A Laplacian Framework for Option Discovery in Reinforcement Learning

Marlos C. Machado [1] **Marc G. Bellemare** [2] **Michael Bowling** [1]

Abstract

Representation learning and option discovery are two of the biggest challenges in reinforcement learning (RL). Proto-value functions (PVFs) are a well-known approach for representation learning in MDPs. In this paper we address the option discovery problem by showing how PVFs implicitly define options. We do it by introducing *eigenpurposes*, intrinsic reward functions derived from the learned representations. The options discovered from eigenpurposes traverse the principal directions of the state space. They are useful for multiple tasks because they are discovered without taking the environment's rewards into consideration. Moreover, different options act at different time scales, making them helpful for exploration. We demonstrate features of eigenpurposes in traditional tabular domains as well as in Atari 2600 games.

1. Introduction

Two important challenges in reinforcement learning (RL) are the problems of representation learning and of automatic discovery of skills. Proto-value functions (PVFs) are a well-known solution for the problem of representation learning (Mahadevan, 2005; Mahadevan & Maggioni, 2007); while the problem of skill discovery is generally posed under the options framework (Sutton et al., 1999; Precup, 2000), which models skills as options.

In this paper, we tie together representation learning and option discovery by showing how PVFs implicitly define options. One of our main contributions is to introduce the concepts of *eigenpurpose* and *eigenbehavior*. Eigenpurposes are intrinsic reward functions that incentivize the agent to traverse the state space by following the principal directions of the learned representation. Each intrinsic reward function leads to a different *eigenbehavior*, which is the optimal policy for that reward function. In this paper we introduce an algorithm for option discovery that leverages these ideas. The options we discover are task-independent because, as PVFs, the eigenpurposes are obtained without any information about the environment's reward structure. We first present these ideas in the tabular case and then show how they can be generalized to the function approximation case.

Exploration, while traditionally a separate problem from option discovery, can also be addressed through the careful construction of options (McGovern & Barto, 2001; Şimşek et al., 2005; Solway et al., 2014; Kulkarni et al., 2016). In this paper, we provide evidence that not all options capable of accelerating planning are useful for exploration. We show that options traditionally used in the literature to speed up planning hinder the agents' performance if used for random exploration during learning. Our options have two important properties that allow them to improve exploration: (i) they operate at different time scales, and (ii) they can be easily sequenced. Having options that operate at different time scales allows agents to make finely timed actions while also decreasing the likelihood the agent will explore only a small portion of the state space. Moreover, because our options are defined across the whole state space, multiple options are available in every state, which allows them to be easily sequenced.

2. Background

We generally indicate random variables by capital letters (*e.g.*, R_t), vectors by bold letters (*e.g.*, $\boldsymbol{\theta}$), functions by lowercase letters (*e.g.*, v), and sets by calligraphic font (*e.g.*, $\mathcal{S}$).

2.1. Reinforcement Learning

In the RL framework (Sutton & Barto, 1998), an agent aims to maximize cumulative reward by taking actions in an environment. These actions affect the agent's next state and the rewards it experiences. We use the MDP formalism throughout this paper. An MDP is a 5-tuple $\langle \mathcal{S}, \mathcal{A}, r, p, \gamma \rangle$. At time t the agent is in state $s_t \in \mathcal{S}$ where it takes action $a_t \in \mathcal{A}$ that leads to the next state $s_{t+1} \in \mathcal{S}$ according to the transition probability kernel $p(s'|s,a)$, which encodes $\Pr(S_{t+1} = s' | S_t = s, A_t = a)$. The agent also observes a reward $R_{t+1} \sim r(s,a)$. The agent's goal is to learn a policy $\mu : \mathcal{S} \times \mathcal{A} \to [0,1]$ that maximizes the expected

[1]University of Alberta [2]Google DeepMind. Correspondence to: Marlos C. Machado <machado@ualberta.ca>.

Appearing in the *Proceedings of the 34^{th} International Conference on Machine Learning*, Sydney, Australia, PMLR 70, 2017.

discounted return $G_t \doteq \mathbb{E}_{p,\mu}\big[\sum_{k=0}^{\infty} \gamma^k R_{t+k+1} | s_t\big]$, where $\gamma \in [0, 1)$ is the discount factor.

It is common to use the policy improvement theorem (Bellman, 1957) when learning to maximize G_t. One technique is to alternate between solving the Bellman equations for the *action-value function* $q_{\mu_k}(s, a)$,

$$q_{\mu_k}(s,a) \doteq \mathbb{E}_{\mu_k,p}\big[G_t | S_t = s, A_t = a\big] = \sum_{s',r} p(s', r|s, a)\big[r + \gamma \sum_{a'} \mu_k(a'|s') q_{\mu_k}(s', a')\big]$$

and making the next policy, μ_{k+1}, greedy w.r.t. q_{μ_k},

$$\mu_{k+1} \doteq \arg\max_{a \in \mathcal{A}} q_{\mu_k}(s, a),$$

until converging to an optimal policy μ_*.

Sometimes it is not feasible to learn a value for each state-action pair due to the size of the state space. Generally, this is addressed by parameterizing $q_\mu(s, a)$ with a set of weights $\boldsymbol{\theta} \in \mathbb{R}^n$ such that $q_\mu(s, a) \approx q_\mu(s, a, \boldsymbol{\theta})$. It is common to approximate q_μ through a linear function, *i.e.*, $q_\mu(s, a, \boldsymbol{\theta}) = \boldsymbol{\theta}^\top \boldsymbol{\phi}(s, a)$, where $\boldsymbol{\phi}(s, a)$ denotes a linear feature representation of state s when taking action a.

2.2. The Options Framework

The options framework extends RL by introducing temporally extended actions called *skills* or *options*. An option ω is a 3-tuple $\omega = \langle \mathcal{I}, \pi, \mathcal{T} \rangle$ where $\mathcal{I} \in \mathcal{S}$ denotes the option's initiation set, $\pi : \mathcal{A} \times \mathcal{S} \to [0, 1]$ denotes the option's policy, and $\mathcal{T} \in \mathcal{S}$ denotes the option's termination set. After the agent decides to follow option ω from a state in $\mathcal{I}$, actions are selected according to π until the agent reaches a state in $\mathcal{T}$. Intuitively, options are higher-level actions that extend over several time steps, generalizing MDPs to semi-Markov decision processes (SMDPs) (Puterman, 1994).

Traditionally, options capable of moving agents to *bottleneck* states are sought after. Bottleneck states are those states that connect different densely connected regions of the state space (*e.g.*, doorways) (Şimşek & Barto, 2004; Solway et al., 2014). They have been shown to be very efficient for planning as these states are the states most frequently visited when considering the *shortest* distance between any two states in an MDP (Solway et al., 2014).

2.3. Proto-Value Functions

Proto-value functions (PVFs) are learned representations that capture large-scale temporal properties of an environment (Mahadevan, 2005; Mahadevan & Maggioni, 2007). They are obtained by diagonalizing a diffusion model, which is constructed from the MDP's transition matrix. A diffusion model captures information flow on a graph, and it is commonly defined by the *combinatorial graph Laplacian* matrix $L = D - A$, where A is the graph's adjacency matrix and D the diagonal matrix whose entries are the row sums of A. Notice that the adjacency matrix A easily generalizes to a weight matrix W. PVFs are defined to be the eigenvectors obtained after the eigendecomposition of L. Different diffusion models can be used to generate PVFs, such as the *normalized graph Laplacian* $L = D^{-\frac{1}{2}}(D - A)D^{-\frac{1}{2}}$, which we use in this paper.

3. Option Discovery through the Laplacian

PVFs capture the large-scale geometry of the environment, such as symmetries and bottlenecks. They are task independent, in the sense that they do not use information related to reward functions. Moreover, they are defined over the whole state space since each eigenvector induces a real-valued mapping over each state. We can imagine that options with these properties should also be useful. In this section we show how to use PVFs to discover options.

Let us start with an example. Consider the traditional 4-room domain depicted in Figure 1c. Gray squares represent walls and white squares represent accessible states. Four actions are available: *up*, *down*, *right*, and *left*. The transitions are deterministic and the agent is not allowed to move into a wall. Ideally, we would like to discover options that move the agent from room to room. Thus, we should be able to automatically distinguish between the different rooms in the environment. This is exactly what PVFs do, as depicted in Figure 2 (left). Instead of interpreting a PVF as a basis function, we can interpret the PVF in our example as a desire to reach the highest point of the plot, corresponding to the centre of the room. Because the sign of an eigenvector is arbitrary, a PVF can also be interpreted as a desire to reach the lowest point of the plot, corresponding to the opposite room. In this paper we use the eigenvectors in both directions (*i.e.*, both signs).

An *eigenpurpose* formalizes the interpretation above by defining an intrinsic reward function. We can see it as defining a *purpose* for the agent, that is, to maximize the discounted sum of these rewards.

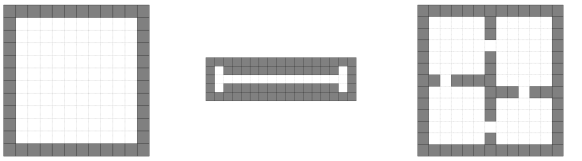
Definition 3.1 (Eigenpurpose)**.** An *eigenpurpose* is the intrinsic reward function $r_i^{\mathbf{e}}(s, s')$ of a proto-value function $\mathbf{e} \in \mathbb{R}^{|\mathcal{S}|}$ such that

$$r_i^{\mathbf{e}}(s, s') = \mathbf{e}^\top(\boldsymbol{\phi}(s') - \boldsymbol{\phi}(s)), \tag{1}$$

where $\boldsymbol{\phi}(x)$ denotes the feature representation of state x.

Notice that an eigenpurpose, in the tabular case, can be written as $r_i^{\mathbf{e}}(s, s') = \mathbf{e}[s'] - \mathbf{e}[s]$.

We can now define a new MDP to learn the option associated with the purpose, $\mathcal{M}_i^{\mathbf{e}} = \langle \mathcal{S}, \mathcal{A} \cup \{\perp\}, r_i^{\mathbf{e}}, p, \gamma \rangle$, where



(a) 10×10 grid (b) I-Maze (c) 4-room domain

Figure 1. Domains used for evaluation.

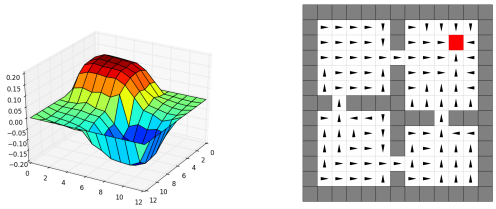


Figure 2. Second PVF (left) and its corresponding option (right) in the 4-room domain. Action *terminate* is depicted in red (top right corner), other actions are depicted as arrows.

the reward function is defined as in (1) and the action set is augmented by the action *terminate* ($\perp$), which allows the agent to leave $\mathcal{M}_i^{\mathbf{e}}$ without any cost. The state space and the transition probability kernel remain unchanged from the original problem. The discount rate can be chosen arbitrarily, although it impacts the timescale the option encodes.

With $\mathcal{M}_i^{\mathbf{e}}$ we define a new state-value function $v_\pi^{\mathbf{e}}(s)$, for policy π, as the expected value of the cumulative discounted intrinsic reward if the agent starts in state s and follows policy π until termination. Similarly, we define a new action-value function $q_\pi^{\mathbf{e}}(s,a)$ as the expected value of the cumulative discounted intrinsic reward if the agent starts in state s, takes action a, and then follows policy π until termination. We can also describe the optimal value function for any eigenpurpose obtained through $\mathbf{e}$:

$$v_*^{\mathbf{e}}(s) = \max_\pi v_\pi^{\mathbf{e}}(s) \quad \text{and} \quad q_*^{\mathbf{e}}(s,a) = \max_\pi q_\pi^{\mathbf{e}}(s,a).$$

These definitions naturally lead us to *eigenbehaviors*.

Definition 3.2 (Eigenbehavior)**.** An *eigenbehavior* is a policy $\chi^{\mathbf{e}} : \mathcal{S} \to \mathcal{A}$ that is optimal with respect to the eigenpurpose $r_i^{\mathbf{e}}$, *i.e.*, $\chi^{\mathbf{e}}(s) = \arg\max_{a\in\mathcal{A}} q_*^{\mathbf{e}}(s,a)$.

Finding the optimal policy $\pi_*^{\mathbf{e}}$ now becomes a traditional RL problem, with a different reward function. Importantly, this reward function tends to be dense, avoiding challenging situations due to exploration issues. In this paper we use policy iteration to solve for an optimal policy.

If each eigenpurpose defines an option, its corresponding eigenbehavior is the option's policy. Thus, we need to define the option's initiation and termination set. An option should be available in every state where it is possible to achieve its purpose, and to terminate when it is achieved.

When defining the MDP to learn the option, we augmented the agent's action set with the *terminate* action, allowing the agent to interrupt the option anytime. We want options to terminate when the agent achieves its purpose, *i.e.*, when it is unable to accumulate further positive intrinsic rewards. With the defined reward function, this happens when the agent reaches the state with largest value in the eigenpurpose (or a local maximum when $\gamma < 1$). Any subsequent reward will be negative. We are able to formalize this condition by defining $q_\chi(s,\perp) \doteq 0$ for all $\chi^{\mathbf{e}}$. When the terminate action is selected, control is returned to the higher level policy (Dietterich, 2000). An option following a policy $\chi^{\mathbf{e}}$ terminates when $q_\chi^{\mathbf{e}}(s,a) \leq 0$ for all $a \in \mathcal{A}$. We define the initiation set to be all states in which there exists an action $a \in \mathcal{A}$ such that $q_\chi^{\mathbf{e}}(s,a) > 0$. Thus, the option's policy is $\pi^{\mathbf{e}}(s) = \arg\max_{a\in\mathcal{A}\cup\{\perp\}} q_\pi^{\mathbf{e}}(s,a)$. We refer to the options discovered with our approach as *eigenoptions*. The eigenoption corresponding to the example at the beginning of this section is depicted in Figure 2 (right).

For any eigenoption, there is always at least one state in which it terminates, as we now show.

Theorem 3.1 (Option's Termination)**.** *Consider an eigenoption $o = \langle \mathcal{I}_o, \pi_o, \mathcal{T}_o \rangle$ and $\gamma < 1$. Then, in an MDP with finite state space, $\mathcal{T}_o$ is nonempty.*

Proof. We can write the Bellman equation in the matrix form: $\mathbf{v} = \mathbf{r} + \gamma T\mathbf{v}$, where $\mathbf{v}$ is a *finite* column vector with one entry per state encoding its value function. From (1) we have $\mathbf{r} = T\mathbf{w} - \mathbf{w}$ with $\mathbf{w} = \boldsymbol{\phi}(s)^\top \mathbf{e}$, where $\mathbf{e}$ denotes the eigenpurpose of interest. Therefore:

$$\begin{aligned}\mathbf{v} + \mathbf{w} &= T\mathbf{w} + \gamma T\mathbf{v} \\ &= (1-\gamma)T\mathbf{w} + \gamma T(\mathbf{v}+\mathbf{w}) \\ &= (1-\gamma)(I-\gamma T)^{-1}T\mathbf{w}.\end{aligned}$$

$$\begin{aligned}||\mathbf{v}+\mathbf{w}||_\infty &= (1-\gamma)||(I-\gamma T)^{-1}T\mathbf{w}||_\infty \\ ||\mathbf{v}+\mathbf{w}||_\infty &\leq (1-\gamma)||(I-\gamma T)^{-1}T||_\infty ||\mathbf{w}||_\infty \\ ||\mathbf{v}+\mathbf{w}||_\infty &\leq (1-\gamma)\frac{1}{(1-\gamma)}||\mathbf{w}||_\infty \\ ||\mathbf{v}+\mathbf{w}||_\infty &\leq ||\mathbf{w}||_\infty\end{aligned}$$

We can shift $\mathbf{w}$ by any finite constant without changing the reward, *i.e.*, $T\mathbf{w}-\mathbf{w} = T(\mathbf{w}+\boldsymbol{\delta})-(\mathbf{w}+\boldsymbol{\delta})$ because $T\mathbf{1}\boldsymbol{\delta} = \mathbf{1}\boldsymbol{\delta}$ since $\sum_j T_{i,j} = 1$. Hence, we can assume $\mathbf{w} \geq \mathbf{0}$. Let $s^* = \arg\max_s \mathbf{w}_{s^*}$, so that $\mathbf{w}_{s^*} = ||\mathbf{w}||_\infty$. Clearly $\mathbf{v}_{s^*} \leq \mathbf{0}$, otherwise $||\mathbf{v}+\mathbf{w}||_\infty \geq |\mathbf{v}_{s^*} + \mathbf{w}_{s^*}| = \mathbf{v}_{s^*} + \mathbf{w}_{s^*} > \mathbf{w}_{s^*} = ||\mathbf{w}||_\infty$, arriving at a contradiction. □

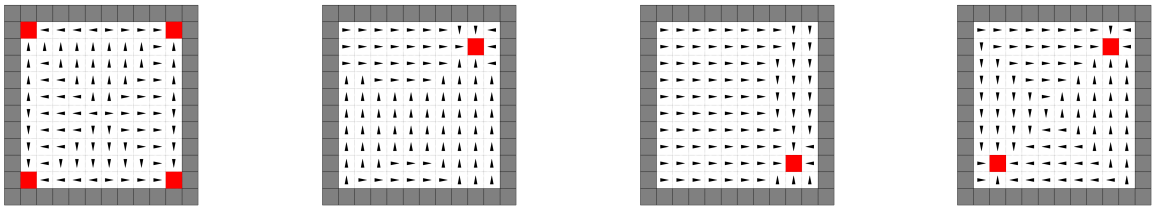


Figure 3. Options obtained from the four smallest eigenvectors in the 10×10 grid. Action *terminate* is depicted in red.

Figure 4. Options obtained from the four smallest eigenvectors in the I-Maze domain. Action *terminate* is depicted in red.

This result is applicable in both the tabular and linear function approximation case. An algorithm that does not rely on knowing the underlying graph is provided in Section 5.

4. Empirical Evaluation

We used three MDPs in our empirical study (*c.f.* Figure 1): an open room, an I-Maze, and the 4-room domain. Their transitions are deterministic and gray squares denote walls. Agents have access to four actions: *up*, *down*, *right*, and *left*. When an action that would have taken the agent into a wall is chosen, the agent's state does not change. We demonstrate three aspects of our framework:[1]

- How the eigenoptions present specific purposes. Interestingly, options leading to bottlenecks are not the first ones we discover.
- How eigenoptions improve exploration by reducing the expected number of steps required to navigate between any two states.
- How eigenoptions help agents to accumulate reward faster. We show how few options may hurt the agents' performance while enough options speed up learning.

4.1. Discovered Options

In the PVF theory, the "smoothest" eigenvectors, corresponding to the smallest eigenvalues, are preferred (Mahadevan & Maggioni, 2007). The same intuition applies to eigenoptions, with the eigenpurposes corresponding to the smallest eigenvalues being preferred. Figures 3, 4, and 5 depict the first eigenoptions discovered in the three domains used for evaluation.

Eigenoptions do not necessarily look for bottleneck states, allowing us to apply our algorithm in many environments in which there are no obvious, or meaningful, bottlenecks. We discover meaningful options in these environments, such as walking down a corridor, or going to the corners of an open room. Interestingly, doorways are not the first options we discover in the 4-room domain (the fifth eigenoption is the first to terminate at the entrance of a doorway). In the next sections we provide empirical evidence that eigenoptions are useful, and often more so than bottleneck options.

4.2. Exploration

A major challenge for agents to explore an environment is to be decisive, avoiding the dithering commonly observed in random walks (Machado & Bowling, 2016; Osband et al., 2016). Options provide such decisiveness by operating in a higher level of abstraction. Agents performing a random walk, when equipped with options, are expected to cover larger distances in the state space, navigating back and forth between subgoals instead of dithering around the starting state. However, options need to satisfy two conditions to improve exploration: (1) they have to be available in several parts of the state space, ensuring the agent always has access to many different options; and (2) they have to operate at different time scales. For instance, in the 4-room domain, it is unlikely an agent randomly selects enough primitive actions leading it to a corner if all options move the agent between doorways. An important result in this section is to show that it is very unlikely for an agent to explore the whole environment if it keeps going back and forth between similar high-level goals.

Eigenoptions satisfy both conditions. As demonstrated in Section 4.1, eigenoptions are often defined in the whole state space, allowing sequencing. Moreover, PVFs can be seen as a "frequency" basis, with different PVFs being associated with different frequencies (Mahadevan & Maggioni, 2007). The corresponding eigenoptions also operate

[1] *Python* code can be found at:
https://github.com/mcmachado/options

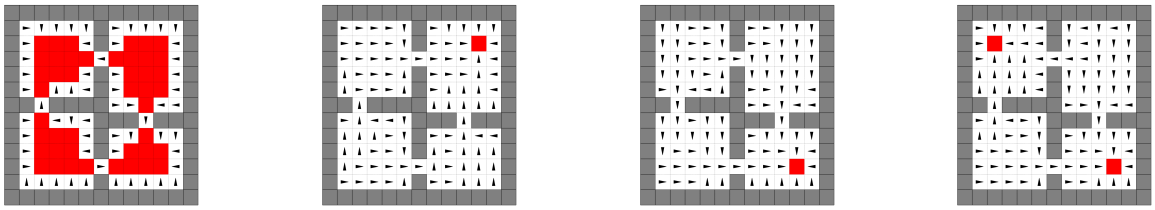
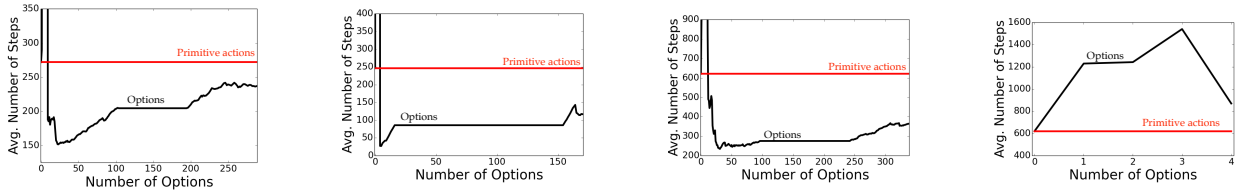


Figure 5. Options obtained from the four smallest eigenvectors in the 4-room domain. Action *terminate* is depicted in red.



(a) 10×10 grid (b) I-Maze (c) 4-room domain (d) Bottleneck options

Figure 6. Expected number of steps between any two states when following a random walk. Figure 6d shows the performance of options that look for doorways in the 4-room domain.

at different frequencies, with the length of a trajectory until termination varying. This behavior can be seen when comparing the second and fourth eigenoptions in the 10×10 grid (Figure 3). The fourth eigenoption terminates, on expectation, twice as often as the second eigenoption.

In this section we show that eigenoptions improve exploration. We do so by introducing a new metric, which we call *diffusion time*. Diffusion time encodes the expected number of steps required to navigate between two states randomly chosen in the MDP while following a random walk. A small expected number of steps implies that it is more likely that the agent will reach all states with a random walk. We discuss how this metric can be computed in the Appendix.

Figure 6 depicts, for our the three environments, the diffusion time with options and the diffusion time using only primitive actions. We add options incrementally in order of increasing eigenvalue when computing the diffusion time for different sets of options.

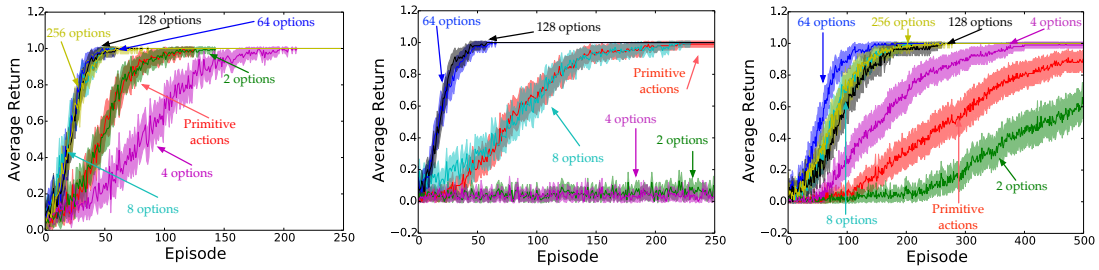
The first options added hurt exploration, but when enough options are added, exploration is greatly improved when compared to a random walk using only primitive actions. The fact that few options hurt exploration may be surprising at first, based on the fact that few useful options are generally sought after in the literature. However, this is a major difference between using options for planning and for learning. In planning, options shortcut the agents' trajectories, pruning the search space. All other actions are still taken into consideration. When exploring, a uniformly random policy over options and primitive actions skews where agents spend their time. Options that are much longer than primitive actions reduce the likelihood that an agent will deviate much from the options' trajectories, since sampling an option may undo dozens of primitive actions. This biasing is often observed when fewer options are available.

The discussion above can be made clearer with an example. In the 4-room domain, if the only options available are those leading the agent to doorways (*c.f.* Appendix), it is less likely the agent will reach the outer corners. To do so the agent would have to select enough consecutive primitive actions without sampling an option. Also, it is very likely agents will be always moving between rooms, never really exploring inside a room. These issues are mitigated with eigenoptions. The first eigenoptions lead agents to individual rooms, but other eigenoptions operate in different time scales, allowing agents to explore different parts of rooms.

Figure 6d supports the intuition that options leading to bottleneck states are not sufficient, by themselves, for exploration. It shows how the diffusion time in the 4-room domain is increased when only bottleneck options are used. As in the PVF literature, the ideal number of options to be used by an agent can be seen as a model selection problem.

4.3. Accumulating Rewards

We now illustrate the usefulness of our options when the agent's goal is to accumulate reward. We also study the impact of an increasing number of options in such a task. In these experiments, the agent starts at the bottom left cor-



(a) 10×10 grid (b) I-Maze (c) 4-room domain

Figure 7. The agents' performance accumulating reward as options are added to the action set in their behavior policy. These results use the eigenpurposes directly obtained from the eigendecomposition as well as their negation.

ner and its goal is to reach the top right corner. The agent observes a reward of 0 until the goal is reached, when it observes a reward of $+1$. We used Q-Learning (Watkins & Dayan, 1992) ($\alpha = 0.1$, $\gamma = 0.9$) to learn a policy over primitive actions. The behavior policy chooses uniformly over primitive actions and options, following them until termination. Figure 7 depicts, after learning for a given number of episodes, the average over 100 trials of the agents' final performance. Episodes were 100 time steps long, and we learned for 250 episodes in the 10×10 grid and in the I-Maze, and for 500 episodes in the 4-room domain.

In most scenarios eigenoptions improve performance. As in the previous section, exceptions occur when only a few options are added to the agent's action set. The best results were obtained using 64 options. Despite being an additional parameter, our results show that the agent's performance is fairly robust across different numbers of options.

Eigenoptions are task-independent by construction. Additional results in the appendix show how the same set of eigenoptions is able to speed-up learning in different tasks. In the appendix we also compare eigenoptions to random options, that is, options that use a random state as subgoal.

5. Approximate Option Discovery

So far we have assumed that agents have access to the adjacency matrix representing the underlying MDP. However, in practical settings this is generally not true. In fact, the number of states in these settings is often so large that agents rarely visit the same state twice. These problems are generally tackled with sample-based methods and some sort of function approximation.

In this section we propose a sample-based approach for option discovery that asymptotically discovers eigenoptions. We then extend this algorithm to linear function approximation. We provide anecdotal evidence in Atari 2600 games that this relatively naïve sample-based approach to function approximation discovers purposeful options.

5.1. Sample-based Option Discovery

In the online setting, agents must sample trajectories. Naturally, one can sample trajectories until one is able to perfectly construct the MDP's adjacency matrix, as suggested by Mahadevan & Maggioni (2007). However, this approach does not easily extend to linear function approximation. In this section we provide an approach that does not build the adjacency matrix allowing us to extend the concept of eigenpurposes to linear function approximation.

In our algorithm, a sample transition is added to a matrix T if it was not previously encountered. The transition is added as the difference between the current and previous observations, *i.e.*, $\phi(s') - \phi(s)$. In the tabular case we define $\phi(s)$ to be the one-hot encoding of state s. Once enough transitions have been sampled, we perform a singular value decomposition on the matrix T such that $T = U\Sigma V^\top$. We use the columns of V, which correspond to the right-eigenvectors of T, to generate the eigenpurposes. The intrinsic reward and the termination criterion for an eigenbehavior are the same as before.

Matrix T is known as the *incidence matrix*. If all transitions in the graph are sampled once, for tabular representations, this algorithm discovers the same options we obtain with the combinatorial Laplacian. The theorem below states the equivalence between the obtained eigenpurposes.

Theorem 5.1. *Consider the SVD of $T = U_T \Sigma_T V_T^\top$, with each row of T consisting of the difference between observations,* i.e., *$\phi(s') - \phi(s)$. In the tabular case, if all transitions in the MDP have been sampled once, the orthonormal eigenvectors of L are the columns of $V_T^\top$.*

Proof. Given the SVD decomposition of a matrix $A = U\Sigma V^\top$, the columns of V are the eigenvectors of $A^\top A$ (Strang, 2005). We know that $T^\top T = 2L$, where $L = D - W$ (Lemma 5.1, *c.f.* Appendix). Thus, the columns of V_T are the eigenvectors of $T^\top T$, which can be rewritten as $2(D - W)$. Therefore, the columns of V_T are also the eigenvectors of L. □

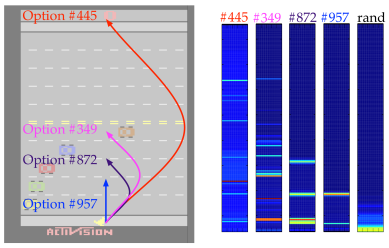


Figure 8. Options in FREEWAY (*c.f.* text for details).

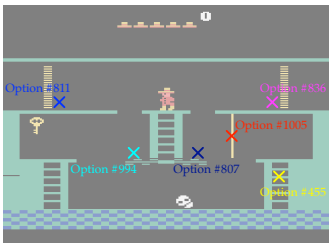


Figure 9. Options in MONTEZUMA'S REV. (*c.f.* text for details).

There is a trade-off between reconstructing the adjacency matrix and constructing the incidence matrix. In MDPs in which states are sparsely connected, such as the I-Maze, the latter is preferred since it has fewer transitions than states. However, what makes this result interesting is the fact that our algorithm can be easily generalized to linear function approximation.

5.2. Function Approximation

An adjacency matrix is not very useful when the agent has access only to features of the state. However, we can use the intuition about the incidence matrix to propose an algorithm compatible with linear function approximation.

In fact, to apply the algorithm proposed in the previous section, we just need to define what constitutes a new transition. We define two vectors, $\mathbf{t}$ and $\mathbf{t}'$, to be identical if and only if $\mathbf{t} - \mathbf{t}' = \mathbf{0}$. We then use a *set* data structure to avoid duplicates when storing $\boldsymbol{\phi}(s') - \boldsymbol{\phi}(s)$. This is a naïve approach, but it provides encouraging evidence eigenoptions generalize to linear function approximation. We expect more involved methods to perform even better.

We tested our method in the ALE (Bellemare et al., 2013). The agent's representation consists of the emulator's RAM state (1,024 bits). The final incidence matrix in which we ran the SVD had 25,000 rows, which we sampled uniformly from the set of observed transitions. We provide further details of the experimental setup in the appendix.

In the tabular case we start selecting eigenpurposes generated by the eigenvectors with smallest eigenvalue, because these are the "smoothest" ones. However, it is not clear such intuition holds here because we are in the function approximation setting and the matrix of transitions does not contain all possible transitions. Therefore, we analyzed, for each game, all 1,024 discovered options.

We approximate these options greedily ($\gamma = 0$) with the ALE emulator's look-ahead. The next action a' for an eigenpurpose $\mathbf{e}$ is selected as $\arg\max_{b \in \mathcal{A}} \int_{s'} p(s'|s,b)\, r_i^{\mathbf{e}}(s,s')$.

Even with such a myopic action selection mechanism we were able to obtain options that clearly demonstrate intent. In FREEWAY, a game in which a chicken is expected to cross the road while avoiding cars, we observe options in which the agent clearly wants to reach a specific lane in the street. Figure 8 (left) depicts where the chicken tends to be when the option is executed. On the right we see a histogram representing the chicken's height during an episode. We can clearly see how the chicken's height varies for different options, and how a random walk over primitive actions (*rand*) does not explore the environment properly. Remarkably, option *#445* scores 28 points at the end of the episode, without ever explicitly taking the reward signal into consideration. This performance is very close to those obtained by state-of-the-art algorithms.

In MONTEZUMA'S REVENGE, a game in which the agent needs to navigate through a room to pickup a key so it can open a door, we also observe the agent having the clear intent of reaching particular positions on the screen, such as staircases, ropes and doors (Figure 9). Interestingly, the options we discover are very similar to those handcrafted by Kulkarni et al. (2016) when evaluating the usefulness of options to tackle such a game. A video of the highlighted options can be found online.[2]

6. Related Work

Most algorithms for option discovery can be seen as *top-down* approaches. Agents use trajectories leading to informative rewards[3] as a starting point, decomposing and refining them into options. There are many approaches based on this principle, such as methods that use the observed rewards to generate intrinsic rewards leading to new value functions (*e.g.*, McGovern & Barto, 2001; Menache et al., 2002; Konidaris & Barto, 2009), methods that use the observed rewards to climb a gradient (*e.g.*, Mankowitz et al., 2016; Vezhnevets et al., 2016; Bacon et al., 2017), or to do

[2] https://youtu.be/2BVicx4CDWA

[3] We define an informative reward to be the signal that informs the agent it has reached a goal. For example, when trying to escape from a maze, we consider 0 to be an informative reward if the agent observes rewards of value -1 in every time step it is inside the maze. A different example is a positive reward observed by an agent that typically observes rewards of value 0.

probabilistic inference (Daniel et al., 2016). However, such approaches are not applicable in large state spaces with sparse rewards. If informative rewards are unlikely to be found by an agent using only primitive actions, requiring long or specific sequences of actions, options are equally unlikely to be discovered.

Our algorithm can be seen as a *bottom-up* approach, in which options are constructed before the agent observes any informative reward. These options are composed to generate the desired policy. Options discovered this way tend to be independent of an agent's intention, and are potentially useful in many different tasks (Gregor et al., 2016). Such options can also be seen as being useful for exploration by allowing agents to commit to a behavior for an extended period of time (Machado & Bowling, 2016). Among the approaches to discover options without using extrinsic rewards are the use of global or local graph centrality measures (Şimşek & Barto, 2004; Şimşek et al., 2005; Şimşek & Barto, 2008) and clustering of states (Mannor et al., 2004; Bacon, 2013; Lakshminarayanan et al., 2016). Interestingly, Şimşek et al. (2005) and Lakshminarayanan et al. (2016) also use the graph Laplacian in their algorithm, but to identify bottleneck states.

Baranes & Oudeyer (2013) and Moulin-Frier & Oudeyer (2013) show how one can build policies to explicitly assist agents to explore the environment. The proposed algorithms self-generate subgoals in order to maximize learning progress. The policies built can be seen as options. Recently, Solway et al. (2014) proved that "optimal hierarchy minimizes the geometric mean number of trial-and-error attempts necessary for the agent to discover the optimal policy for any selected task (...)". Our experiments confirm this result, although we propose *diffusion time* as a different metric to evaluate how options improve exploration.

The idea of discovering options by learning to control parts of the environment is also related to our work. Eigenpurposes encode different rates of change in the agents representation of the world, while the corresponding options aim at maximizing such change. Others have also proposed ways to discover options based on the idea of learning to control the environment. Hengst (2002), for instance, proposes an algorithm that explicitly models changes in the variables that form the agent's representation. Recently, Gregor et al. (2016) proposed an algorithm in which agents discover options by maximizing a notion of empowerment (Salge et al., 2014), where the agent aims at getting to states with a maximal set of available intrinsic options.

Continual Curiosity driven Skill Acquisition (CCSA) (Kompella et al., In Press) is the closest approach to ours. CCSA also discovers skills that maximize an intrinsic reward obtained by some extracted representation. While we use PVFs, CCSA uses Incremental Slow Feature Analysis (SFA) (Kompella et al., 2011) to define the intrinsic reward function. Sprekeler (2011) has shown that, given a specific choice of adjacency function, PVFs are equivalent to SFA (Wiskott & Sejnowski, 2002). SFA becomes an approximation of PVFs if the function space used in the SFA does not allow arbitrary mappings from the observed data to an embedding. Our method differs in how we define the initiation and termination sets, as well as in the objective being maximized. CCSA acquires skills that produce a large variation in the slow-feature outputs, leading to options that seek for bottlenecks. Our approach does not seek for bottlenecks, focusing on traversing different directions of the learned representation.

7. Conclusion

Being able to properly abstract MDPs into SMDPs can reduce the overall expense of learning (Sutton et al., 1999; Solway et al., 2014), mainly when the learned options are reused in multiple tasks. On the other hand, the wrong hierarchy can hinder the agents' learning process, moving the agent away from desired goal states. Current algorithms for option discovery often depend on an initial informative reward signal, which may not be readily available in large MDPs. In this paper, we introduced an approach that is effective in different environments, for a multitude of tasks.

Our algorithm uses the graph Laplacian, being directly related to the concept of proto-value functions. The learned representation informs the agent what are meaningful options to be sought after. The discovered options can be seen as traversing each one of the dimensions in the learned representation. We believe successful algorithms in the future will be able to simultaneously discover representations and options. Agents will use their learned representation to discover options, which will be used to further explore the environment, improving the agent's representation.

Interestingly, the options first discovered by our approach do not necessarily find bottlenecks, which are commonly sought after. In this paper we showed how bottleneck options can hinder exploration strategies if naively added to the agent's action set, and how the options we discover can help an agent to explore. Also, we have shown how the discovered options can be used to accumulate reward in a multitude of tasks, leveraging their exploratory properties.

There are several exciting avenues for future work. As noted, SFA can be seen as an approximation to PVFs. It would be interesting to compare such an approach to eigenoptions. It would also be interesting to see if the options we discover can be generated incrementally and with incomplete graphs. Finally, one can also imagine extensions to the proposed algorithm where a hierarchy of options is built.

Acknowledgements

The authors would like to thank Will Dabney, Rémi Munos and Csaba Szepesvári for useful discussions. This work was supported by grants from Alberta Innovates Technology Futures and the Alberta Machine Intelligence Institute (Amii). Computing resources were provided by Compute Canada through CalculQuébec.

References

Bacon, Pierre-Luc. On the Bottleneck Concept for Options Discovery: Theoretical Underpinnings and Extension in Continuous State Spaces. Master's thesis, McGill University, 2013.

Bacon, Pierre-Luc, Harb, Jean, and Precup, Doina. The option-critic architecture. In *Proceedings of the National Conference on Artificial Intelligence (AAAI)*, 2017.

Baranes, Adrien and Oudeyer, Pierre-Yves. Active learning of inverse models with intrinsically motivated goal exploration in robots. *Robotics and Autonomous Systems*, 61(1):49–73, 2013.

Bellemare, Marc G., Naddaf, Yavar, Veness, Joel, and Bowling, Michael. The Arcade Learning Environment: An Evaluation Platform for General Agents. *Journal of Artificial Intelligence Research*, 47:253–279, 2013.

Bellman, Richard E. *Dynamic Programming*. Princeton University Press, Princeton, NJ, 1957.

Şimşek, Özgür and Barto, Andrew G. Using Relative Novelty to Identify Useful Temporal Abstractions in Reinforcement Learning. In *Proceedings of the International Conference on Machine Learning (ICML)*, 2004.

Şimşek, Özgür and Barto, Andrew G. Skill Characterization Based on Betweenness. In *Proceedings of Advances in Neural Information Processing Systems (NIPS)*, 2008.

Şimşek, Özgür, Wolfe, Alicia P., and Barto, Andrew G. Identifying Useful Subgoals in Reinforcement Learning by Local Graph Partitioning. In *Proceedings of the International Conference on Machine Learning (ICML)*, 2005.

Daniel, Christian, van Hoof, Herke, Peters, Jan, and Neumann, Gerhard. Probabilistic Inference for Determining Options in Reinforcement Learning. *Machine Learning*, 104(2):337–357, 2016.

Dietterich, Thomas G. Hierarchical Reinforcement Learning with the MAXQ Value Function Decomposition. *Journal of Artificial Intelligence Research (JAIR)*, 13: 227–303, 2000.

Gregor, Karol, Rezende, Danilo, and Wierstra, Daan. Variational Intrinsic Control. *CoRR*, abs/1611.07507, 2016.

Gross, Jonathan L. and Yellen, Jay. *Graph Theory and Its Applications*. Chapman and Hall/CRC, 2 edition, 2006.

Hengst, Bernhard. Discovering Hierarchy in Reinforcement Learning with HEXQ. In *Proceedings of the International Conference on Machine Learning (ICML)*, 2002.

Kompella, Varun Raj, Luciw, Matthew D., and Schmidhuber, Jürgen. Incremental Slow Feature Analysis. In *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI)*, pp. 1354–1359, 2011.

Kompella, Varun Raj, Stollenga, Marijn, Luciw, Matthew, and Schmidhuber, Juergen. Continual Curiosity-Driven Skill Acquisition from High-Dimensional Video Inputs for Humanoid Robots. *Artificial Intelligence*, In Press. ISSN 0004-3702. Available online 12 February 2015.

Konidaris, George and Barto, Andrew. Skill Discovery in Continuous Reinforcement Learning Domains using Skill Chaining. In *Proceedings of Advances in Neural Information Processing Systems (NIPS)*, pp. 1015–1023, 2009.

Kulkarni, Tejas D., Narasimhan, Karthik R., Saeedi, Ardavan, and Tenenbaum, Joshua B. Hierarchical Deep Reinforcement Learning: Integrating Temporal Abstraction and Intrinsic Motivation. *ArXiv e-prints*, 2016.

Lakshminarayanan, Aravind, Krishnamurthy, Ramnandan, Kumar, Peeyush, and Ravindran, Balaraman. Option Discovery in Hierarchical Reinforcement Learning using Spatio-Temporal Clustering. *CoRR*, abs/1605.05359, 2016. Presented at the ICML-16 Workshop on Abstraction in Reinforcement Learning.

Machado, Marlos C. and Bowling, Michael. Learning Purposeful Behaviour in the Absence of Rewards. *CoRR*, abs/1410.4604, 2016. Presented at the ICML-16 Workshop on Abstraction in Reinforcement Learning.

Mahadevan, Sridhar. Proto-Value Functions: Developmental Reinforcement Learning. In *Proceedings of the International Conference on Machine Learning (ICML)*, pp. 553–560, 2005.

Mahadevan, Sridhar and Maggioni, Mauro. Proto-value Functions: A Laplacian Framework for Learning Representation and Control in Markov Decision Processes. *Journal of Machine Learning Research (JMLR)*, 8:2169–2231, 2007.

Mankowitz, Daniel J., Mann, Timothy Arthur, and Mannor, Shie. Adaptive Skills Adaptive Partitions (ASAP). In *Proceedings of Advances in Neural Information Processing Systems (NIPS)*, pp. 1588–1596, 2016.

Mannor, Shie, Menache, Ishai, Hoze, Amit, and Klein, Uri. Dynamic Abstraction in Reinforcement Learning via Clustering. In *Proceedings of the International Conference on Machine Learning (ICML)*, 2004.

McGovern, Amy and Barto, Andrew G. Automatic Discovery of Subgoals in Reinforcement Learning using Diverse Density. In *Proceedings of the International Conference on Machine Learning (ICML)*, 2001.

Menache, Ishai, Mannor, Shie, and Shimkin, Nahum. Q-Cut - Dynamic Discovery of Sub-goals in Reinforcement Learning. In *Proceedings of the European Conference on Machine Learning (ECML)*, 2002.

Moulin-Frier, Clément and Oudeyer, Pierre-Yves. Exploration Strategies in Developmental Robotics: A Unified Probabilistic Framework. In *Proceedings of the Joint IEEE International Conference on Development and Learning and Epigenetic Robotics (ICDL-EpiRob)*, pp. 1–6, 2013.

Oh, Junhyuk, Chockalingam, Valliappa, Singh, Satinder P., and Lee, Honglak. Control of Memory, Active Perception, and Action in Minecraft. In *Proceedings of the International Conference on Machine Learning (ICML)*, pp. 2790–2799, 2016.

Osband, Ian, Roy, Benjamin Van, and Wen, Zheng. Generalization and Exploration via Randomized Value Functions. In *Proceedings of the International Conference on Machine Learning (ICML)*, pp. 2377–2386, 2016.

Precup, Doina. *Temporal Abstraction in Reinforcement Learning*. PhD thesis, University of Massachusetts Amherst, 2000.

Puterman, Martin L. *Markov Decision Processes: Discrete Stochastic Dynamic Programming*. John Wiley & Sons, Inc., New York, NY, USA, 1994.

Salge, Christoph, Glackin, Cornelius, and Polani, Daniel. Empowerment – An Introduction. In *Guided Self-Organization: Inception*, pp. 67–114. Springer, 2014.

Solway, Alec, Diuk, Carlos, Córdova, Natalia, Yee, Debbie, Barto, Andrew G., Niv, Yael, and Botvinick, Matthew M. Optimal Behavioral Hierarchy. *PLOS Computational Biology*, 10(8):1–10, 2014.

Sprekeler, Henning. On the Relation of Slow Feature Analysis and Laplacian Eigenmaps. *Neural Computation*, 23 (12):3287–3302, 2011.

Strang, Gilbert. *Linear Algebra and Its Applications*. Brooks Cole, 2005.

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

Sutton, Richard S., Precup, Doina, and Singh, Satinder. Between MDPs and semi-MDPs: A Framework for Temporal Abstraction in Reinforcement Learning. *Artificial Intelligence*, 112(12):181 – 211, 1999.

Szepesvári, Csaba. *Algorithms for Reinforcement Learning*. Synthesis Lectures on Artificial Intelligence and Machine Learning. Morgan & Claypool, 2010.

Vezhnevets, Alexander, Mnih, Volodymyr, Osindero, Simon, Graves, Alex, Vinyals, Oriol, Agapiou, John, and Kavukcuoglu, Koray. Strategic Attentive Writer for Learning Macro-Actions. In *Proceedings of Advances in Neural Information Processing Systems (NIPS)*, pp. 3486–3494, 2016.

Watkins, Christopher J. C. H. and Dayan, Peter. Technical Note: Q-Learning. *Machine Learning*, 8(3-4), May 1992.

Weber, Marcus, Rungsarityotin, Wasinee, and Schliep, Alexander. Perron Cluster Analysis and Its Connection to Graph Partitioning for Noisy Data. Technical Report 04-39, ZIB, Takustr.7, 14195 Berlin, 2004.

Wiskott, Laurenz and Sejnowski, Terrence J. Slow Feature Analysis: Unsupervised Learning of Invariances. *Neural Computation*, 14(4):715–770, 2002.

Appendix: Supplementary Material

This supplementary material contains details omitted from the main text due to space constraints. The list of contents is below:

- Supporting lemmas and their respective proofs, as well as a more detailed proof of Theorem 3.1;
- Description of how to easily compute the *diffusion time* in tabular MDPs;
- The options leading to bottleneck states (doorways) we used in our experiments;
- Performance comparisons between eigenoptions and options generated to reach randomly selected states;
- Demonstration of the applicability of eigenoptions in multiple tasks with a new set of experiments;
- Further details on the empirical setting used in the Arcade Learning Environment.

A. Lemmas and Proofs

Lemma 11.1. *Suppose $(I + A)$ is a non-singular matrix, with $||A|| \leq 1$. We have:*

$$||(I + A)^{-1}|| \leq \frac{1}{1 - ||A||}.$$

Proof. [4]

$$\begin{aligned}
(I + A)(I + A)^{-1} &= I \\
I(I + A)^{-1} + A(I + A)^{-1} &= I \\
(I + A)^{-1} &= I - A(I + A)^{-1} \\
||(I + A)^{-1}|| &= ||I - A(I + A)^{-1}|| \\
&\leq ||I|| + ||A(I + A)^{-1}|| && \text{because } ||A + B|| \leq ||A|| + ||B|| \\
&\leq 1 + ||A||||(I + A)^{-1}|| && \text{because } ||AB|| \leq ||A|| \cdot ||B|| \\
||(I + A)^{-1}|| - ||A||||(I + A)^{-1}|| &\leq 1 \\
(1 - ||A||)||(I + A)^{-1}|| &\leq 1 \\
||(I + A)^{-1}|| &\leq \frac{1}{1 - ||A||} && \text{if } ||A|| \leq 1.
\end{aligned}$$

□

Lemma 11.2. *The induced infinity norm of $(I - \gamma T)^{-1}T$ is bounded by*

$$||(I - \gamma T)^{-1}T||_\infty \leq \frac{1}{(1 - \gamma)}.$$

Proof.

$$\begin{aligned}
||(I - \gamma T)^{-1}T||_\infty &\leq ||(I - \gamma T)^{-1}||_\infty ||T||_\infty && \text{because } ||AB||_\infty \leq ||A||_\infty \cdot ||B||_\infty \\
||(I - \gamma T)^{-1}T||_\infty &\leq \frac{1}{1 - ||-\gamma T||_\infty} ||T||_\infty && \text{Lemma 3.1} \\
||(I - \gamma T)^{-1}T||_\infty &\leq \frac{1}{1 - \gamma||T||_\infty} ||T||_\infty && \text{because } ||\lambda B|| = |\lambda|||B|| \\
||(I - \gamma T)^{-1}T||_\infty &\leq \frac{1}{(1 - \gamma)}
\end{aligned}$$

□

[4] Our proof follows closely the proof of Parnell in lecture notes available at http://www-solar.mcs.st-and.ac.uk/~clare/Lectures/num-analysis.html.

Theorem 11.1 (Option's Termination)**.** *Consider an eigenoption $o = \langle \mathcal{I}_o, \pi_o, \mathcal{T}_o \rangle$ and $\gamma < 1$. Then, in an MDP with finite state space, $\mathcal{T}_o$ is nonempty.*

Proof. This proof is more detailed than the one presented in the main paper. We can write the Bellman equation in the matrix form: $\mathbf{v} = \mathbf{r} + \gamma T \mathbf{v}$, where $\mathbf{v}$ is a *finite* column vector with one entry per state encoding its value function. From equation (1) in the main paper we have $\mathbf{r} = T\mathbf{w} - \mathbf{w}$ with $\mathbf{w} = \boldsymbol{\phi}(s)^\top \mathbf{e}$, where $\mathbf{e}$ denotes the eigenpurpose of interest. Therefore:

$$
\begin{aligned}
\mathbf{v} &= T\mathbf{w} - \mathbf{w} + \gamma T\mathbf{v} \\
\mathbf{v} + \mathbf{w} &= T\mathbf{w} + \gamma T\mathbf{v} \\
&= T\mathbf{w} + \gamma T\mathbf{v} + \gamma T\mathbf{w} - \gamma T\mathbf{w} \\
&= (1-\gamma)T\mathbf{w} + \gamma T(\mathbf{v} + \mathbf{w}) \\
\mathbf{v} + \mathbf{w} - \gamma T(\mathbf{v} + \mathbf{w}) &= (1-\gamma)T\mathbf{w} \\
(I - \gamma T)(\mathbf{v} + \mathbf{w}) &= (1-\gamma)T\mathbf{w} \\
\mathbf{v} + \mathbf{w} &= (1-\gamma)(I-\gamma T)^{-1} T\mathbf{w} && (I-\gamma T)^{-1} \text{ is guaranteed to be nonsigular because} \\
& && ||T|| \leq 1, \text{ where } ||T|| = \sup_{\mathbf{v}:||\mathbf{v}||_\infty = 1} ||T\mathbf{v}||_\infty. \text{ By} \\
& && \text{Neumann series we have } (I-\gamma T)^{-1} = \sum_{n=0}^{\infty} \gamma^n T^n \\
||\mathbf{v} + \mathbf{w}||_\infty &= (1-\gamma)||(I-\gamma T)^{-1} T\mathbf{w}||_\infty && \text{using the induced norm} \\
||\mathbf{v} + \mathbf{w}||_\infty &\leq (1-\gamma)||(I-\gamma T)^{-1} T||_\infty ||\mathbf{w}||_\infty && \text{because } ||A\mathbf{x}|| \leq ||A|| \cdot ||\mathbf{x}|| \\
||\mathbf{v} + \mathbf{w}||_\infty &\leq (1-\gamma)\frac{1}{(1-\gamma)}||\mathbf{w}||_\infty && \text{Lemma 3.2} \\
||\mathbf{v} + \mathbf{w}||_\infty &\leq ||\mathbf{w}||_\infty
\end{aligned}
$$

We can shift $\mathbf{w}$ by any finite constant without changing the reward, *i.e.* $T\mathbf{w} - \mathbf{w} = T(\mathbf{w}+\boldsymbol{\delta}) - (\mathbf{w}+\boldsymbol{\delta})$ because $T\mathbf{1}\boldsymbol{\delta} = \mathbf{1}\boldsymbol{\delta}$ since $\sum_j T_{i,j} = 1$. Therefore, we can assume $\mathbf{w} \geq \mathbf{0}$. Let $s^* = \arg\max_s \mathbf{w}_{s^*}$, so that $\mathbf{w}_{s^*} = ||\mathbf{w}||_\infty$. Clearly $\mathbf{v}_{s^*} \leq \mathbf{0}$, otherwise $||\mathbf{v} + \mathbf{w}||_\infty \geq |\mathbf{v}_{s^*} + \mathbf{w}_{s^*}| = \mathbf{v}_{s^*} + \mathbf{w}_{s^*} > \mathbf{w}_{s^*} = ||\mathbf{w}||_\infty$, arriving at a contradiction.

□

Lemma 12.1. *In the tabular case, if all transitions in the MDP have been sampled once, $T^\top T = 2L$.*

Proof. Let t_{ij} and tt_{ij} denote the entries in the i-th row and j-th column of matrices T and $T^\top T$. We can write tt_{ij} as:

$$tt_{ij} = \sum_k t_{ik} \times t_{jk}. \tag{2}$$

In the tabular case, t_{ij} has three possible values:

- $t_{ij} = +1$, meaning that the agent arrived in state j at time step i,
- $t_{ij} = -1$, meaning that the agent left state j at time step i,
- $t_{ij} = 0$, meaning that the agent did not arrive nor leave state j at time step i.

We decompose $T^\top T$ in two matrices, K and Z, such that $T^\top T = K + Z$. Here Z is a diagonal matrix such that $z_{ii} = tt_{ii}$, for all i; and K contains all elements from $T^\top T$ that lie outside the main diagonal.

When computing the elements of Z we have $i = j$. Thus $z_{ii} = \sum_k t_{ik}^2$. Because we square all elements, we are in fact summing over all transitions leaving (-1^2) and arriving (1^2) in state i, counting the node's degree twice. Thus, $Z = 2D$.

When not computing the elements in the main diagonal, for the element tt_{ij}, we add all transitions that leave state i arriving in state j (-1×1), and those that leave state j arriving in state i (1×-1). We assume each transition has been sampled once, thus:

$$tt_{ij} = \begin{cases} -2, & \text{if the transition between states } i \text{ and } j \text{ exists,} \\ 0, & \text{otherwise.} \end{cases}$$

Therefore, we have $K = -2W$ and $T^\top T = K + Z = 2(D - W)$. □

B. Diffusion Time Computation

In the main paper we introduced *diffusion time* as a new metric to evaluate exploration, but we did not discuss how it can be computed. Diffusion time encodes the expected number of time steps required to navigate between any two states in the MDP when following a random walk. In tabular domains, we can easily compute the diffusion time with dynamic programming. To do so we define a new MDP such that the value function of a state s, under a uniform random policy, encodes the expected number of steps required to navigate between state s and a chosen goal state. We can then compute the expected number of steps between any two states by averaging, for each possible goal, the value of all other states.

The MDP in which the value function of state s encodes the expected number of time steps from s to a goal state has $\gamma = 1$ and a reward function where the agent observes $+1$ at every time step in which it is not in the goal state. Policy evaluation in this case encodes the expected number of time steps the agent will take before arriving to the goal state. To compute the diffusion time we iterate over all possible states, defining them as terminal states, and averaging the value function of the other states in that MDP.

C. Options Leading to Doorways in the 4-room Domain

Figure 10 depicts the four options we refer to in Section 4 as the options leading to bootleneck states, *i.e.*, doorways. Each option is defined in a room and it moves the agent toward the closest doorway. These options were inspired by Solway et al. (2014)'s discussion about the optimal options discovered by their algorithm.

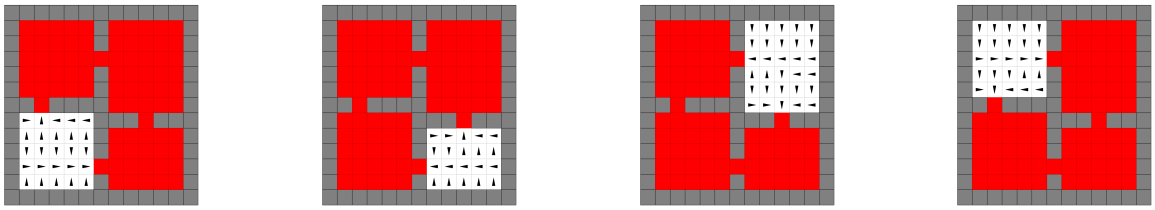


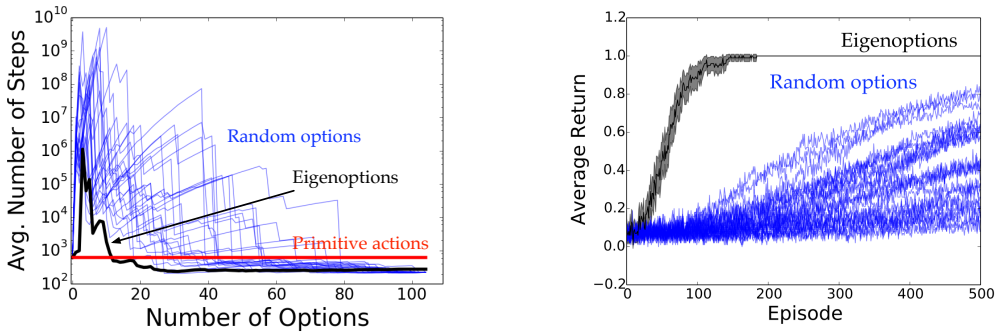
Figure 10. Options leading to bottleneck states. Each option is defined in a single room, moving the agent to the closest doorway.

D. Comparison to Random Options

In this section we show the importance of using information about diffusion in the environment to define the option's purposes. This information impacts the sequence of subgoal locations the options' seek after, as well as the time scales they operate at. The ordering in which the eigenoptions are discovered and the different time scales they operate at can have a major impact on the agents' performance.

We demonstrate the importance of using the environment's diffusion information by comparing our approach to *random options*, a simple baseline that does not use such information. This baseline defines an option to be the policy, defined in the whole state space, that terminates in a randomly selected state of the environment. We performed our experiments in the tabular case because it is not clear how we can extend this baseline to settings in which states cannot be enumerated.

Figure 11a depicts the diffusion time (*c.f.* Section B) of random options and eigenoptions in the 4-room domain. We used the same method described in Section 4.2 to obtain the eigenoptions' performance. For the random options results, we added them incrementally to the agent's action set until having added all possible options. We repeated this process 24 times to verify the impact of adding random options in a different order. Each blue line represents the performance of one



(a) Diffusion time. The y-axis is in logarithmic scale.

(b) Learning curve using 64 options.

Figure 11. Diffusion time and learning performance of eigenoptions and of random options in the 4-room domain.

of the evaluated sequences. The results clearly show that eigenoptions do more than going to a randomly selected state. Most of the obtained sequences of random options fail to reduce the agent's diffusion time. They increase it by several orders of magnitude (notice the y-axis is in logarithmic scale) until having enough options available to the point that the graph is almost fully connected, that is, when the agent basically has an option leading it to each possible state in the MDP.

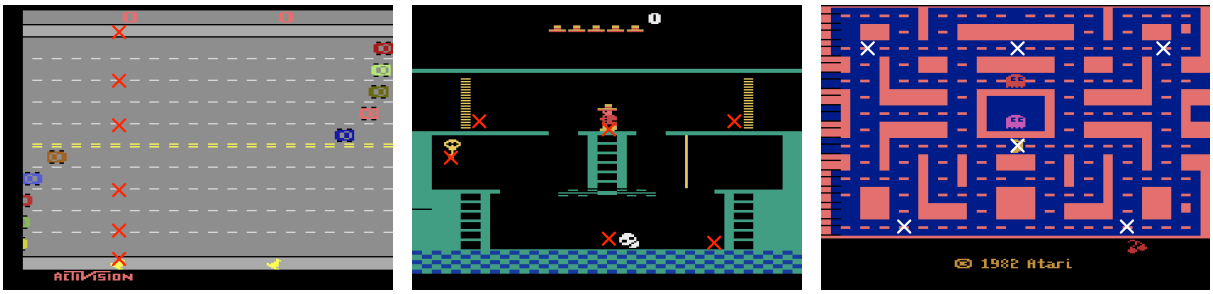
Figure 11b was generated following the protocol described in Section 4.3. It depicts the learning curve of agents equipped with eigenoptions and of agents equipped with random options. As before, the blue lines indicate the agent's performance in individual runs. We can see that no individual run is competitive to eigenoptions. When fewer options are used (not shown), the variance across individual runs is even larger, depending on whether one of the random options terminates near the goal state. In some runs the agent never even learns to reach the goal. Therefore, as in the diffusion time, on average, random options are not competitive to eigenoptions, demonstrating the importance of the diffusion model we use.

D. Empirical Evaluation of the Agent's Performance in Multiple Tasks

In Section 4 we argued that eigenoptions are useful for multiple tasks, based on results showing that eigenoptions allow us to find and to accumulated rewards faster. Here we explicit demonstrate the uselfuness of eigenoptions to multiple tasks. We evaluate the agents' performance for different starting and goal states in the 4-room domain. As in Section 4.3, we use Q-Learning ($\alpha = 0.1, \gamma = 0.9$) to learn a policy over primitive actions. The behavior policy chooses uniformly over primitive actions and options, following them until termination. Episodes were 100 time steps long, and we learned for 250 episodes. For clarity, we zoom in the plots on the interval in which agents are still learning.

Figure 14 depicts, after learning for a pre-determined number of episodes, the average over 100 trials of the agents' final performance, as well as the starting (S) and goal (G) states. Based on our previous results, we fixed the number of used eigenoptions to 64 (32 options and their negations). In this set of experiments we also compare our approach to traditional bottleneck options (Figure 10).

The obtained results show that switching the positions of the starting and goal states have no effect in the performance of our algorithm. Also, in almost all settings, the agents augmented by eigenoptions outperfom those equipped only with primitive actions. The comparison between eigenoptions and options that look for bottleneck states is more subtle. As expected, agents equipped with eigenoptions outperform agents equipped with options leading to bottleneck states in settings in which the goal state is far from the doorways, as discussed in the main paper. In scenarios where the goal state is closer to bottleneck states, the options leading to doorways are more competitive. Importantly, this analysis is based on the results when using 64 eigenoptions, which may not encode all options required to go to a specific region of the state space.



(a) FREEWAY (b) MONTEZUMA'S REVENGE (c) MS PAC-MAN

Figure 12. Pre-defined start states in Atari 2600 games.

E. Experimental Setup in the Arcade Learning Environment

We defined six different starting states in each Atari 2600 game, letting the agent take random actions from that point until termination. The agent follows a pre-determined sequence of actions leading it to each starting state. We store the observed transitions leading the agent to the start states as well as those obtained from the random actions. In the main paper we provided results for FREEWAY and MONTEZUMA'S REVENGE. In this section we also provide results for MS PAC-MAN. The starting states for all three games are depicted in Figure 12.

The agent plays rounds of six episodes, with each episode starting from a different start state, until it observes at least 25,000 new transitions. The final incidence matrix in which we ran the SVD had 25,000 rows, which we sampled uniformly from the set of observed transitions. The agent used the deterministic version of the Arcade Learning Environment (ALE), the games' minimal action set and, a frame skip of 1.

We used three games to evaluate the options we discover in the sample-based setting with linear function approximation. We discussed the results for FREEWAY and MONTEZUMA'S REVENGE in the main paper. The results we obtained in MS. PAC-MAN are similar to those we already discussed. MS. PAC-MAN is a game in which the agent needs to navigate through a maze eating pellets while avoiding ghosts. As in the other games, the agent has the clear intent of reaching particular positions in the screen, such as corners and intersections. Figure 4 depicts the positions in which agents tend to spend most of their time on. A video of the highlighted options can be found online.[5]

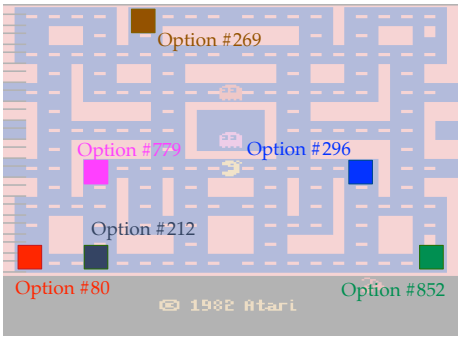


Figure 13. Options in MS. PAC-MAN (*c.f.* text for details).

[5] https://youtu.be/2BVicx4CDWA

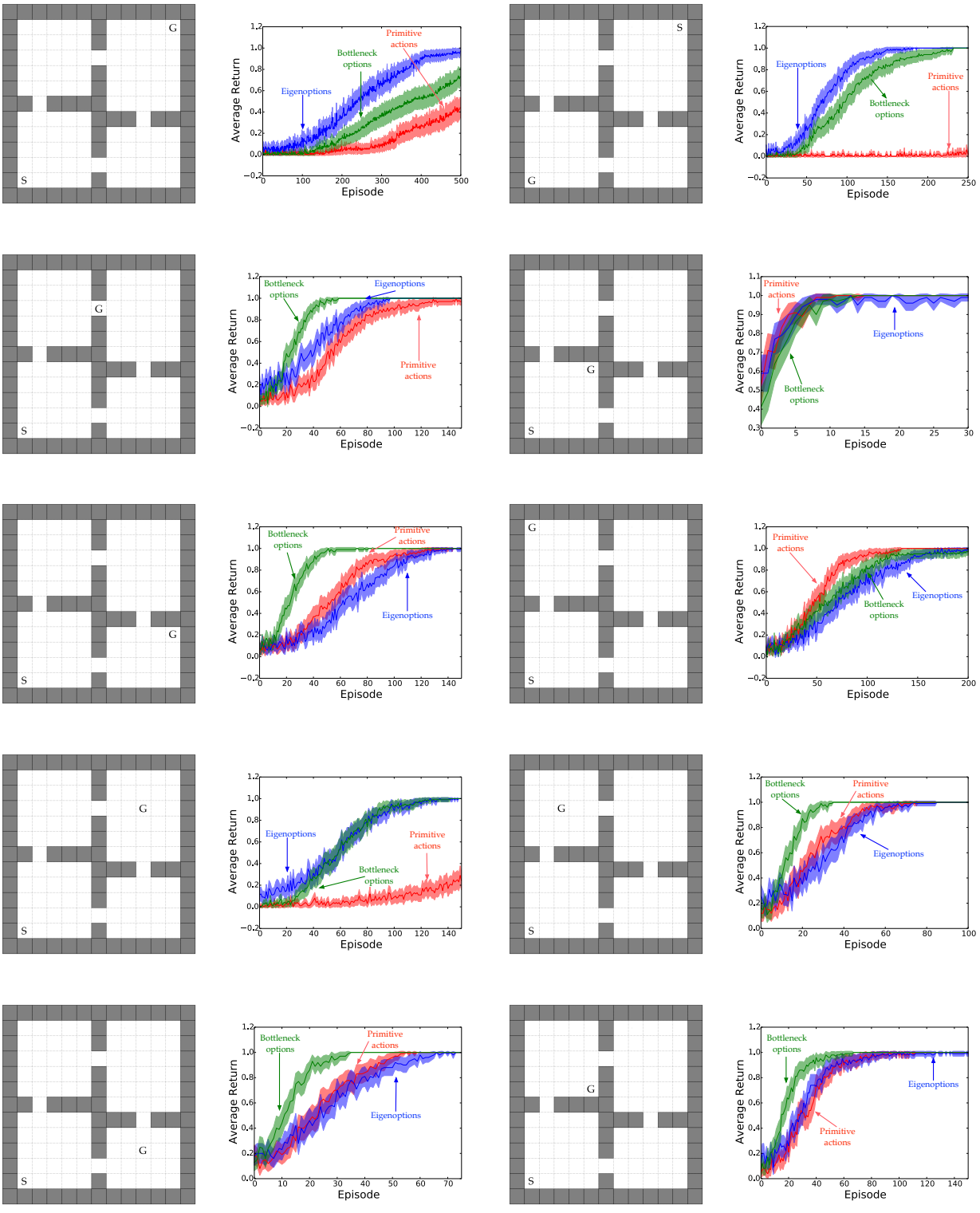


Figure 14. Agents performance in different tasks when using eigenoptions, bottleneck options, and primitive actions.