Using Proto-Value Functions for Curriculum Generation in Goal-Conditioned RL

Henrik Metternich[1], **Ahmed Hendawy**[1,2*], **Jan Peters**[1,2], **Carlo D'Eramo**[2,3]

[1]Technische Universität Darmstadt, Germany

[2]Hessian.AI, Germany [3]University of Würzburg, Germany

Abstract

In this paper, we investigate the use of Proto Value Functions (PVFs) for measuring the similarity between tasks in the context of Curriculum Learning (CL). PVFs serve as a mathematical framework for generating basis functions for the state space of a Markov Decision Process (MDP). They capture the structure of the state space manifold and have been shown to be useful for value function approximation in Reinforcement Learning (RL). We show that even a few PVFs allow us to estimate the similarity between tasks. Based on this observation, we introduce a new algorithm called Curriculum Representation Policy Iteration (CRPI) that uses PVFs for CL, and we provide a proof of concept in a Goal-Conditioned Reinforcement Learning (GCRL) setting.

1 Introduction

Reinforcement Learning (RL) [17] is a branch of machine learning that deals with sequential decision-making and has been successfully applied to a wide range of problems [8, 10]. One of the main challenges is that RL algorithms often require many interactions with the environment to learn a good policy. Especially in long-horizon tasks, this can be very time-consuming, expensive, or simply infeasible. Curriculum Learning (CL) [16] is a learning paradigm that aims to address this problem by presenting the agent with a sequence of tasks of increasing difficulty. The success of CL depends on the choice of the curriculum and the order in which the tasks are presented to the agent. In particular, the tasks should be similar to each other, but also different enough to provide new challenges to the agent. This is a difficult problem because it is not obvious how to measure the similarity between learning tasks. One way to approach this problem is to find a suitable representation of the tasks and then measure the similarity between these representations. Ideally, similar tasks should have similar representations and vice versa. In this context, Proto Value Functions (PVFs) [12, 14] emerge as a promising candidate for representing reinforcement learning tasks. PVFs serve as a mathematical framework for generating basis functions $\phi(s)$ for a given Markov Decision Process (MDP) based on diffusion models [5], more specifically the graph Laplacian. They capture the structure of the state-space manifold by design, and thus create very compact basis functions for value function approximation over the state space. We further detail PVFs in Appendix A. Because of their compact representation, PVFs have been successfully applied to many different problems, such as transfer learning [6, 2, 3], option discovery [11], and reward shaping [19, 18]. While PVFs can be computed analytically in small discrete MDPs, they have to be approximated from observed transitions in bigger or even continuous MDPs [15, 14]. The efficient computation of PVFs in large or continuous MDP is still an open problem and is addressed by recent research [19, 18]. In this paper, we investigate a graph-based approach to CL. In particular, we extend Representation Policy Iteration (RPI)[13, 14], by leveraging the use of a curriculum based on a graph structure imposed by PVFs.

*Ahmed Hendawy (ahmed.hendawy@tu-darmstadt.de) is the corresponding author.

NeurIPS 2023 Workshop on Goal-Conditioned Reinforcement Learning.

2 Curriculum Representation Policy Iteration

We consider a set of goal-directed tasks, formalized as a Contextual Markov Decision Process (CMDP)[7]. A CMDP is a tuple $(\mathcal{C}, \mathcal{S}, \mathcal{A}, \mathcal{M})$, where $\mathcal{C}$ is a set of contexts, $\mathcal{S}$ is a set of states, $\mathcal{A}$ is a set of actions and $\mathcal{M}$ is a set of MDPs. For each context $c \in \mathcal{C}$, we obtain an MDP $\mathcal{M}^c = (\mathcal{S}, \mathcal{A}, p^c, r^c)$. By choosing the context $\mathcal{C}$ to be a subset of the state space $\mathcal{C} \subseteq \mathcal{S}$, we can directly encode a set of goal-directed tasks, where $c \in \mathcal{C}$ is the goal of the task $\mathcal{M}^c$. We assume that the transition probabilities p^c are identical for all tasks, such that the PVFs can be shared across them. We can construct a basis $\phi(s)$ over the state space from the n smallest PVFs, resulting in a representation w^c for each task c by approximating its value function $V^c(s) \approx \phi(s)^T w^c$. These representations implicitly give rise to a graphical structure of the tasks. Each task can be considered a node. Nodes are connected if their representations w^c are close to each other with respect to a distance metric. Based on this graph-structure we can define a curriculum, by finding a path through the graph that connects an initial trivial task with the final task of interest. In practice, we compute the PVFs, the task representations w^c, and with that the similarity graph based on the current *partial* knowledge of the environment, initially collected in the trivial first task. We then use the approximate similarity graph to construct a curriculum. By solving tasks in the curriculum, new information is obtained which we use to update the PVFs and with that the similarity graph. This process is repeated until the agent has solved the final task. The resulting algorithm, Curriculum Representation Policy Iteration (CRPI), is visualized in Figure 1. Following is an algorithmic description of CRPI (Algorithm 1), together with further textual explanations. Additional algorithmic details of CRPI and RPI can be found in Appendix B.

Algorithm 1 Curriculum RPI

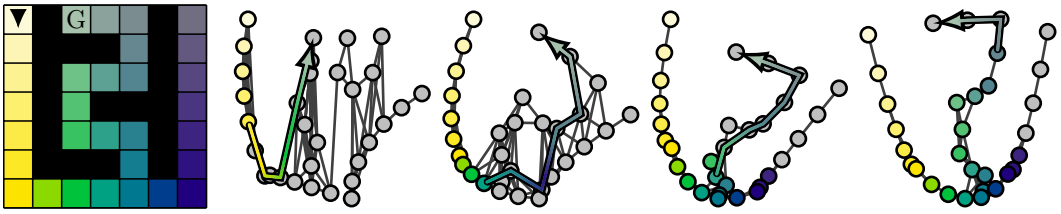
Input: $\mathcal{C} \subseteq \mathcal{S}$ a set of contexts
$c_0, c_{goal} \in \mathcal{C}$ start and goal state
$\mathcal{M}^{\cdot}$ a set of tasks, where $\mathcal{M}^c = (\mathcal{S}, \mathcal{A}, p, r^c)$
$\mathcal{D}$ initial data constisting of random trajectories

$c = c_0$
repeat
  visited = $\{s | (s, a, r, s') \in \mathcal{D}\}$
  if $c \in$ visited **then**
    $\mathcal{G}$ = SimilarityGraph($\mathcal{D}$, $\mathcal{C}$, $\mathcal{M}$)
    $c, c_1, \ldots, c_{goal}$ = ShortestPath($\mathcal{G}$, c, c_{goal})
    $c = c_1$
  end if
  π = LSPI($\mathcal{D}^c$, PVFs($\mathcal{D}^c$, $\mathcal{M}$), ϵ)
  $\mathcal{D} = \mathcal{D} \cup$ Sample($\mathcal{M}^c$, π)
until $c = c_{goal} \wedge c_{goal} \in$ visited

return π

Opposed to Representation Policy Iteration (RPI), which trains the agent in a single learning task, Curriculum Representation Policy Iteration (CRPI) iteratively chooses a new task from a set of tasks $\mathcal{C} \subseteq \mathcal{S}$ to present the agent with a task sequence of increasing difficulty. In addition to the set of contexts, the algorithm also requires to specify an initial and a goal task $c_0, c_{goal} \in \mathcal{C}$. The tasks are represented as CMDPs $\mathcal{M}^c = (\mathcal{S}, \mathcal{A}, p, r^c)$, where the reward function r^c is a function of the context $c \in \mathcal{C}$. We assume knowledge of the reward function r^c in order to *relabel* a transition $(s, a, r_{c_1}(s, a), s')$ in context c_1 into a valid transition in another context c_2, i.e., $(s, a, r_{c_2}(s, a), s')$. This assumption is not novel and has been, e.g., exploited in the well konwn hindsight experience replay algorithm [1]. Finally, the algorithm requires an initial dataset $\mathcal{D}$, consisting of random trajectories. These trajectories are required to compute the initial PVFs and similarity graph (Appendix B).

The algorithm starts by initializing the current task c to an initial, trivial task c_0. Then it iteratively updates the current task c until it reaches the goal task c_{goal}. The current task c is only updated if it has been solved, i.e. if the agent has visited the goal state c at least once. Hence, in each iteration the algorithm first checks if the c has been solved. If this is the case, it computes the similarity graph $\mathcal{G}$ using the current dataset $\mathcal{D}$, the set of contexts $\mathcal{C}$, and the corresponding tasks $\mathcal{M}$ as described in Algorithm 4. It then calculates the shortest path from the current task c to the goal task c_{goal} in the



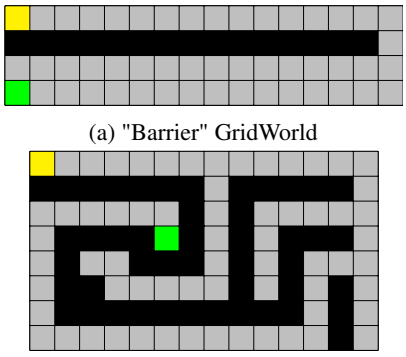
(a) "Maze" Gridworld. (b) Iteration 3. (c) Iteration 5. (d) Iteration 7. (e) Iteration 12.

Figure 1: The iterative nature of CRPI. There is a one-to-one correspondence between states (Figure 1a) and tasks (Figures 1b to 1e), indicated by the color. Gray color represents unvisited states. G indicates the goal. The start position is indicated by the triangle. Based on the current similarity graph, we can construct a curriculum, from which we can choose the next task to solve. By solving this task, we obtain new information which we can use to update the similarity graph.

similarity graph $\mathcal{G}$. The next task is then set to the first task on the shortest path. If the current task c has not been solved, the algorithm simply continues to train on the current task c. In both cases, the algorithm then uses the current task c to compute a new policy π using Least Squares Policy Iteration (LSPI) and the current dataset $\mathcal{D}$. This new policy π is then executed in the current task c. The resulting trajectories are added to the dataset $\mathcal{D}$. At the end of each iteration, the algorithm checks if the goal task c_{goal} has been reached. If so, the algorithm terminates and returns the final policy π.

3 Evaluation

We evaluated out approach in three different GridWorld-Environments. The first one is the Gridworld environment shown already shown in Figure 1a, whose combination of $L2$ reward and walls present a prototypical exploration challenge. The agent starts in the top left corner and has to reach the goal at the end of the wiggled corridor. We compare the CRPI method shown in Figure 1 to RPI [14]. Analogous to RPI, we make use of LSPI [9] and PVFs to solve the individual tasks. In addition, we evaluate an ablation of our approach, where we use Fourier basis functions instead of PVFs to construct the curriculum. As shown in Figure 3, CRPI allows to find better policies than RPI. Furthermore, PVFs seem better suited for approximating the structure between learning tasks from limited data, resulting in better curricula compared to Fourier features.



(a) "Barrier" GridWorld

(b) "Big Maze" GridWorld

Figure 2: The "Barrier" Gridworld MDP. The starting position is indicated in yellow and the goal is highlighted in green. The reward function for a given state is the negative L2 distance to the goal.

The second evaluation environment is the "Barrier" MDP, shown in Figure 2a. This environment is another classical example of an exploration task, since the agent has to first move away from the reward signal in order to eventually reach the goal. As we can see in Figure 3b, CRPI using PVFs is clearly able to reliably solve the task. As expected, the performance decreases at first, as the agent has to navigate around the barrier. However, after a few iterations, the agent is able to reach the goal state and the performance increases. In contrast to that, RPI is not able to solve the task at all. The reward stays constant, as the agent is not able to exit the barrier. The performance of CRPI using Fourier basis functions is significantly worse than the performance of CRPI using PVFs. This is due to the fact that the similarity graph obtained from the Fourier features is significantly worse than the ones obtained from PVFs. This consequently leads to the agent picking difficult tasks, which it is not able to solve, ultimately resulting in inferior performance compared to using PVFs.

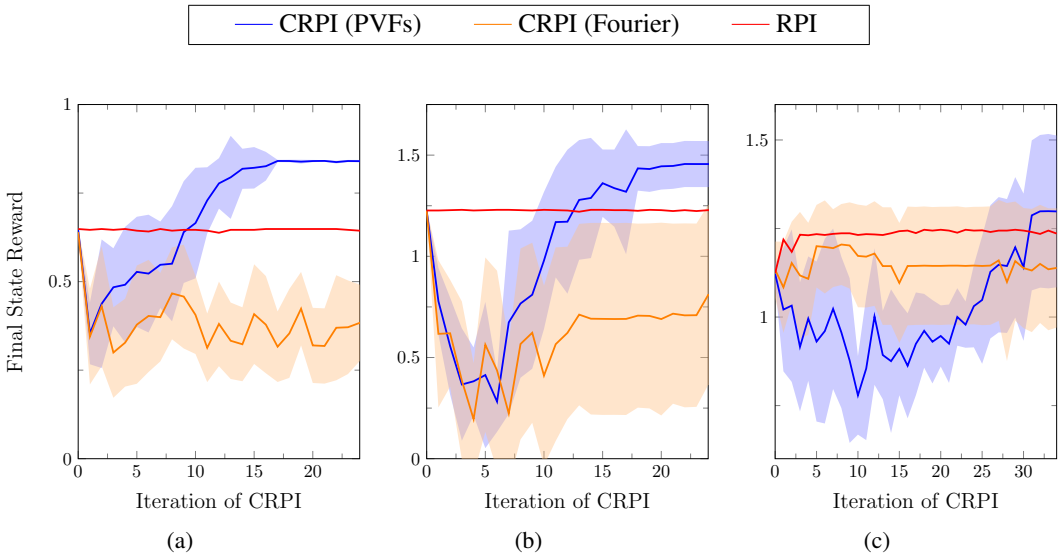


Figure 3: The reward of the last state in each episode for the CRPI and RPI algorithm on the "Maze" (Figure 1a), "Barrier" (Figure 2a), and "Big Maze" (Figure 2b) MDP. The evaluations are shown in Figures 3a, 3b, and 3c respectively. The reward is calculated by computing the negative L_2 distance between the final state of the episode and the goal state and averaged over 10 runs of each algorithm. The shaded areas represent the 95% confidence intervals.

Similar results can be observed for the ”Big-Maze” environment (Figure 2b) that we display in Figure 3c. Here, the margin between CRPI using PVFs and the RPI baseline is much smaller since the required detour to reach the goal state is longer than in both previous environments. Nevertheless, the mean performance of CRPI still outperforms the baseline, indicating that at least a few runs were able to solve the task. In comparison, RPI always converges to the same local optima, not reaching the goal state. The evaluation shows that CRPI, given the right basis functions, is able to use the curriculum to learn a policy that solves the task.

4 Conclusion

In this paper, we showed that PVFs can be used to measure the similarity between learning tasks. Consequently, we introduced a method that builds curricula using PVFs. We have provided a proof of concept, by applying CRPI to GridWorld environments. By leveraging the curriculum CRPI is able to solve several exploration tasks, that RPI was not able to solve, assuming same amount of random exploration. However, as it turns out the construction of the similarity graph is very sensitive to approximation errors. While computing the actual representation requires only a few PVFs to obtain a useful curriculum, running LSPI for each task will not converge properly unless given access to (almost) the full set of basis functions. Therefore it will be interesting to see whether or not CRPI will scale beyond this initial proof of concept. In future work, we will investigate the use of PVFs for CL in more complex environments with larger discrete- or continuous state-action spaces.

Acknowledgments and Disclosure of Funding

This work was funded by the German Federal Ministry of Education and Research (BMBF) (Project: 01IS22078). This work was also funded by Hessian.ai through the project ’The Third Wave of Artificial Intelligence – 3AI’ by the Ministry for Science and Arts of the state of Hessen. In addition, we thank Pascal Klink for the fruitful discussion about the paper.

References

[1] M. Andrychowicz, F. Wolski, A. Ray, J. Schneider, R. Fong, P. Welinder, B. McGrew, J. Tobin, O. Pieter Abbeel, and W. Zaremba. Hindsight experience replay. *Advances in neural information processing systems*, 2017.

[2] A. Barreto, W. Dabney, R. Munos, J. J. Hunt, T. Schaul, H. P. van Hasselt, and D. Silver. Successor features for transfer in reinforcement learning. *Advances in neural information processing systems*, 30, 2017.

[3] A. Barreto, D. Borsa, J. Quan, T. Schaul, D. Silver, M. Hessel, D. Mankowitz, A. Zidek, and R. Munos. Transfer in deep reinforcement learning using successor features and generalised policy improvement. In *International Conference on Machine Learning*, pages 501–510. PMLR, 2018.

[4] S. J. Bradtke and A. G. Barto. Linear least-squares algorithms for temporal difference learning. *Machine learning*, 22:33–57, 1996.

[5] R. R. Coifman, S. Lafon, A. B. Lee, M. Maggioni, B. Nadler, F. Warner, and S. W. Zucker. Geometric diffusions as a tool for harmonic analysis and structure definition of data: Diffusion maps. *Proceedings of the national academy of sciences*, 102(21):7426–7431, 2005.

[6] K. Ferguson and S. Mahadevan. Proto-transfer learning in markov decision processes using spectral methods. *Computer Science Department Faculty Publication Series*, page 151, 2006.

[7] A. Hallak, D. Di Castro, and S. Mannor. Contextual markov decision processes. *arXiv preprint arXiv:1502.02259*, 2015.

[8] J. Kober, J. A. Bagnell, and J. Peters. Reinforcement learning in robotics: A survey. *The International Journal of Robotics Research*, 32(11):1238–1274, 2013.

[9] M. G. Lagoudakis and R. Parr. Least-squares policy iteration. *The Journal of Machine Learning Research*, 4:1107–1149, 2003.

[10] T. P. Lillicrap, J. J. Hunt, A. Pritzel, N. Heess, T. Erez, Y. Tassa, D. Silver, and D. Wierstra. Continuous control with deep reinforcement learning. *arXiv preprint arXiv:1509.02971*, 2015.

[11] M. C. Machado, M. G. Bellemare, and M. Bowling. A laplacian framework for option discovery in reinforcement learning. In *International Conference on Machine Learning*, pages 2295–2304. PMLR, 2017.

[12] S. Mahadevan. Proto-value functions: Developmental reinforcement learning. In *Proceedings of the 22nd international conference on Machine learning*, pages 553–560, 2005.

[13] S. Mahadevan. Representation policy iteration. *arXiv preprint arXiv:1207.1408*, 2012.

[14] S. Mahadevan and M. Maggioni. Proto-value functions: A laplacian framework for learning representation and control in markov decision processes. *Journal of Machine Learning Research*, 8(10), 2007.

[15] S. Mahadevan, M. Maggioni, K. Ferguson, and S. Osentoski. Learning representation and control in continuous markov decision processes. In *AAAI*, volume 6, pages 1194–1199, 2006.

[16] S. Narvekar, B. Peng, M. Leonetti, J. Sinapov, M. E. Taylor, and P. Stone. Curriculum learning for reinforcement learning domains: A framework and survey. *The Journal of Machine Learning Research*, 21(1):7382–7431, 2020.

[17] R. S. Sutton and A. G. Barto. *Reinforcement learning: An introduction*. MIT press, 2018.

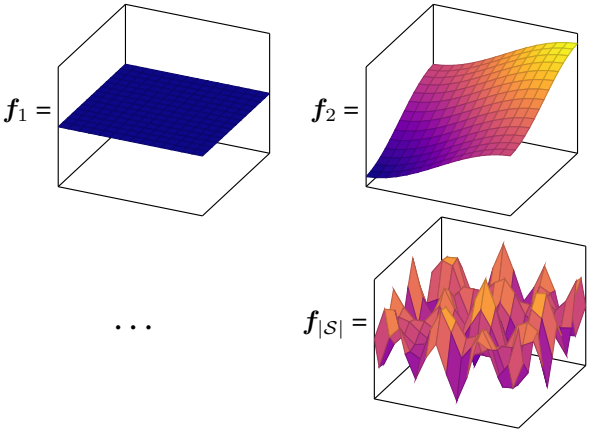
[18] K. Wang, K. Zhou, Q. Zhang, J. Shao, B. Hooi, and J. Feng. Towards better laplacian representation in reinforcement learning with generalized graph drawing. In *International Conference on Machine Learning*, pages 11003–11012. PMLR, 2021.

[19] Y. Wu, G. Tucker, and O. Nachum. The laplacian in rl: Learning representations with efficient approximations. *arXiv preprint arXiv:1810.04586*, 2018.

A Proto-Value Functions

PVFs are a mathematical framework for generating basis functions for a given MDP. In the context of RL, PVFs were first introduced by Mahadevan [12][14]. In contrast to other common basis functions, PVFs are not learned from the rewards, but instead from analyzing the state space manifold. In a discrete MDP, the state space manifold can be represented as an adjacency graph $A = (V, E)$, where the nodes are the states $V = \mathcal{S}$ and the edges are the transitions between the states $E \subseteq \mathcal{S} \times \mathcal{S}$. The PVFs can be obtained by solving a generalized eigenvalue problem on the graph Laplacian matrix $L = D - A$, where D is the degree matrix of the graph. This yields a set of eigenvalues $\lambda_1, \ldots, \lambda_{|\mathcal{S}|}$ and a set of eigenfunctions $\boldsymbol{f}_1, \ldots, \boldsymbol{f}_{|\mathcal{S}|}$. The PVFs constitute a complete basis for the space of value functions on $\mathcal{S}$. Consequently, any function can be specified in terms of a linear combination of the PVFs to an arbitrary precision. However, in practice, and this is the interesting property, only a small number of PVFs are required to obtain a good approximation. This property makes PVFs so useful for function approximation in RL and sparked our interest in using them for measuring task similarity. The eigenvalue of each PVF can be interpreted as a sort of frequency, where lower-valued eigenvectors capture the coarse structure of the environment, while higher-valued eigenvectors encode more detailed aspects.

$$\phi(s) = \begin{pmatrix} \boldsymbol{f}_1(s) \\ \boldsymbol{f}_2(s) \\ \vdots \\ \boldsymbol{f}_n(s) \\ f_{n+1}(s) \\ \vdots \\ f_{|\mathcal{S}|}(s) \end{pmatrix}$$



, where

$$\begin{array}{ccccccc} \boldsymbol{f}_1, & & \boldsymbol{f}_2, & & \boldsymbol{f}_3, & \ldots & \boldsymbol{f}_{|\mathcal{S}|} \\ \lambda_1, & \leq & \lambda_2, & \leq & \lambda_3, & \ldots & \lambda_{|\mathcal{S}|} \end{array}$$

Figure 4: An illustration of the intuition behind the basis construction using PVFs. The PVFs are computed from a square Gridworld with no barriers. The frequency of the PVFs is directly related to their eigenvalues. The smallest PVF $\boldsymbol{f}_1$ is always a constant-valued function with eigenvalue 0.

Figure 5 shows the first eight PVF of the "Big Barrier" Gridworld, shown in Figure 2a. Keep in mind that the values at the location of a barrier (black) is always 0. As we can see, the PVFs adapt to structure of the environments state space. It is clearly visible that the oscillation frequencies of the basis functions scale in relation with increasing magnitude of the corresponding eigenvalues.

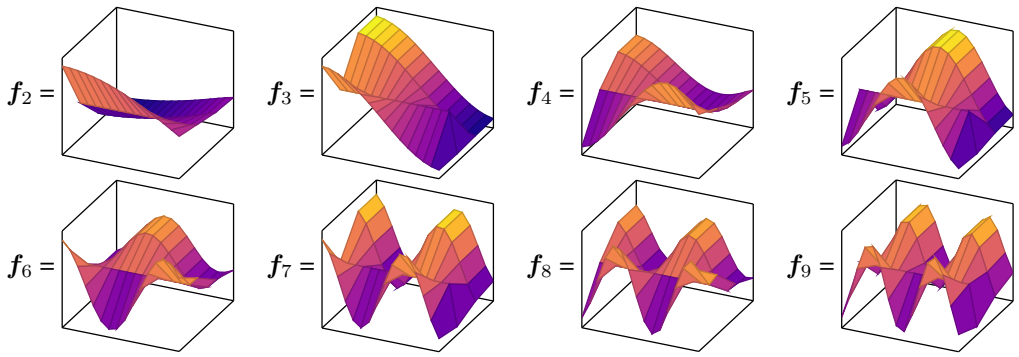


Figure 5: Eight PVFs of the "Big Barrier" Environment. The PVFs are ordered by their corresponding eigenvalues. The first PVF is not shown due to the fact that it is always constant-valued.

B Algorithmic Approach

This section presents a comprehensive view of the algorithmic aspects of CRPI. We present a high level view of CRPI in Algorithm 1. In addition to the main algorithm, we also include all of the foundations and utility algorithms which it is based on. This includes a short overview of LSPI and RPI, shown in Algorithm 2 and 3 respectively, and an algorithmic explanation of the similarity graph construction in Algorithm 4.

Algorithm 2 LSPI

Input: $\mathcal{D}$ set of samples (s, a, r, s')
ϕ feature function
ϵ stopping criterion

w' = w_0
repeat
  w = w'
  $\pi(s)$ = $\arg\max_a w^T \phi(s, a)$
  w' = LSTD$Q(\mathcal{D}, \phi, \pi)$
until $||w - w'|| < \epsilon$

return π

Least Squares Policy Iteration (LSPI)[9], as shown in Algorithm 2, can be summarized as an iterative application of Least Squares Temporal Difference Q-Learning (LSTDQ)[4], that applies a greedy update to the assumed sampling policy of the dataset. LSTDQ solves a quadratic optimization problem, that minimizes the error between the true Q-function Q^π and the approximation $w^T \phi$. The solution to this objective is given by a linear equation $Aw = b$, where A and b are defined as follows

$$A = \sum_{(s,a,r,s') \in \mathcal{D}} \phi(s, a) \left(\phi(s, a) - \gamma \phi(s', \pi(s'))\right)^T$$

$$b = \sum_{(s,a,r,s') \in \mathcal{D}} \phi(s, a) r.$$

Here π is assumed to be the sampling policy of the Q-function we want to estimate. Since LSPI greedily updates the sampling policy without changing the dataset, LSPI is strongly biased by the initial sampling distribution.

Algorithm 3 RPI

Input: $\mathcal{D}$ set of samples (s, a, r, s')
ϵ stopping criterion

w' = w_0
ϕ = PVFs($\mathcal{D}$, $\mathcal{M}$)
repeat
  w = w'
  $\pi(s)$ = $\arg\max_a w^T \phi(s, a)$
  w' = LSTD$Q(\mathcal{D}, \phi, \pi)$
  $\mathcal{D}$ = $\mathcal{D} \cup$ Sample($\mathcal{M}$, π)
  ϕ = PVFs($\mathcal{D}$, $\mathcal{M}$)
until $||w - w'|| < \epsilon$

return π

Representation Policy Iteration (RPI)[12, 14] is an extension of LSPI, that uses PVFs as the basis functions of the linear approximator. In contrast to LSPI, the basis functions are not fixed, but can be updated during the execution by collecting samples with an intermediate policy.

Algorithm 4 for computing the similarity graph of tasks starts by computing a representation w^c for each task $\mathcal{M}^c$ using the relabeled dataset $\mathcal{D}^c$. The representations are computed by running LSPI

Algorithm 4 SimilarityGraph

Input: $\mathcal{D}$ set of samples (s, a, r, s')
$\mathcal{C}$ a set of contexts
$\mathcal{M}^{\cdot}$ a set of tasks, where $\mathcal{M}^c = (\mathcal{S}, \mathcal{A}, p, r^c)$
n_{PVFs} number of PVFs used

ϕ' = PVFs($\mathcal{D}, \mathcal{M}$)
$\phi = \phi'_{1,\ldots,n_{\text{PVFs}}}$
for $c \in \mathcal{C}$ **do**
  $\mathcal{D}^c$ = Relabel($\mathcal{D}$)
  π^c = LSPI($\mathcal{D}^c$, ϕ', ϵ)
  $w^c = (\phi^T\phi)^{-1}\phi^T V^{\pi^c}$
end for
$E = \emptyset$
repeat
  $\beta = \min \{||w^{c_1} - w^{c_2}|| \mid (c_1, c_2) \in \mathcal{C} \times \mathcal{C} \setminus E\}$
  $E = \{(c_1, c_2) \mid c_1, c_2 \in \mathcal{C}, ||w^{c_1} - w^{c_2}|| \leq \beta\}$
until $E^* = \mathcal{C} \times \mathcal{C}$

return $(\mathcal{C}, E)$

to obtain an approximation of the optimal policy and it's value function V^{π_c} for $\mathcal{M}^c$, from which then the representation w^c can be obtained via linear regression. We then iteratively grow a graph $\mathcal{G}$ starting from a graph with no edges, i.e., $\mathcal{G} = (\mathcal{C}, \emptyset)$, and including edges between two contexts $c_1 \in \mathcal{C}$ and $c_2 \in \mathcal{C}$ whose representations differ by no more than a threshold β. We repeatedly increase the threshold β to the minimum representation distance $\|w^{c_1} - w^{c_2}\|$ of contexts c_1 and c_2 that are not yet connected by an edge. We then add all edges that are below the updated threshold to the set of edges E. This process is repeated until the graph is connected, i.e. the reflexive transitive closure E^* of the edge set E is exactly the set of all edges $\mathcal{C} \times \mathcal{C}$. The resulting graph $\mathcal{G} = (\mathcal{C}, E)$ is then returned.