Meta Dynamic Programming

Pierluca D'Oro **Pierre-Luc Bacon**
Mila, Université de Montréal
`{pierluca.doro, pierre-luc.bacon}@mila.quebec`

Abstract

To accelerate the pace at which they acquire new information, reinforcement learning algorithms can select which data to use first for training. In this paper, we outline a general methodology to perform this selection process, hinting at a generation of agents which deeply think about their current and future learning state while selecting their training data. In the context of prioritization methods for asynchronous dynamic programming, we propose a meta-level technique for state selection. We show that the method, called meta dynamic programming, together with its approximations, can provide promising performance improvements while being grounded on a theoretically sound metacognitive formalization.

1 Introduction

High-level cognition permeates many aspects of human intelligence, but it is not always easy to reproduce and leverage in artificial agents. One of the most crucial of these aspects is the use of higher levels of cognition for selecting which experience to use next for learning: typically, a human uses some form of model of its future learning to assess which experience can be maximally useful given his/her specific learning aptitudes [2, 5]. The goal of this paper is to discuss a formalization for the problem of selecting on which data point (i.e., state) to learn next in the context of asynchronous dynamic programming [8].

Acceleration of dynamic programming via state prioritization comes often from heuristics based on the structure of the underlying Markov Decision Process (MDP) [13]. But, as previously observed in the context of curriculum learning [4, 6], another, equally important, structure can be leveraged: the one created by the learning process as a whole [11]. We propose to construct a meta MDP on top of the original one, in which the reward function is the learning progress [7, 9, 11] of the original agent and the dynamics is the one implied by the learning algorithm. A meta policy, trained in the meta MDP, understands the learning state of the agent, as well as the dynamics of its future improvements, and dynamically selects the state that will be used to update its control policy.

We apply this technique to both policy iteration and value iteration. We prove that, under a budget of updates, the sequence of states suggested by the optimal meta policy leads to the largest possible value achievable by policy iteration and we show that, in practice, our algorithms significantly accelerate dynamic programming.

2 Background

We will discuss in the rest of the paper two classes of MDPs [8]. An infinite-horizon Markov Decision Process is a tuple $\mathcal{M} = (\mathcal{S}, \mathcal{A}, P, r, \gamma)$, where $\mathcal{S}$ is the state space, $\mathcal{A}$ is the action space, $P \in \mathbb{R}^{|\mathcal{S}| \times |\mathcal{A}| \times |\mathcal{S}|}$ is the transition matrix, $r \in \mathbb{R}^{|\mathcal{S}| \times |\mathcal{A}|}$ is the reward matrix and γ is the discount factor. In this class of MDPs, the sequence of interactions of the agent with the environment has an infinite length. A finite-horizon Markov Decision Process is a tuple $\mathcal{M}_H = (\mathcal{S}, \mathcal{A}, P, r, H)$, where H is the horizon and the other elements of the tuple are defined similarly to infinite-horizon

MDPs. We will mostly focus on the first class of MDPs, in which the agent selects its actions according to a policy $\pi \in \mathbb{R}^{|\mathcal{S}|\times|\mathcal{A}|}$. Each policy can be characterized by using a value function $V^\pi \in \mathbb{R}^{|\mathcal{S}|}$, defined, for each state, as $V^\pi(s) = \mathbb{E}_{p,\pi}[\sum_{t=0}^{\infty}\gamma^t r(S_t, A_t)|S_0 = s]$, where parentheses denote accessing specific elements of a matrix. Denoting by $P^\pi \in \mathbb{R}^{|\mathcal{S}|\times|\mathcal{S}|}$ and $r^\pi \in \mathbb{R}^{|\mathcal{S}|\times|\mathcal{S}|}$ the transition and reward matrices conditioned on a particular policy, the value function obeys to the *policy evaluation equation* $V^\pi = r^\pi + \gamma P^\pi V^\pi$. To find the optimal policy with maximizes V^π at each state, we consider asynchronous versions of value iteration, which directly updates a value function as $V(s) = \max_a\{r(s,a) + \gamma\sum_{s'} P(s'|s,a)V(s')\}$ and policy iteration, which iteratively evaluates a policy and updates it as $\pi(s) = \arg\max_a\{r(s,a) + \gamma\sum_{s'} P(s'|s,a)V^\pi(s')\}$. The goal of prioritization methods is to select, at each iteration, a state s to update the policy or value function.

3 Meta Dynamic Programming

The idea behind meta dynamic programming is to build a *meta MDP* on top of the original MDP, with specifically designed state space, action space, meta dynamics and reward function. To construct the meta MDP, we start from its final goal: learning a *meta policy* μ which optimally selects the new state to provide to the asynchronous dynamic programming algorithm in order to learn fast.

Thanks to this perspective, we already implicitly defined the action space $\mathcal{A}_{\mathfrak{M}} \equiv \mathcal{S}$, which effectively corresponds to the state space of the original MDP. The missing components for the construction of the meta MDP, which are the state space, the dynamics and the reward function, directly depend upon the learning state and dynamics, and thus upon the underlying dynamic programming algorithm used for optimizing a policy in the original MDP. For this reason, we do now a particularization of the precise definition of the meta MDP in both the asynchronous policy iteration and asynchronous value iteration cases.

3.1 Meta Policy Iteration

In the meta MDP, we want the meta policy μ to take its decisions on which state to pick next to depend on the current *learning state* in the underlying MDP. This is, in general, because a state that can lead to fast training in a given moment in the optimization process can also lead to slow training in another one.

In policy iteration, it is natural to maintain an explicit representation of the policy π across iterations. This representation fully captures the current learning state of the agent: we would like our meta policy μ to take decisions on the next state to be sampled depending on the current policy. Although the space of stochastic policies is potentially infinite, the policy iteration algorithm proceeds in greedification steps, thus moving from a deterministic policy to another. For this reason, we can define the state space of the meta MDP to be $\mathcal{S}_{\mathfrak{M}} \equiv \Pi_{\mathcal{M}}$, where $\Pi_{\mathcal{M}}$ is the space of deterministic policies in MDP $\mathcal{M}$.

Let us now define the dynamics of the meta MDP, which is given by the (deterministic) asynchronous policy iteration step, given a policy π and a state s:

$$\mathfrak{p}_{\mathcal{M}}(\pi'|\pi, s) \triangleq \mathbb{1}\left[\pi'(\bar{s}) = \begin{Bmatrix} \arg\max_a\{r(\bar{s},a) + \gamma\sum_{s'} P(s'|\bar{s},a)V^\pi(s')\}, & \text{if } \bar{s} = s \\ \pi(\bar{s}), & \text{if } \bar{s} \neq s \end{Bmatrix}\right]. \quad (1)$$

Since the objective of the meta policy is to accelerate learning in the original MDP, it is natural to conceive a meta reward function based on the *learning progress* of the underlying policy. Defining a scalar measure of progress requires to have a hierarchy, or simply a distribution over states. We assume there exists a distribution $u \in \Delta(\mathcal{S})$ over states, that, in the simplest and perhaps most common case, can be assumed to be uniform. In this case, the meta reward can be defined as:

$$\mathfrak{r}_{\mathcal{M}}(\pi, s, \pi') \triangleq \mathbb{E}_{s'\sim u}\left[V^{\pi'}(s') - V^\pi(s')\right]. \quad (2)$$

Note that it is possible to obtain an equivalent problem by considering a reward function $\mathfrak{r}_{\mathcal{M}}(\pi, s) \triangleq \mathbb{E}_{\pi'\sim\mathfrak{p}_{\mathcal{M}}(\pi'|\pi,s)}[\mathfrak{r}_{\mathcal{M}}(\pi, s, \pi')]$, which does not depend on the next policy. Lastly, a discount factor β can be introduced. In a parallel way w.r.t. the underlying MDP, it can have the effect of favoring states which will lead to fast improvements for policy iteration. With all the components in place, the meta MDP can be formally defined as a tuple $\mathfrak{M} = (\Pi_{\mathcal{M}}, \mathcal{S}, \mathfrak{p}_{\mathcal{M}}, \mathfrak{r}_{\mathcal{M}}, \beta)$. The meta policy will be

Algorithm 1 Meta Policy Iteration

```
Input: MDP $\mathcal{M} = (\mathcal{S}, \mathcal{A}, P, r, \gamma)$, initial meta policy $\mu$, initial policy $\pi$
  Build meta MDP $\mathfrak{M}_{\mathcal{M}} = (\Pi_{\mathcal{M}}, \mathcal{S}, \mathfrak{p}_{\mathcal{M}}, \mathfrak{r}_{\mathcal{M}}, \beta)$
  while not converged do
    $s \leftarrow \arg\max_{\bar{s} \in \mathcal{S}} \mu(\pi, \bar{s})$    ▷ Select state
    $V \leftarrow (I - \gamma P^{\pi})^{-1} r$    ▷ Evaluate policy
    $\pi(s) \leftarrow \arg\max_{a \in \mathcal{A}} r(s, a) + \gamma \sum_{s' \in \mathcal{S}} P(s'|s, a) V(s')$    ▷ Improve policy on selected state
    $\mathcal{V} \leftarrow (I - \gamma \mathfrak{p}^{\mu}_{\mathcal{M}})^{-1} \mathfrak{r}_{\mathcal{M}}$    ▷ Evaluate meta policy
    $\mu(\pi, s) \leftarrow \arg\max_{\bar{s} \in \mathcal{S}} \mathfrak{r}_{\mathcal{M}}(\pi, \bar{s}) + \beta \sum_{\pi' \in \Pi_{\mathcal{M}}} \mathfrak{p}_{\mathcal{M}}(\pi'|\pi, \bar{s}) \mathcal{V}(\pi')$ ▷ Improve meta policy on current $\pi$
  end while
```

a matrix $\mu \in \mathbb{R}^{|\Pi_{\mathcal{M}}| \times |S|}$ and, being $\mathfrak{M}$ formally an MDP, it can be naturally evaluated by defining a value function $\mathcal{V}^{\mu} \in \mathbb{R}^{|\Pi_{\mathcal{M}}|}$ as $\mathcal{V}^{\mu}(\pi) = \mathbb{E}_{\mathfrak{p}_{\mathcal{M}}, \mu}[\sum_{k=0}^{\infty} \beta^k \mathfrak{r}_{\mathcal{M}}(\bar{\pi}_k, S_k) | \bar{\pi}_0 = \pi]$. $\mathcal{V}^{\mu}$ provides the cumulative learning progress expected starting from a particular learning state, as represented by π, given a future optimization process determined by the combination of the policy iteration procedure (i.e., the meta dynamics $\mathfrak{p}_{\mathcal{M}}$) and the state sampling strategy (i.e., the meta policy μ).

The optimal meta policy μ^* can be found by using any dynamic programming algorithm, for instance policy iteration itself. We call *Meta Policy Iteration* (MPI) the algorithm resulting from the combination of learning in the meta MDP $\mathfrak{M}_{\mathcal{M}}$ and selecting samples according to the meta policy μ to learn a policy π in the underlying MDP $\mathcal{M}$. A pseudocode for MPI can be found in Algorithm 1.

Which kind of properties should we expect from the policies found by MPI? To see it more clearly, let us think about a variation of the algorithm, in which the meta MDP is constructed as a finite-horizon MDP $\mathfrak{M}_{\mathcal{M},K} = (\Pi_{\mathcal{M}}, \mathcal{S}, \mathfrak{p}_{\mathcal{M}}, \mathfrak{r}_{\mathcal{M}}, K)$. In practice, this setting implies assuming that, instead of being theoretically infinite in duration, the optimization process has a finite duration K. In other words, we are interested in the best policy π that can be found in K steps only. In this context, the policy obtained by optimizing according the optimal meta policy μ^* can be characterized by the following proposition.

Proposition 3.1. *Let $\Pi^K_{\mathcal{M},\pi_0}$ be the space of all the policies which can be obtained by performing K steps of asynchronous policy iteration in the MDP $\mathcal{M}$ from an initial policy π_0, μ^* be the optimal policy in the meta MDP $\mathfrak{M}_{\mathcal{M},K}$, π_K be the policy obtained after K steps of asynchronous policy iteration with states sampled according to meta policy μ^*. It holds that:*

$$\pi_K \in \arg\max_{\pi \in \Pi^K_{\mathcal{M},\pi_0}} \mathbb{E}_{s \sim u}[V^{\pi}(s)]$$

Proposition 3.1 states that, if only a finite budget of policy iteration updates is available, then improving the policy by using a the strategy implied by the optimal meta policy μ^* will lead to the best possible policy affordable in that context. Intuitively, the same idea smoothly generalizes to the discounted setting, in which the discount β allows for a tradeoff between the quality of the resulting policy and the number of iterations required to reach it.

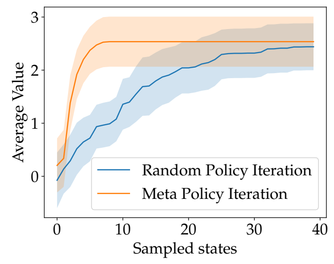
3.2 Approximations for Meta Value Iteration

We described how the meta MDP can be designed for the asynchronous policy iteration algorithm. By leveraging the same construction, we can derive a meta MDP for value iteration as well. While the meta action space is clearly still the set of states that can be provided to one step of asynchronous value iteration (i.e., the original state space $\mathcal{S}$ of $\mathcal{M}$), the most natural meta state space, which captures the learning state, is in this case the space of all possible value functions $\mathcal{W}_{\mathcal{M}}$ in the MDP $\mathcal{M}$. Recent work studied the geometric properties of this space [3], but we will not focus our discussion on those. The same reasoning applied to meta policy iteration leads us to the following meta dynamics and reward functions:

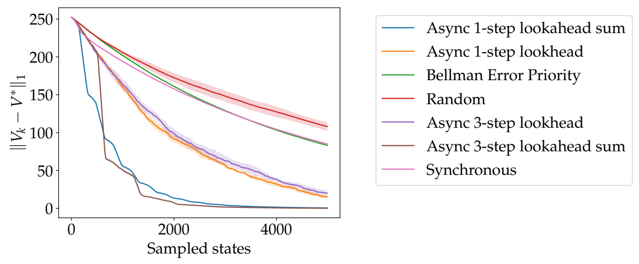
$$\bar{\mathfrak{p}}_{\mathcal{M}}(V'|V, s) \triangleq \mathbb{1}\left[V'(\bar{s}) = \left\{ \begin{array}{ll} \max_a \{r(\bar{s}, a) + \gamma \sum_{s'} P(s'|\bar{s}, a) V(s')\}, & \text{if } \bar{s} = s \\ V(\bar{s}), & \text{if } \bar{s} \neq s \end{array} \right\}\right], \quad (3)$$

$$\bar{\mathfrak{r}}_{\mathcal{M}}(V, s) \triangleq \mathbb{E}_{V' \sim \bar{\mathfrak{p}}_{\mathcal{M}}(V'|V,s)}[\bar{\mathfrak{r}}_{\mathcal{M}}(V, s, V')] \triangleq \mathbb{E}_{V' \sim \bar{\mathfrak{p}}_{\mathcal{M}}(V'|V,s)}[\mathbb{E}_{s' \sim u}[V'(s') - V(s')]]. \quad (4)$$

The meta MDP for value iteration can be thus defined as $\bar{\mathfrak{M}} = (\mathcal{W}_{\mathcal{M}}, \mathcal{S}, \bar{\mathfrak{p}}_{\mathcal{M}}, \bar{\mathfrak{r}}_{\mathcal{M}}, \beta)$. There is, however, a computational difference w.r.t. the construction from MPI: while the space of deterministic



(a) Meta Policy Iteration



(b) Approximate Meta Value Iteration

Figure 1: (1a): Results for meta policy iteration in randomly generated MDPs (20 runs, 95% C.I.). (1b): Results for approximate meta value iteration in the forest management MDP (5 runs, 95 % C.I.).

policies $\Pi_{\mathcal{M}}$ is large but finite, the space of value functions $\mathcal{W}_{\mathcal{M}}$ is always infinite. Thus, even if the underlying MDP $\mathcal{M}$ is finite, approximate dynamic programming methods should be used for solving the corresponding value iteration meta MDP.

A rough approximation to constructing the meta MDP altogether is to pick the state s which maximizes the expected value improvement (either terminal or cumulative) after K steps of synchronous value iteration. The state to optimize the value function is hence selected by preferring the ones which, after the asynchronous update on it, yield the best value function in the future. This is related to running a uniformly random policy in the meta MDP, and effectively accounts for the uncertainty of the part of the optimization process which is not unrolled. Note that for $K = 0$, if only the immediate improvement is considered, this is equivalent to using the *Bellman error* as a priority, a classic approach in both asynchronous dynamic programming [13] and deep reinforcement learning [10].

4 Empirical Validation

We test the meta policy iteration algorithm in randomly generated discrete MDPs using the toolbox in [1], with $|\mathcal{S}| = 11$, $|\mathcal{A}| = 2$ and $\gamma = 0.97$. We compare it to asynchronous policy iteration with randomly sampled states and obtain the results reported in Figure 1a. They show that, on average, MPI is able to reach the optimal policy in just a few steps, verifying in practice the theoretical advantage outlined in Proposition 3.1.

To assess the performance of the approximate meta value iteration algorithm, we leverage a more complex MDP with $|\mathcal{S}| = 192$, $|\mathcal{A}| = 64$ and $\gamma = 0.97$. The MDP is a realistic formalization of a forest management problem discussed in [14], in which the agent faces a tradeoff between the profits that it can make by selling the timber from a forest and the ecological damage that can result from deforestation. The comparison in Figure 1b includes *k-step lookahead* algorithms, which take the state s leading to the largest learning progress at the k-th step, and *k-step lookahead sum* algorithms, which consider the sum of the learning progresses up to the iteration k as a priority. To guarantee convergence, we use an ϵ-greedy meta policy with $\epsilon = 0.1$ in both the approaches. The results show that, while the Bellman Error alone is only able to match the performance of synchronous value iteration, using the meta MDP, albeit with a rough approximation, allows to converge much faster.

5 Conclusions

In this work, we presented a meta MDP construction, as well as a class of algorithms, to accelerate asynchronous dynamic programming. We showed, both in theory and in practice, that, by just modifying the sequence of states provided to the optimization algorithms, the dynamics of the learning process can get close to optimal under a given data budget. Despite the relative simplicity of the method and the clear scalability limits of the version we presented, we believe the conceptual tools we developed for asynchronous dynamic programming can inspire and motivate similar metacognitive strategies for reinforcement learning [12], in the context of exploration and memory replay.

References

[1] I. Chades, Guillaume Chapron, Marie-Josée Cros, Frédérick Garcia, and Régis Sabbadin. Mdptoolbox: a multi-platform toolbox to solve stochastic dynamic programming problems. *Ecography*, 37:916–920, 2014.

[2] Michael T Cox. Metacognition in computation: A selected research review. *Artificial intelligence*, 169(2):104–141, 2005.

[3] Robert Dadashi, Adrien Ali Taiga, Nicolas Le Roux, Dale Schuurmans, and Marc G Bellemare. The value function polytope in reinforcement learning. In *International Conference on Machine Learning*, pages 1486–1495. PMLR, 2019.

[4] Alex Graves, Marc G Bellemare, Jacob Menick, Remi Munos, and Koray Kavukcuoglu. Automated curriculum learning for neural networks. In *international conference on machine learning*, pages 1311–1320. PMLR, 2017.

[5] Thomas L Griffiths, Frederick Callaway, Michael B Chang, Erin Grant, Paul M Krueger, and Falk Lieder. Doing more with less: meta-reasoning and meta-learning in humans and machines. *Current Opinion in Behavioral Sciences*, 29:24–30, 2019.

[6] Lu Jiang, Deyu Meng, Qian Zhao, Shiguang Shan, and Alexander G Hauptmann. Self-paced curriculum learning. In *Twenty-Ninth AAAI Conference on Artificial Intelligence*, 2015.

[7] Manuel Lopes, Tobias Lang, Marc Toussaint, and Pierre-Yves Oudeyer. Exploration in model-based reinforcement learning by empirically estimating learning progress. In *Neural Information Processing Systems (NIPS)*, 2012.

[8] Martin L. Puterman. Markov decision processes: Discrete stochastic dynamic programming. In *Wiley Series in Probability and Statistics*, 1994.

[9] Tom Schaul, Diana Borsa, David Ding, David Szepesvari, Georg Ostrovski, Will Dabney, and Simon Osindero. Adapting behaviour for learning progress. *arXiv preprint arXiv:1912.06910*, 2019.

[10] Tom Schaul, John Quan, Ioannis Antonoglou, and David Silver. Prioritized experience replay. In *ICLR*, 2016.

[11] Jürgen Schmidhuber. Curious model-building control systems. In *Proc. international joint conference on neural networks*, pages 1458–1463, 1991.

[12] Richard S Sutton and Andrew G Barto. *Reinforcement learning: An introduction*. MIT press, 2018.

[13] David Wingate, Kevin D Seppi, and Sridhar Mahadevan. Prioritization methods for accelerating mdp solvers. *Journal of Machine Learning Research*, 6(5), 2005.

[14] Mo Zhou and Joseph Buongiorno. Forest landscape management in a stochastic environment, with an application to mixed loblolly pine–hardwood forests. *Forest Ecology and Management*, 223(1-3):170–182, 2006.

A Appendix

We report here the proof of Proposition 3.1.

Proposition 3.1. *Let $\Pi^K_{\mathcal{M},\pi_0}$ be the space of all the policies which can be obtained by performing K steps of asynchronous policy iteration in the MDP $\mathcal{M}$ from an initial policy π_0, μ^* be the optimal policy in the meta MDP $\mathfrak{M}_{\mathcal{M},K}$, π_K be the policy obtained after K steps of asynchronous policy iteration with states sampled according to meta policy μ^*. It holds that:*

$$\pi_K \in \operatorname*{arg\,max}_{\pi \in \Pi^K_{\mathcal{M},\pi_0}} \mathbb{E}_{s\sim u}[V^\pi(s)]$$

Proof. Let us write the meta value function $\mathcal{V}^{\mu,0}$ for the first step of a finite-horizon meta MDP:

$$\begin{aligned}
\mathcal{V}^{\mu,0}(\pi_0) &= \mathbb{E}_{\mathfrak{p}_\mathcal{M},\mu}\left[\sum_{k=0}^{K-1} \mathfrak{r}_\mathcal{M}(\pi_k, s_k)\right] = \mathbb{E}_{\mathfrak{p}_\mathcal{M},\mu}\left[\sum_{k=0}^{K-1} \mathbb{E}_{s\sim u}\left[V^{\pi_{k+1}}(s) - V^{\pi_k}(s)\right]\right] && (5)\\
&= \mathbb{E}_{\mathfrak{p}_\mathcal{M},\mu}\left[\mathbb{E}_{s\sim u}\left[\sum_{k=0}^{K-1}\left(V^{\pi_{k+1}}(s) - V^{\pi_k}(s)\right)\right]\right] && (6)\\
&= \mathbb{E}_{\mathfrak{p}_\mathcal{M},\mu}\left[\mathbb{E}_{s\sim u}\left[V^{\pi_K}(s) - V^{\pi_0}(s)\right]\right] && (7)\\
&= \mathbb{E}_{\mathfrak{p}_\mathcal{M},\mu}\left[\mathbb{E}_{s\sim u}\left[V^{\pi_K}(s)\right]\right] - \mathbb{E}_{s\sim u}\left[V^{\pi_0}(s)\right] && (8)
\end{aligned}$$

Being μ^* the optimal meta policy, it holds by definition of optimality that:

$$\begin{aligned}
\mathcal{V}^{\mu^*,0}(\pi_0) &= \max_\mu \left\{\mathbb{E}_{\mathfrak{p}_\mathcal{M},\mu}\left[\mathbb{E}_{s\sim u}\left[V^{\pi_K}(s)\right]\right] - \mathbb{E}_{s\sim u}\left[V^{\pi_0}(s)\right]\right\} && (9)\\
&= \max_\mu \left\{\mathbb{E}_{\mathfrak{p}_\mathcal{M},\mu}\left[\mathbb{E}_{s\sim u}\left[V^{\pi_K}(s)\right]\right]\right\} - \mathbb{E}_{s\sim u}\left[V^{\pi_0}(s)\right] && (10)\\
&= \max_{\pi \in \Pi^K_{\mathcal{M},\pi_0}} \mathbb{E}_{s\sim u}[V^\pi(s)] - \mathbb{E}_{s\sim u}\left[V^{\pi_0}(s)\right] && (11)
\end{aligned}$$

In the perspective of the meta MDP, π_K is simply the last meta state that is reached by the policy. Under the optimal meta policy, this last meta state will be therefore the one which maximizes the function above, which only depends on the last state:

$$\pi_K \in \operatorname*{arg\,max}_{\pi \in \Pi^K_{\mathcal{M},\pi_0}} \mathbb{E}_{s\sim u}[V^\pi(s)] - \mathbb{E}_{s\sim u}\left[V^{\pi_0}(s)\right] = \operatorname*{arg\,max}_{\pi \in \Pi^K_{\mathcal{M},\pi_0}} \mathbb{E}_{s\sim u}[V^\pi(s)]. \quad (12)$$

□