Using label propagation for learning temporally abstract actions in reinforcement learning

Pierre-Luc Bacon
School of Computer Science
McGill University
Montreal, QC, Canada

Doina Precup
School of Computer Science
McGill University
Montreal, QC, Canada
dprecup@cs.mcgill.ca

ABSTRACT

Temporal abstraction plays a key role in scaling up reinforcement learning algorithms. While learning and planning with given temporally extended actions has been well studied, the topic of how to construct this type of abstraction automatically from data is still open. We propose to use the label propagation algorithm for community detection in order to construct extended actions, within the framework of options. We illustrate the benefit of the approach in small computational experiments and discuss its relationship to existing methods for subgoal discovery.

1. INTRODUCTION

Reinforcement learning is an active area of machine learning research, which aims to describe how agents can learn actively and incrementally from interaction with their environment. A crucial aspect needed to scale up reinforcement learning algorithms is *abstraction*, i.e., building higher-level representations from the sensory stream that the agent receives, and from its (often low-level) actions. We focus on temporal abstraction, in which high-level controllers or behaviours are obtained by "chaining together" low-level actions, and we adopt the options framework [23]. By developing options, or *skills*, an agent can learn more efficiently by reusing prior knowledge, instead of continuously reasoning from scratch at the atomic scale of primitive actions. For example, a manipulator might have an option for reaching and opening a door knob using the sensory information from tactile sensors and the action set defined over the possible motor commands at each joint. By learning such an option and later using it as prior knowledge for some new (but related) task, the solution to the new task can be obtained much faster, and requires less new data (which could be expensive to acquire).

A lot of research has focused on developing learning and planning methods when options are given (e.g, in the form of subgoals that are worthwhile to achieve). However, a designer may not always be able to express easily such subgoals. Several papers have addressed the problem of learning subgoals from the agent's interaction with its environment, e.g. [20, 15, 21, 13, 27]. Most of these approaches rely on the idea of *bottleneck* or *access* states, which are important when an agent travels around its state space. Despite many papers on this topic, there is still no consensus on which method might be superior. One of the issues is that all methods require a fairly large amount of data to be acquired in order for the bottleneck state detection and the option construction to succeed. We are interested in developing methods which are more data-efficient, and which may even work in an incremental setting.

We approach this problem by considering the interaction graph discovered while the learning agent is actively exploring its environment. This is a graph in which different states of the agent are connected if transitions between these states were observed when the agent took some action. We use the network perspective to analyze the state interaction graph. The intuition is that "communities" of states, i.e. densely connected subsets of states linked together by sparse connections [14], provide natural initiation sets for options, and that states sitting "between" communities are natural subgoals. This definition is thus closely related to idea of *access states* from [1].

An approach similar in flavor is that of Simsek & Barto [20], which relies on the complete representation of the interaction graph to detect bottleneck states using the betweenness centrality measure. However, assuming that most effective solutions for a task might lie in a sparse subgraph of the state space, a complete representation of the interaction graph might be impossible to obtain or unnecessary for discovering good skills.

In contrast, we present an algorithm meant to operate while we monitor the interaction of an RL agent with its environment. We aim to reconstruct incrementally part of the interaction graph and extract useful skills from it. Our approach is based on a near-linear time label propagation algorithm for detecting community structures in networks [19]. This is an initial foray into the use of label propagation, and other algorithms could be used instead as well.

The paper is structured as follows. Section 2 reviews background, notation and related work. Section 3 develops the label propagation algorithm for option construction. In Section 4 we present an empirical illustration of the approach in a small discrete problem. In Section 5 we explain how the approach can be generalized to continuous state space and present preliminary evidence of its utility in this case. In Section 6 we conclude and discuss future work.

Appears in: *Proceedings of the Workshop on Multiagent Interaction Networks (MAIN 2013), held in conjunction with AAMAS 2013*, May 7, 2013, Saint Paul, Minnesota, USA.

2. BACKGROUND

2.1 Reinforcement learning

In reinforcement learning, an agent interacts with its environment at discrete time steps. At each time t, it receives a collection of sensations, which we call the state s_t, and based on this observation, it takes an action a_t. As a result, the environment provides a numerical reward signal r_{t+1}, a new state s_{t+1} is observed, and the process repeats. A standard assumption used to formalize this problem is that the environment is in fact a Markov Decision Process (MDP). An MDP is a tuple $\langle S, A, P, R, \gamma \rangle$ consisting of a set of states S, a set of primitive actions A, transition probabilities $P : S \times A \times, S \mapsto [0, 1]$, expected reward function $R : S \times A \mapsto \mathbb{R}$ and discount factor $\gamma \in [0, 1]$. The learning task for an RL agent is to learn a policy $\pi : S \mapsto A$ that dictates the choice of action in any given state so as to maximize the total reward received over time. We consider the discounted reward criterion, in which the action-value function for a given policy π, $Q^\pi : S \times A \to \mathbb{R}$ is defined as:

$$Q^\pi(s, a) = \mathbb{E}(r_{t+1} + \gamma r_{t+2} + ...|s_t = s, a_t = a) \quad (1)$$

In any discrete MDP, there exists an optimal action-value function, which satisfies the following set of Bellman equations:

$$Q^*(s, a) = R(s, a) + \gamma \sum_{s\prime \in S} P(s\prime|s, a) \max_{b \in A} Q^*(s\prime, b), \forall s \in S, a \in A \quad (2)$$

A widely used algorithm for learning an action-value function is Q-Learning [25], which has been shown to converge to the optimal action-value function when actions are sampled infinitely often in every state [6]. Q-Learning is an incremental dynamic programming algorithm that successively improves the estimate on the action-value function by computing:

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha(r_{t+1} + \gamma \max_{a\prime \in A} Q(s_{t+1}, a\prime) - Q(s_t, a_t)) \quad (3)$$

where α is the learning rate and γ is the discount factor (which determines the extent to which the agent should strive for long term reward or act myopically).

2.2 Options Framework

Options [23] are a popular way of defining temporally extended actions in reinforcement learning.

Since we adopted the MDP assumption, we use Markov options. A Markov option o consists of a tuple $\langle \mathcal{I}, \pi, \beta \rangle$ where the *initiation set* $\mathcal{I} \subseteq S$ specifies in which states the option can be invoked, $\pi : S \mapsto O$ is a sub-policy defined for o and $\beta : S \mapsto [0, 1]$ defines the probability of terminating the execution of π in any given state. The primitive action set A can now be replaced with a set of options. Note that a primitive action a can be redefined as an option by setting $\beta(s) = 1 \; \forall s \in S$, making the corresponding option available everywhere a can be invoked: $\mathcal{I} = \{s : a \in \mathcal{A}_s\}$ and finally, having a sub-policy which selects a for every state: $\pi(s, a) = a \; \forall s \in S$

In [23] the authors show that and MDP together with a set of options O forms a Semi-Markov Decision Process (SMDP) $\langle S, O, \gamma, R_O, P_O \rangle$, where S and γ are as above, and the reward function $R_O : S \times O \mapsto \mathbb{R}$ and transition sub-probability function $P_O : S \times O \times S \mapsto [0, 1]$ are computed from R, P and the definitions of the options in O. An option-value function can be defined in this SMDP as:

$$Q^\mu(s, o) = \mathbb{E}\{r_{t+1} + \gamma r_{t+2} + \gamma^2 r_{t+3} + ...|\mathbb{E}(o\mu, s, t)\} \quad (4)$$

In equation (4), $o\mu$ corresponds to choosing option o, executing its sub-policy until termination, and then choosing another options according to the main policy over options μ. Q-Learning can also be extended to SMDPs and options, and the options-value function is learnt incrementally with the following update rule:

$$Q(s, o) = Q(s, o) + \alpha(r + \gamma^k \max_{a \in O} Q(s\prime, a) - Q(s, o)) \quad (5)$$

The update above is only evaluated after the option has terminated. In contrast, intra-option learning [22] can take place throughout the execution of an option. It leads to more efficient use of experience, which can also be shared simultaneously among all options that are consistent with it.

2.3 Related Work

We now review some approaches for automatically constructing options (or skills) from data. The idea of *bottleneck* regions of the interaction graph has been instrumental in the development of most of these techniques. The canonical example of bottlenecks can be found in a navigation task within an environment with multiple-rooms, where subgoals naturally corresponds to the doorways connecting two rooms (i.e. regions of the state space). Bottlenecks have been defined as those states which appear frequently on successful trajectories to a goal but not on unsuccessful ones [13, 21] or as nodes which allow for densely connected regions of the interaction graph to reach other such regions [15, 27, 7]. Bottleneck states are found using graph centrality measures in [20, 18] and capture most of the properties outlined above by defining important nodes as those which lie frequently on shortest paths between states.

It appears that all of these approaches can also be categorized on the basis of whether the total reward obtained on a trajectory is taken into account in the identification of useful subgoals. Solutions that consider only the structural properties of the interaction graph assume that the state-space dynamics is the only determining factor in defining options. The other class of approaches treats both dynamics and rewards, accounting for the fact that bottlenecks of the state-space dynamics do not necessarily correspond to desirable subgoals to develop for a task.

The idea of sampling trajectories while the agent is interacting with the environment is central to most approaches. One drawback of this class of methods is that primitive actions might not suffice to attain the task goal or yield to successful (defined in a domain-dependent way) trajectories. Hence, these methods can require large amounts of data, possibly too expensive to acquire. Furthermore, useful solutions might lie within sparse subgraphs of the entire interaction graph (a similar intuition behind manifold embedding techniques of machine learning).

McGovern & Barto [13] formulate the problem of finding subgoals as a multiple-instance learning problem over bag of feature vectors collected by interacting with the environment. Two sets of bags are obtained in this way, from observations collected along successful and unsuccessful trajectories respectively. The notion of diverse density is then applied either by exhaustive search or gradient descent to

find regions of the feature space with the most positive instances and the least negative ones. Because negative bags must only contain unsuccessful trajectories, while positive bags have to contain at least one successful feature vector, this method is sensitive to noise.

In [1], the authors identify subgoals by looking for *access states* which lead to the regions of the state space that have not been visited recently and trying to capture the notion of *relative novelty* The time frame within which to search for such novelty events is a parameter of this algorithm and influences the results obtained.

The graph-cut algorithm proposed in [15] tries to identify access states of those densely connected regions linked together through a few important edges. Wolfe & Barto [27] apply the same graph-cut partitioning but only over local transition graphs obtained from the sampled trajectories. A similar approach of finding strongly connected components (SCC) of the interaction graph is adopted in [7]. Since most MDP problems exhibit ergodicity, the applicability of this algorithm seems limited. Ergodic chains would then lead to large components being detected by SCC.

By representing the entire interaction graph, Simsek & Barto [20] compute betweenness centrality to identify important subgoals. They also propose a potential incremental formulation of their algorithm that finds local maxima of betweenness within the subgraph obtained by collecting short trajectories. However, it has also been argued [18] that the centrality measure called *connection graph stability* leads to better subgoal identification than betweenness or closeness centrality.

The problem of subgoal discovery has only been recently considered under the formulation of graph clustering. Mathew & al. [12] also build an interaction graph out of which clusters of nodes are found using the PCCA+ [26] algorithm. The concept of metastability motivates this approach and tries to find regions of the state space in which a dynamical system spends more time. PCCA+ is a spectral clustering technique which comes at a higher cost than the approach proposed here.

3. SKILLS DISCOVERY

We base our definition of subgoals on the intuitive notions of access states from [1, 27]. Such states are found to mediate access to other regions of the graph. More precisely, we posit that they lie on paths connecting densely connected regions with sparse connections between them. We present a skill discovery algorithm based on label propagation (LPA) for detecting community structures. An overview of the skills detection algorithms based on centrality measures is also presented for comparison purposes with LPA in the next section.

3.1 Community Structures

Communities are groups of nodes (also known as *modules*) with dense connections within the group but sparse links to other groups. The algorithms presented in [16] iteratively remove edges based on their measure of *betweenness*, defined as the relative number of all the shortest paths going through a given edge. Edges with the highest values are removed first and betweenness is recomputed after every deletion.

The notion of *modularity* is introduced [16] to measure the quality of the current partitioning of a network into communities and it is used to define the termination condition. Modularity is defined as the difference between the fraction of the edges belonging to communities and their expected number in a network which would have the same community partitioning but with random edges assignment. Hence, a modularity value of 0 corresponds to a community partitioning which does not differ from random networks, whereas a value of 1 indicates the strongest presence of community structures. While splitting the network into smaller communities, a sudden increase in the modularity is considered as an indicator that an optimum might have been attained. Simulated Annealing (SA) was applied in [14] as a global optimization method to find such optima. Unfortunately, even with Brandes' algorithm [2] for computing betweenness centrality in $O(nm)$, this community detection technique remains prohibitively expensive. In particular, this complexity is not well-suited for the amounts of data that would be acquired by an agent during its environment interactions.

3.2 Label Propagation Algorithm (LPA)

Raghavan et al. [19] introduced a near-linear time algorithm for finding community structures that does not rely on any prior knowledge about the networks nor any definition of an objective function for global optimization purposes. This algorithm starts by assigning unique initial community labels to every node. At each iteration, neighboring nodes try to reach a consensus on their labels. LPA can be summarized in the following steps:

1. Assign an initial unique label to every vertices in the graph G
2. Arrange the vertices in a random sequence X
3. For every vertex $v \in X$, assign it the label which appears the most frequently in the neighborhood of v, breaking ties uniformly at random.
4. Repeat steps 2 and 3 as long as there are vertices whose labels differ from the most frequent label in their neighborhood.

Under certain graph structures, such as bipartite graphs, LPA might not converge to a stable solution and a termination condition would have to be defined arbitrarily. We will not handle this issue in this paper, but rather concentrate on the problem of updating the labelling efficiently. By collecting trajectories from an RL agent, the resulting interaction graph will be continuously altered: edges will be added and others will be removed if deemed unnecessary. We would thus want to update the current labelling online in an incremental fashion. [17] extends LPA by labelling nodes with a pair that consists of a time and a *label* components. It then proceeds to local updates in a similar way to LPA:

1. Given a new edge, label its endpoints u and v as (t, u) and (t, v) where t is the current time
2. Add u and v to a local set X_l together with their neighbors
3. Shuffle X_l and apply the same label assignment rule as LPA except that ties are resolved based on the label with the **largest** t.
4. Add all node that have changed their label to a new set X_l.

5. Repeat steps 2 to 4 until a consensus is reached

Updating the labels after an edge has been added is $O(m_l + n_l)$ where m_l is the number of vertices in the set X_l and n_l, the edges.

3.3 Using Weight Information

While collecting trajectories, it can be useful to set the weight on an edge, e.g. as the number of times a transition has been experienced over it. This allows us not only to prune the interaction graph from possibly irrelevant edges (and thus lower the computation burden) but also to bias the label propagation algorithm towards those nodes with the highest importance in terms of weight. We modified the label selection criterion of LPA such that the label occurring with the *most cumulative weight* among a node's neighbors is the one that gets assigned. Note that this is a strict generalization of the previous approach, which can be rephrased as all weights being equal to 1.

3.4 Finding Subgoals within Communities

The set of states within a community naturally fits the idea of the initiation set component in the options framework. It is however less obvious to come up with a precise definition of which node within a community should be a subgoal. We use a similar approach as [7] and try to find those *edge points* that have at least one outgoing edge to another community of a significant size. In the aforementioned paper, the authors make the assumption that community sizes are normally distributed and only consider those which are least $Pr(X < \mu + 2\sigma) \approx 0.95$. Since a community can also have multiple such edge points, we decided to only use the one with the highest weight to another community.

3.5 Centrality Measures

For comparison with our LPA algorithm, we also used some graph centrality techniques for options detection [20, 18]

$$C_B(v) = \sum_{s \neq t \neq} \frac{\sigma_{st}(v)}{\sigma_{st}} \tag{6}$$

$$C_C(v) = \frac{1}{\sum_{t \in V} d_G(v,t)} \tag{7}$$

Betweenness centrality (eq. 6) is defined as the fraction of the shortest paths σ_{st} between any possible pair passing through a vertex v. Naive computation of betweenness centrality would be $\Theta(n^3)$ but [2] showed that the complexity be can reduced to $O(nm)$ or $O(nm + n^2 \log n)$ in the unweigthed and weighted case respectively. Closeness centrality (eq. 7), corresponds to the inverse of the distances from a given vertex to every other vertex in the graph.

We use the scoring measure of [18] in order discover maxima of either betweenness or closeness centrality:

$$S(v) = C(v) \left(\frac{C(v)}{\max_{u \in N(v)}(C(v))} \right)^2 \tag{8}$$

In equation (8), $C(v)$ is either the betweenness or closeness centrality and $N(v)$ is the set of neighbors for node v. We set a threshold on the minimum score that a node must achieve comparatively to the scores computed for the other nodes. For example, we might require that the score should be 97% larger than for any other node.

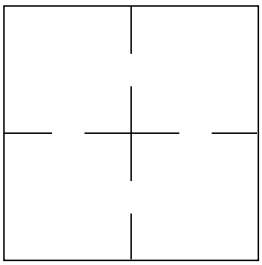

Figure 1: The domain that we used has four 5×5 rooms connected using four doors. The corresponding state space for this problem has 103 states.

4. ILLUSTRATION

As a proof of concept, we tested our algorithm in a grid world environment, similarly to many other papers on option construction. The environment, depicted in Figure 4, consists of four rooms in which the agent must navigate while avoiding collisions with un-penetrable walls. Every move incurs a penalty of -1, while reaching the goal results in a positive reward of 50. The environment is stochastic with transition probabilities set to 0.7.

The graph discovery procedure that we then applied over this domain was the following:

1. Collect trajectories for 100 episodes for a primitive Q-Learning agent with an epsilon greedy exploration strategy.
2. Build the transition graph
3. Run weighted label propagation
4. Identify subgoals
5. Create options

We used our automatic detection algorithm to find potential subgoals in the interaction graph obtained from a hundred trajectories of a regular Q-learning agent. The resulting subgoals and communities are depicted in Figure 4. Q-Learning for options (eq. 5) was then executed for another hundred episodes in order to learn the options' subpolicies. The initiation set $\mathcal{I}$ for an option was defined as the states within a community and the termination probabilities specified by β were set to 1 at the subgoal and 0 everywhere else. We compared the performance of our LPA algorithm against skill discovery methods based on betweenness and closeness centrality. Skills based on centrality methods were selected using different thresholds: 0.90, 0.92, 0.95, and 0.97. The performance of 1000 independent agents for each set of options was averaged over 100 episodes.

As shown in Figure 4, the options discovered by our LPA-based algorithm are useful enough to reduce the number of steps needed to attain the goal compared to the performance of an agent using only a flat policy over primitive actions. However, our algorithm is outperformed by the centrality-based approaches. We think that this could be explained by the simple nature of the problem domain, for which the 9 communities detected by LPA might lead to an overly complicated solution. Due to the small number of states in this domain, the computation time of the two approaches is

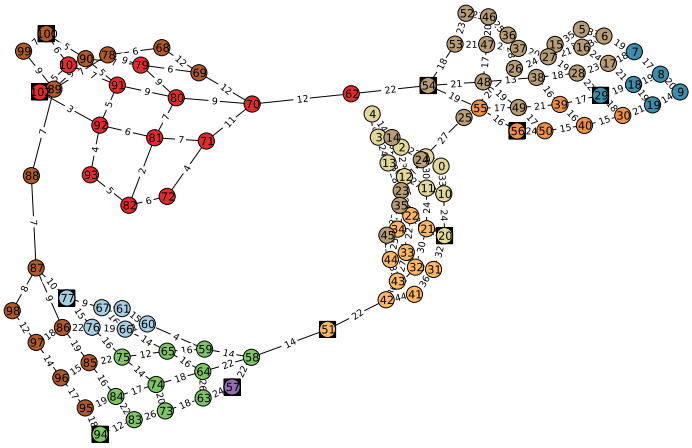

Figure 2: Nine communities were detected by LPA when the subgoal detection threshold was set to 0.68 (one point of standard deviation) and edges with only one transition were removed. A spring layout was used to draw the graph. The subgoals are highlighted in black.

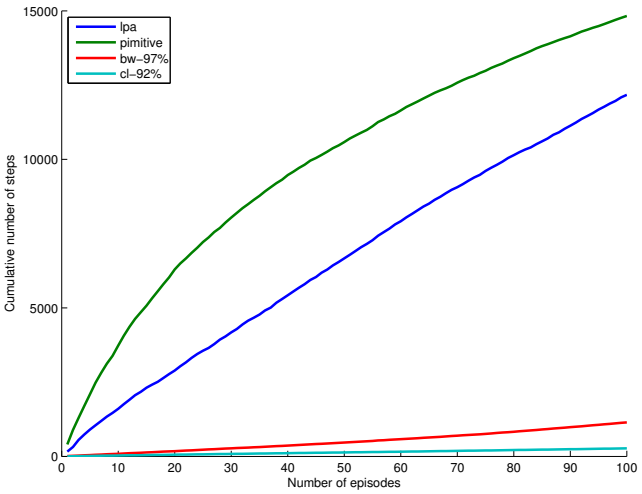

Figure 4: Our label propagation algorithm for skills discovery leads to shorter trajectories to the goal than a primitive agent but is outperformed by other approaches based on betweenness or closeness centrality.

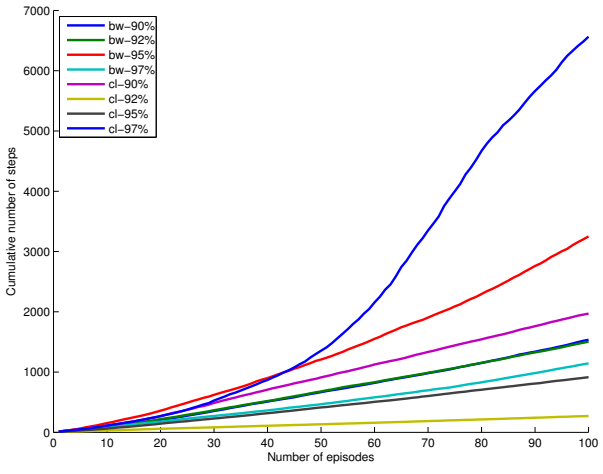

Figure 3: Cumulative number of steps as a function of the number of episodes for RL agents with behavior policies based on betweenness and closeness centrality. The set of options obtained by selecting only the subgoals corresponding to maxima of closeness centrality under a threshold of 0.92 seems to produce the most efficient behavior; the worst is obtained for betweenness at 0.90.

very similar. The scalability of LPA will however be better appreciated in the next experiment.

5. SCALING TO CONTINUOUS MARKOV DECISION PROCESSES

We conducted a preliminary experiment to assess the scalability of our skill discovery algorithm in a continuous state-space. Transitioning from the discrete world to the continuous one, however, requires a new graph construction approach. Furthermore, the identified subgoal states and initialization sets must be generalized to regions of the continuous state space.

We chose to apply the continuous extension of our community detection approach to the Pinball ball domain described in [8]. Pinball is an episodic 4-dimensional domain in which the agent must navigate a ball to the goal through arbitrarily shaped obstacles. Since collisions do not incur a penalty, an agent can strategically bounce the ball off the obstacles to reach the goal more efficiently. The complex dynamics resulting from elastic collisions make this domain difficult.

For efficiency reasons, we decided to adopt a biased sampling strategy by collecting trajectories of a learning agent. We used an RL-Glue [24] implementation of a Sarsa(λ) agent with Fourier bases provided by the main author of [9]. Once the dataset of trajectories is collected, there remains the problem of defining neighboring relationships in the graph. For a single trajectory, one could simply use the temporal ordering of states within the trajectory as the graph structure. However, in the continuous setting, this would likely result in a disconnected graph when trying to merge multiple trajectories, as the same state may never be encountered twice. We thus use the graph construction approach of [11] connecting each node to its k-nearest neighbors (according to the Euclidean distance) and setting the edge weights to

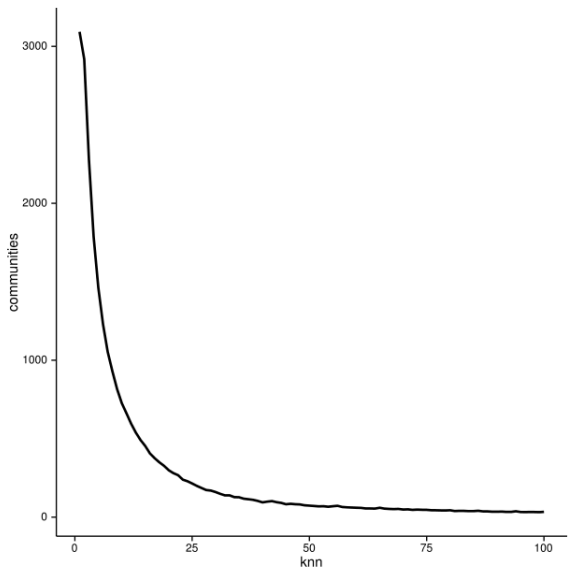

Figure 5: Number of communities detected as a function of the number of nearest neighbors used in the graph construction.

$W(i,j) = e^{-\frac{||x_i - x_j||_{\mathbb{R}^n}^2}{\sigma}}$.

Our preliminary results with the igraph [4] implementation of LPA can easily handle (within a few seconds) the above graph construction over 30000 nodes. The number of communities found is however quite high, even after having applied the aggregation procedure described in [19]. Increasing the number of nearest neighbors during the graph construction phase has a very strong effect on the number of communities detected, as shown in Figure 5. As expected, using more neighbors creates denser areas of the graph, resulting in fewer and larger communities being detected. The shape of the graph suggests that k might need to be chosen based on the desired number of options one would need. Different graph construction methods may be worth exploring here as well.

In the current version of this experiment, we have not yet completed the step of taking these communities and generating actual subgoals from them. As in the discrete case, we could identify subgoals from the state-space graph by considering the nodes lying on the *boundary* of a community *i.e.* the nodes adjacent to other nodes of a different community. But since none of them will be exactly encountered again by a learning agent, we need to devise a proper state abstraction mechanism for generating subgoals. We envision three possible approaches:

Feature Construction
: A discretization scheme could be obtained by partitioning the state-space using kd-trees for example. More recently, Random Projection Trees (RP-Trees) [5] have been been shown to scale better in higher dimensions while naturally adapting to the intrinsic low-dimensionality of the data. Density estimation methods could also help for this problem. We could, for example, put a multivariate Gaussian over each identified subgoal.

Non-parametric Clustering
: Within a community, we could apply a non-parametric clustering algorithm to obtain a partition between subgoals and non-subgoals states. Given new observations encountered by a learning agent, termination could be decided based on the distance to the closest cluster.

Classifier Learning
: A classification method (eg. SVM, decision tree, logistic regression) could be learned for every option to discriminate its subgoals from other states. This idea has already been explored in earlier work [8] for defining the initiation set of an option.

Since our approach to option discovery relies on a graph representation of the state-space, it could subsequently be used for learning a control policy using Representation Policy Iteration (RPI) [11] based on Least-Squares Policy Iteration (LSPI) [10]. Note that one only has to form the Laplacian of the community and compute its eigenvectors to obtain Proto-Value Function (PVF) bases. This idea has already been exploited in the metastability framework of [12]. Due to the possibly large size of the subgraphs, computing the eigenvectors could turn out to be too expensive and any other suitable learning algorithm could be used instead.

As for the definition of the initialization set, the problem is closely related to the one of recognizing subgoal states and the same set of proposed approaches could be attempted. Proper definition of the initialization set is often neglected in most of the work about options, which are often assumed to be available everywhere. However, we think that the concept of communities offers a natural definition and it is worthwhile to leverage it.

6. DISCUSSION AND FUTURE WORK

The community detection approach to option construction presented in this paper has the advantage of providing a clear definition for what the initiation set for an option should be: any member of a community should also belong to the initiation set of the corresponding option. On the other hand, the question of identifying subgoals within those regions has a less definite answer. We think an important reason which explain the observed results has to do with the heuristic that we designed for selecting a subgoal within the multiple edge points of a community. This definition might have to be revised in future work.

As expected, because of the stochastic nature of the algorithm, we observed instability in the solutions returned by LPA. The extension to label propagation proposed in [3] shows that by using graph coloring for determining the order of label updates, stability and convergence can be guaranteed. If it could be formulated incrementally, semi-synchronous label propagation could be a useful improvement for our application.

The four-rooms domain was used to compare the performance of our algorithm against other skill detection techniques. We think it would be crucial to investigate more complex problem domains, which might yield a different performance profile. It is reasonable to believe that an autonomous skill detection system could make use of a combination of different subgoals identification techniques over this same community detection idea.

Finally, an important critique of our approach, but also to the broader class to which it belongs, is that we make use

of only the structural properties of the state space. However, we note that reward or value information could easily be incorporated in the definition of the weights used by the algorithm. For example, the weights could be based on the values of the neighboring states, rather than on the frequency of the transitions. This could lead to interesting state abstractions, in which states of similar current valuation would be grouped together. We leave the exploration of such an approach for future work. Our immediate plan is to complete the current version of the algorithm and investigate its theoretical properties in the continuous state space.

7. REFERENCES

[1] A. G. Barto. Using Relative Novelty to Identify Useful Temporal Abstractions in Reinforcement Learning. *Learning*, 2002.

[2] U. Brandes. A faster algorithm for betweenness centrality. *Journal of Mathematical Sociology*, 25(1994):163–177, 2001.

[3] G. Cordasco and L. Gargano. Community detection via semi-synchronous label propagation algorithms. In *Business Applications of Social Network Analysis (BASNA), 2010 IEEE International Workshop on*, pages 1 –8, 2010.

[4] G. Csardi and T. Nepusz. The igraph software package for complex network research. *International Journal of Complex Systems*, 2006.

[5] S. Dasgupta and Y. Freund. Random projection trees and low dimensional manifolds. In *STOC*, 2008.

[6] P. Dayan. Technical Note Q , -Learning. *Machine Learning*, 292(3):279–292, 1992.

[7] S. J. Kazemitabar and H. Beigy. Using Strongly Connected Components as a Basis for Autonomous Skill Acquisition in. In *Proceedings of the 6th International Symposium on Neural Networks on Advances in Neural Networks*, pages 794–803, 2009.

[8] G. Konidaris and A. G. Barto. Skill discovery in continuous reinforcement learning domains using skill chaining. In *NIPS*, volume 22, pages 1015–1023, 2009.

[9] G. Konidaris, S. Osentoski, and P. Thomas. Value Function Approximation in Reinforcement Learning using the Fourier Basis. In *AAAI*, pages 380–385, 2011.

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

[11] S. Mahadevan. Proto-value Functions : A Laplacian Framework for Learning Representation and Control in Markov Decision Processes. *Journal of Machine Learning Research*, 8:2169–2231, 2007.

[12] V. Mathew, K. Peeyush, and B. Ravindran. Abstraction in Reinforcement Learning in Terms of Metastability. In *EWRL*, pages 1–14, 2012.

[13] A. McGovern and A. G. Barto. Automatic Discovery of Subgoals in Reinforcement Learning using Diverse Density. *Machine Learning*, 2001.

[14] A. Medus, G. Acuña, and C. Dorso. Detection of community structures in networks via global optimization. *Physica A: Statistical Mechanics and its Applications*, 358(2-4):593–604, 2005.

[15] I. Menache, S. Mannor, and N. Shimkin. Q-Cut - Dynamic Discovery of Sub-Goals in Reinforcement Learning. In *Proceedings of the 13th European Conference on Machine Learning*, pages 295—-306. Springer-Verlag, 2002.

[16] M. Newman and M. Girvan. Finding and evaluating community structure in networks. *Physical Review E*, 69(2):1–15, 2004.

[17] S. Pang, C. Chen, and T. Wei. A Realtime Clique Detection Algorithm: Time-Based Incremental Label Propagation. *2009 Third International Symposium on Intelligent Information Technology Application*, pages 459–462, 2009.

[18] A. A. Rad and M. Hasler. Automatic Skill Acquisition in Reinforcement Learning using Connection Graph Stability Centrality. *Sciences-New York*, pages 697–700, 2010.

[19] U. Raghavan, R. Albert, and S. Koumara. Near linear time algorithm to detect community structures in large-scale networks. *Physical Review E*, pages 1–12, 2007.

[20] O. Simsek and A. G. Barto. Skill characterization based on betweenness. In *NIPS*, pages 1–8, 2009.

[21] M. Stolle and D. Precup. Learning options in reinforcement learning. In *Symposium on Abstraction, Reformulation, and Approximation*, pages 212–223. Springer, 2002.

[22] R. Sutton, D. Precup, and S. Singh. Intra-option learning about temporally abstract actions. In *ICML*, pages 556–564, 1998.

[23] R. S. Sutton, D. Precup, and S. Singh. Between MDPs and semi-MDPs: A framework for temporal abstraction in reinforcement learning. *Artificial Intelligence*, 112(1-2):181–211, 1999.

[24] B. Tanner and A. White. RL-Glue : Language-Independent Software for Reinforcement-Learning Experiments. *Journal of Machine Learning Research*, 10:2133–2136, 2009.

[25] C. Watkins. *Learning from Delayed Rewards*. PhD thesis, Cambridge University, England, 1989.

[26] M. Weber, W. Rungsarityotin, and A. Schliep. Perron Cluster Analysis and Its Connection to Graph Partitioning for Noisy Data. Technical Report November, Zuse Institute Berlin ZIB, Berlin, 2004.

[27] A. P. Wolfe and A. G. Barto. Identifying useful subgoals in reinforcement learning by local graph partitioning. In *ICML*, pages 816–823, 2005.