PROBABILISTIC PLANNING WITH SEQUENTIAL MONTE CARLO

Anonymous authors
Paper under double-blind review

ABSTRACT

Planning is a fundamental ability of intelligent agents, which allows them to reason about their future behaviour and efficiently learn new tasks. Despite its importance, planning remains an open problem in complex environments. Current challenges include model compounding errors in roll-outs and exponential search space over trajectories. In this work, we propose a novel formulation of planning, which views it as a probabilistic inference problem over future optimal trajectories. This enables us to use sampling methods, and thus, tackle planning in continuous and high-dimensional spaces using a fixed computational budget. We design a new algorithm, Sequential Monte Carlo Planning (SMCP), by leveraging modern methods in Sequential Monte Carlo (SMC), Bayesian smoothing, and *control as inference*. We draw parallels between our algorithm and popular planning methods such as Monte Carlo Tree Search (MCTS) (Kearns et al., 2002) and Random Shooting methods (Rubinstein & Kroese, 2004). Furthermore, we show that SMCP can capture multimodal policies and can quickly learn continuous control tasks.

1 INTRODUCTION

In order for machine learning agents to exhibit intelligent behaviour, they must be able to learn quickly, predict the consequences of their actions, and explain how they will react in a given situation. These characteristics are best achieved when the agent is able to efficiently use an approximate model of the world.

Recent successes in planning were achieved with deep learning (DL), such examples include Alpha Go (Silver et al., 2017) which used Monte Carlo Tree Search (MCTS) (Kearns et al., 2002) and robot manipulation which used model predictive control (MPC) (Finn & Levine, 2017).

These successes, however, may only have been possible because the use of perfect or simple models, and have been difficult to replicate in complex environments due to model error compounding or environments with complex dynamics such as contact forces. Additionally, these methods rely on strong assumptions such as unimodality when resampling actions via MPC (using Cross Entropy Methods) and MCTS-UCT (Upper Confidence for Trees)(Kocsis & Szepesvári, 2006) assumes discrete states. Efficient planning, therefore, remains an open problem, particularly in cases when states and actions are continuous and high-dimensional

Here, we frame planning as a density estimation problem over optimal future trajectories. Since trajectories consist of an intertwined sequence of states and actions, estimating this density is complex. To solve this issue, we leverage tools from statistics, such as Sequential Monte Carlo (SMC) methods to build complex and sequential distributions over trajectories. Sequential Monte Carlo methods are flexible and efficient methods used to approximate complex distributions by drawing from simpler ones sequentially. Gu et al. (2015) recently developed techniques advancing SMC by approximating the optimal proposal distribution using variational inference methods. In the field of *control as inference*, Haarnoja et al. (2018) independently proposed Soft Actor Critic (SAC) as a way to approximate the optimal policy. We propose a method that combines the aforementioned methods to make an efficient and robust planning algorithm: SMC planning (SMCP). SMC allows us to naturally spend more computing resources on promising trajectories and capture the uncertainties in the model while SAC maximizes the entropy of its policy, thus allowing us to efficiently explore the distribution of trajectories.

Contribution. We depict the problem of planning as one of density estimation that can be estimated using SMC methods. We introduce a novel planning strategy based on the SMC class of algorithms, in which we treat the policy as the proposed distribution to be learned. Following, we draw connections between our algorithm and popular planning algorithms: MCTS and Random Shooting (RS). We investigate how our method empirically compares with existing

model-based methods and a strong model-free baseline on the standard benchmark Mujoco benchmark (Todorov et al., 2012).

2 BACKGROUND

2.1 CONTROL AS INFERENCE

We consider the general case of a Markov Decision Process (MDP) $\{\mathcal{S}, \mathcal{A}, p_{\text{env}}, r, \gamma, \mu\}$ where $\mathcal{S}$ and $\mathcal{A}$ represent the state and action spaces respectively. We use the letters $\mathbf{s}$ and $\mathbf{a}$ to denote states and actions, which we consider to be continuous vectors. Further notations include: $p_{\text{env}}(\mathbf{s}'|\mathbf{s}, \mathbf{a})$ as the state transition probability of the environment, $r(\mathbf{s}, \mathbf{a})$[1] as the reward function, and $\gamma \in [0, 1)$ as the discount factor. μ denotes the probability distribution over initial states.

This work focuses on an episodic formulation, with a fixed end-time of T. We define a trajectory as a sequence of state-action pairs $\mathbf{x}_{t:T} = \{(\mathbf{s}_t, \mathbf{a}_t), \ldots, (\mathbf{s}_T, \mathbf{a}_T)\}$, and we use the notation π for a policy which represents a distribution over actions conditioned on a state. Here π is parametrized by a neural network with parameters θ. The notation $q_\theta(\mathbf{x}_{1:T}) = \mu(\mathbf{s}_1) \prod_{t\geq 1}^{T-1} p_{\text{env}}(\mathbf{s}_{t+1}|\mathbf{s}_t, \mathbf{a}_t)\pi_\theta(\mathbf{a}_t|\mathbf{s}_t)$ denotes the probability of a trajectory $\mathbf{x}_{1:T}$ under policy π_θ.

Traditionally, in reinforcement learning (RL) problems, the goal is to find the optimal policy that maximizes the expected return $\mathbb{E}_{q_\theta}[\sum_{t=0}^{\infty} \gamma^t r_t]$. However, it is useful to frame RL as an inference problem within a probabilistic graphical framework (Rawlik et al., 2012; Toussaint & Storkey, 2006; Levine, 2018). First, we introduce an auxiliary binary random variable $\mathcal{O}_t$ denoting the "optimality" of a pair $(\mathbf{s}_t, \mathbf{a}_t)$ at time t and define its probability[2] as $p(\mathcal{O}_t = 1|\mathbf{s}_t, \mathbf{a}_t) = \exp(r(\mathbf{s}_t, \mathbf{a}_t))$. $\mathcal{O}$ is a convenience variable only here for the sake of modelling. By considering the variables $(\mathbf{s}_t, \mathbf{a}_t)$ as latent and $\mathcal{O}_t$ as observed, we can construct a Hidden Markov Model (HMM) as depicted in figure 2.1. Notice that the link $\mathbf{s} \to \mathbf{a}$ is not present in figure 2.1 as the dependency of the optimal action on the state depends on the future observations. In this graphical model, the optimal policy is expressed as $p(\mathbf{a}_t|\mathbf{s}_t, \mathcal{O}_{t:T})$.

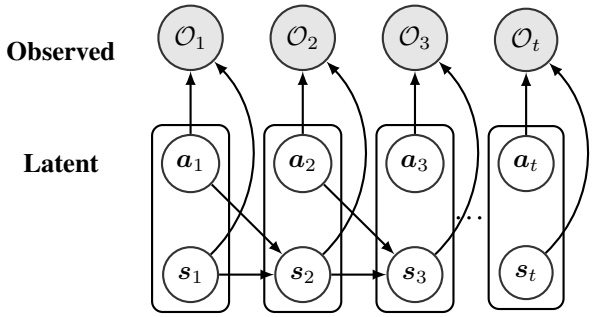


Figure 2.1: $\mathcal{O}_t$ is an observed *optimality* variable with probability $p(\mathcal{O}_t|\mathbf{s}_t, \mathbf{a}_t) = \exp(r(\mathbf{s}_t, \mathbf{a}_t))$

The posterior probability of this graphical model can be written as:

$$p(\mathbf{x}_{1:T}|\mathcal{O}_{1:T}) \propto p(\mathbf{x}_{1:T}, \mathcal{O}_{1:T}) = \mu(\mathbf{s}_1) \prod_{t=1}^{T} p_{\text{env}}(\mathbf{s}_{t+1}|\mathbf{a}_t, \mathbf{s}_t) \exp\big(\sum_{t=1}^{T} r(\mathbf{s}_t, \mathbf{a}_t)\big). \tag{2.1}$$

It appears clearly that finding optimal trajectories is equivalent to finding plausible trajectories yielding a high return.

Many *control as inference* methods can be seen as approximating the density by optimizing its variational lower bound: $\log p(\mathcal{O}_{1:T}) \geq \mathbb{E}_{\mathbf{x}_{1:T}\sim q_\theta}[\sum_{t=1}^{T} r(\mathbf{s}_t, \mathbf{a}_t) - \log \pi_\theta(\mathbf{a}_t|\mathbf{s}_t)]$ (Rawlik et al., 2012; Toussaint, 2009). Instead of directly differentiating the variational lower bound, it is possible to take a message passing approach such as the one used in Soft Actor-Critic (SAC) (Haarnoja et al., 2018) and directly estimate the optimal policy $p(\mathbf{a}_t|\mathbf{s}_t, \mathcal{O}_{t:T})$.

2.2 SEQUENTIAL MONTE CARLO

Since distributions over trajectories are complex, it is often difficult or impossible to directly draw samples from them. Fortunately in statistics, there are successful strategies for drawing samples from complex sequential distributions, such as SMC methods (Doucet et al., 2001). These techniques require knowledge of the density of a target distribution up to a multiplicative constant.

[1] For the state-action pair $(\mathbf{s}_t, \mathbf{a}_t)$ at time t, we will use the notation $r_t = r(\mathbf{s}_t, \mathbf{a}_t)$

[2] as in Levine (2018), if the rewards are bounded above, we can always remove a constant so that the probability is well defined.

For simplicity, in the remainder of this section we will overload the notation and refer to the target distribution as $p(\mathbf{x})$ and the proposal distribution as $q(\mathbf{x})$. In the next section, we will define the distributions p and q in the context of planning.

Importance sampling (IS): In the case where $\mathbf{x}$ can be efficiently sampled from another simpler distribution q i.e. the proposal distribution, we can estimate the likelihood of any point $\mathbf{x}$ under p straightforwardly by computing the *unnormalized importance sampling weights* $w(\mathbf{x}) \propto \frac{p(\mathbf{x})}{q(\mathbf{x})}$ and using the identity $p(\mathbf{x}) = \frac{w(\mathbf{x})}{\int w(\mathbf{x})q(\mathbf{x})d\mathbf{x}} q(\mathbf{x})$. We will refer to $\bar{w}(\mathbf{x}) = \frac{w(\mathbf{x})}{\int w(\mathbf{x})q(\mathbf{x})d\mathbf{x}}$ as the *normalized importance sampling weights*. In practice, one draws N samples from q: $\{\mathbf{x}^{(n)}\}_{n=1}^N \sim q$; these are referred to as *particles*. The set of particles $\{\mathbf{x}^{(n)}\}_{n=1}^N$ associated with the weights $\{\bar{w}^{(n)}\}$ are simulations of samples from p. That is, we approximate the density p with a weighted sum of diracs from samples of q:

$$p(\mathbf{x}) \approx \sum_{n=1}^{N} \bar{w}^{(n)} \delta_{\mathbf{x}^{(n)}}(\mathbf{x}), \text{ with } \mathbf{x}^{(n)} \text{ sampled from } q$$

where $\delta_{\mathbf{x}^0}(\mathbf{x})$ denotes the Dirac delta mass located as $\mathbf{x}_0$.

Sequential Importance Sampling (SIS): When our problem is sequential in nature $\mathbf{x} = \mathbf{x}_{1:T}$, the complexity of the sampling can grow super-linearly with T. By exploiting the sequential structure, the unnormalized weights can be updated iteratively in an efficient manner: $w_t(\mathbf{x}_{1:t}) = w_{t-1}(\mathbf{x}_{1:t-1}) \frac{p(\mathbf{x}_t|\mathbf{x}_{1:t-1})}{q(\mathbf{x}_t|\mathbf{x}_{1:t-1})}$. We call this the **update step**. This enables us sample sequentially $\mathbf{x}_t \sim q(\mathbf{x}_t|\mathbf{x}_{1:t-1})$ to finally obtain the set of particles $\{\mathbf{x}_{1:T}^{(n)}\}$ and their weights $\{\bar{w}_T^{(n)}\}$ linearly in the horizon T.

Sequential Importance Resampling (SIR) / Particle Filter: When the horizon T is long, samples from q usually have a low likelihood under p, and thus the quality of our approximation decreases exponentially with T. More concretely, the unnormalized weights $w_t^{(n)}$ converge to 0 with $t \to \infty$. This usually causes the normalized weight distribution to degenerate, with one weight having a mass of 1 and the others a mass of 0. This phenonmemon is known as *weight impoverishment*.

One way to address weight impoverishment is to add a **resampling step** where each particle is resampled to higher likelihood regions at each time step. This can typically reduce the variance of the estimation from growing *exponentially* with t to growing *linearly*. Resampling can also have adverse effects, for instance all the particles could be resampled on the most likely particle, leading to a particle degeneracy. We tackle this issue by selecting the temperature of the resamping distribution to not be too low.

3 Planning as probabilistic inference

In the context of *control as inference*, it is natural to see planning as the act of approximating a distribution of optimal future trajectories via simulation. In order to plan, an agent must possess a model of the world that can accurately capture consequences of its actions. In cases where multiple trajectories have the potential of being optimal, the agent must rationally partition its computational resources to explore each possibility. Given finite time, the agent must limit its planning to a finite horizon h. We, therefore, define *planning* as the act of approximating the optimal distribution over trajectories of length h. In the control-as-inference framework, this distribution is naturally expressed as $p(\mathbf{a}_1, \mathbf{s}_2, \ldots \mathbf{s}_h, \mathbf{a}_h|\mathcal{O}_{1:T}, \mathbf{s}_1)$, where $\mathbf{s}_1$ represents our current state.

3.1 Planning and Bayesian smoothing

As we consider the current state $\mathbf{s}_1$ given, it is equivalent and convenient to focus on $p(\mathbf{x}_{1:h}|\mathcal{O}_{1:T})$. Bayesian smoothing is an approach to the problem of estimating the distribution of a latent variable conditioned on all past and future observations. This method decomposes the posterior into:

$$p(\mathbf{x}_{1:h}|\mathcal{O}_{1:T}) \propto \underline{p(\mathbf{x}_{1:h}|\mathcal{O}_{1:h})} \cdot \underline{p(\mathcal{O}_{h+1:T}|\mathbf{x}_h)} \tag{3.1}$$

This corresponds to a forward-backward messages factorization in a Hidden Markov Model as depicted in figure 3.1.

Filtering is the task of estimating $p(\mathbf{x}_t|\mathcal{O}_{1:t})$: the probability of a latent variable conditioned on all past observations. In contrast, **smoothing** estimates $p(\mathbf{x}_t|\mathcal{O}_{1:T})$: the density of a latent variable conditioned on all the past and future measurements.

In the belief propagation algorithm for HMMs, these probabilities correspond to the forward message $\alpha_h(\mathbf{x}_h) = p(\mathbf{x}_{1:h}|\mathcal{O}_{1:h})$ and backward message $\beta_h(\mathbf{x}_h) = p(\mathcal{O}_{h+1:T}|\mathbf{x}_h)$, both of which are computed recursively. While in discrete spaces, these forward and backward messages can be estimated using the sum-product algorithm, its complexity scales with the square of the space dimension. We will now devise efficient strategies for computing both the forward and backward messages.

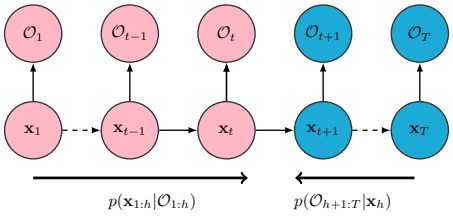


Figure 3.1: Factorization of the HMM into forward (red) and backward (blue) messages. Estimating the forward message is filtering, estimating the value of the latent knowing all the observations is smoothing.

3.2 The backward message and the value function

The backward message $p(\mathcal{O}_{h+1:T}|\mathbf{x}_h)$ can be understood as the answer to: "What is the probability of following an optimal trajectory from the next time step on until the end of the episode, given my current state?" Importantly, this term is closely related to the notion of *value function* in RL. Indeed, in the control-as-inference framework, the state- and action-value functions are defined as $V(\mathbf{s}_h) = \log p(\mathcal{O}_{h:T}|\mathbf{s}_h)$ and $Q(\mathbf{s}_h, \mathbf{a}_h) = \log p(\mathcal{O}_{h:T}|\mathbf{s}_h, \mathbf{a}_h)$ respectively. They are solutions of a soft-Bellman equation that differ a little from the traditional Bellman equation (O'Donoghue et al., 2016; Nachum et al., 2017; Schulman et al., 2017; Abdolmaleki et al., 2018). A more in depth explanation can be found in (Levine, 2018). We can show subsequently that:

$$p(\mathcal{O}_{h+1:T}|\mathbf{x}_h) = \mathbb{E}_{\mathbf{s}_{h+1}|\mathbf{x}_h}\left[\exp\left(V(\mathbf{s}_{h+1})\right)\right] \tag{3.2}$$

if we choose the action prior to be uniform. Full details can be found in appendix A.3. Estimating the backward message is then equivalent to learning a value function. Consequently we use SAC (Haarnoja et al., 2018) to efficiently estimate the value function.

3.3 Filtering with Sequential Monte Carlo Methods

Unlike the backward message, the forward message does not have a straightforward interpretation in RL. One way to estimate it reliably without making simplifying assumptions is to apply SMC methods covered in section 2.2. To be consistent with the terminology of section 2.2, we call $p(\mathbf{x}_{1:h}|\mathcal{O}_{1:h})$ the target distribution and $q_\theta(\mathbf{x}_{1:h})$ the proposal distribution.

The importance sampling weights $w_t \propto \frac{p(\mathbf{x}_{1:t}|\mathcal{O}_{1:t})}{q_\theta(\mathbf{x}_{1:t})}$ adhere to the following recursive expression:

$$w_t \propto \exp\left(r_t - \log \pi_\theta(\mathbf{a}_t|\mathbf{s}_t)\right) \cdot w_{t-1} \tag{3.3}$$

assuming that we had learned a perfect model of the environment. The derivation of the recursion is provided in Appendix A.2. The perfect-model hypothesis is not needed when we have access to a simulator of the environment, as in AlphaGo (Silver et al., 2017) or Guo et al. (2014) who used the Arcade Learning Environment (ALE) (Bellemare et al., 2013) simulator.

Notice that the weight update at each step corresponds to a multiplication by the exponential of the entropy regularized reward, which is consistent with the reward signal used in Maximum Entropy Reinforcement Learning (Ziebart, 2010).

3.4 Sequential Monte Carlo Planning Algorithm

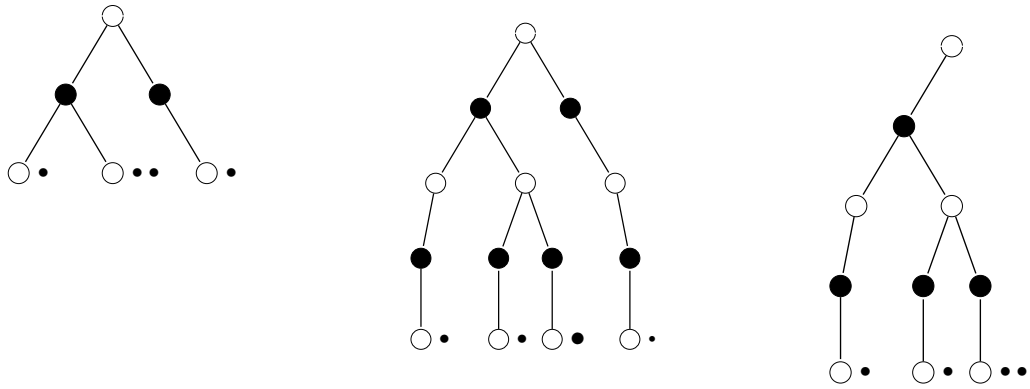
We can now use the computations of the forward and backward message to derive the full algorithm. We consider the root state of the planning to be the current state $\mathbf{s}_t$. We aim at building a set of particles $\{\mathbf{x}_{t:t+h}^{(n)}\}_{n=1}^N$ and their weights $\{\hat{w}_T^{(n)}\}_{n=1}^N$ representative of the planning density $p(\mathbf{x}_{t:t+h}|\mathcal{O}_{1:T})$ over optimal trajectories. We use SAC (Haarnoja et al., 2018) for the policy and value function, but any other policy can be used for the proposal distribution.

We summarize the proposed algorithm in Algorithm 1. At each step, we sample from the proposal distribution or model-free agent (**line 5**) and evaluate the expected likelihood of the step of being optimal using a given model of the environment, i.e. $\hat{w}_T \propto \exp(r_t + V(\mathbf{s}_{t+1}) - \log \pi(\mathbf{a}_t|\mathbf{s}_t))$ (**line 8-9**). The trajectories are then weighted by their likelihood of being optimal till the end of the episode. We subsequently (re-)sample trajectories from a multinomial distribution with probabilities proportional to this likelihood (**line 10**). Each (re-)sampled trajectory particle is assigned a fixed weight of 1 (**line 11**), as these particles together now constitute an unweighted empirical distribution of $p(\mathbf{x}_{1:t}|\mathcal{O}_{1:t})$, whereas before sampling we had a weighted empirical distribution. After the tree search has been performed for h steps, we possess an unweighted distribution over trajectories and can sample from it (**line 14**). We then execute the first action from the sampled trajectory into the environment. We restart the procedure at the following time step. The pseudo code is provided in Algorithm 1. An alternative algorithm that does not use the resampling step is highlighted in Algorithm 2.

A schematic view of the algorithm can also be found on figure 3.2.

Algorithm 1 Planning using Sequential Importance Resampling

1: **for** t in $\{0, \dots, T\}$ **do**
2: $\quad \mathbf{s}_t^{(n)} = \mathbf{s}_t, \forall n, 1 \leq n \leq N.$
3: $\quad w_0 = 1.$
4: $\quad$ **for** i in $\{t, \dots, t+h\}$ **do**
5: $\quad\quad \{\mathbf{a}_i^{(n)} \sim \pi^{SAC}(\mathbf{a}_i^{(n)}|\mathbf{s}_i^{(n)})\}_{n=0}^N$
6: $\quad\quad \{\mathbf{s}_{i+1}^{(n)}, r_i^{(n)} \sim p_{\text{model}}(\cdot|\mathbf{s}_i^{(n)}, \mathbf{a}_i^{(n)})\}_{n=0}^N$
7: $\quad\quad w_i \propto w_{i-1} \cdot \exp(r_i - \lambda \log \pi^{SAC}(\mathbf{a}_i|\mathbf{s}_i))$
8: $\quad\quad \hat{w}_T \propto w_i \cdot \underline{\exp(V^{SAC}(\mathbf{s}_{t+1}))}$
9: $\quad\quad$ # Resample the particles $\propto$ weight $\hat{w}_T$
10: $\quad\quad \{\mathbf{x}_{1:i+1}^{(n)}\}_{n=1}^N \sim \text{Mult}(n; \hat{w}_T^{(1)}, \dots, \hat{w}_T^{(N)})$
11: $\quad\quad w_i \leftarrow 1$
12: $\quad$ **end for**
13: $\quad \hat{w}_T \propto w_{t+h}$
14: $\quad$ Sample $n \sim \text{Categorical}(\hat{w}_T^{(1)}, \dots, \hat{w}_T^{(N)})$.
15: $\quad$ Select $\mathbf{a}_t$, first action of $\mathbf{x}^{(n)}$.
16: $\quad \mathbf{s}_{t+1}, r_t \sim p_{\text{env}}(\cdot|\mathbf{s}_t, \mathbf{a}_t)$
17: $\quad \mathbf{s}_t \leftarrow \mathbf{s}_{t+1}.$
18: **end for**



(a) **Configuration at time $t-1$:** we have the root white node $\mathbf{s}_{t-1}$, the actions $\mathbf{a}_{t-1}^{(n)}$ are black nodes and the leaf nodes are the $\mathbf{s}_t^{(n)}$. We have one particle on the leftmost branch, two on the central branch and one on the rightmost branch.

(b) **Update:** New actions and states are sampled from the proposal distribution and model. The particle sizes are proportional to their density $p(\mathbf{x}_{1:t}|\mathcal{O}_{1:T})$.

(c) **Resampling:** after sampling with replacement the particles relatively to their weight, the less promising branch was cut while the most promising has now two particles.

Figure 3.2: Schematic view of Sequential Monte Carlo planning. In each tree, the white nodes represent states and black nodes represent actions. Each bullet point near a state represents a particle, meaning that this particle contains the total trajectory of the branch. The root of the tree represents the root planning state, we expand the tree downward when planning.

4 RELATED WORK

4.1 PLANNING ALGORITHMS

In this section, we highlight the connections between our SMCP algorithm and other popular planning algorithms such as Monte Carlo Tree Search and Random Search.

As shown in figure 3.2, SMC-Planning stochastically explores the most promising trajectories in the tree, cutting the less promising branches off when their particles are not resampled. MCTS with UCT, on the other hand, uses a confidence interval based approach, optimistically expanding branches that may have the best return (Agrawal, 1995; Szita & Lörincz, 2006). Indeed SMC-Planning can be viewed as the maximum entropy tree search equivalent of MCTS. While it may not perform as well as MCTS in discrete environments with known dynamics, our algorithm can also be applied to continuous tasks.

Note also the similarity between the loss used in Expert Iteration with MCTS (Anthony et al., 2017) and AlphaGo Zero (Silver et al., 2017). These algorithms learn policies that minimize the cross entropy between the MCTS policy $\mathbf{p}$ and the neural network policy $\boldsymbol{\pi}_\theta$: $\mathbf{p}^\top \log \boldsymbol{\pi}_{\boldsymbol{\theta}}$. This is similar to the way in which our algorithm learns the proposal distribution as it minimizes the expected KL divergence between the optimal policy and our current policy.

There are also similarities with the random search methods in the continuous setting. Random search methods draw a sequence of actions independent from the states and evaluate the log likelihood after a horizon h i.e. $\sum_{i=1}^{h} r_i$. The trajectory that is most likely to be optimal without conditioning further than h steps is selected. In this work, we compare our algorithm with the Cross Entropy Method (CEM) (Rubinstein & Kroese, 2004; Szita & Lörincz, 2006) which is a more sophisticated random search that uses evolutionary methods.

4.2 PLANNING AS INFERENCE

Attias (2003) proposed a definition of planning most similar to ours. He computed a posterior over actions conditioned on reaching the goal state within a specific number of steps. Much work aimed at solving *control as inference* problems have focused on maximum entropy policies.

Planning can also be tackled with trajectory optimization methods. When the systems is linear Gaussian (linear Gaussian transitions, observations, reward function and Gaussian noise), Linear Quadratic Regulator methods can compute directly the optimal planning. However we do not assume we have access to the dynamics, nor that it is linear gaussian. While extensions of LQR to more complex environments exist (Todorov & Li, 2005; Toussaint, 2009), these methods typically do not handle well arbitrary reward functions.

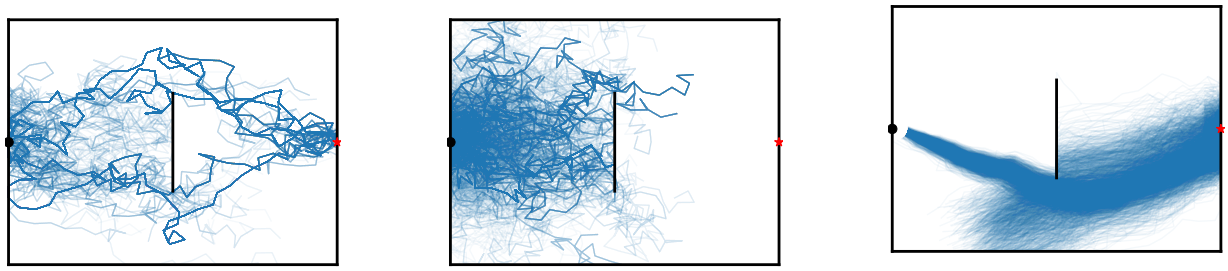
More generally, the problem of RL has also been portrayed as an inference problem by (Dayan & Hinton, 1997; Rawlik et al., 2010; 2012).

Botvinick & Toussaint (2012) summarized the connection between RL and neuroscience views of planning as inference. From these perspectives, planning as inference is seen to occur when the decision-making agent makes use an internal cognitive model to represent the future as a joint probability distribution over actions, outcomes states and rewards. The agent can then sample from its internal model to perform a tree search or could condition on future states of interest and perform inverse inference to discover the course of actions necessary to obtain its goal.

4.3 INFERENCE IN STATE SPACE MODELS

Sequential Monte Carlo methods have grown in popularity since their inception, in localisation (Arulampalam et al., 2002) and signal processing (Del Moral, 1996), due to their ease of use and flexibility. They have the advantage being able to cover multiple modes and scale well with horizon (Doucet et al., 2001). This is particularly relevant in real time localization tasks where one must estimate the position and velocity of an agent (often a robot), given only the previous measurements (signals from GPS, accelerometers or gyro sensors for instance).

Bayesian smoothing (Särkkä, 2013) has also been applied to Part of Speech (PoS) tagging. For instance, in a sentence, parts of speech (nouns, verbs, articles, etc.) are considered to be the latent variables and the spoken words are considered as the observed variables. Similar to planning as inference, PoS tagging requires consideration of consistency i.e. conditioning on future words in a sentence in order to infer current words.



(a) Sequential Importance Resampling (SIR): when resampling the trajectories at each time step, the agent is able to focus on the promising trajectories and does not collapse on a single mode.

(b) Sequential Importance Sampling (SIS): if we do not perform the resampling step the agent spends most of its computation on uninteresting trajectories and was not able to explore as well.

(c) CEM: here the agent samples all the actions at once from a Gaussian with learned mean and covariance. We needed to update the parameters 50 times for the agent to find one solution, but it forgot the other one.

Figure 5.1: Comparison of three methods on the toy environment. The agent (•) must go the goal ($\star$) while avoiding the wall (|) in the center. The proposal distribution is taken to be an isotropic gaussian. Here we plot the planning distribution imagined at $t = 0$ for three different agents. Only the agent using Sequential Importance Resampling was able to find good trajectories while not collapsing on a single mode.

Gu et al. (2015) proposed minimizing the KL divergence between the true and the approximate posterior to learn a good proposal distribution. It is similar to how we learn our proposal distribution, however, we minimized the $\mathrm{KL}(\mathbf{q}\|\mathbf{p})$ instead of the inverse KL. Our method is more appropriate when performing exploration.

5 Experiments

5.1 Didactic example

In this section, we aim to show that SMCP can deal with multimodal policies when planning. We applied two version of SMCP: i) with a resampling step (SIR) ii) without a resampling step (SIS) and compare it to CEM on a simple 2D point mass environment 5.1. Here, the agent can control the displacement on (x, y) within the square $[0, 1]^2$, $\mathbf{a} = (\Delta x, \Delta y)$ with maximum magnitude $||\mathbf{a}|| = 0.05$. The starting position (•) of the agent is $(x = 0, y = 0.5)$, while the goal ($\star$) is at $\mathbf{g} = (x = 1, y = 0.5)$. The reward is the agent's relative closeness to the goal: $r_t = 1 - \frac{||\mathbf{s}_{t+1} - \mathbf{g}||^2}{||\mathbf{s}_t - \mathbf{g}||^2}$. However, there is a partial wall at the centre of the square leading to two optimal trajectories, one choosing the path below the wall and one choosing the path above.

The proposal is an isotropic normal distribution for each planning algorithm, and since the environment's dynamics are known, there is no need for learning: the only difference between the three methods is how they handle planning. We used 1500 particles for each method, and updated the parameters of CEM 50 times. Our experiment 5.1 shows how having particles can deal with multimodality and how the resampling step can help to focus on the most promising trajectories.

5.2 Continuous Control Benchmark

The experiments were conducted on the Open AI Gym benchmark suite (Brockman et al., 2016). The aim of our experiments were to understand how incorporating planning increases the learning speed of RL agents. We evaluated our algorithm on a suite of Mujoco tasks. These environments provide a complex benchmark with continuous states and actions that requires exploration in order to achieve state-of-the-art performances. We used minimal tweaking of the hyper parameters of SMCP as we fixed the horizon length to 25, the temperature to 10 and the reward scaling suggested by Haarnoja et al. (2018) for all experiments.

We included as baselines: SAC (Haarnoja et al., 2018), CEM (Finn & Levine, 2017) and RS (Nagabandi et al., 2017). SAC was included because: i) it has one of the highest performances on Mujoco tasks, which make it a very strong

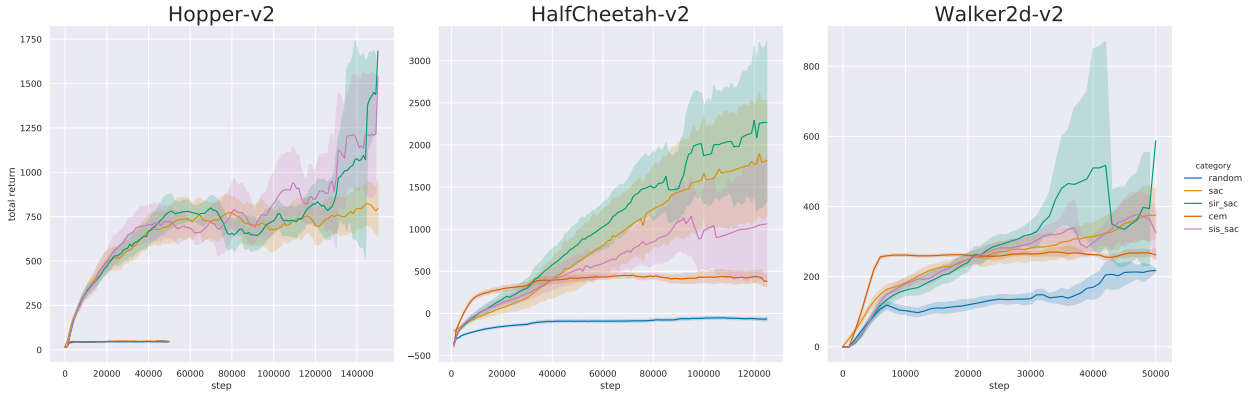


Figure 5.2: Training curves on the Mujoco continuous control benchmarks. SMCP (green) learns faster than the SAC model-free baseline and achieve higher asymptotic performances than the model-based methods (CEM and RS).

baseline, and ii) it is a component of our algorithm, as we use it as a proposal distribution in the planning phase. The CEM and RS methods were included since they are amongst the most popular planning algorithms in the deep RL community.

Our results suggest that SMCP does not learn as fast as CEM initially as it heavily relies on estimating a good value function. However, SMCP quickly achieved higher performances than CEM. SMCP also learned faster than SAC because it was able to leverage information from the model early in training.

Note that our results differ slightly from the results usually found in the model-based RL literature. This is because we are tackling a more difficult problem: estimating the transitions and the reward function. We are using unmodified versions of the environments which introduce many challenges. For instance, our reward function is very noisy. Usually, the environments are modified such that the reward can be computed directly from the state e.g. Chua et al. (2018).

6 Conclusion and Future Work

The main objective of this research was to leverage sampling methods to develop an efficient planning method that could be applied to complex continuous domains. Our work has led to three major contributions: (1) drawing connections between planning and Bayesian smoothing, (2) development of a planning algorithm referred to as "Sequential Monte Carlo planning" (SMCP), and (3) successful application of SMCP in complex continuous control benchmark tasks.

An advantage of our approach is its ability to hold distributions over potential trajectories, unlike modern RL methods, which depend heavily on mode seeking behaviour. Approximating a distribution over potentially good trajectories, instead of seeking the mode, leads to natural exploration strategies which translates to faster learning in difficult tasks.

Additionally, our approach allows us to understand how an agent is going to behave several time steps into the future. This characteristic is important when deploying RL agents in high stake scenarios as they will not fail unexpectedly.

Our approach is expensive to train due the necessity of training all components of model free agents (i.e. policy and value networks) and any additional generative models. We also need to query these models multiple times before the agent takes an action.

Despite the achievements of this work, there still remains work to be done to achieve effective planning under realistic constraints. Some challenges to begin with include: i) planning in visual environments, ii) to leverage the different uncertainties captured by the model and the agent, and iii) be less dependent on the proposal distribution. But ultimately, we believe that our algorithm could be used as a stepping stone to achieve the standard of planning necessary to deploy real world reinforcement learning agents.

ACKNOWLEDGMENTS

Use unnumbered third level headings for the acknowledgments. All acknowledgments, including those to funding agencies, go at the end of the paper.

REFERENCES

Abbas Abdolmaleki, Jost Tobias Springenberg, Yuval Tassa, Remi Munos, Nicolas Heess, and Martin Riedmiller. Maximum a posteriori policy optimisation. *arXiv preprint arXiv:1806.06920*, 2018.

Rajeev Agrawal. Sample mean based index policies by o (log n) regret for the multi-armed bandit problem. *Advances in Applied Probability*, 27(4):1054–1078, 1995.

Thomas Anthony, Zheng Tian, and David Barber. Thinking fast and slow with deep learning and tree search. In *Advances in Neural Information Processing Systems*, pp. 5360–5370, 2017.

M Sanjeev Arulampalam, Simon Maskell, Neil Gordon, and Tim Clapp. A tutorial on particle filters for online nonlinear/non-gaussian bayesian tracking. *IEEE Transactions on signal processing*, 50(2):174–188, 2002.

Hagai Attias. Planning by probabilistic inference. In *AISTATS*. Citeseer, 2003.

Marc G Bellemare, Yavar Naddaf, Joel Veness, and Michael Bowling. The arcade learning environment: An evaluation platform for general agents. *Journal of Artificial Intelligence Research*, 47:253–279, 2013.

Matthew Botvinick and Marc Toussaint. Planning as inference. *Trends in cognitive sciences*, 16(10):485–488, 2012.

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

Kurtland Chua, Roberto Calandra, Rowan McAllister, and Sergey Levine. Deep reinforcement learning in a handful of trials using probabilistic dynamics models. *arXiv preprint arXiv:1805.12114*, 2018.

Peter Dayan and Geoffrey E Hinton. Using expectation-maximization for reinforcement learning. *Neural Computation*, 9(2):271–278, 1997.

Pierre Del Moral. Non-linear filtering: interacting particle resolution. *Markov processes and related fields*, 2(4): 555–581, 1996.

Arnaud Doucet, Nando De Freitas, and Neil Gordon. An introduction to sequential monte carlo methods. In *Sequential Monte Carlo methods in practice*, pp. 3–14. Springer, 2001.

Chelsea Finn and Sergey Levine. Deep visual foresight for planning robot motion. In *Robotics and Automation (ICRA), 2017 IEEE International Conference on*, pp. 2786–2793. IEEE, 2017.

Shixiang Gu, Zoubin Ghahramani, and Richard E Turner. Neural adaptive sequential monte carlo. In *Advances in Neural Information Processing Systems*, pp. 2629–2637, 2015.

Xiaoxiao Guo, Satinder Singh, Honglak Lee, Richard L Lewis, and Xiaoshi Wang. Deep learning for real-time atari game play using offline monte-carlo tree search planning. In *Advances in neural information processing systems*, pp. 3338–3346, 2014.

Tuomas Haarnoja, Aurick Zhou, Pieter Abbeel, and Sergey Levine. Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor. *arXiv preprint arXiv:1801.01290*, 2018.

Michael Kearns, Yishay Mansour, and Andrew Y Ng. A sparse sampling algorithm for near-optimal planning in large markov decision processes. *Machine learning*, 49(2-3):193–208, 2002.

Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

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

Sergey Levine. Reinforcement learning and control as probabilistic inference: Tutorial and review. *arXiv preprint arXiv:1805.00909*, 2018.

Ofir Nachum, Mohammad Norouzi, Kelvin Xu, and Dale Schuurmans. Bridging the gap between value and policy based reinforcement learning. In *Advances in Neural Information Processing Systems*, pp. 2775–2785, 2017.

Anusha Nagabandi, Gregory Kahn, Ronald S Fearing, and Sergey Levine. Neural network dynamics for model-based deep reinforcement learning with model-free fine-tuning. *arXiv preprint arXiv:1708.02596*, 2017.

Brendan O'Donoghue, Remi Munos, Koray Kavukcuoglu, and Volodymyr Mnih. Combining policy gradient and q-learning. *arXiv preprint arXiv:1611.01626*, 2016.

Konrad Rawlik, Marc Toussaint, and Sethu Vijayakumar. An approximate inference approach to temporal optimization in optimal control. In *Advances in neural information processing systems*, pp. 2011–2019, 2010.

Konrad Rawlik, Marc Toussaint, and Sethu Vijayakumar. On stochastic optimal control and reinforcement learning by approximate inference. In *Robotics: science and systems*, volume 13, pp. 3052–3056, 2012.

RY Rubinstein and DP Kroese. A unified approach to combinatorial optimization, monte-carlo simulation, and machine learning. *Springer-Verlag New York, LLC*, 2004.

Simo Särkkä. *Bayesian filtering and smoothing*, volume 3. Cambridge University Press, 2013.

John Schulman, Xi Chen, and Pieter Abbeel. Equivalence between policy gradients and soft q-learning. *arXiv preprint arXiv:1704.06440*, 2017.

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

István Szita and András Lörincz. Learning tetris using the noisy cross-entropy method. *Neural computation*, 18(12): 2936–2941, 2006.

Emanuel Todorov and Weiwei Li. A generalized iterative lqg method for locally-optimal feedback control of constrained nonlinear stochastic systems. In *American Control Conference, 2005. Proceedings of the 2005*, pp. 300–306. IEEE, 2005.

Emanuel Todorov, Tom Erez, and Yuval Tassa. Mujoco: A physics engine for model-based control. In *Intelligent Robots and Systems (IROS), 2012 IEEE/RSJ International Conference on*, pp. 5026–5033. IEEE, 2012.

Marc Toussaint. Robot trajectory optimization using approximate inference. In *Proceedings of the 26th annual international conference on machine learning*, pp. 1049–1056. ACM, 2009.

Marc Toussaint and Amos Storkey. Probabilistic inference for solving discrete and continuous state markov decision processes. In *Proceedings of the 23rd international conference on Machine learning*, pp. 945–952. ACM, 2006.

Brian D Ziebart. Modeling purposeful adaptive behavior with the principle of maximum causal entropy. 2010.

A APPENDIX

A.1 ABBREVIATION AND NOTATION

Table A.1: Abbreviation

SMCP:	Sequential Monte Carlo Planning.
SAC:	Soft Actor Critic.
CEM:	Cross Entropy Method.
RS:	Random Shooting.
MCTS:	Monte Carlo Tree Search.
SMC:	Sequential Monte Carlo.
SIR:	Sequential Importance Resampling.
SIS:	Sequential Importance Sampling.

Table A.2: Notation

$\mathbf{x}_{1:T}$	$\triangleq$	$\{s_i, a_i\}_{i=1}^T$.
$\mathcal{O}_t$	$\triangleq$	Optimality variable.
$p(\mathcal{O}_t \vert s_t, a_t)$	$\triangleq$	$\exp(r(s_t, a_t))$ Probability of a pair state action of being optimal.
p_{env}	$\triangleq$	Transition probability of the environment. Takes state and action (s_t, a_t) as argument and return next state and reward (s_{t+1}, r_t).
p_{model}	$\triangleq$	Model of the environment. Takes state and action (s_t, a_t) as argument and return next state and reward (s_{t+1}, r_t).
w	$\triangleq$	Normalized weight of the importance sampling.
$p(\mathbf{x})$	$\triangleq$	Density of interest.
$q(\mathbf{x})$	$\triangleq$	Approximation of the density of interest.
$t \in \{1, \dots T\}$	$\triangleq$	time steps.
$n \in \{1, \dots N\}$	$\triangleq$	particle number.

A.2 RECURSIVE WEIGHTS

$$
\begin{aligned}
w_t(\mathbf{x}_{1:t}) &= \frac{p(\mathbf{x}_{1:t} \vert \mathcal{O}_{1:t})}{q(\mathbf{x}_{1:t})} \\
&= \frac{p(\mathbf{x}_{1:t-1}) \vert \mathcal{O}_{1:t-1} p_{\text{env}}(\mathbf{s}_{t+1} \vert \mathbf{s}_t, \mathbf{a}_t) p(\mathcal{O}_t \vert \mathbf{s}_t, \mathbf{a}_t}{q(\mathbf{x}_{1:t-1}) q(\mathbf{x}_t \vert \mathbf{x}_{1:t-1})} \\
&= \frac{p(\mathbf{x}_{1:t-1}) \vert \mathcal{O}_{1:t-1})}{q(\mathbf{x}_{1:t-1})} \frac{p_{\text{env}}(\mathbf{s}_{t+1} \vert \mathbf{s}_t, \mathbf{a}_t) \exp(r_t)}{p_{\text{model}}(\mathbf{s}_{t+1} \vert \mathbf{s}_t, \mathbf{a}_t) \pi_\theta(\mathbf{a}_t \vert \mathbf{s}_t)} \\
&= w_{t-1} \frac{p_{\text{env}}(\mathbf{s}_{t+1} \vert \mathbf{s}_t, \mathbf{a}_t))}{p_{\text{model}}(\mathbf{s}_{t+1} \vert \mathbf{s}_t, \mathbf{a}_t)} \exp(r_t - \log \pi_\theta(\mathbf{a}_t \vert \mathbf{s}_t))
\end{aligned} \tag{A.1}
$$

A.3 VALUE FUNCTION: BACKWARD MESSAGE

.

$$
\begin{aligned}
p(\mathcal{O}_{1:t+1}|\mathbf{x}_t) &= \int_{\mathbf{x}_{t+1}} p(\mathbf{x}_{t+1}|\mathbf{x}_t) \exp Q(\mathbf{s}_{t+1}, \mathbf{a}_{t+1}) d\mathbf{x}_{t+1} \\
&= \int_{\mathbf{s}_{t+1}} p_{\text{env}}(\mathbf{s}_{t+1}|\mathbf{s}_t, \mathbf{s}_t) \left[\int_{\mathbf{a}_{t+1}} p(\mathbf{a}_{t+1}|\mathbf{x}_t) \exp Q(\mathbf{s}_{t+1}, \mathbf{a}_{t+1}) d\mathbf{a}_{t+1} \right] d\mathbf{s}_{t+1} \\
&= \int_{\mathbf{s}_{t+1}} p_{\text{env}}(\mathbf{s}_{t+1}|\mathbf{s}_t, \mathbf{a}_t) \exp V(\mathbf{s}_{t+1}) d\mathbf{s}_{t+1} \qquad \text{(A.2)}
\end{aligned}
$$

By definition of the value function in (Levine, 2018).

A.4 EXPERIMENT DETAILS

Random samples: 1000 transitions are initially collected a random policy to pretrain the model and the proposal distribution. After which the agents start following their respective policy.

Data preprocessing: We normalize the observation to have zero mean and standard deviation 1. We add zero mean Gaussian noise to the training data for the model to improve its robustness.

Architecture: For the model with used 3 hidden layers with 256 hidden units as in Chua et al. (2018) and ReLU activation. For the agent we used a single hidden layer with 256 hidden units for the three networks: the value function, the policy and the soft Q function.

Model Predictive Control: The model is used to predict the return for the horizon H of N particles. We then sample or take the best trajectory and return the first action of this trajectory. In our experiments, we fix the maximum number of particles for every methods to 2500. For SMCP, the temperature was fixed to 10.

Soft Actor Critic: We used a custom implementation with a Gaussian policy for both the SAC baseline and the proposal distribution used for both versions of SMCP. We used Adam (Kingma & Ba, 2014) with a learning rate of 0.0003.

Model: We train the model p_{model} to minimize the negative log likelihood of $p(s_{t+1}|s_t + \delta_t(s_t, a_t), \sigma_t(s_t, a_t))$. For every experiment, we used a fully connected neural network with 3 hidden layers of 256 hidden units. We train the model to predict the distribution on the next state and learn a deterministic reward function from the current state and simulated next state. Additionally, we manually add the penalty on action magnitude. At the end of each episode we train the model for 10 epochs. Since the training is fairly short, we stored every transitions into the buffer. The model is defined as:

$$
\Delta s_t \sim p(\cdot|s_t, a_t) \qquad \text{(A.3)}
$$

$$
r_t = g(s_t, \Delta s_t) - \alpha \|a\|^2 \qquad \text{(A.4)}
$$

where α was taken from the Mujoco gym environments. Each neural network has 256 hidden units and 2 hidden layers. We used Adam (Kingma & Ba, 2014) with a learning rate of 0.001.

Environment	**Temperature**	**Horizon length**	**SAC Reward Scaling**	**Reward Function** (α)
Hopper	10	25	5	0.001
Walker	10	25	5	0.001
Half Cheetah	10	25	5	0.5

A.5 SEQUENTIAL IMPORTANCE SAMPLING PLANNING

Algorithm 2 Sequential Importance Sampling - Planning

for t in $\{0, \ldots, T\}$ **do**
  $\mathbf{s}_t^{(n)} = \mathbf{s}_t, \forall n, 1 \leq n \leq N.$
  $w_0 = 1.$
  for i in $\{t, \ldots, t+h\}$ **do**
    $\{\mathbf{a}_i^{(n)} \sim \pi^{SAC}(\mathbf{a}_i^{(n)} | \mathbf{s}_i^{(n)})\}_{n=0}^N$
    $\{\mathbf{s}_{i+1}^{(n)}, r_i^{(n)} \sim p_{\text{model}}(\cdot | \mathbf{s}_i^{(n)}, \mathbf{a}_i^{(n)})\}_{n=0}^N$
    $w_i = w_{i-1} \cdot \exp(r_i - \lambda \log \pi^{SAC}(\mathbf{a}_i | \mathbf{s}_i))$
  end for
  $\hat{w}_T \propto w_{t+h} \cdot \exp(V^{SAC}(\mathbf{s}_{t+h+1}))$
  Sample $n \sim$ Categorical$(\hat{w}_T^{(1)}, \ldots, \hat{w}_T^{(N)})$.
  Select $\mathbf{a}_t$, first action of $\mathbf{x}^{(n)}$.
  $\mathbf{s}_{t+1}, r_t \sim p_{\text{env}}(\cdot | \mathbf{s}_t, \mathbf{a}_t)$
  $\mathbf{s}_t \leftarrow \mathbf{s}_{t+1}.$
end for