Efficient hierarchical subgoal discovery for reinforcement learning

Adrian Millea
a.millea14@imperial.ac.uk
Department of Computing, Imperial College London, SW7 2AZ, London

Abstract

In complex environments, where rewards are sparse and there are many states, the problem of meaningful behaviour for a reinforcement learning agents amounts to finding additional structures in the environment that can guide the agent's actions, often referred to as pseudo-rewards. These can be continuous functions, added at every timestep, usually defined by an expert designer, or in some cases can be discrete. We will focus on this latter type, where the pseudo-rewards are received by the agent when it finds itself in specific, interesting states. Methods generally employed for finding these states are global and thus expensive, especially where the number of states of the environment is large. We provide a simple solution to this problem employing exclusively local methods, that is able to discover subgoal states efficiently.

Keywords: **deep reinforcement learning, value-based RL, options, locality, subgoal discovery, pseudo-rewards, successor representation, temporal abstraction**

1 Introduction

Recent advances in Deep Reinforcement Learning (DRL) has shown super human performance in a number of domains, especially Atari games. These games are interesting to test on as they provide various levels of difficulty with respect to temporal abstraction, reward sparsity, complex different policies, etc. One of the major shortcomings of such methods is that they cannot deal with rewards that are too sparse in time. To overcome this, a way of structuring the environment meaningfully is needed. The general approach until now has been to discover peculiar, rare, or bottleneck states, as they are called. Finding these interesting states, also called subgoals, or options, can be quite expensive and methods that do usually employ complex clustering algorithms, or expensive operations such as eigendecomposition. The main point is that all methods are global, in the sense that the state-space is explored extensively to get enough data such that these methods find useful subgoals. We propose a new and different approach that computes some simple quantities locally for each state. This general problem in RL is known as the option discovery problem.

2 Background and related work

We state next briefly some of the approaches existent in the literature for option discovery. The problem of discovering options is tightly related to the exploration-exploitation dilemma in RL. We need a exploration mechanism that enables us to efficiently go through the state-action space such that we can identify candidate options. A question that naturally arises in this context is what particular characteristics does an option need to posses to be considered useful for the agent. An option is considered useful if it can implement some behavioural pattern that can be reused by the agent in different contexts, so in effect we are interested in what subtasks an option implements, so we can look at the final subgoal of an option to figure out some of its characteristics. One perspective, is to consider as subgoals particular states or state-sequences which enable access to a wider variety of next states, so effectively maximizing the state-space subset accessible from this subgoal state, and thus maximizing information accessible to the agent. This is a widely used concept in curiosity and efficient exploration, it is often referred to as the information-seeking principle. So an option could enable the agent to learn to transition efficiently to these subgoals. These are often called bottleneck states. We give next some of the approaches existent in the literature for finding such states:

- [McGovern and Barto, 2001] look at rare transitions between subsets of state-space, and consider these points of transition as bottleneck states
- [Hengst, 2004] looks at the sequences of actions that cause rare changes in the state representation. By employing a factored state representation, rare transitions can be identified
- In [Şimşek and Barreto, 2009] they derive a graphical representation of the state-action space and then use node centrality, a graph theoretic measure to identify nodes with high centrality
- In [Menache et al., 2002] they use clustering methods to identify strongly connected components of the underlying MDP and then consider as bottleneck states those states that connect two or more clusters.
- In [Brunskill and Li, 2014] they use a greedy approach to discovering options inspired by PAC-learning, by making use of sample complexity, a measure of learning speed, counting the number of non-optimal actions [Kakade et al., 2003].

As we can see there is an overarching principle, we are searching for peculiar states, states that are different and that enable the agent to gather more information about the state-action space or get access to a new part of the state space. In the context of machine learning we could even see them as some type of outliers in the state-action space. A hierarchical approach to discovering options is through spatio-temporal clustering through PCCA+ (Perron Clustering) [Lakshminarayanan et al., 2016]. In this work the authors assign states to abstract states in a fuzzy manner, where each state has a membership function which governs how much a particular state is part of an abstract state. Then they find the policy over abstract states through hill-climbing on the membership functions. This is quite elegant and moreover it is enhanced by the use of the powerful spectral clustering technique (PCCA+) and by a action conditional convolutional network [Oh et al., 2015], which extracts high level features that also take time into account. The authors show the advantage of discovering the different structural and functional abstractions, and especially differentiating between them, as we are also advocating throughout this paper.

One often used method for finding subgoals, or options, or states that have interesting properties is to use the minimum normalized cut. This has been used for discovering options [Kulkarni et al., 2016b, Şimşek et al., 2005] but also for discovering other types of structures, for example, objects in an image [Shi and Malik, 2000]. The main idea used in DRL [Kulkarni et al., 2016b] is to first collect a large number of states $\mathrm{T} = \{m_{s_1,a_1}, m_{s_2,a_2}, ..., m_{s_n,a_n}\}$, following a random policy, and then generate an affinity matrix W by applying a radial basis function (with Euclidean metric) to every pair $(m_{s_i,a_i}, m_{s_j,a_j})$ in T, to generate for each such pair a similarity w_{ij}. Now, form the diagonal matrix $D = \sum_j w_{ij}$ and take the second largest eigenvalue of the matrix $D^{-1}(D - W)$ which will give an approximation of the minimum normalized cut of partitioning T. Then the subgoals are the points, in this case these are states, which lie on the endpoints of the cut. After randomly sampling T, statistics of how many times a state lies along the cut are collected and then the top-k such states are selected as subgoals. These prove to be useful subgoals for speeding up learning. A very similar idea, a bit simpler, is presented in [Kulkarni et al., 2016a] where the authors perform image segmentation (in Montezuma's revenge) to localize individual objects in the image and then use those as subgoals. Ideally one would like a relatively efficient way of devising subgoals but also comprehensive. This is an active area of research.

3 Subgoal discovery

Very recent high-quality research has shown that the hippocampus might function as a predictive map, where the successor representation (SR) is employed [Stachenfeld et al., 2017]:

$$M(s, s') = \mathbb{E}\left(\sum_{t=0}^{\infty} \gamma^t \mathbb{I}(s_t = s') | s_0 = s\right]$$

which gives the value function as:

$$V(s) = \sum_{s'} M(s, s') R(s')$$

The SR catches the discounted future state occupancy. Even though the idea and underlying physiological evidence are quite convincing and simple, the method has a number of insufficiencies, for example, this matrix should be computed at every state for all next states, which is intractable for a big number of states, which is usually the case in complex environments. For different γ the SR looks quite different, with bigger values showing long temporal dependency between states. The authors show further how to get bottleneck states usually used as subgoals, by using the neurons (grid fields) that encode a low dimensional eigendecomposition of the SR. Even though the low-dimensional clustering of states shows indeed an interesting property at different γs, the authors don't use the multiple levels in any way for decision making. Moreover, the process of using the eigenvectors in the normalized cut algorithm to discover the bottleneck states is quite expensive. We strive to provide a framework which has a number of advantages:

- it uses memory with a local approximation of the SR, instead of going through all states (this is a necessary condition for large state-spaces)
- the SR can look at more steps into the future
- discovery of subgoals does not use any expensive global algorithm (clustering, eigendecomposition, etc.)

3.1 Locality

Inspired by [Schapiro et al., 2016] we define a set of neighbour sets for each state (successor states):

$$\begin{aligned}
N_1(s) &= \{s' \in \mathcal{S} | \forall s' \text{ with } (s, s') \in \mathcal{M}_B\} \\
N_2(s) &= \{s'' \in \mathcal{S} | \forall s', s'' \text{ with } (s, s') \text{ and } (s', s'') \in \mathcal{M}_B\} \\
&\dots
\end{aligned}$$

where $\mathcal{M}_B$ is the set of valid transitions seen in the environment (or base memory) and $\mathcal{S}$ is the set of states in the environment. We now have a representation of each state as sets of neighbour states. We are interested to discover peculiar states by looking at this representation, in effect, computing distances between states. However, by doing this locally we manage the quadratic growth of the computational needs for computing a similarity matrix, since our sets of states for which we compute similarities is quite small. Even though at first we might discover nuisance subgoal states, the abstraction and locality will allow us to reach useful subgoals rapidly. Differences between states can be computed using for example the following dissimilarity function:

$$d_n(s_i, s_j) = |N_n(s_i) \backslash N_n(s_j)| + |N_n(s_j) \backslash N_n(s_i)|$$

where $n \in \mathbb{N}$ is the index of the neighbour function used (n denotes the number of neighbours in the future we look at, we use $n \in \{1, 2, 3\}$ in our experiments). Since N_n are sets, $\backslash$ denotes set subtraction and $|\cdot|$ denotes cardinality of set. Experiments show that this simple technique for discovering subgoals works and is extremely efficient. We note that to the best of our knowledge this is the only local technique for discovering subgoals and does not imply any expensive clustering algorithm or eigendecomposition. Of course, the neighbour sets N_n can be combined in different ways to obtain different distance functions which might be more appropriate for the environment (weighted average, max, etc). We note that with higher n the agent should be able to discover paths that constitute bottlenecks (e.g. a narrow tunnel, a ladder, etc.), not just single states.

3.2 Heterogeneity

Fortunately, these distances need to be computed once in a while, as the agent accumulates experience. Moreover, since we compute these quantities locally, we need a sufficiently good representation for the tractability of these computations, meaning we can project high dimensional state-spaces to lower dimensions even though we lose information, as long as the general structuring remains. For example we can use cheap random projections, which assures us through the J-L Lemma that we lose a limited amount of information. After investigating these distance function as indicative for peculiar states, hoping that we will get certain minimal values for our subgoals of interest, we see that we should take into account also the sizes of the neighbour sets when computing distances. This is because subgoal states usually have a smaller number of neighbours and there are often states which have such a small number of neighbours in common, even though the overall set of neighbours is larger. As o conclusion, one needs a more appropriate distance function to weigh this set difference correctly. As an alternative we consider the following quantity (we call it *heterogeneity*, because it denotes how heterogeneous the set of neighbours of each state is) and we define it as:

$$h_n(s) = \frac{1}{|N_n(s)|^2} \sum_{s_i \in N_n(s)} \sum_{s_j \in N_n(s)} d_n(s_i, s_j)$$

where n denotes the index of the neighbour function used in the dissimilarity calculation. We see how in all these distance computations we leverage extensively the locality of states, i.e. the fact that each state has a small number of neighbours. The heterogeneity measures the *average pairwise dissimilarity* of the set of the neighbours for the state s. We test this measure on a small 4-room gridworld. We show in Figure 3.2 the computation of heterogeneity after 50 episodes, 100 steps per episode, with random actions and a random start position in each episode. We show the values when using N_1, N_2, N_3 and their mean. We note that this computation is quite general, for various state spaces and dissimilarity functions, however it entails one important assumption: the number of neighbour states of any state is relatively small, however in practice this assumption is quite reasonable. We also note that for this small state space 50 episodes seems to be enough to discover the subgoals, however for more complex environments we will surely use smarter exploration strategies than the random one, which will compensate. After sufficient exploration of a neighbourhood, the heterogeneity does not change anymore. This is where we consider the maximum values seen in the h. Bottleneck states usually connect different partitions of the environment which means that the average pairwise dissimilarity is quite high, significantly higher than for the majority of states. We thus consider as subgoals the states that have a maximum value of h. We thus define the set of subgoal states as:

$$G_n = \{g \in \mathcal{S} | h_n(g) > h_n(s), \forall s \in \mathcal{S}\}$$

3.3 Value functions

We augment our agent with a subgoal value function, through which we give a pseudo-reward to the agent when it reaches subgoals state. We connect the different states through the neighbour function

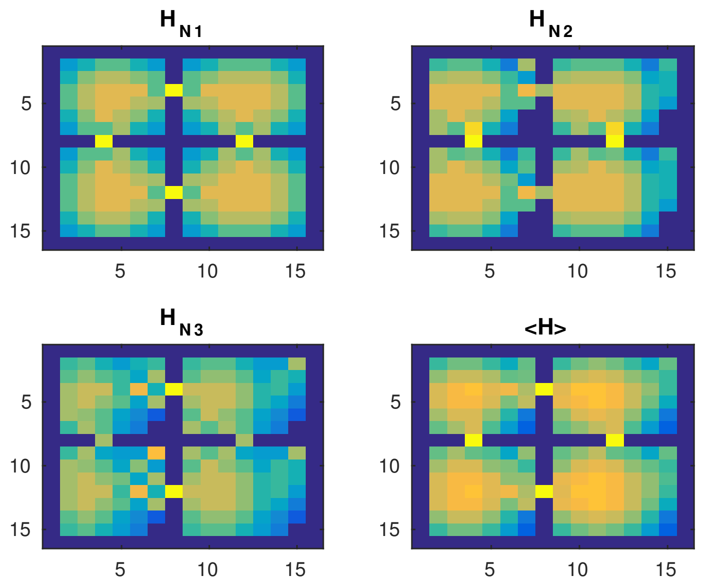

Figure 1: Heterogeneity on 16×16 4-room gridworld clearly discovers subgoals.

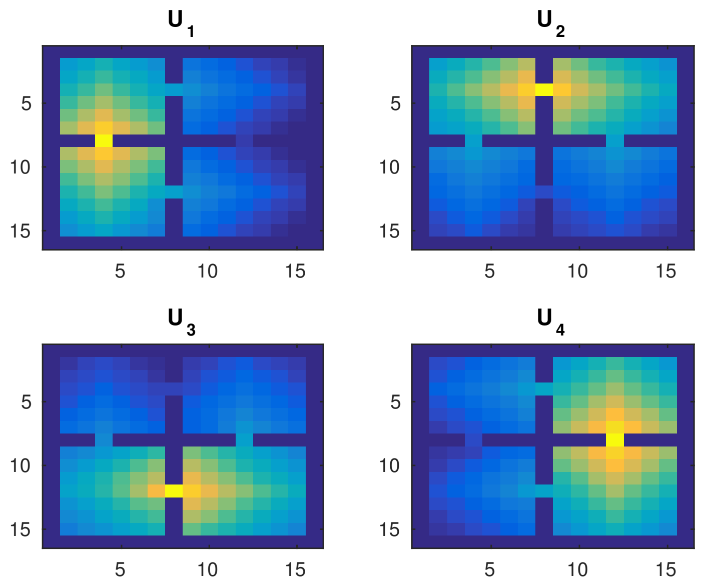

Figure 2: Subgoal value function on 16×16 4-room gridworld.

used previously, $N(s)$. Considering that agent has just reached state state s', the update rule is similar to the Q-learning rule:

$$U(g, s') = (1 - \alpha_{sub}) * U(g, s') + \alpha_{sub} * (r_{sub} + \gamma_{sub} * max_s U(g, N_n(s')))$$

where $g \in G_n$, α_{sub} is the learning rate specific to the subgoal value function, and γ_{sub} the respective discount factor. To represent the distance to the subgoal we model the pseudo-reward as following:

$$r_{sub} = \begin{cases} -\epsilon & \text{if } s' \neq g \\ pr & \text{if } s' = g \end{cases}$$

where ϵ is a small positive value (we can remove this altogether, however it seems that the greedy policy on U is better behaved when using it), and pr is the pseudo-reward, a bigger positive value. This means that when wanting to reach a subgoal g it suffices to select the greedy policy with respect to $U(g, \cdot)$. This gives an easy way of constructing policies for reaching specific states (subgoals) in the environment. However, other methods could be employed, like imitation learning or inverse models (citations needed). Moreover, since we're using function approximation, i.e. a deep neural network for U, assuming we're dealing with an unseen goal, we can still start from a relatively informed policy of a similar subgoal. Now we can look at subgoals as abstract states, and now neighbours of subgoals are other subgoals, thus describing more succinctly the state space and creating a hierarchical structure for behaviour as well. In the same manner more levels of abstraction can be created, giving the following update rule for level i:

$$U(g_{i+1}, g_i) = (1 - \alpha_{sub}) * U(g_{i+1}, g_i) + \alpha_{sub} * (r_{sub} + \gamma_{sub} * max_{g'_i} U(g_{i+1}, N_1(g_i))$$

where $N_1(g)$ is the neighbour function now applied to the subgoal space, given by:

$$N_1(g) = \{g' \in \mathcal{G} | \forall g' \text{ with } (g, g') \in \mathcal{M}_A\}$$

where $\mathcal{M}_A$ is now the abstract memory of valid higher-level transitions between subgoals. We can use the neighbour function similarly as for states to compute dissimilarities between subgoals. Thus higher level subgoals can appear. In this way we connect high level abstract states such that the agent is able to have meaningful behaviour even in the absence of external reward. However, we need to be able to easily apply the learned behaviour also to external rewards, sparse as they might be. We can consider a state which receives reward as a subgoal state at the highest level, replacing the default r_{sub} with the external r. Having a small number of subgoals at the highest level means the agent will propagate the reward quickly through all subgoals, which in turn will propagate the rewards down the hierarchy, until the agent reinforces primitive states. Thus, an external reward will trigger an update at all levels of abstraction, so reinforcing subgoals on the path to the reward at all levels of the hierarchy. We see that a natural association between pseudo-rewards and external rewards occurs, with the external one pertaining to the highest level of abstraction.

3.4 Policies

After reaching a subgoal, the behavioural policy needs to take into account the recency of visited subgoals, with subgoals being visited less recently having higher probability of being the next target subgoal (we choose stochastic policies for improved exploration). We can embed this constraint by multiplying the value function of the subgoals with a positive recency weight w_{g_i}, such that $\sum_{i=1}^{t} w_{g_i} = 1$, and $w_{g_0} > w_{g_1} > ... > w_{g_t}$, where t is the timestep associated with the selection of the tth subgoal in the episode, an abstract timestep. We choose a simple rule, once a subgoal has been a target subgoal in the current episode, the probability that the respective subgoal will be selected again is divided by d (in our experiments $d = 2$). Let the subgoals in a trajectory τ be denoted by $\mathcal{G}_\tau = \{g_0, g_1, ..., g_t\}$. The probability that a subgoal g_i is selected in state s as target subgoal at timestep $t+1$ is given by:

$$p(g_i = g_{t+1}) \propto w_{g_i} \times U(g_i, s), \text{ and } w_{g_i} = \begin{cases} w_{g_i}/d, & \text{if } g_t = g_i \\ w_{g_i} + \epsilon, & \text{if } g_{t-j} = g_i, \forall 0 < j < t \\ w_{g_i} = w_{g_i}, & \text{if } g_i \notin \mathcal{G} \end{cases}$$

where ϵ is a small positive value. Sampling uniformly from the cdf of this empirical distribution gives at time step t a subgoal g_t. We observe that the index i refers to the subgoal index in the set $\mathcal{G}$, i.e. $g_i \in \mathcal{G}$, whereas t refers to the index of the subgoal in the current trajectory, so a time based index, i.e. $g_t \in \mathcal{G}_\tau$.

3.4.1 Update

For an efficient update to take place down to the primitive states, the agent holds in memory the current trajectory, in terms of subgoals and primitive states (this is a simpler form of eligibility) to update the whole chain once the external reward is received. Assuming the agent has discovered all levels of abstraction, and has just received an external reward, we show the update that happens in the hierarchy in Figure **??**. This is but a simple form of $n-step$ return, but where the generally sum of discounted rewards are replaced with a single step sparse reward, which in turn triggers an update for the step below. We can simplify the problem further, instead of iteratively updating more values for the different states, we can multiply the vector of values for a set of states (pertaining to a single subgoal) with a monotonic sequence of discounted rewards. This is for practical reasons, as we can update a whole set of states in one single update, whereas the iterative version needs n-steps, to achieve the same final effect. We can see this as having n-step backups (e.g. Sarsa n-step) but for each subgoal and each trajectory this n is different, the size of the actual trajectory steps that led to the subgoal.

3.4.2 Actions

Until now we have not even mentioned the actions. As we see, in our model the action selection is completely decoupled from value function learning and subgoal discovery. We consider the problem of action selection as a secondary, because of the limited number of actions we have available in each state. Knowing in which state we are, s, and in which state we want to be, s', we would like to find the action that will take us there, thus we want to find an action $a = T(s, s')$. If the environment is deterministic and has some kind of structure (e.g. up-down-left-right) we can easily learn action models. If not, the problem becomes more complex but can still be solved through trial-and-error. Assuming the number of actions is finite, not necessary small, we can see this problem as a classification problem, where the actions are the labels and the concatenation of s and s' constitutes the input. This is also known as learning an inverse model in RL(find citation).

4 Exploration

For the subgoal discovery part, we used We can use the representation given by the neighbour function as a simple but efficient exploration strategy. When looking at how the heterogeneity changes from one step to the next ($dh(s) = h_t(s) - h_{t-1}(s)$)we see that if the change is big in magnitude this means the neighbourhood which gives these values was not explored enough, so one should explore more in the direction of changing heterogeneity. One can look at this as a prediction error. If $h(s)$ does not change it means that the agent is predicting well the neighbouring states, if $h(s)$ does change it means the neighbouring states are still being discovered, and in general bigger changes suggest more exploration is needed. We thus form a new value function E for exploration which uses the change in

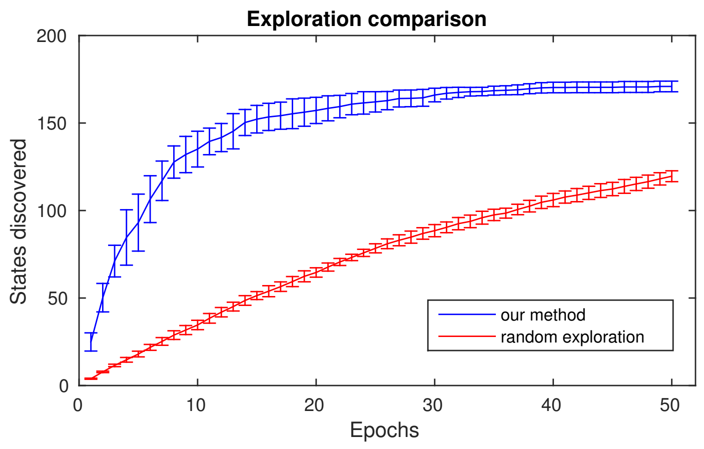

Figure 3: Comparison between random exploration and our method. See text for details.

heterogeneity as reward:

$$E(s) = (1-\alpha_e)E(s) + \alpha_e(dh(s) + \gamma_e \mathbb{E}[E(N_1(s)])$$
$$\text{with greedy next state selection in state s:}$$
$$s' = max_s E(N_1(s)), a = T(s, s')$$

We note here that in the case of exploration, a bigger n for the neighbour function $N_n(s)$ might be more beneficial, even though more expensive to compute. However, in practice, we notice a considerable difference between random exploration and using E with N_1. We show this in Figure 3. The plot shows means and standard deviations for 10 repetitions and 50 epochs. Each epoch has 100 steps, independent of the reward reached or not. In the implementation, the maximum number of states is 16×16 minus the number of states which represent walls (83 in this case), which is 173, and is achieved 4 out of 10 repetitions. Heterogeneity is computed once every 100 steps.

5 Experiments

5.1 4-room gridworld

5.2 Atari 2600

6 Conclusion

What our framework is doing is first to consider the problem of learning the structure of the environment independent of the reward. After subgoals have been discovered and policies are known for reaching different subgoals, once the external reward is received, a simple hierarchical propagation of the reward through the already learnt structure is sufficient to learn a policy that leads the agent to the reward. Action learning is decoupled from learning the structure of the environment and is learnt separately, which makes learning modular and more flexible. Besides our architectural choices, our main contribution is the state representation used, which is in terms of neighbouring states. The idea of neighbour function can then be used at all levels of the hierarchy to learn subgoals, which are in effect just states that have different properties in terms of how dissimilar their neighbours are, at multiple levels of the hierarchy. To our knowledge this is the first framework to provide a simple and efficient way to structure the environment into an arbitrary level hierarchy, enabling in the same time one-shot learning.

References

[Brunskill and Li, 2014] Brunskill, E. and Li, L. (2014). Pac-inspired option discovery in lifelong reinforcement learning. In *ICML*, pages 316–324.

[Hengst, 2004] Hengst, B. (2004). Model approximation for hexq hierarchical reinforcement learning. In *European Conference on Machine Learning*, pages 144–155. Springer.

[Kakade et al., 2003] Kakade, S. M. et al. (2003). *On the sample complexity of reinforcement learning*. PhD thesis, University of London.

[Kulkarni et al., 2016a] Kulkarni, T. D., Narasimhan, K. R., Saeedi, A., and Tenenbaum, J. B. (2016a). Hierarchical deep reinforcement learning: Integrating temporal abstraction and intrinsic motivation. *arXiv preprint arXiv:1604.06057*.

[Kulkarni et al., 2016b] Kulkarni, T. D., Saeedi, A., Gautam, S., and Gershman, S. J. (2016b). Deep successor reinforcement learning. *arXiv preprint arXiv:1606.02396*.

[Lakshminarayanan et al., 2016] Lakshminarayanan, A. S., Krishnamurthy, R., Kumar, P., and Ravindran, B. (2016). Option discovery in hierarchical reinforcement learning using spatio-temporal clustering. *arXiv preprint arXiv:1605.05359*.

[McGovern and Barto, 2001] McGovern, A. and Barto, A. G. (2001). Automatic discovery of subgoals in reinforcement learning using diverse density.

[Menache et al., 2002] Menache, I., Mannor, S., and Shimkin, N. (2002). Q-cutdynamic discovery of sub-goals in reinforcement learning. In *European Conference on Machine Learning*, pages 295–306. Springer.

[Oh et al., 2015] Oh, J., Guo, X., Lee, H., Lewis, R. L., and Singh, S. (2015). Action-conditional video prediction using deep networks in atari games. In *Advances in Neural Information Processing Systems*, pages 2863–2871.

[Schapiro et al., 2016] Schapiro, A. C., Turk-Browne, N. B., Norman, K. A., and Botvinick, M. M. (2016). Statistical learning of temporal community structure in the hippocampus. *Hippocampus*, 26(1):3–8.

[Shi and Malik, 2000] Shi, J. and Malik, J. (2000). Normalized cuts and image segmentation. *IEEE Transactions on pattern analysis and machine intelligence*, 22(8):888–905.

[Şimşek and Barreto, 2009] Şimşek, Ö. and Barreto, A. S. (2009). Skill characterization based on betweenness. In *Advances in neural information processing systems*, pages 1497–1504.

[Şimşek et al., 2005] Şimşek, Ö., Wolfe, A. P., and Barto, A. G. (2005). Identifying useful subgoals in reinforcement learning by local graph partitioning. In *Proceedings of the 22nd international conference on Machine learning*, pages 816–823. ACM.

[Stachenfeld et al., 2017] Stachenfeld, K. L., Botvinick, M. M., and Gershman, S. J. (2017). The hippocampus as a predictive map. *bioRxiv*, page 097170.