

CAMEO: Curiosity Augmented Metropolis for Exploratory Optimal Policies

Simo Alami, Fernando Llorente, Rim Kaddah, Luca Martino, Jesse Read

▶ **To cite this version:**

Simo Alami, Fernando Llorente, Rim Kaddah, Luca Martino, Jesse Read. CAMEO: Curiosity Augmented Metropolis for Exploratory Optimal Policies. EUSIPCO, Aug 2022, Belgrade, Serbia. hal-03675575v2

HAL Id: hal-03675575
https://hal.science/hal-03675575v2

Submitted on 14 Feb 2023

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

CAMEO: Curiosity Augmented Metropolis for Exploratory Optimal Policies

Simo Alami Chehboune
Ecole Polytechnique/IRT SystemX
Palaiseau, France
mohamed.alami-chehboune@polytechnique.edu

Fernando Llorente
Universidad Carlos III de Madrid
Leganés, Spain
felloren@est-econ.uc3m.es

Rim Kaddah
IRT SystemX
Palaiseau, France
rim.kaddah@irt-systemx.fr

Luca Martino
Universidad Rey Juan Carlos
Fuenlabrada, Spain
luca.martino@urjc.es

Jesse Read
LIX, Ecole Polytechnique, IP-Paris
Palaiseau, France
jread@lix.polytechnique.fr

***Abstract*—Reinforcement Learning has drawn huge interest as a tool for solving optimal control problems. Solving a given problem (task or environment) involves converging towards an optimal policy. However, there might exist multiple optimal policies that can dramatically differ in their behaviour; for example, some may be faster than the others but at the expense of greater risk. We consider and study a distribution of optimal policies. We design a curiosity-augmented Metropolis algorithm (CAMEO), such that we can sample optimal policies, and such that these policies effectively adopt diverse behaviours, since this implies greater coverage of the different possible optimal policies. In experimental simulations we show that CAMEO indeed obtains policies that all solve classic control problems, and even in the challenging case of environments that provide sparse rewards. We further show that the different policies we sample present different risk profiles, corresponding to interesting practical applications in interpretability, and represents a first step towards learning the distribution of optimal policies itself.**

***Index Terms*—Reinforcement Learning, Curiosity model, Metropolis, MCMC**

I. Introduction

Reinforcement Learning (RL) is a sequential decision framework where a model (agent) has to solve a task in multiple steps and find the optimal strategy (policy, π) to do so. The agent interacts directly with its environment and learns through this experience. After each time step t that the agent performs an action a_t, the environment returns a new state s_{t+1} and a reward r_t. The agent then seeks to maximise the expected return i.e. the discounted sum of rewards received over time: $G = \mathbb{E}[r_0 + \gamma r_1 + \gamma^2 r_2 + \dots \gamma^t r_t + \dots | s_0]$; where gamma is a discount factor of future rewards. We define the value of state s, $V^\pi(s)$, as the expected return obtained when following policy π from state s; and the Q-value $Q^\pi(s,a)$ as the state value obtained after performing action a in state s.
$V^\pi(s) = \mathbb{E}_\pi\left[\sum_{k=0}^{\infty} \gamma^k r(s_{t+k+1})|s\right] = \sum_a \pi(a|s) Q^\pi(s,a)$

An optimal policy π^* is a policy that reaches the maximal value possible for all states. Such a policy always exists but might not be unique [14]. For instance, given policies π_{θ_i} parameterized by θ_i, in parameter space Ω, there are many θ_i that represent an optimal policy. Most of the literature focuses on finding models and algorithms that learn policies iteratively and converge towards a unique optimal policy [16]. But considering that many optimal policies exist, we propose a process to generate different optimal policies.

Indeed, even-though all desired policies may be optimal, they may differ in their behaviour towards solving the task. We can argue that an optimal policy is the fastest and most consistent one that solves the given task. However, these two concepts can be antagonist depending on the policy's risk profile. Two policies can be optimal but one might take more risk than the other and solve the task faster. Sutton and Barto [16] give an example of policies obtained using Q-learning or SARSA and show that in the cliff problem (see table I), both methods solve the task but Q-learning takes more risk by walking near the edge while SARSA takes a longer, safer, path.

As many optimal policies exist, then a distribution of optimal policies also exists. Learning such a distribution would allow to sample different optimal policies and choose the one which profile suits the best the task needs. For instance, the mode of the learnt distribution corresponds to the minimum variance optimal policy. However, to learn a distribution, many samples are needed. This paper addresses the question of whether or not it is possible to sample many different optimal policies. While all existing approaches focus on converging towards one optimal policy, we try to find a solution to generate many different ones that could correspond to different profiles.

We adapted the Metropolis algorithm [6] for RL in order to obtain a Markov chain of optimal policies whose stationary distribution corresponds to the distribution of optimal policies. We thus propose to sample a suitable distribution featuring *intrinsic* and *extrinsic* rewards (see Eq. (8)). We show that the obtained policies effectively adopt different behaviours while solving the desired tasks.

II. Related work and brief introduction to Reinforcement Learning

We cited Q-learning [17] and SARSA [16] above as classic methods used in RL. These two methods build Q-tables that store for every state-action pair the mean reward obtained afterwards until converging towards good estimations. After convergence, the optimal policy is the one that always follows the action that returns the highest value. While these methods try to evaluate Q values directly, other parameterize the policy itself like policy gradient methods [15]. The advantage of the latter being that these are capable to learn stochastic policies but also perform in continuous action spaces.

Deep Learning had a massive impact on RL and had been naturally incorporated into the cited approaches. This led to the emergence of Deep Q-learning [11], which uses deep neural networks to evaluate Q-values in large state spaces, or to Deep Policy Gradients that use a neural network parameters as policy parameters. Some other methods bridged the gap between these approaches like Actor Critic methods that use 2 neural networks, one that acts as policy and an other that evaluates state values [2].

Generative Adversarial Networks (GAN) [5] are closely related to Actor Critic where the actor network can be assimilated to a generator and the critic to a discriminator [13]. Indeed, GAN have been used in imitation learning for instance where expert demonstrations are used to learn a policy [7]. In this case the generator has to generate trajectories while the discriminator has to decide whether the trajectory[1] comes from the available demonstration or not. However instead of learning a new policy or outputting many different ones, the aim is to mimic the expert behaviour. Yet again, there is a close connection between this approach and GAN [4].

However these methods converge towards a unique optimal policy, rather than generate different optimal ones. All these advances show that using generative approaches to learn an optimal policy is possible but for our knowledge, there is yet no generative model that is used to generate different optimal policies that output different behaviours and different risk profiles. This work is a first step towards generating such samples in the hope of being then able to learn a distribution of optimal policies in future work.

III. Metropolis and Direct Policy Search

Our main objective is to sample from the distribution of optimal policies. In this section we present a simple adaptation of Metropolis algorithm to RL problems.

A. Metropolis Algorithm

Let θ denote the parameters of a policy π_θ. Rather than searching directly for optimal θ, several works consider sampling policy parameters θ_i using Markov Chain Monte Carlo algorithms [1], [8]. For instance, in [8], Hoffman et al. showed that for direct policy search, sampling directly from a trans-dimensional distribution that is proportional to the reward performs better than classic simulations methods. Therefore, we model a distribution $f(\theta)$, i.e the target distribution, that is proportional to the expectation of a monotonically increasing function U of the empirical return, i.e., it should be maximal when the return is maximal; a utility function $U(\tau)$.

$$f(\theta) \propto p(\theta)\eta(\theta). \tag{1}$$

where $p(\theta)$ is a prior over θ (in the simple case, uniform) and η the performance of θ wrt the environment:

$$\eta(\theta) = \mathbb{E}_{p(\tau|\theta)}[U(\tau)] = \int U(\tau)p(\tau|\theta)d\tau, \tag{2}$$

where τ denotes a trajectory (i.e. the succession of states visited), $p(\tau|\theta)$ is the distribution of τ (i.e., the one induced by following policy π_θ), and $U(\tau)$ the utility of τ. Provided that $\eta(\theta)$ is non-negative, we can sample it via a Metropolis-Hastings (MH) algorithm.

Denoting the empirical return by $\widetilde{G}(\tau) = \sum_{t=0}^{L-1} \gamma^t r_t$, a usual choice is $U(\tau) = \exp(\widetilde{G}(\tau)/T)$. Therefore:

$$f(\theta;T) \propto p(\theta)\eta_T(\theta) = p(\theta)\mathbb{E}_{p(\tau|\theta)}[e^{\widetilde{G}(\tau)/T}] \tag{3}$$

where T denotes an inverse temperature parameter.

In practice, in order to apply the MH algorithm on $f(\theta)$, the evaluation of $\eta(\theta)$ must be substituted with an unbiased estimate over N episodes,

$$\begin{aligned}\eta(\theta) &= \mathbb{E}_{p(\tau|\theta)}[U(\tau)] \\ &\approx \bar{U}_N(\theta) = \frac{1}{N}\sum_{i=1}^{N} U(\tau_i), \quad \tau_i \sim p(\tau|\theta),\end{aligned} \tag{4}$$

where $\bar{U}_N(\theta)$ denotes the empirical utility for θ over N episodes τ_i. Algorithm 1 consider the noisy evaluation of the utility function in Eq. (4), and is called *Monte Carlo-within-Metropolis* [9], [10].

IV. Curiosity Augmented Metropolis

A. Prior Bias

In section V-A, we showed that $\{\theta_k\}_{k=1}^K$ are highly correlated; due to a scaling factor among them. Indeed, as we considered that $p(\theta) \sim \mathcal{U}[-a, a]$, there is no bound to θ_i values and therefore we obtain $\theta_{k+1} = \lambda\theta_k$, with λ a constant. A solution is to constraint $\theta_k \in [-1, 1]^D$ but it is not possible to use $p(\theta) \sim \mathcal{U}[-1, 1]$, as it would cancel out anyway during the calculation of β (see algorithm 1). However it is possible to measure the variance of θ_k from $[-1, 1]$:

$$\sigma^2_{\theta_k} = \frac{1}{D}\sum_{j=0}^{D} \mathbb{1}_{\theta_{kj}\notin[-1,1]}(\theta^2_{kj} - 1)^2, \tag{5}$$

and therefore we define: $p(\theta) = e^{-\sigma^2_\theta}$

[1] *In RL literature, a trajectory is a successions of state action pairs or states alone. A trajectory can be longer than one episode. In this work, we constraint a trajectory to not exceed the length of an episode. Therefore, in this context, the words episode and trajectory are equivalent*

Algorithm 1 Monte Carlo-within-Metropolis for RL

Require: K: the number of iterations, N: number of episodes
  Initialise Agent π
  $\theta_0 \sim \mathcal{N}(\mathbf{0}, \sigma_p^2 \mathbf{I}_D)$
  for k from 1 to K **do**
    $\theta' \sim \mathcal{N}(\theta_{k-1}, \sigma_p^2 \mathbf{I}_D)$
    Run N episodes with $\pi_{\theta_{k-1}}$ and compute $\bar{U}_N(\theta_{k-1})$
    Run N episodes with $\pi_{\theta'}$ and compute $\bar{U}_N(\theta')$
    $\beta \leftarrow \dfrac{p(\theta')\bar{U}_N(\theta')}{p(\theta_{k-1})\bar{U}_N(\theta_{k-1})}$
    $\alpha \leftarrow \min(1, \beta)$
    $\epsilon \sim \mathcal{U}_{[0,1]}$
    if $\epsilon < \alpha$ **then**
      $\theta_k \leftarrow \theta'$
    else
      $\theta_k \leftarrow \theta_{k-1}$
    end if
  end for
  return $\{\theta_k\}_{k=1}^K$

B. Trajectory bootstrap

In Algorithm 1, we had to run N episodes twice to estimate the mean return of θ_{k-1} and θ'. We could store the return of θ_{k-1} and use it in subsequent runs. However, the method shown in Algorithm 1 was preferred to penalise the possible variance of the returns obtained using θ_{k-1}.

An alternative to still penalise variance while avoiding to run the N episodes twice is to make use of importance sampling and the advantage function [16]. Intuitively, it corresponds to the extra reward that the agent could obtain by taking action a over a random action: $A_\theta(s,a) = Q_\theta(s,a) - V_\theta(s)$. More specifically, by defining G as a function of θ:

$$G(\theta) = \int \widetilde{G}(\tau) p(\tau|\theta) d\tau \tag{6}$$

the *advantage* of θ' over θ_{k-1}:

$$\begin{aligned}
G(\theta') &= G(\theta_{k-1}) + \mathbb{E}_{\theta'}\left[\sum_{t=0} \gamma^t A_{\theta'}(s_t, a_t)\right] \\
&= G(\theta_{k-1}) + \sum_s \rho_{\theta'}(s) \sum_a \pi_{\theta'}(a|s) A_{\theta'}(s_t, a_t) \\
&\approx G(\theta_{k-1}) + \sum_s \rho_{\theta_{k-1}(s)} \sum_a \pi_{\theta'}(a|s) A_{\theta'}(s_t, a_t) \\
&= G(\theta_{k-1}) + \mathbb{E}_{s\sim\rho(\theta_{k-1})}\left[\sum_a \pi_{\theta'}(a|s) A_{\theta'}(s_t, a_t)\right] \\
&= G(\theta_{k-1}) + \mathbb{E}_{s\sim\rho(\theta_{k-1})}\mathbb{E}_{a\sim\pi_{\theta_{k-1}}}\left[\frac{\pi_{\theta'}(a|s)}{\pi_{\theta_{k-1}}(a|s)} A_{\theta'}(s_t, a_t)\right] \\
&\approx G(\theta_{k-1}) \\
&\quad + \mathbb{E}_{s,a\sim\rho(\theta_{k-1}),\pi_{\theta_{k-1}}}\left[\Pi(r + \gamma(V_{\pi_{\theta'}}(s') - V_{\pi_{\theta_{k-1}}}(s))\right],
\end{aligned} \tag{7}$$

where $\Pi = \frac{\pi_{\theta'}(a|s)}{\pi_{\theta_{k-1}}(a|s)}$. ρ is the state visitation frequency and we used $\rho_{\theta'}(s) \approx \rho_{\theta_{k-1}}(s)$ because we already have trajectories sampled from $\pi_{\theta_{k-1}}$, so it is easier to obtain $\rho_{\theta_{k-1}}(s)$ than $\rho_{\theta'}(s)$. Finally, TD Error [16] was used as an estimator for the advantage in place of Q-values in order reduce variability. This way, the return obtained using θ' can be estimated using the behaviour of the agent parameterised by θ' on the trajectories obtained under θ_{k-1}, noted $\pi_{\theta'}(\tau_{\theta_{k-1}})$.

C. Curiosity Module

As shown in section V-A, the previous implementation fails on Griworld and Cliff environments. One of the main reason explaining this failure is that the rewards in these environments are sparse. This means that a reward of -1 at each time step does not bring much information about the state of the agent and how close it is from the goal. The only valuable information are obtained either on the pit or when reaching the goal. In a gradient free approach like Metropolis where there is no criteria to drive the search, it means that the agent should first reach the goal by luck before improving. Moreover, the fact that there is no valuable information gathered at each run makes the process get stuck at some points in parameter space as all θ' are drawn from $\mathcal{N}(\theta_k, \sigma_p^2 \mathbf{I}_D)$.

It is therefore necessary to implement a mechanism that drives exploration dynamically. Inspired by the approach in [12], we propose a curiosity module that consists of a Neural Network that learns to predict the next state given the last state of an agent. This way, if the network learns to predict the trajectory of θ_k and θ' tries something new, it will fail and output a large prediction error, called the intrinsic reward. The intrinsic reward is then added to the extrinsic reward, i.e the reward returned by the environment:

$$R(\tau) = \mu\widetilde{G}(\tau) + (1-\mu)\mathcal{L}^{(k)}(\tau), \tag{8}$$

with $\mathcal{L}^{(k)}$ the prediction error (i.e. the loss) at step k and $\mu \in [0,1]$. The utility is thus computed on this modified reward

$$\widetilde{U}_N(\theta) = \frac{1}{N}\sum_{i=1}^N U(\tau_i) \quad \tau_i \sim p(\tau|\theta), \tag{9}$$

where, if we take U to be an exponential as above, we have

$$U(\tau_i) = \exp\{\mu\widetilde{G}(\tau_i) + (1-\mu)\mathcal{L}(\tau_i)\}. \tag{10}$$

The output θ_k are all stochastic optimal policies that fool the curiosity module and that explore different trajectories. Algorithm 2 details the procedure.

V. Experiments and Results

Our proposed framework has been tested in Classic Control Gym environments, Cartpole, Acrobot and cliff [3] as well as on a gridworld. Table I shows environments details.

A. Simple Metropolis implementation

In this section we present the results obtained using the simple implementation detailed in algorithm 1. The agent is a neural network composed of 1 hidden layer of 8 neurons and a ReLU activation function. The average return was estimated on 20

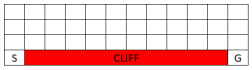
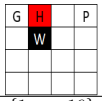
Environments	Cliff	Cartpole	Acrobot	Gridworld
Snapshot				
State Space $s_t \in$	$\{1, \dots, 48\}$	$\mathbb{R}^4$	$[-1, 1]^4 \times [-4\pi, 4\pi] \times [-9\pi, 9\pi]$	$\{1, \dots, 16\}$
Action Space $a_t \in$	$\{0, 1, 2, 3\}$	$\{0, 1\}$	$\{0, 1, 2\}$	$\{0, 1, 2, 3\}$
Reward $r_t =$	-1 per move, 10 for the goal and -10 for the pit	+1 per time step	-1 per time step	-1 per move, 10 for the goal and -10 for the pit

TABLE I: Specifications of Environments

Algorithm 2 CAMEO

Require: K: the number of iterations, N: number of episodes, μ a constant
Initialise Agent π and Φ the curiosity neural network
$\theta_0 \sim \mathcal{N}(\mathbf{0}, \sigma_p^2 \mathbf{I}_D)$
for k from 1 to K **do**
  $\theta' \sim \mathcal{N}(\theta_{k-1}, \sigma_p^2 \mathbf{I}_D)$
  Run N episodes with $\pi_{\theta_{k-1}}$ and store trajectories $\tau_{\theta_{k-1}}$
  for each trajectory $\tau^{(i)}_{\theta_{k-1}}$ **do**
    $\mathcal{L}(\tau^{(i)}_{\theta_{k-1}}) \leftarrow MSE(\Phi(\tau^{(i)}_{\theta_{k-1}}), \tau^{(i)}_{\theta_{k-1}})$
    $\mathcal{L}(\tau^{(i)}_{\theta'}) \leftarrow MSE(\Phi(\pi_{\theta'}(\tau^{(i)}_{\theta_{k-1}})), \pi_{\theta'}(\tau^{(i)}_{\theta_{k-1}}))$
    Train Φ using $\mathcal{L}(\tau^{(i)}_{\theta'})$
    $\mathcal{L}(\tau_{\theta_{k-1}}) \leftarrow \mathcal{L}(\tau_{\theta_{k-1}}) + \mathcal{L}(\tau^{(i)}_{\theta_{k-1}})$
    $\mathcal{L}(\tau_{\theta'}) \leftarrow \mathcal{L}(\tau_{\theta'}) + \mathcal{L}(\tau^{(i)}_{\theta'})$
  end for
  Compute $\widetilde{G}(\tau_{\theta_{k-1}})$
  $\widetilde{G}(\tau_{\theta'}) \leftarrow \widetilde{G}(\pi_{\theta'}(\tau_{\theta_{k-1}}))$
  $R(\tau_{\theta_{k-1}}) \leftarrow \mu \cdot \widetilde{G}(\tau_{\theta_{k-1}}) + (1-\mu) \cdot \mathcal{L}(\tau_{\theta_{k-1}})$
  $R(\tau_{\theta'}) \leftarrow \mu \cdot \widetilde{G}(\tau_{\theta'}) + (1-\mu) \cdot \mathcal{L}(\tau_{\theta'})$
  $\widetilde{U}(\theta_{k-1}) \leftarrow \frac{1}{N} \cdot \big(U(R(\tau_{\theta_{k-1}}))\big)$
  $\widetilde{U}(\theta') \leftarrow \frac{1}{N} \cdot (U(R(\tau_{\theta'})))$
  $\beta \leftarrow \frac{p(\theta')\widetilde{U}_N(\theta')}{p(\theta_{k-1})\widetilde{U}_N(\theta_{k-1})}$
  $\alpha \leftarrow \min(1, \beta)$
  $\epsilon \sim \mathcal{U}_{[0,1]}$
  if $\epsilon < \alpha$ **then** $\theta_k \leftarrow \theta'$
  else $\theta_k \leftarrow \theta_{k-1}$
  end if
end for
return $\{\theta_k\}_{k=1}^K$

episodes for every θ_i. The metropolis algorithm was done on 200 timesteps for Cartpole and 500 hundred for Acrobot.

Figure 1 presents the results obtained on Cartpole and Acrobot environments. Cartpole is solved while we converge towards a mean average return of -80 which is on par with great implementations according to gym leaderboard. Best implementation achieves a score of -40; we believe that our implementation can reach this score with enough iterations, which is the main drawback of MCMC methods.

When visualizing the succeeding agents on Cartpole and

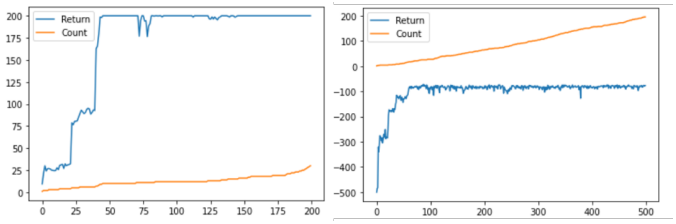


Fig. 1: Average return for every new θ (in blue) and incremental count of the number of θ_i retained (in orange) on Cartpole (left) and Acrobot (right) using simple implementation.

Acrobot, it is not obvious if they effectively adopt different behaviours. We therefore plot the cosine similarity between all pairs of retained θ_i in figure 2. It appears that succeeding θ_i are heavily correlated.

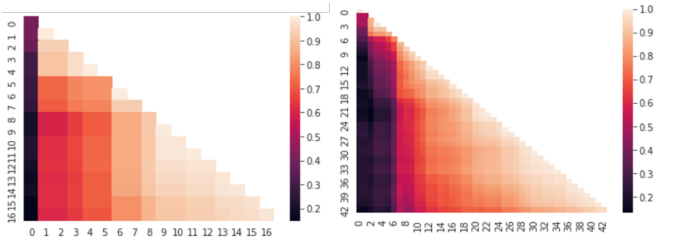


Fig. 2: Cosine similarity between pairs of retained θ_i on Cartpole (left) and Acrobot (right) using simple implementation.

However, the simple approach fails when confronted to Gridworld or Cliff. In both cases, the agent remains stuck, always performing the same action. The reasons for this failure are explained and tackled in section IV-C.

B. Results for CAMEO

In this section we present the results obtained for CAMEO on Gridworld and Cliff. We recall that for these environments, the rewards are sparse, and therefore the prior adaptation, as well as the curiosity module are necessary. Here we kept the same architecture for the agent while the curiosity module is a neural network of 1 hidden layer of size 150 with a ReLU activation function. As shown in figure 3, the model succeeds in solving those two environments. We do not present the results on Cartpole and Acrobot as they are similar to those obtained without curiosity augmentation.

It is interesting to note in figure 3 that the prediction error peaks when the model tries something new, even if it is not

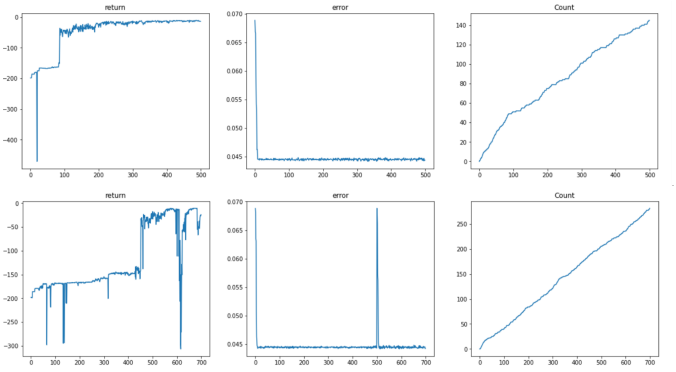


Fig. 3: CAMEO Results on Cliff (above) and Gridworld (below). The figure presents the mean return, the Prediction error and the count of θ_i retained over time steps

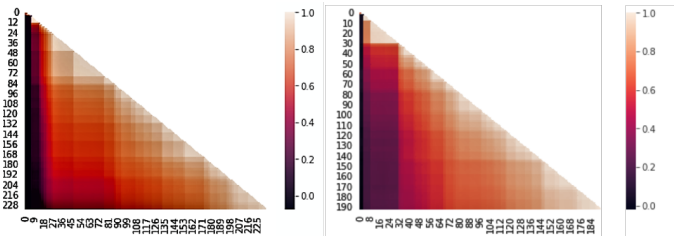


Fig. 4: Cosine similarity between pairs of retained θ_i on Gridworld (left) and Cliff (right) using CAMEO implementation.

a good move but also that it recovers quickly afterwards. Moreover, we can note that the rate of θ_i retained is nearly linear, which shows that the ratio of rejection is low and therefore that the sampling process is efficient. Finally, figure 4 shows that the θ_i retained are less correlated than in the simple implementation, which suggests that the curiosity module and the prior are effective. However non correlated weights do not always imply a different behaviour. Figure 5 shows the aggregated state visitation frequency of 100 different policies that solve the problems. The most efficient (shortest) paths are the most taken but the state visitation frequencies are non negligible for other paths. Therefore, the learned policies effectively correspond to different behaviours.

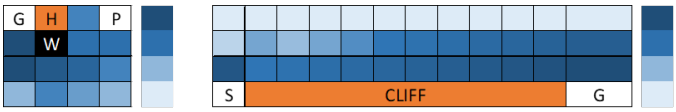


Fig. 5: State visitation frequency aggregated on 100 policies obtained using CAMEO on Gridworld and Cliff. Less visited states are in light blue and most visited ones in dark shade

VI. Conclusion

In the context of reinforcement learning, we were able to sample optimal policies on the fly and showed that the resulting behaviours are diverse. We also showed that our approach succeeded to output optimal policies even when the rewards structure of our problems were sparse, by using a curiosity module that encouraged exploration dynamically.

Our approach still bears some limitations as the policy spaces of studied environments are discrete. As is generally the case with Monte Carlo methods, convergence is more challenging in high dimensions. To mitigate this, it is possible to replace the standard proposal distribution with a policy guided proposal that draws educated samples. The new distribution could therefore explore larger spaces by focusing on areas of interest. This will be subject to future work.

References

[1] Vahid Tavakol Aghaei, Ahmet Onat, and Sinan Yıldırım. A markov chain monte carlo algorithm for bayesian policy search. *Systems Science & Control Engineering*, 6(1):438–455, 2018.

[2] Shalabh Bhatnagar, Richard Sutton, Mohammad Ghavamzadeh, and Mark Lee. Natural Actor-Critic Algorithms. *Automatica*, 45(11), July 2009.

[3] Greg Brockman, Vicki Cheung, Ludwig Pettersson, Jonas Schneider, John Schulman, Jie Tang, and Wojciech Zaremba. Openai gym, 2016.

[4] Chelsea Finn, Paul Christiano, Pieter Abbeel, and Sergey Levine. A connection between generative adversarial networks, inverse reinforcement learning, and energy-based models, 2016.

[5] Ian J. Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial networks, 2014.

[6] W.K. Hastings. Monte carlo sampling methods using markov chains and their applications. *Biometrika*, 57(1):97–109, 1970.

[7] Jonathan Ho and Stefano Ermon. Generative adversarial imitation learning, 2016.

[8] Matthew Hoffman, Arnaud Doucet, Nando Freitas, and Ajay Jasra. Bayesian policy learning with trans-dimensional mcmc. In J. Platt, D. Koller, Y. Singer, and S. Roweis, editors, *Advances in Neural Information Processing Systems*, volume 20. Curran Associates, Inc., 2008.

[9] F. Llorente, L. Martino, J. Read, and D. Delgado. A survey of Monte Carlo methods for noisy and costly densities with application to reinforcement learning. *arXiv preprint arXiv:2108.00490*, 2021.

[10] F. Llorente, L. Martino, J. Read, and D. Delgado. Optimality in Noisy Importance Sampling. *Signal Processing*, page 108455, 2022.

[11] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin Riedmiller. Playing atari with deep reinforcement learning, 2013.

[12] Deepak Pathak, Pulkit Agrawal, Alexei A. Efros, and Trevor Darrell. Curiosity-driven exploration by self-supervised prediction, 2017.

[13] David Pfau and Oriol Vinyals. Connecting generative adversarial networks and actor-critic methods, 2017.

[14] Martin L. Puterman. *Markov Decision Processes: Discrete Stochastic Dynamic Programming*. John Wiley and Sons, Inc., New York, NY, USA, 1st edition, 1994.

[15] Richard Sutton, David Mcallester, Satinder Singh, and Yishay Mansour. Policy gradient methods for reinforcement learning with function approximation. *Adv. Neural Inf. Process. Syst*, 12, 02 2000.

[16] Richard S. Sutton and Andrew G. Barto. *Reinforcement Learning: An Introduction*. The MIT Press, second edition, 2018.

[17] Christopher J. C. H. Watkins and Peter Dayan. Q-learning. *Machine Learning*, 8(3):279–292, May 1992.