L. P. Kaelbling, M. L. Littman, and A. R. Cassandra. Planning and acting in partially observable stochastic domains. *Artificial intelligence*, 101(1-2):99–134, 1998.

[10] J. Pineau, G. Gordon, S. Thrun, et al. Point-based value iteration: An anytime algorithm for pomdps. In *Ijcai*, volume 3, pages 1025–1032, 2003.

[11] T. Smith and R. G. Simmons. Heuristic search value iteration for pomdps. In *Conference on Uncertainty in Artificial Intelligence*, 2004.

[12] M. T. Spaan and N. Spaan. A point-based pomdp algorithm for robot planning. In *IEEE International Conference on Robotics and Automation, 2004. Proceedings. ICRA'04. 2004*, volume 3, pages 2399–2404. IEEE, 2004.

[13] G. Shani, R. I. Brafman, and S. E. Shimony. Forward search value iteration for pomdps. In *IJCAI*, pages 2619–2624, 2007.

[14] D. Hsu, W. S. Lee, and N. Rong. A point-based pomdp planner for target tracking. In *2008 IEEE International Conference on Robotics and Automation*, pages 2644–2650. IEEE, 2008.

[15] H. Kurniawati, D. Hsu, and W. S. Lee. Sarsop: Efficient point-based pomdp planning by approximating optimally reachable belief spaces. In *Robotics: Science and systems*, volume 2008. Citeseer, 2008.

[16] 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):354–359, 2017.

[17] Reinforcement learning with deep energy-based policies. *ICML*.

[18] D. Van Ravenzwaaij, P. Cassey, and S. D. Brown. A simple introduction to markov chain monte–carlo sampling. *Psychonomic bulletin & review*, 25(1):143–154, 2018.

[19] K. Sohn, H. Lee, and X. Yan. Learning structured output representation using deep conditional generative models. *Advances in neural information processing systems*, 28, 2015.

[20] A. Somani, N. Ye, D. Hsu, and W. S. Lee. Despot: Online pomdp planning with regularization. *Advances in neural information processing systems*, 26, 2013.

[21] L. Kocsis and C. Szepesvári. Bandit based monte-carlo planning. In *Machine Learning: ECML 2006: 17th European Conference on Machine Learning Berlin, Germany, September 18-22, 2006 Proceedings 17*, pages 282–293. Springer, 2006.

[22] 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):484–489, 2016.

[23] D. Silver, T. Hubert, J. Schrittwieser, I. Antonoglou, M. Lai, A. Guez, M. Lanctot, L. Sifre, D. Kumaran, T. Graepel, et al. Mastering chess and shogi by self-play with a general reinforcement learning algorithm. *arXiv preprint arXiv:1712.01815*, 2017.

[24] 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. *Nature*, 588(7839):604–609, 2020.

[25] B. Kim, L. Kaelbling, and T. Lozano-Pérez. Guiding search in continuous state-action spaces by learning an action sampler from off-target search experience. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 32, 2018.

[26] B. Kim and L. Shimanuki. Learning value functions with relational state representations for guiding task-and-motion planning. *Conference on Robot Learning*, 2019.

[27] B. Kim, L. Kaelbling, and T. Lozano-Pérez. Adversarial actor-critic method for task and motion planning problems using planning experience. *AAAI Conference on Artificial Intelligence*, 2020.

[28] D. Driess, O. Oguz, J.-S. Ha, and M. Toussaint. Deep visual heuristics: Learning feasibility of mixed-integer programs for manipulation planning. In *Proc. of the IEEE Int. Conf. on Robotics and Automation (ICRA)*, 2020.

[29] D. Driess, J.-S. Ha, and M. Toussaint. Deep visual reasoning: Learning to predict action sequences for task and motion planning from an initial scene image. In *Proc. of Robotics: Science and Systems (R:SS)*, 2020.

[30] R. Platt Jr, R. Tedrake, L. Kaelbling, and T. Lozano-Perez. Belief space planning assuming maximum likelihood observations. 2010.

[31] E. Coumans and Y. Bai. Pybullet, a python module for physics simulation for games, robotics and machine learning. http://pybullet.org, 2016–2021.

[32] K. He, G. Gkioxari, P. Dollár, and R. Girshick. Mask r-cnn. In *Proceedings of the IEEE international conference on computer vision*, pages 2961–2969, 2017.

[33] Y. Gal and Z. Ghahramani. Dropout as a bayesian approximation: Representing model uncertainty in deep learning. *ArXiv*, abs/1506.02142, 2015.

[34] C. R. Qi, H. Su, K. Mo, and L. J. Guibas. Pointnet: Deep learning on point sets for 3d classification and segmentation. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 652–660, 2017.