Employing Monte Carlo Methods in Model-Based Reinforcement Learning Progress Report

Orhan Sönmez
orhan.sonmez@boun.edu.tr

January 12, 2018

Abstract

Most of the current state-of-the-art reinforcement learning algorithms are based on the Bellman equations and utilize variety of methods to converge to the optimal solution. However, some of the recent approaches transform the reinforcement learning problem into an equivalent likelihood maximization problem, such that it allows the adoption of classical probabilistic inference methods. Similarly, in this thesis we focus on the employment of Monte Carlo methods as approximate probabilistic inference tools in order to solve the reinforcement learning problem.

We propose the RLEMMC algorithm which is an expectation maximization algorithm that uses a Monte Carlo sampling scheme in its expectation step. Then, we derive and employ different Monte Carlo algorithms such as importance sampling, Metropolis-Hastings, sequential Monte Carlo samplers and sequentially interacting Markov chain Markov for this setting. For the policy representation, we utilize a k-nearest neighbour approximation and likewise we derive the corresponding maximization step. Finally, we applied our algorithms to some of the benchmark reinforcement learning problems in the OpenAI framework.

1 Introduction

Reinforcement learning is one of the hot topics in machine learning which can casually be described as finding a strategy that can determine the optimal decisions for an agent in acting an environment. Its main difference from the supervised learning is that the agent only uses its own experience gained through the interactions with the environment instead of a given training set.

By its definition, reinforcement learning is a sequential decision making under uncertainty with autonomous learning and it is defined over Markov decision processes (MDPs). Unfortunately, other than linear dynamical systems with Gaussian noises, it doesn't have a closed form solution [3].

Hence, it had been widely solved by converging to the solutions of the Bellman equations using different optimization techniques [1],[25]. These approaches are called model-free approaches where the dynamics of the environment aren't taken into consideration. Instead, the utility of each state or each state-action pair is estimated. Later, the policy of the agent is determined using these estimated utility values.

Furthermore, another type of approach is the model-based approach where the system dynamics of the environment are also modelled and estimated. Later, they are used in the process of learning the optimal policy. Here, we propose a similar model-based reinforcement learning algorithm that adopts a probabilistic approach for learning the optimal policy.

1.1 Related Work

The probabilistic approaches to the reinforcement learning problem have also been gaining more attention. The seminal work of these approaches is [6] where an expectation-maximization (EM) [13] algorithm is used for the first time to solve the problem. Then, Ng and Jordan [20] proposed to transform any arbitrary partially observable Markov decision process (POMDP) into an equivalent POMDP with deterministic transitions. They introduced a policy iteration scheme that approximates the value function of each policy by averaging over scenarios that sample the initial state of POMDP and calculating the total future reward deterministically. Also, Attias [2] used the Viterbi algorithm to solve a planning problem and obtain theoretical results for the discrete state space finite horizon case.

Toussaint and Storkey [26]. By defining a mixture model over MDPs, they introduced an equivalent probabilistic reformulation and an EM based inference to the reinforcement learning problem. In the E-step of this EM algorithm, they utilized a forward-pass filter to approximate the state distribution for each future time step, while truncating the distant horizons. Finally, the expected sufficient statistics are computed under the posterior state distributions. Later on, Furmston and Barber [14] introduced Q-functions which are more efficient ways to calculate the state distributions by using both forward and backward passes.

The probabilistic formulation introduced by [26] has been slightly modified by several other methods. Vlassis and Toussaint [27] included auxiliary random variables defined over the horizon of MDPs and enhanced the efficiency of the model by enabling the adoption of intermediate rewards in addition to terminal rewards. Moreover, a stochastic approximation EM (SAEM) is introduced by them to resolve the model-free setting of the reinforcement learning problem. Also, Rawlik et. al [23] extended the graphical model with binary reward random variables whose probabilities are inversely proportional to exponentials of their corresponding costs. Finally, they proposed a fixed-point iteration scheme.

Monte Carlo sampling is a viable approximation technique in higher dimensions and many such sampling methods have been proposed for the planning problem. Hoffman and Jasra[16] utilized a reversible jump Markov chain Monte Carlo (RJMCMC) [15] to estimate the corresponding expectation in the E-step of the EM algorithm introduced by [26] for the infinite horizon case. In order to do so, they sampled from the posterior distribution with RJMCMC at each iteration of the EM algorithm and used these samples to obtain a Monte Carlo estimate of the sufficient statistics. Later, Hoffman et. al [17] increased the efficiency of this work by introducing a new target distribution and freeing the policy parameters from the trajectory sampling. De Villiers et. al [7] introduced an artificial extended space which consists of states and control signals. Then, they employed Monte Carlo methods to solve state inference and control optimization simultaneously. Lazaric et. al [18] utilized sequential Monte Carlo methods in a model-free actor-critic technique to estimate the policy of the actor given the value function approximation by the critic.

Gaussian process (GP) [21] is another approximation method that is utilized in reinforcement learning after the initial work of Rasmussen and Kuss [22]. After that, Deisenroth et. al [10] utilized GPs to approximate the value function over the state space and then use it to determine the optimal policy. Later, in PILCO [11] adopted GPs both for learning the system dynamics of the environment and representing the policy. Using the properties of Gaussian distributions, they came up with a closed-form solution to estimate the policy gradient. Finally, they iteratively improved the policy using the estimate gradient information. This approach and its extensions are applied to robotics due to its cost effectiveness [9],[8].

2 Problem Definition

The reinforcement learning problem is defined over MDPs as finding the policy of the decision making agent that maximizes total expected reward received by the agent during the process. Other than some special cases, reinforcement learning problems can not be solved in closed form. Thus, they have been generally solved by approaching the solutions of the Bellman equations in different manners [3]. However, we consider this problem as a probabilistic inference problem after an appropriate reformulation and approach it with probabilistic inference literature.

In the upcoming subsections, we are going to foremost introduce MDPs and the definition of the reinforcement learning problem over them. Then, we are going to finalize the section after presenting the probabilistic inference reformulation of the reinforcement learning problem and the corresponding new maximization problem.

2.1 Markov Decision Processes

MDPs are frameworks to model the behaviour of a utility-based agent in an environment. Let, $\mathcal{X}$ and $\mathcal{A}$ are the state and the action spaces respectively, the process starts with the agent in an initial state of $x_0 \in \mathcal{X}$ introduced by the environment. From there on, at each time step t, the agent is at a state $x_t \in \mathcal{X}$ and makes an action $a_t \in \mathcal{A}$ determined by its policy π. With respect to these, the agent receives a reward $r_t \in \mathbb{R}$ and transits into the next state $x_{t+1} \in \mathcal{X}$ for time step $t + 1$. The process continues till it reaches its possibly infite horizon T. Formally, the states and actions are generated by an MDP according to the following generative model given a predetermined horizon T,

$$\begin{aligned} x_0 &\sim P(x_0) & & \\ a_t &\sim P(a_t|x_t;\pi) & & t = 0, ..., T \\ x_{t+1} &\sim P(x_{t+1}|x_t, a_t) & & t = 0, ..., T-1 \\ r_t &\sim P(r_t|x_t, a_t) & & t = 0, ..., T \end{aligned} \tag{1}$$

where $P(x_0)$ denotes the initialization model and $P(x_{t+1}|x_t, a_t)$ is the transition model of the environment. Here, action selection procedure of the agent is modelled as a probability distribution $P(a_t|x_t;\pi)$ parametrized by the policy π of the agent and the rewards r_t at each time step are determined according to the utility of the agent by a possibly deterministic reward function $P(r_t|x_t, a_t)$ given the state x_t and action a_t. The graphical model of the MDP can be seen in Figure 1.

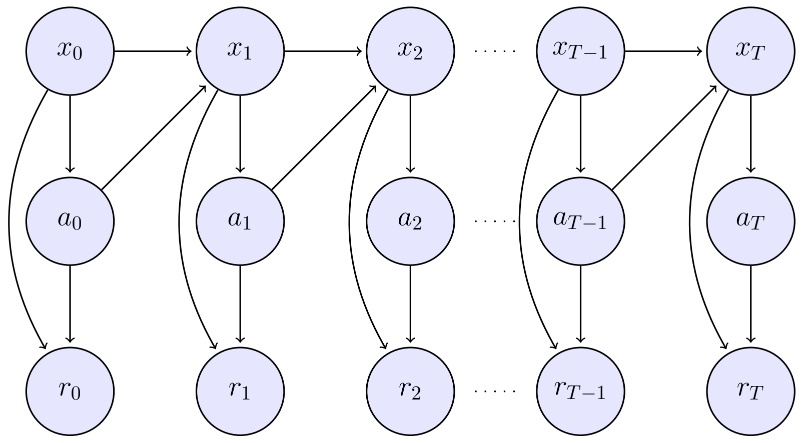


Figure 1: Graphical model of a Markov decision process with an horizon $T = \tau$.

Note that a state-action trajectory with an horizon T has the following full-joint probability to be sampled from a given MDP while following the policy π of the agent,

$$P(x_{0:T}, a_{0:T}; \pi) = P(x_0)\Big[\prod_{t=0}^{T-1} P(a_t|x_t;\pi)P(x_{t+1}|x_t, a_t)\Big]P(a_T|x_T;\pi) \tag{2}$$

where $(x_{0:T}, a_{0:T})$ is a contracted notation of the longer form $(x_0, ..., x_T, a_0, ..., a_T)$.

2.2 Reinforcement Learning

The reinforcement learning problem is to find the optimal policy that maximizes the expected total reward received by the agent over a MDP. In the most general case, the horizon of the MDP can either be finite or infinite. In the finite horizon case usually the sum of the rewards is used directly to represent the total reward of a trajectory. However, in the infinite horizon case where horizon goes to infinity, the sum of rewards would diverge to infinity. In order to avoid this divergence and also to weigh rewards according to their proximity, a discount factor $0 < \gamma \leq 1$ is defined that is used to discount the future rewards exponentially with respect to their time steps.

So, the general reinforcement learning problem is formally defined over

MDPs as the following maximization problem,

$$\pi^* = \arg\max_{\pi} \left\langle \sum_{t=0}^{T} \gamma^t r_t \right\rangle_{P(x_{0:T}, a_{0:T}; \pi)} \tag{3}$$

where $0 < \gamma \leq 1$ for the finite horizon case and $0 < \gamma < 1$ for the infinite case. Here, π^* is called the optimal policy and the goal of the learner is to find this policy.

In our approach, we are working on the planning aspect of the reinforcement problem where we assume the initialization model and the transition model of the environment are either known or already estimated by the agent. Estimation of these distributions is well-known and intentionally out of the scope of this paper.

2.3 Probabilistic Approach

Instead of using the Bellman equations [3], we adopted a probabilistic approach to solve the reinforcement learning problem. We base our approach on the work of [26] that introduced a mixture model defined over the terminal reward MDPs with different horizons that enables the reformulation of the reinforcement learning problem into an equivalent probabilistic inference problem. Graphical models of these terminal reward MDPs are given in Figure 2.

In this new mixture model, the prior probability of a terminal reward MDP is chosen geometrically proportional to its horizon such as,

$$P(T = t) \propto \gamma^t \tag{4}$$

where $P(T = t)$ denotes the probability of terminal reward MDP with an horizon of $T = t$.

Additionally, without loss of generality a stochastic binary reward setting $r \in \{0, 1\}$ is assumed for these terminal reward MDPs. However, this assumption can always be enforced, since any set of deterministic rewards $R(x_t, a_t)$ defined over the state-action space can be scaled to $[0, 1]$ interval and probability of receiving a positive reward $P(r = 1|x_t, a_t)$ could then be set equal to these scaled rewards in order to obtain this setting.

Finally, under these conditions, [26] proved that solving the following maximization problem,

$$\pi^* = \arg\max_{\pi} P(r = 1; \pi) \tag{5}$$

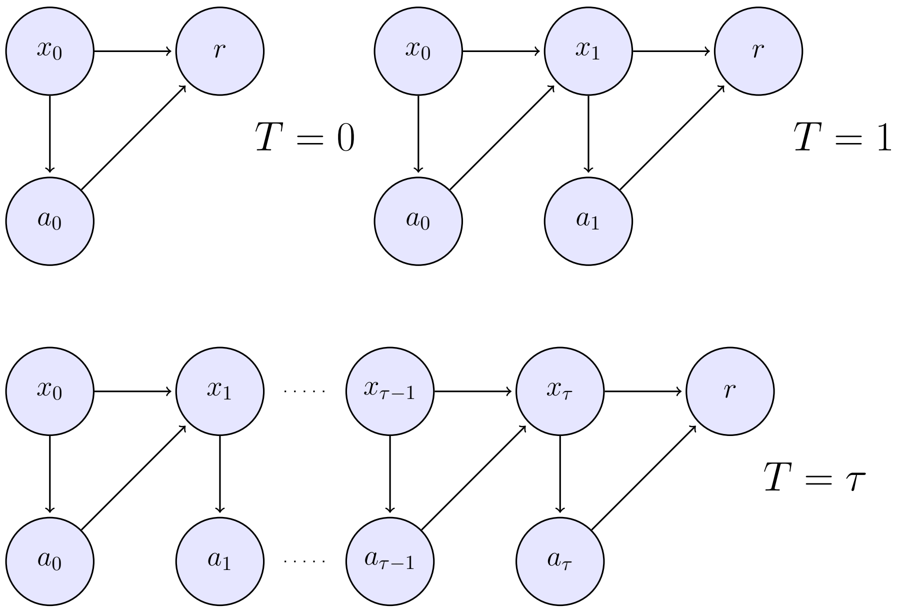


Figure 2: Terminal reward Markov decision processes with horizons of $T = 0$, $T = 1$ and $T = \tau$ as examples.

defined over the new mixture model, is actually equivalent to the reinforcement problem defined in Equation (3). For the sake completeness, we sketch an alternative statement of this proof in Appendix A.

Note that the marginal likelihood in Equation (5) is actually a summation over all possible state-action trajectories as,

$$P(r = 1; \pi) = \sum_{t=0}^{\tau} \sum_{x_{0:t}} \sum_{a_{0:t}} P(r = 1, x_{0:T}, a_{0:T}, T = t; \pi) \qquad (6)$$

and exact calculation is typically intractable. Thus, approximate probabilistic inference methods have to be employed in order to find the optimal policy π^*.

3 Method

We propose an expectation-maximization based algorithm for the planning part of the reinforcement learning problem where the system dynamics are either known or already estimated. The main contribution of our algorithm is the adoption of different Monte Carlo methods such as importance sampling, Metropolis-Hastings, sequential Monte Carlo Samplers [12] and sequentially interacting Markov chain Monte Carlo (SIMCMC) [4] in the E-step of the expectation-maximization algorithm derived to solve the RL problem.

3.1 Expectation-Maximization

Toussaint and Storkey [26] derived an EM algorithm to maximize the likelihood defined in Equation (5). The details of this derivation is sketched in Appendix B for the completeness of the paper. The update rule of the corresponding EM algorithm is derived as,

$$\pi^{(k+1)} \leftarrow \arg\max_{\pi} \langle \log P(r=1, x_{0:T}, a_{0:T}, T; \pi) \rangle_{P(x_{0:T}, a_{0:T}, T | r=1; \pi^{(k)})} \quad (7)$$

where superscript (k) denotes the iteration index of the EM algorithm. At the E-step of this EM algorithm, we sample from the posterior distribution and then at the M-step of it, we estimate the maximizer depending on the policy representation.

3.1.1 Expectation Step

As the exact calculation of this expectation is intractable due to curse of the dimensionality, at each E-step we sample from the posterior distribution with respect to the current policy $\pi^{(k)}$ as,

$$(x_{0:T}^{(s)}, a_{0:T}^{(s)}, T^{(s)}) \sim P(x_{0:T}, a_{0:T}, T | r=1; \pi^{(k)}) \qquad s = 1, ..., S \quad (8)$$

where the superscript (s) denotes the sample index and S is the total sample count.

For E-step of the EM algorithm, we propose different sampling schemes. For the sake of explicitness, we skip the sampling process to the next section and now and assume it is possible to sample from the posterior distribution. After the sampling, we can explicitly estimate the expectation term in the update rule in a Monte Carlo manner.

$$\langle \log P(r=1, x_{0:T}, a_{0:T}, T; \pi) \rangle \approx \frac{1}{S} \sum_{s=1}^{S} \log P(r=1, x_{0:T}^{(s)}, a_{0:T}^{(s)}, T^{(s)}; \pi) \quad (9)$$

where the expectation is over the posterior distribution $P(x_{0:T}, a_{0:T}, T | r=1; \pi^{(k)})$ as defined in Equation (7).

3.1.2 Maximization Step

At each M-step of the EM algorithm, after sampling from the posterior distribution in Equation (8) in the E-step, we have to maximize the approximated expectation with respect to the parameters of the policy π as given below.

$$\pi \leftarrow \arg\max_{\pi} \frac{1}{S} \sum_{s=1}^{S} \log P(r=1, x_{0:T}^{(s)}, a_{0:T}^{(s)}, T^{(s)}; \pi) \tag{10}$$

As the maximization will be held with respect to the policy parameters, keeping only the terms related to π, the maximization problem simplifies into the following,

$$\pi^{(k)} \leftarrow \arg\max_{\pi} \frac{1}{S} \sum_{s=1}^{S} \sum_{t=0}^{T^{(s)}} \log P(a_t^{(s)} | x_t^{(s)}; \pi) \qquad \text{(Terms related with } \pi^{(k)}\text{)} \tag{11}$$

which means that any policy $\pi^{(k)}$ such that $\forall t \forall s\ \pi^{(k)}(x_t^{(s)}) = a_t^{(s)}$ is a maximizer. Thus, M-step of the EM algorithm actually turns into a policy learning problem given the desired state-action mappings.

In our approach, we employ a k-nearest neighbour approximation for the policy. Hence, the policy can be approximated simply by picking the sampled states as the data points of the approximator and their corresponding actions as their values.

3.2 Algorithm

Applying the Monte Carlo approximation introduced in Section 3.1.1 to the EM algorithm with the update rule of Equation (7), we propose the RLEMMC algorithm to solve the reinforcement learning problem as follows,

Algorithm 1 RLEMMC

```
Initial Policy $\pi \leftarrow$ Uniform policy
while $\pi$ not converged do
    E-step: Sample $x_{0:T}^{(s)}, a_{0:T}^{(s)}, T^{(s)}$ from posterior $P(x_{0:T}, a_{0:T}, T | r = 1; \pi)$
    M-step: Learn policy $\pi$ using samples $x_{0:T}^{(s)}, a_{0:T}^{(s)}, T^{(s)}$
end while
Optimal Policy: $\pi^* \leftarrow \pi$
```

However, as mentioned before sampling from this posterior is not a trivial task. In order to do, we have applied several different Monte Carlo techniques and come up with different variations of RLEMMC algorithm. The next section of this report is devoted to these Monte Carlo techniques and their application to our setting.

4 Monte Carlo Methods

4.1 Importance Sampling

If we would like to estimate a target probability distribution $p(x)$ by actually sampling from a proposal probability distribution $q(x)$, importance sampling is intuitively the first approach to be considered.

The importance sampling process starts with sampling a predefined S samples from the proposal distribution $q(x)$.

$$x^{(s)} \sim q(x) \qquad\qquad s = 1, ..., S \tag{12}$$

Then, depending on both the target and the proposal distributions, the weight of each sample $w(x^{(s)})$ is calculated as,

$$w(x^{(s)}) = \frac{p(x^{(s)})}{q(x^{(s)})} \tag{13}$$

Furthermore, in order to obtain a probability distribution, the weights are normalized and the normalized weight of each sample $W(x^{(s)})$ is calculated.

$$W(x^{(s)}) = \frac{w(x^{(s)})}{\sum_s w(x^{(s)})} \tag{14}$$

Finally, the empirical distribution $\hat{p}(x)$ that approximates the target distribution $p(x)$ is obtained as a weighted sum of delta-dirac functions located at the samples.

$$\hat{p}(x) = \sum_s W(x^{(s)}) \delta_{x^{(s)}}(x) \tag{15}$$

4.1.1 Application to RLEMMC

Once we have a policy, it is straightforward to follow the policy and generate some sample trajectories. Thus, we picked our prior distribution as our proposal distribution for the importance sampling scheme.

$$q(x_{0:T}, a_{0:T}, T) = P(x_{0:T}, a_{0:T}, T; \pi) \tag{16}$$

and we obtain each sample $s = (x_{0:T}^{(s)}, a_{0:T}^{(s)}, T^{(s)})$ as follows,

$$\begin{aligned}
T^{(s)} &\sim P(T) \\
x_0^{(s)} &\sim P(x_0) \\
a_t^{(s)} &\sim P(a_t | x_t = x_t^{(s)}; \pi) && 0 \leq t \leq T^{(s)} \\
x_{t+1}^{(s)} &\sim P(x_t | x_t = x_t^{(s)}, a_t = a_t^{(s)}) && 0 \leq t \leq T - 1^{(s)} \qquad (17)
\end{aligned}$$

Weight Function Afterwards, calculating the weight function for the importance sampling is trivial due to our choice of prior distribution as the proposal distribution, using the Bayes rule and the Markov property of MDPs.

$$w(x_{0:T}^{(s)}, a_{0:T}^{(s)}, T^{(s)}) = \frac{P(x_{0:T}^{(s)}, a_{0:T}^{(s)}, T^{(s)} | r = 1; \pi)}{P(x_{0:T}^{(s)}, a_{0:T}^{(s)}, T^{(s)}; \pi)} = P(r = 1 | x_T^{(s)}, a_T^{(s)}, T^{(s)}) \qquad (18)$$

Then, the normalized weights can be calculated by dividing each individual weights to the sum of all the weights.

$$W(x_{0:T}^{(s)}, a_{0:T}^{(s)}, T^{(s)}) = \frac{w(x_{0:T}^{(s)}, a_{0:T}^{(s)}, T^{(s)})}{\sum_s w(x_{0:T}^{(s)}, a_{0:T}^{(s)}, T^{(s)})} \qquad (19)$$

Resampling Normally, these normalized weights are already enough to be used at the estimation of the desired expectation at the E-step of the RLEMMC algorithm. However, since at the M-step of the algorithm, we would like to learn the policy given state-action mappings, having unweighed samples generally eases the learning process depending on our choice of the policy approximation. As we employ k-nearest neighbour approximation for our policy, it is desired to have unweighed data points. So, we resample the samples with respect to their with respect to their normalized weights $W(x_{0:T}^{(s)}, a_{0:T}^{(s)}, T^{(s)})$ to have an unweighted Monte Carlo estimate.

Algorithm Finally, we propose our importance sampling derivation for the RLEMMC algorithm as in Algorithm 2.

Algorithm 2 RLEMMC-IS

```
Initial Policy $\pi \leftarrow$ Uniform policy
while $\pi$ not converged do
    Sample $x^{(s)}_{0:T}, a^{(s)}_{0:T}, T^{(s)}$ from the prior $P(x_{0:T}, a_{0:T}, T; \pi)$
    Weigh each sample such $w(x^{(s)}_{0:T}, a^{(s)}_{0:T}, T^{(s)}) = P(r = 1|x^{(s)}_{T}, a^{(s)}_{T}, T^{(s)})$
    Normalize the weights to obtain $W(x^{(s)}_{0:T}, a^{(s)}_{0:T}, T^{(s)})$
    Resample samples w.r.t. $W(x^{(s)}_{0:T}, a^{(s)}_{0:T}, T^{(s)})$
    Learn policy $\pi$ using resampled samples $x^{(s)}_{0:T}, a^{(s)}_{0:T}, T^{(s)}$
end while
Optimal Policy: $\pi^* \leftarrow \pi$
```

4.2 Metropolis-Hastings

Metropolis-Hastings [19] is a Monte Carlo technique to sample for a target distribution $p(x)$ with using a proposal function $q(x)$.

In Metropolis-Hastings algorithm, sampling process is held sequentially. After starting with an arbitrary sample $x^{(0)}$ the support of $p(x)$, in order to obtain the next sample, first of all a sample candidate is proposed by the proposal function using the current sample $x^{(s)}$. To sample the next sample $x^{(s+1)}$,

$$x' \sim q(x'|x^{(s)})$$

where x' is the candidate sample and the superscript s denotes the index of the current sample.

Later, the candidate sample is accepted or rejected as the next sample with respect to the following acceptance probability,

$$\alpha_{x^{(s)} \to x'} = \min\left(1, \frac{p(x')q(x^{(s)}|x')}{p(x)q(x'|x^{(s)})}\right)$$

such that $x^{(s+1)} = x'$ with probability $\alpha_{x^{(s)} \to x'}$ and $x^{(s+1)} = x^{(s)}$ otherwise.

4.2.1 RLEMMC Derivation

For RLEMMC algorithm, we need to sample trajectories from the posterior $P(x_{0:T}, a_{0:T}, T|r = 1; \pi)$ at the E-step of the algorithm. To shorten the notation, we call a trajectory sample simply as z such that $z \equiv (x_{0:T}, a_{0:T}, T)$. Hence, we now have to aim to sample from $P(z|r = 1; \pi)$.

In our approach, we derived two different Metropolis-Hastings sampling schemes with two different proposal functions for the RLEMMC algorithm.

4.2.2 Prior as Proposal

A naive but to the point selection of the proposal would be selecting the prior distribution as our proposal function that disregards the given trajectory and samples a new trajectory from the scratch.

$$q_{prior}(z) \equiv P(z'; \pi)$$

Acceptance Probability When this is the case, using the Bayes Rule, the acceptance probability simplifies into minimum of 1 and the ratio of the reward probabilities of the current and the candidate trajectory samples.

Acceptance Probability

$$\begin{aligned}
\alpha_{z^{(s)} \to z'} &= \min\left(1, \frac{p(z')q(z^{(s)}|z')}{P(z^{(s)})q(z'|z^{(s)})}\right) \\
&= \min\left(1, \frac{P(z'|r=1;\pi)P(z^{(s)};\pi)}{P(z^{(s)}|r=1;\pi)P(z';\pi)}\right) \\
&= \min\left(1, \frac{P(r=1|z')P(z';\pi)P(z^{(s)};\pi)}{P(r=1|z^{(s)})P(z^{(s)};\pi)P(z';\pi)}\right) \\
&= \min\left(1, \frac{P(r=1|z')}{P(r=1|z^{(s)})}\right)
\end{aligned}$$

4.2.3 Keep-Then-Prior as a Proposal

A slightly more complex approach choice of a proposal would be keeping some random amount of states and actions at the beginning of the given trajectory and sampling the rest from the prior. This enables to keep some information from the sampled trajectories meanwhile still having an easy to compute acceptance probability. The sampling process of this proposal function can be formulated subsequently,

$$\begin{aligned}
\tau &\sim U[1...T] \\
(x'_{0:\tau}, a'_{0:\tau-1}) &= (x_{0:\tau}, a_{0p:\tau-1}) \\
a'_t &\sim P(a_t|x'_t;\pi) && t = \tau, ..., T \\
x'_t &\sim P(x_t|x'_{t-1}, a'_{t-1}) && t = \tau+1, ..., T \\
z' &= (x'_{0:T}, a'_{0:T}, T)
\end{aligned}$$

Acceptance Probability Also, in the similar manner like the selection of prior as the proposal, the acceptance probability simplifies into minimum of 1 and the ratio of the reward probabilities of the current and the candidate trajectory samples.

$$\alpha_{z^{(s)} \to z'} = \min \left(1, \frac{P(r = 1|z')}{P(r = 1|z^{(s)})} \right)$$

4.2.4 Algorithm

Hence, without depending on the selection of the proposal function, our Metropolis-Hastings derivation for the RLEMMC algorithm is obtained as follows,

Algorithm 3 RLEMMC-MH

```
1: $\pi \leftarrow$ Uniform policy
2: **while** $\pi$ not converged **do**
3:     Sample the candidate trajectory $z' \sim q(z'|z^{(s)})$
4:     Calculate the acceptance probability $\alpha_{z^{(s)} \to z'}$
5:     **if** Accept **then**
6:         $z^{(s+1)} \leftarrow z'$
7:     **else**
8:         $z^{(s+1)} \leftarrow z^{(s)}$
9:     **end if**
10:    Learn policy $\pi$ using the samples $z^{(s)}$
11: **end while**
12: Optimal Policy: $\pi^* \leftarrow \pi$
```

4.3 Sequential Monte Carlo Samplers

Sequential Monte Carlo samplers (SMCS) [12] is a sequential Monte Carlo technique that utilizes sequential importance sampling and resampling through auxiliary densities in order to sample from a target density. Its main advantage is that an extended artificial space is also defined to allow arbitrary Markov kernels to be used in proposal functions without sacrificing the closed form property.

Before sampling from an arbitrary target distribution $p_{target}(z)$ using SMCS, we firstly require an arbitrary proposal distribution $p_{source}(z)$ that we can directly sample from. Then, a predefined number of auxiliary bridge densities ϕ_n should be declared,

$$p_{source}(z) \propto \phi_0(z_0) \rightarrow \phi_1(z_1) \rightarrow ... \rightarrow \phi_N(z_N) \propto p_{target}(z) \tag{20}$$

such that samples generated by each bridge density is used in the sampling process of its successor density. Here, N denotes the total number of auxiliary bridge densities including the proposal distribution.

Furthermore, an artificial target joint distribution $\tilde{\phi}_N(z_{0:N})$ is defined over an an extended space of $\{z_0, ..., z_N\}$ as follows,

$$\tilde{\phi}_N(z_{0:N}) = \phi_N(z_N) \prod_{k=0}^{N-1} L_k(z_k|z_{k+1}) \tag{21}$$

where $L_k(z_k|z_{k+1})$ for $k = 0, 1, ..., N-1$ are arbitrary backward Markov transition kernels. Remark that, by construction the marginal of the distribution is,

$$\sum_{z_{0:N-1}} \tilde{\phi}_N(z_{0:N}) = \phi_N(z_N) \tag{22}$$

where $\phi_N(z_N)$ is explicitly set proportional to the actual target distribution $p_{target}(z)$. Similarly, other auxiliary distributions are introduced for $n = 0, 1, ..., N-1$,

$$\tilde{\phi}_n(z_{0:n}) = \phi_n(z_n) \prod_{k=0}^{n-1} L_k(z_k|z_{k+1}) \tag{23}$$

Now, let us consider how to generate samples from each of these extended spaces. By definition $\tilde{\phi}_0(z_0)$ is equivalent to $\phi_0(z_0)$ and it can be approximated by $\hat{\phi}_0(z_0)$ simply by using the samples drawn from the source density p_{source}. For the rest of the bridge densities $\tilde{\phi}_n(z_{0:n})$, assume that we have S samples, namely $\tilde{z}_{0:n-1}^{(s)}$, generated from $\tilde{\phi}_{n-1}(z_{0:n-1})$ with weights $w_{n-1}(z_{0:n-1})$. Later, an arbitrary Markov kernel $K_n(z_n|z_{n-1})$ is defined to be used in the proposal and candidate samples are proposed in the subsequent manner.

$$\begin{aligned} z_n^{*(s)} &\sim K_n(z_n|\tilde{z}_{n-1}^{(s)}) && s = 1, ..., S \\ \tilde{z}_{0:n}^{(s)} &= (z_n^{*(s)}, \tilde{z}_{0:n-1}^{(s)}) && s = 1, ..., S \end{aligned} \tag{24}$$

Note that, all $\tilde{z}_{0:n}^{(s)}$ are actually sampled from the following proposal distribution $\tilde{q}_n(z_{0:n})$.

$$\tilde{q}_n(z_{0:n}) = \tilde{\phi}_{n-1}(z_{0:n-1}) K_n(z_n|z_{n-1}) \tag{25}$$

Thus, using this samples to generate samples from $\tilde{\phi}_n(z_{0:n})$ with importance sampling, the samples have to be weighted and the weight function $W_n(z_{0:n})$ is determined as,

$$w_n(z_{0:n}) = w_{n-1}(z_{0:n-1})\frac{\tilde{\phi}_n(z_{0:n})}{\tilde{q}_n(z_{0:n})} = w_{n-1}(z_{0:n-1})\frac{\phi_n(z_n)L_{n-1}(z_{n-1}|z_n)}{\phi_{n-1}(z_{n-1})K_n(z_n|z_{n-1})} \tag{26}$$

after expanding $\tilde{\phi}_n(z_{0:n})$ and $\tilde{q}_n(z_{0:n})$ terms and simplifying the equation. Here, the tricky part is the selection of appropriate L_{n-1} and K_n Markov kernels in order to keep the calculation of the weight function $w_n(z_{0:n})$ in closed form.

Propagating samples generated from the source distribution p_{source} through the bridge densities in the preceding manner, weighted samples are obtained from the artificial extended distribution $\tilde{\phi}_N(z_{0:N})$. As noted in Equation (22), the marginal of this distribution is equivalent to $\phi_N(z_N)$ which is equivalent to our target distribution p_{target} by definition. Hence, particles $\tilde{z}_N^{(s)}s$ of $\tilde{z}_{0:N}^{(s)}$ are actually samples generated from our target density p_{target} with corresponding weights.

4.3.1 Application to RLEMMC

Before starting to explain our proposed SMCS method, we define z as,

$$z \equiv (x_{0:T}, a_{0:T}, T) \tag{27}$$

in order to simplify the notation. Without changing anything else, further in this section we will address state-action trajectories simply as z.

Source and Target Densities We employed SMCS in the E-step of the EM algorithm derived for the reinforcement learning problem. The corresponding update rule of this algorithm in each iteration is given in Equation (7). According to that, we have to sample from the subsequent target posterior,

$$p_{target}(z) \equiv P(z|r = 1; \pi) \tag{28}$$

using SMCS as in Equation (8). Note that, each sample here actually correspond to a whole state-action trajectory.

As the first step of constructing a SMCS, we first have to pick a source density and then define the auxiliary bridge densities. We chose our source density as the prior distribution $P(z_n;\pi)$ as we can simply sample from it just by following a given policy π.

$$p_{source}(z) \equiv P(z;\pi) \tag{29}$$

Bridge Densities Then, the bridge densities ϕ_n for are constructed via tempering between the source and the target densities as follows,

$$\phi_n(z_n) = P(z_n;\pi)^{1-\eta(n)} P(z_n|r=1;\pi)^{\eta(n)} \tag{30}$$

for a monotonically increasing function of $\eta(n)$ with values of $\eta(0) = 0$, $\eta(N) = 1$. With this construction, the first bridge function ϕ_0 becomes equivalent to our source density and the last one ϕ_N becomes equivalent to our target density. By construction, the definitions of the bridge densities can be simplified with using the Bayes rule as,

$$\phi_n(z_n) \propto P(z_n;\pi)P(r=1|z_n;\pi)^{\eta(n)} \tag{31}$$

Remark that the latter term here can easily be calculated using the Markov property of MDPs as follows,

$$P(r=1|z_n;\pi) = P(r=1|x_T,a_T) \tag{32}$$

where x_T and a_T are the last state and the last action in the state-action trajectory z_n respectively.

Forward and Backward Kernels The tricky part in the SMCS construction is the selection of appropriate forward kernels and backward kernels. In order to increase the sample efficiency, forward kernels K_n has to be selected in a way to generate nice proposal samples with respect to ϕ_n. Hence, we chose K_n as the Metropolis-Hastings kernels that have ϕ_n as their invariant distributions. Then, to calculate the weight in Equation (26) in closed form, appropriate backward kernels have to selected. Hence, we picked L_n to cancel out the forward kernels such that,

$$L_{n-1}(z_{n-1}|z_n) = K_n(z_n|z_{n-1}) \qquad \text{for } n = 1,...,N \tag{33}$$

We designed a straightforward Metropolis-Hastings kernel K_n with invariant distribution ϕ_n. Given a state-action trajectory z_{n-1}, kernel typically picks a time step uniformly at random, keeps the part of it before that time step and then samples the rest of the trajectory from the prior distribution in order to obtain the candidate sample z'_n. Finally, the candidate sample is either accepted or rejected as the final sample $\tilde{z}_n$ according to the calculated acceptance probability.

Formally, each kernel K_n generates the candidate sample z'_n as follows,

$$\begin{aligned}
\tau &\sim U[1...T] \\
(x'_{0:\tau}, a'_{0:\tau-1}) &= (x_{0:\tau}, a_{0:\tau-1}) \\
a'_t &\sim P(a_t|x'_t; \pi) && \text{for } t = \tau..T \\
x'_{t+1} &\sim P(x_{t+1}|x'_t, a'_t) && \text{for } t = \tau..T-1 \\
z'_n &\equiv (x'_{0:T}, a'_{0:T})
\end{aligned} \tag{34}$$

Then, the corresponding acceptance probability is calculated as,

$$\alpha(z_{n-1} \to z'_n) = \min\left\{1, \frac{\phi_n(z'_{n-1})q(z_{n-1}|z'_n)}{\phi_n(z_{n-1})q(z'_n|z_{n-1})}\right\} \tag{35}$$

Additionally, due to the choice of sampling from the prior, it further simplifies down to,

$$\alpha(z_{n-1} \to z'_n) = \min\left\{1, \frac{P(r=1|z'_n)^{\eta(n)}}{P(r=1|z_{n-1})^{\eta(n)}}\right\} \tag{36}$$

Finally, with probability $\alpha(z_{n-1} \to z'_n)$ $\tilde{z}_n^{(s)}$ is set to z'_n, otherwise it is set to the previous sample z_{n-1}.

Given previous the trajectory $z_{n-1}^{(s)}$, we sample $\tilde{z}_n^{(s)}$ using the forward kernel such that,

$$\tilde{z}_n^{(s)} \sim K_n(z_n|z_{n-1}^{(s)}) \tag{37}$$

and construct the proposal sample,

$$\tilde{z}_{0:n}^{(s)} \equiv (z_{0:n-1}^{(s)}, \tilde{z}_n^{(s)}) \tag{38}$$

Weight Function The weights are obtained using the weight function w_n defined in Equation (26). Since kernels K_n and L_{n-1} is chosen to cancel out each other, the weight function is simplified to,

$$w_n(z_{0:n}) = w_{n-1}(z_{0:n-1})\frac{P(r=1|z_n)^{\eta(n)}P(z_n;\pi)}{P(r=1|z_{n-1})^{\eta(n-1)}P(z_{n-1};\pi)} \tag{39}$$

Resampling However, as unweighted samples are required, a resampling processes is hold over the samples $\tilde{z}_{0:n}^{(s)}$ after the importance sampling. Hence, the actual particles $\tilde{z}_{0:n}^{(s)}$ from $\tilde{\phi}_n(z_{0:n})$ are sampled from the weighted empirical distribution of the candidate samples.

$$z_{0:n}^{(s)} \sim \sum_{s=1}^{S} W_n(\tilde{z}_{0:n}^{(s)}) \delta_{\tilde{z}_{0:n}^{(s)}}(z_{0:n}) \tag{40}$$

where $W_n(\tilde{z}_{0:n}^{(s)})$ are the normalized weights of each sample obtained as,

$$W_n(\tilde{z}_{0:n}^{(s)}) = \frac{w_n(\tilde{z}_{0:n}^{(s)})}{\sum_{s=1}^{S} w_n(\tilde{z}_{0:n}^{(s)})} \tag{41}$$

Algorithm Hence, the flow of our SMCS derivation for RLEMMC algorithm is as in Algorithm 4.

Algorithm 4 RLEMMC-SMCS

```
1: $\pi \leftarrow$ Uniform policy
2: while $\pi$ not converged do
3:     for $n = 0$ to $N$ do
4:         if $n == 0$ then
5:             Sample $z_0^{(s)}$ following the policy $\pi$
6:         else
7:             Sample $z_n^{(s)}$ from $K_n(z_n|z_{n-1}^{(s)})$
8:         end if
9:         Calculate weights $w_n(z_{0:n})$
10:        Calculate normalized weights $W_n(z_{0:n})$
11:        Resample $z_{0:n}$ using normalized weights $W_n(z_{0:n})$
12:    end for
13:    Learn policy $\pi$ using samples $z_N$
14: end while
15: Optimal Policy: $\pi^* \leftarrow \pi$
```

4.4 Sequentially Interacting Markov Chain Monte Carlo

Sequentially interacting Markov Chain Monte Carlo (SIMCMC) [4] is a sequential Monte Carlo method which consists of parallel MCMC-like samplers. Roughly, each of these samplers generate samples sequentially from its predefined distribution to help sampling from the actual target density.

Bridge Densities Before sampling from a target distribution $p_{target}(z)$ using SIMCMC, we first require a proposal distribution $p_{source}(z)$ that we can directly sample from. Then, a predefined number of auxiliary bridge densities ϕ_n should be declared,

$$p_{source}(z) \propto \phi_0(z_0) \to \phi_1(z_1) \to \dots \to \phi_N(z_N) \propto p_{target}(z) \tag{42}$$

such that samples generated by each bridge density is used in the sampling process of its successor density. Here, N denotes the total number of auxiliary bridge densities including the proposal distribution.

Through at the sampling scheme each bridge density is approximated by an empirical distribution at any given time. For all ϕ_n, its empirical approximation $\hat{\phi}_n$ is a Monte Carlo estimate using the generated samples,

$$\hat{\phi}_n(z_n) = \frac{1}{S_n} \sum_{s=1}^{S_n} \delta_{z_n^{(s)}}(z_n) \tag{43}$$

where S_n is current number of samples generated from the bridge density ϕ_n.

Sampling Notice that we can directly sample from the initial bridge density ϕ_0,

$$z_0^{(s)} \sim \hat{\phi}_0(z_0) \tag{44}$$

since by definition it is proportional to our proposal distribution p_{source}. For all other bridge densities ϕ_n, using the empirical density $\hat{\phi}_{n-1}$ of the previous bridge density, the following sampling scheme is adopted to propose the candidate samples $z_n^{*(s)}$,

$$\tilde{z}_{n-1}^{(s)} \sim \hat{\phi}_{n-1}(z_{n-1}) \tag{45}$$

$$z_n^{*(s)} \sim q_n(z_n | \tilde{z}_{n-1}^{(s)}) \tag{46}$$

where $q_n(\cdot)$ is the corresponding proposal function to be used for ϕ_n and $\tilde{z}_{n-1}$ is the ancestor of candidate sample $z_n^{*(s)}$. Hence, each sample other than the very first ones in each bridge density depends on the samples that are sampled before it in the previous bridge density as shown in Figure 3. That's why these samplers doesn't exactly correspond to a MCMC method and are called as MCMC-like samplers instead.

$$\begin{tikzcd}
z_0^{(1)} \arrow[dr] \arrow[ddr] & z_1^{(1)} \arrow[d] \arrow[dr] \arrow[ddr] & z_2^{(1)} \arrow[d] \arrow[dr] \arrow[ddr] & z_3^{(1)} \arrow[d] \\
z_0^{(2)} \arrow[r] \arrow[dr] & z_1^{(2)} \arrow[r] \arrow[d] \arrow[dr] & z_2^{(2)} \arrow[r] \arrow[d] \arrow[dr] & z_3^{(2)} \arrow[d] \\
z_0^{(3)} \arrow[r] & z_1^{(3)} \arrow[r] & z_2^{(3)} \arrow[r] & z_3^{(3)}
\end{tikzcd}$$

Figure 3: An example graphical model of SIMCMC sampling scheme for the first 3 samples and for $N = 3$ bridge densities.

Acceptance Probability After generating the candidate sample $\alpha(z_n^{(s-1)}, z_n^{*(s)})$, the acceptance probability α has to be calculated given the weights of each sample as below,

$$\alpha(z_n^{(s-1)}, z_n^{*(s)}) = \min\left\{1, \frac{w(z_n^{*(s)})}{w(z_n^{(s-1)})}\right\} \tag{47}$$

where the weight of an arbitrary sample $z_n^{(s)}$ is obtained using the bridge densities and its ancestor sample as follows,

$$w(z_n^{(s)}) = \frac{\phi_n(z_n^{(s)})}{\phi_{n-1}(\tilde{z}_{n-1}^{(s)})q_n(z_n^{(s)}|\tilde{z}_{n-1}^{(s)})} \tag{48}$$

Finally, $z_n^{(s)}$ is assigned to $z_n^{*(s)}$ with probability $\alpha_n(z_n^{(s-1)}, z_n^{*(s)})$ or to $z_n^{(s-1)}$ with probability $1 - \alpha_n(z_n^{(s-1)}, z_n^{*(s)})$.

Properties On contrast to the other sampling methods, the number of samples that would be generated from each bridge density doesn't have to known in advance. Instead, the SIMCMC sampling scheme consists of sampling rounds such that a single sample is generated for each bridge density starting from ϕ_0 to ϕ_N during each round. The process may proceed up to an unknown arbitrary number of rounds while improving the accuracy of the Monte Carlo estimate each round.

4.4.1 Application to RLEMMC

Before starting to introduce our SIMCMC-based sampling scheme for reinforcement learning, in order to keep the notation simple, we treat the state-action trajectories $(x_{0:T}, a_{0:T}, T)$ as particles z such that

$$z \equiv (x_{0:T}, a_{0:T}, T) \tag{49}$$

Note that according to this notation we would now like to sample from the target posterior distribution of $P(z|r = 1; \pi)$ in Equation (8).

Bridge Densities First of all, we define N bridge densities $\phi_0(z_0), ..., \phi_{n-1}(z_{n-1})$ and the target distribution $\phi_N(z_N)$ as follows,

$$\phi_n(z_n) \propto P(z_n; \pi)P(r = 1|z_n; \pi)^{\eta(n)} \qquad n = 0, ..., N \tag{50}$$

where $\eta(\cdot)$ is a monotonically increasing tempering function defined over $\{0, 1, ..., N\}$ with bound conditions of,

$$0 \equiv \eta(0) < \eta(1) < ... < \eta(N) \equiv 1 \tag{51}$$

By this construction, initial bridge density $\phi_0(z_0)$ actually corresponds to our prior trajectory distribution $P(z_n; \pi)$ that we can simply generate samples from. Similarly, the last bridge density $\phi_N(z_N)$ is proportional to our target posterior distribution.

Also, notice that the latter term in the bridge density definition can easily be calculated for each n using the Markov property of MDPs as follows,

$$P(r = 1|z_n; \pi) = P(r = 1|x_T, a_T) \tag{52}$$

where this (x_T, a_T) tuple is the terminal state-action tuple in z_n.

Sampling In our approach, we used the following proposal function which keeps the beginning of a state-action trajectory as it is and samples the rest of it from the prior.

$$\tau \sim U[1...T] \tag{53}$$

$$(x'_{0:\tau}, a'_{0:\tau-1}) = (x_{0:\tau}, a_{0p:\tau-1}) \tag{54}$$

$$a'_t \sim P(a_t|x'_t; \pi) \qquad t = \tau, ..., T \tag{55}$$

$$x'_t \sim P(x_t|x'_{t-1}, a'_{t-1}) \qquad t = \tau + 1, ..., T$$

$$z' = (x'_{0:T}, a'_{0:T}, T) \tag{56}$$

According to the proposal function in Equation (53), the weight of sample $z_n^{(s)}$ can be explicitly calculated subsequently.

$$w(z_n^{(s)}) = \tau \frac{P(r=1|x_T, a_T)^{\eta(n)}}{P(r=1|\tilde{x}_T, \tilde{a}_T)^{\eta(n-1)} P(\tilde{x}_{\tau+1:T}, \tilde{a}_{\tau:T}|\tilde{x}_\tau; \pi)} \tag{57}$$

4.5 Algorithm

Our SIMCMC derivation for the RLEMMC algorithm has an algorithmic flow as shown in the Algorithm 5.

Algorithm 5 RLEMMC-SIMCMC

```
1: $\pi \leftarrow$ Uniform policy
2: while $\pi$ not converged do
3:   for $s = 1$ to $S$ do
4:     for $n = 0$ to $N$ do
5:       if $n == 0$ then
6:         Sample $z_0^{(s)}$ following the policy $\pi$
7:       else
8:         Sample the ancestor $\tilde{z}_{n-1}^{(s)}$ from the empirical density $\hat{\phi}_{n-1}$
9:         Sample $z_n^{(s)}*$ using proposal $q_n$ given ancestor $\tilde{z}_{n-1}^{(s)}$
10:        Calculate the weights $w(z_n^{(s)*})$ and $w(z_n^{(s-1)})$
11:        Calculate the acceptance probability $\alpha(z_n^{(s-1)}, z_n^{(s)*})$
12:        if Accept then
13:          $z_n^{(s)} \leftarrow z_n^{(s)*}$
14:        else
15:          $z_n^{(s)} \leftarrow z_n^{(s-1)}$
16:        end if
17:      end if
18:    end for
19:  end for
20:  Learn policy $\pi$ using samples $z_N$
21: end while
22: Optimal Policy: $\pi^* \leftarrow \pi$
```

5 Experiments

5.1 Two Dimensional Continuous Grid

Futhermore, in order to apply our work to continuous domains, we first chose a simple toy problem of two-dimensional continuous grid similar to the one in [16]. In this example, a agent that is only capable of moving forward has to determine its movement direction in advance to obtain maximum expected reward given system parameters.

Initialization probabilities of the fixed-dimensional trajectories are Gaussian distributions and transition among states are linear with Gaussian noise. Due to the simplicity of the problem, there is a unique deterministic policy π among the state space that defines the angle θ_π of the agent. Finally, we assume the probability of receiving a reward is proportional to a Gaussian distribution. Hence, the generative model for the problem can be summarized as,

$$\begin{aligned} x_0 &\sim \mathcal{N}(x_0; \mu_0, \Sigma_0) \\ a_t &= \begin{bmatrix} \cos(\theta_\pi) \\ \sin(\theta_\pi) \end{bmatrix} \\ P(x_{t+1}|x_t, a_t) &= \mathcal{N}(x_{t+1}; x_t + a_t, \Sigma) \end{aligned} \tag{58}$$

and the probabilities of receiving a rewards are calculated as,

$$P(r = 1|x_{0:T}, a_{0:T}) \propto \exp\left(-\frac{1}{2}(x_T - \mu_r)^\top \Sigma_r^{-1}(x_T - \mu_r)\right) \tag{59}$$

where μ_0, Σ_0 are the mean and the covariance of the initialization distribution respectively, Σ is the covariance of the Gaussian transition noise and μ_r, Σ_r are the mean and the covariance of the Gaussian reward function respectively.

We used a continuous version of the importance sampling scheme in order the approximate the expectations in Equation (**??**) of the EM algorithm defined in Equation (7). After sampling the trajectories $(x_{0:T}^{(s)}, a_{0:T}^{(s)})$ from the prior distribution using the generative model introduced in Equation (58), corresponding weights are calculated proportional to the reward probabilities as,

$$w(x_{0:T}^{(s)}) = P(r = 1|x_{0:T}^{(s)}) \tag{60}$$

In the M-step, the only policy parameter θ_π is estimated as the weighted average of the most likely angles $\theta_x(x_{0:T}^{(s)})$ in order to generate the given trajectories as follows,

$$\theta_\pi = \frac{\sum_{s=1}^{S} W(x_{0:T}^{(s)}) \theta_x(x_{0:T}^{(s)})}{\sum_{s=1}^{S} W(x_{0:T}^{(s)})} \tag{61}$$

where $\theta_x(x_{0:T}^{(s)})$ can be explicitly calculated as,

$$\theta_x(x_{0:T}^{(s)}) = \arctan\left(\frac{x_{T,2}^{(s)} - x_{0,2}^{(s)}}{x_{T,1}^{(s)} - x_{0,1}^{(s)}} \right) \tag{62}$$

where second sub indices refer to the first and the second dimension of the two-dimensional state space.

Thus, we applied the method to a scenario, where $\mu_0 = \begin{bmatrix} 0 \\ 0 \end{bmatrix}$ and $\mu_r = \begin{bmatrix} 7 \\ 7 \end{bmatrix}$ for a finite a horizon of $T = 10$. Since, the problem is simplistic, we used only 5 samples per iterations and run the algorithm for 3 iterations. The convergence results can be seen in Figure 5.1.

5.2 Mountain-Car Problem

As a more sophisticated problem, we applied our methods to the benchmark Mountain-Car problem [24] which has a continuous two-dimensional state-space and non-linear transitions.

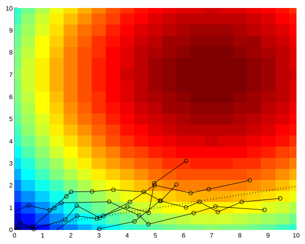
Problem Definition In this problem, a car tries to climb up to the top of a mountain by accelerating forward or backward as shown in Figure 5.2. However, scenario is organized in a way that if the car first goes backwards and gains some speed only then it would have enough speed to climb up the hill. Hence, the greedy solution of keep accelerating forward is not an optimal solution.

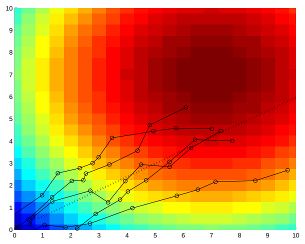
In order to formalize this problem, we only need to represent the horizontal position and the velocity of the car as at each time step as x_t,

$$x_t = \begin{bmatrix} pos_t \\ vel_t \end{bmatrix} \tag{63}$$

Additionally, car has 3 possible actions as accelerating forward, accelerating backward and doing nothing. Hence, at time step action a_t takes values among,

$$a_t \in \{-1, 0, 1\} \tag{64}$$


(a) Current policy and the generated samples at 1st iteration


(b) Current policy and the generated samples at 2nd iteration

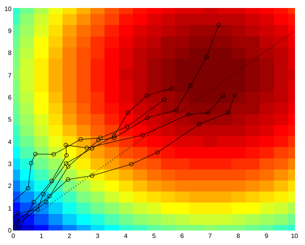

(c) Current policy and the generated samples at 3rd iteration

Figure 4: EM iterations and samples generated with importance sampling for 2D continuous grid problem. Dotted lines denoted the angles defined by the corresponding policy. The Gaussian reward function is represented by the color scheme where the hotter colors denote higher rewards and coolers denote lower rewards.

Also, let us denote the function of the hill as $h(\cdot)$ that only depends on the horizontal position. Then, the gravitational and frictional forces applied to car in the opposite direction of the movement given the position of the car can be calculated via trigonometric operations as follows,

$$g(pos_t) = G\frac{\frac{\partial h(pos_t)}{\partial pos}}{\sqrt{1 + \frac{\partial h(pos_t)}{\partial pos}}} \tag{65}$$

$$f(pos_t) = F\frac{1}{\sqrt{1 + \frac{\partial h(pos_t)}{\partial pos}}} \tag{66}$$

where G is the gravity, ϵ_t is the two-dimensional Gaussian transition noise with Σ covariance and F is the friction coefficient of the surface.

Thus, non-linear transition function of the car while moving up and down

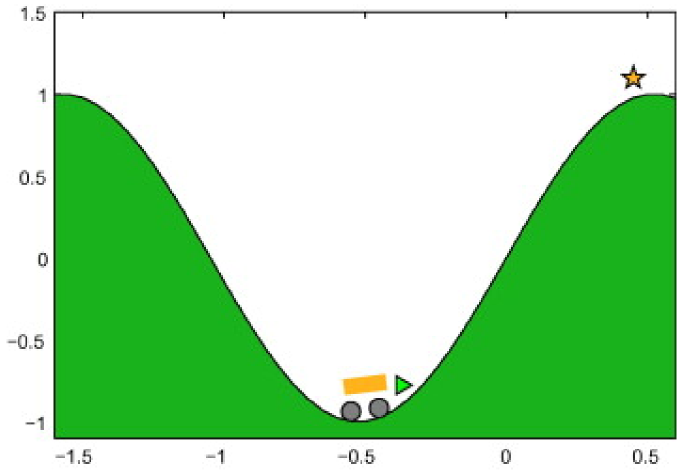


Figure 5: Illustration of the Mountain-Car Problem.

the hill can be explicitly calculated as,

$$x_{t+1} = \begin{bmatrix} 1 & \Delta t \\ 0 & 1 \end{bmatrix} x_t + \begin{bmatrix} 0 \\ M\Delta t \end{bmatrix} a_t + \begin{bmatrix} 0 \\ -(g(pos_t) + f(pos_t))\Delta t \end{bmatrix} + \epsilon_t \tag{67}$$

where M coefficient related with the motor power of the car and Δt is the time granularity.

Furthermore, the reward probabilities are distributed according to a Gaussian distribution with a mean of μ_r at the top of the mountain and covariance Σ_r.

Approach Finally, since the state-state space is continuous, there can not be a tabular policy. Either, the state space has to be discretized or a policy function has to learn over the state space. In the experiment, we used a k-nearest neighbour [5] with an Euclidean distance as the policy approximation.

We derived an implemented an SMC sampler method in order to find the optimal policy in mountain-car problem.

Bridge Functions When we denote a trajectory $(x_{0:T}, a_{0:T})$ as z, we defined N bridge functions as follows,

$$\phi_n(z_n) = P(r = 1|z_n)^{\eta(n)} P(z_n; \pi) \tag{68}$$

for $\eta(0) = 0$ and $\eta(N) = 1$. Hence, the initial proposal distribution is equal to the prior distribution,

$$\phi_0(z_0) = P(z_0; \pi) \tag{69}$$

Also, in order to see the effect of bridging, function of rewards over position can be graphed as in Figure 5.2 since the reward functions are Gaussian.

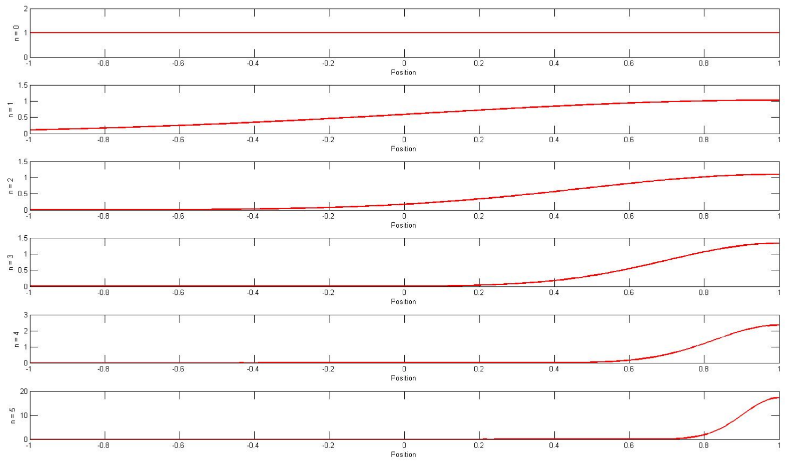


Figure 6: An example of tempered reward functions $P(r = 1|z_n)^{\eta(n)}$ for N = 5

Forward and Backward Kernels Now, we define the forward K_n and backward kernels L_{n-1} equal to the MCMC kernel with invariant distribution of ϕ_n as described below.

We have derived and implemented a Metropolis-Hastings kernel and also theoretically derived a RJMCMC kernel for the transdimensional case.

Metropolis-Hastings Kernel Our proposal function given a trajectory $z_{n-1} = (x_{0:T}, a_{0:T})$ chooses a time step uniformly at random and then samples the rest of the trajectory from the prior and build the proposal sample of MCMC kernel z'_{n-1} as follows,

$$\begin{aligned}\tau &\sim U[1...T]\\ (x'_{0:\tau}, a'_{0:\tau-1}) &= (x_{0:\tau}, a_{0:\tau-1})\\ x'_t &\sim P(x_t|x'_{t-1}, a'_{t-1})\\ a'_t &\sim P(a_t|x'_t; \pi)\\ z'_{n-1} &= (x'_{0:T}, a'_{0:T}) \end{aligned} \tag{70}$$

Hence, the acceptance probability of the proposed trajectory is obtained as,

$$\alpha = \min\left\{1, \frac{P(r=1|z'_{n-1})^{\eta(n)}}{P(r=1|z_{n-1})^{\eta(n)}}\right\} \tag{71}$$

Weight Function After sampling $\tilde{z}_n^{(s)}$ for each sample $z_{0:n-1}^{(s)}$ using the described MCMC kernel as,

$$\tilde{z}_n^{(s)} \sim K_n(z_n|z_{n-1}^{(s)}) \tag{72}$$

we form the proposals for the importance sampling step of the SMC sampler.

$$\tilde{z}_{0:n}^{(s)} \equiv (z_{0:n-1}^{(s)}, \tilde{z}_n^{(s)}) \tag{73}$$

Since, forward and backward kernels are selected appropriately the weight function is simplified to,

$$W(\tilde{z}_{0:n}^{(s)}) = \frac{\phi_n(\tilde{z}_n^{(s)})}{\phi_{n-1}(z_{n-1}^{(s)})} \tag{74}$$

Results We experimented our SMC sampler on the mountain-car problem for a Gaussian initialization probability with mean $\mu_0 = \begin{bmatrix} 0 \\ 0 \end{bmatrix}$ and a Gaussian reward function that only considers the position with a mean of $\mu_r = 0.7$.

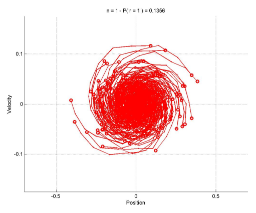
Additionally, we employed 5 bridge functions in addition to the prior such that

$$\begin{aligned}\eta(0) &= 0\\ \eta(1) &= 0.1\\ \eta(2) &= 0.33\\ \eta(3) &= 1\\ \eta(4) &= 3\\ \eta(5) &= 10\end{aligned} \tag{75}$$

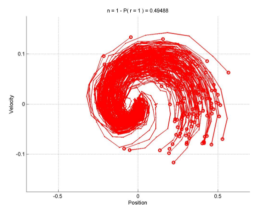
Here, we employed further annealing for the fourth and the fifth bridge functions to favor samples with higher reward probabilities in order to fasten the convergence rate.

Also, we utilize the Metropolis-Hastings kernel described in Section 5.2 since our problem is fixed-dimensional.

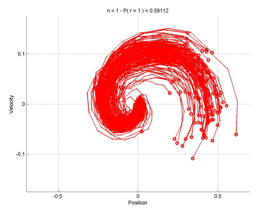
In Figure 5.2, SMC sampler steps with 200 samples for the very first EM iteration can be seen. Even at the end of this first EM step, trajectories are converged to an apparently useful suboptimal solution. However, we further continue EM algorithm for 4 additional iterations and obtain policy trajectories as shown in Figure 5.2.



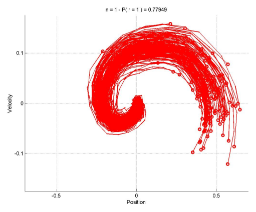
(a)



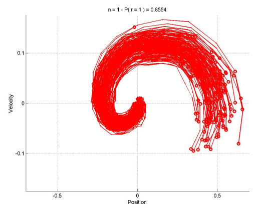
(b)



(c)

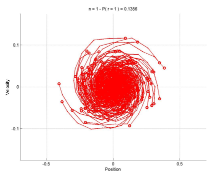


(d)

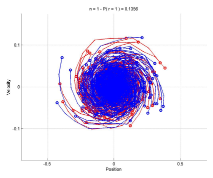


(e)

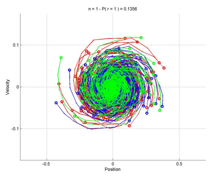
Figure 7: Samples generated with the current policy at the beginning of each EM iteration



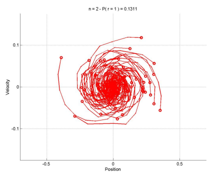
(a)



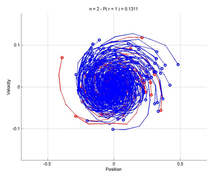
(b)



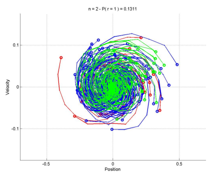
(c)



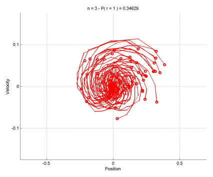
(d)



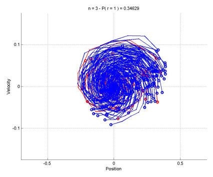
(e)



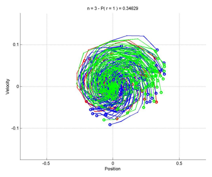
(f)



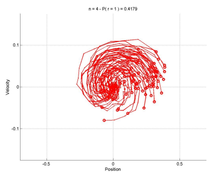
(g)



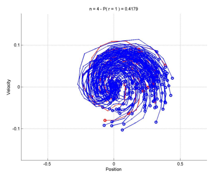
(h)



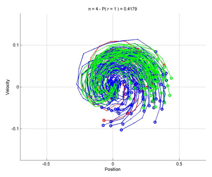
(i)



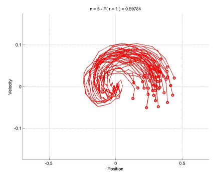
(j)



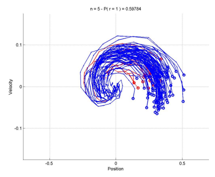
(k)



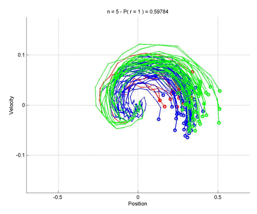
(l)



(m)



(n)



(o)

Figure 8: E-step of the very first EM iteration using SMC samplers with 5 bridge functions where red trajectories denote the samples at the beginning of a SMC samplers iterations, blue trajectories represent the proposals that are generated using the forward kernel and green trajectories are the resampled trajectories at the end of the iteration.

5.3 Noisy Cart Pole Problem

We applied our derived RLEMMC algorithms with different Monte Carlo methods at their E-steps to the Noisy version of the Cart Pole problem in the famous OpenAI framework for reinforcement learning.

In this problem, the state space has 4 dimensions; namely cart position, cart velocity, pole angle and pole velocity. The aim of the agent is to hold the pole as much perpendicular while not letting the cart go too much away from the starting position by only pushing cart either to the left or to the right.

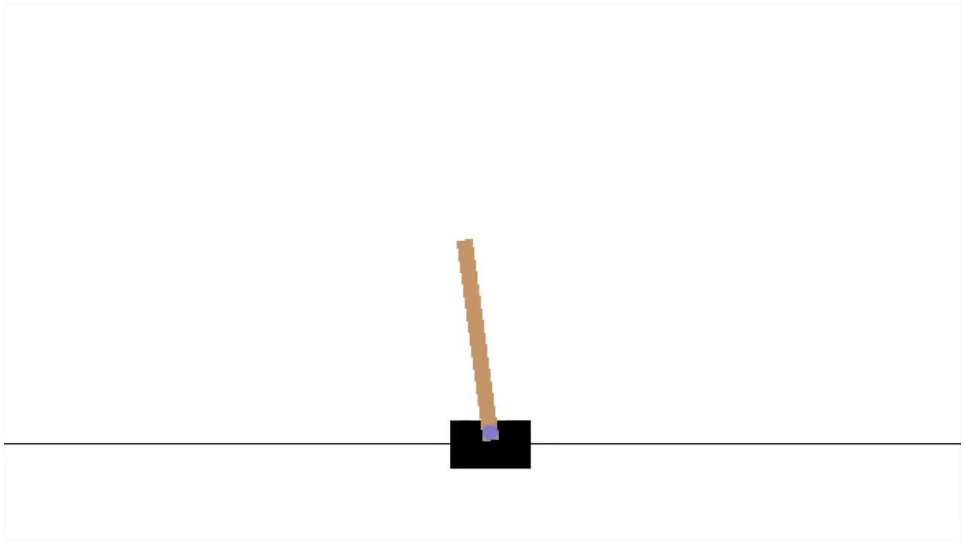


Figure 9: Illustration of the Cart Pole Problem.
3

We ran our algorithms for an horizon of 250 timesteps while using as small as 1000 samples at each iteration. Even this small number of samples, the algorithms using our more sophisticated methods, SMCS and SIMCMC manage to converge to an acceptable solution just in couple of iterations. Meanwhile, the algorithms using importance sampling and Metropolis-Hastings at the E-step require either more iterations or more sample per iteration to reach the goal state which is defined by OpenAI as keeping the cart pole up for 195 timesteps. The convergence results of the experiment is shown in Figure 5.3

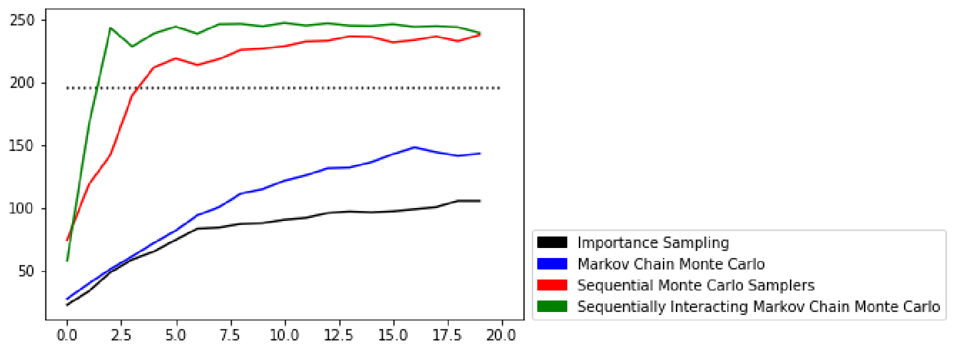


Figure 10: Illustration of the Mountain-Car Problem.

6 Discussion

The experiments show that the RLEMMC algorithm converges to a suboptimal solution even though the expectation at the E-step of the algorithm is also approximated instead of an exact calculation. Hence, the algorithm can be count as an optimistic policy search.

However, when the reward space is complex and positive rewards are rare events in the state space and through the horizon, classical Monte Carlo algorithms like importance sampling and Metropolis-Hastings have some convergence issues. However, introducing a tempering over the rewards via bridge densities, sequential Monte Carlo methods manage to overcome this problem.

6.1 Future Work

First of all, in order to increase the amount of the experiments to an acceptable level, I aim to apply our algorithm to more environments from the OpenAI framework, both with discrete state spaces like Frozen Lake and with continuous state space like Acrobot, Pendulum etc.

Having better experiments which was the main reason of rejection of my previously submitted journal, next semester my main goal is to resubmit a journal paper about the RLEMMC algorithms using SMCS and SIMCMC at their E-steps.

So far, I have only tried two different proposal functions, namely prior as proposal and keep-then-prior as introduced in Section 4.2.1 as a Metropolis-Hastings kernel. I plan to investigate different proposal to increase the converge rates of both methods SMCS and SIMCMC that employs Metropolis-Hastings kernel.

Also, the policy approximation for the RLEMMC algorithm is not deeply investigated. State-of-the-art approximators like deep neural networks and Gaussian processes can be employed there as well.

References

[1] E Alpaydin. *Introduction to Machine Learning*, volume 56. The MIT Press, 2004.

[2] H Attias. Planning by probabilistic inference. In *Proc. of the 9th Int. Workshop on Artificial*, 2003.

[3] Dimitri P. Bertsekas. *Dynamic Programming and Optimal Control 3rd Edition, Vol. I*, volume 2 of *Athena Scientific optimization and computation series*. Athena Scientific, 2007.

[4] Anthony Brockwell, Pierre Del Moral, and Arnaud Doucet. Sequentially interacting Markov chain Monte Carlo methods. *The Annals of Statistics*, 38(6):3387–3411, dec 2010.

[5] T. Cover and P. Hart. Nearest neighbor pattern classification. *Information Theory, IEEE Transactions on*, 13, 1967.

[6] Peter Dayan and Geoffrey E. Hinton. Using Expectation-Maximization for Reinforcement Learning. *Neural Computation*, 9(2):271–278, feb 1997.

[7] J.P. de Villiers, S.J. Godsill, and S.S. Singh. Particle predictive control. *Journal of Statistical Planning and Inference*, 141(5):1753–1763, may 2011.

[8] Marc Deisenroth, Dieter Fox, and Carl Rasmussen. Gaussian Processes for Data-Efficient Learning in Robotics and Control. *Transactions on Pattern Analysis and Machine Intelligence*, 37(2):1–20, 2015.

[9] Marc P Deisenroth, Carl Edward Rasmussen, and Dieter Fox. Learning to Control a Low-Cost Manipulator using Data-Efficient Reinforcement Learning. 2011.

[10] Marc Peter Deisenroth, Carl Edward Rasmussen, and Jan Peters. Gaussian process dynamic programming. *Neurocomputing*, 72(7-9):1508–1524, mar 2009.

[11] MP Deisenroth and CE Rasmussen. PILCO: A Model-Based and Data-Efficient Approach to Policy Search. In *Proceedings of the 28th International Conference on Machine Learning*, pages 465—-472, 2011.

[12] Pierre Del Moral and Arnaud Doucet. Sequential monte carlo samplers. *Journal of the Royal Statistical Society - Series B: Statistical Methodology*, 68(3):411–436, 2006.

[13] A P Dempster, N M Laird, and D B Rubin. Maximum likelihood from incomplete data via the EM algorithm. *Journal of the Royal Statistical Society Series B Methodological*, 39(1):1–38, 1977.

[14] Thomas Furmston and David Barber. Efficient Inference in Markov Control Problems. In *Proceedings of the Twenty-Seventh Conference Annual Conference on Uncertainty in Artificial Intelligence (UAI-11)*, pages 221–229. AUAI Press, 2011.

[15] Peter J. Green. Reversible jump Markov chain Monte Carlo computation and Bayesian model determination. *Biometrika*, 82(4):711–732, 1995.

[16] Matt Hoffman and Ajay Jasra. Trans-dimensional MCMC for Bayesian Policy Learning. *Neural Information Processing Systems*, 20:1–8, 2008.

[17] Matt Hoffman, Hendrik Kueck, Nando De Freitas, and Arnaud Doucet. New inference strategies for solving Markov Decision Processes using reversible jump MCMC. In *Conference on Uncertainty in Artificial Intelligence*, 2009.

[18] Alessandro Lazaric, Marcello Restelli, and Andrea Bonarini. Reinforcement Learning in Continuous Action Spaces through Sequential Monte Carlo Methods. *Nips*, pages 833–840, 2008.

[19] J. S. Liu. *Monte Carlo Strategies in Scientific Computing*. Springer Series in Statistics. Springer, 2001.

[20] A Y Ng and M Jordan. PEGASUS: A policy search method for large MDPs and POMDPs. *Policy*, pages:406–415, 2000.

[21] CE Rasmussen. Gaussian processes in machine learning. *ADVANCED LECTURES ON MACHINE LEARNING*, 3176:63–71, 2004.

[22] CE Rasmussen and Malte Kuss. Gaussian processes in reinforcement learning. *Advances in neural information processing ...*, 2004.

[23] Konrad Rawlik, Marc Toussaint, and Sethu Vijayakumar. Approximate Inference and Stochastic Optimal Control. *Computing Research Repository*, abs/1009.3:1–20, 2010.

[24] Richard S Sutton and Andrew G Barto. *Reinforcement Learning : An Introduction.* MIT Press, 1998.

[25] Csaba Szepesvári. *Algorithms for Reinforcement Learning*, volume 4 of *Synthesis Lectures on Artificial Intelligence and Machine Learning.* Morgan & Claypool Publishers, 2010.

[26] 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*, pages 945–952, New York, New York, USA, 2006. ACM.

[27] Nikos Vlassis and Marc Toussaint. Model-free reinforcement learning as mixture learning. In *Proceedings of 26th International Conference on Machine Learning*, 2009.

A Defined New Likelihood and Reinforcement Learning Problem Reformulation

Let us expand the defined new likelihood $P(r=1;\pi)$ using its definition for a given horizon $T=\tau$.

$$\begin{aligned}
P(r=1;\pi) &= \sum_{t=0}^{\tau}\sum_{x_{0:t}}\sum_{a_{0:t}} P(r=1, x_{0:T}, a_{0:T}, T=t;\pi) \\
&= \sum_{t=0}^{\tau}\sum_{x_{0:t}}\sum_{a_{0:t}} P(r=1, x_{0:T}, a_{0:T}|T=t;\pi)P(T=t) \\
&\propto \sum_{t=0}^{\tau}\sum_{x_{0:t}}\sum_{a_{0:t}} \gamma^t P(r=1, x_{0:T}, a_{0:T}|T=t;\pi) && \text{Since } P(T=t)\propto\gamma^t \\
&= \sum_{t=0}^{\tau}\gamma^t\sum_{x_{0:t}}\sum_{a_{0:t}} P(r=1, x_{0:T}, a_{0:T}|T=t;\pi) \\
&= \sum_{t=0}^{\tau}\gamma^t\sum_{x_{0:t}}\sum_{a_{0:t}} P(r=1|x_{0:T}, a_{0:T}, T=t;\pi)P(x_{0:T}, a_{0:T}|T=t;\pi) \\
&= \sum_{t=0}^{\tau}\gamma^t \left\langle P(r=1|x_{0:T}, a_{0:T}, T=t;\pi)\right\rangle_{P(x_{0:T},a_{0:T}|T=t;\pi)} \\
&= \sum_{t=0}^{\tau}\gamma^t \left\langle r_t\right\rangle_{P(x_{0:T},a_{0:T}|T=t;\pi)} && \text{Since } r\in\{0,1\} \\
&= \sum_{t=0}^{\tau}\gamma^t \left\langle r_t\right\rangle_{P(x_{0:T},a_{0:T}|T=\tau;\pi)} \\
&= \left\langle \sum_{t=0}^{\tau}\gamma^t r_t\right\rangle_{P(x_{0:T},a_{0:T}|T=\tau;\pi)} && (76)
\end{aligned}$$

As the defined new likelihood is actually proportional to the expected total reward in the reinforcement learning problem definition, their maximizers are equivalent.

$$\arg\max_{\pi} P(r=1;\pi) = \arg\max_{\pi}\left\langle \sum_{t=0}^{T}\gamma^t r_t\right\rangle_{P(x_{0:T},a_{0:T}|T=\tau;\pi)} \tag{77}$$

B EM Derivation for the Mixture of Terminal Reward MDPs

For sake of simplicity, we use abbreviations for state and action trajectories such that X and A denote the state trajectories $x_{0:T}$ and the action trajectories $a_{0:T}$ respectively for this proof only. Also, all summations below are done over the whole space of the corresponding random variable. Let's start with expanding the likelihood maximization problem in Equation 5 and then put it into the log form.

$$\begin{aligned}\pi^* &= \arg\max_{\pi} P(r=1;\pi)\\ &= \arg\max_{\pi} \sum_T \sum_X \sum_A P(r=1,X,A,T;\pi)\\ &= \arg\max_{\pi} \log \sum_T \sum_X \sum_A P(r=1,X,A,T;\pi)\end{aligned}$$

Now, let's consider this log expression to be maximized and assume q is an arbitrary proposal function.

$$\begin{aligned}&\log \sum_T \sum_X \sum_A P(r=1,X,A,T;\pi)\\ &\qquad= \log \sum_T \sum_X \sum_A P(r=1,X,A,T;\pi)\frac{q(X,A,T)}{q(X,A,T)}\\ &\qquad= \log \left\langle \frac{P(r=1,X,A,T;\pi)}{q(X,A,T)} \right\rangle_{q(X,A,T)}\end{aligned}$$

Using Jensen's Inequality,

$$\geq \left\langle \log \frac{P(r=1,X,A,T;\pi)}{q(X,A,T)} \right\rangle_{q(X,A,T)} \tag{78}$$

Now, let us consider the optimal proposal function q^* that maximizes this lower bound. It can be calculated by taking the derivate of the above equation with respect to q and then equating it to 0. It is found as below,

$$\begin{aligned}q^*(X,A,T) &= P(X,A,T|r=1;\pi)\\ &= \langle \log P(r=1,X,A,T;\pi)\rangle_{P(X,A,T|r=1;\pi)}\\ &\qquad - \langle \log q(X,A,T)\rangle_{P(X,A,T|r=1;\pi)}\\ &=^{+} \langle \log P(r=1,X,A,T;\pi)\rangle_{P(X,A,T|r=1;\pi))}\end{aligned}$$

where $=^+$ denotes left hand side equals to right hand side plus a constant which doesn't affect the maximization. Hence, each iteration k of the derived EM algorithm, policy π is updated as follows,

$$\pi^{(k+1)} \leftarrow \arg\max_{\pi} \langle \log P(r=1, X, A, T; \pi) \rangle_{P(X,A,T|r=1;\pi^{(k)})} \tag{79}$$