Learning Options from Demonstration using Skill Segmentation

Matthew Cockcroft[†], Shahil Mawjee[†], Steven James, and Pravesh Ranchod
School of Computer Science and Applied Mathematics
University of the Witwatersrand
Johannesburg, South Africa
{matthew.cockcroft1, shahil.mawjee}@students.wits.ac.za, {steven.james, pravesh.ranchod}@wits.ac.za

***Abstract*—We present a method for learning options from segmented demonstration trajectories. The trajectories are first segmented into skills using nonparametric Bayesian clustering and a reward function for each segment is then learned using inverse reinforcement learning. From this, a set of inferred trajectories for the demonstration are generated. Option initiation sets and termination conditions are learned from these trajectories using the one-class support vector machine clustering algorithm. We demonstrate our method in the four rooms domain, where an agent is able to autonomously discover usable options from human demonstration. Our results show that these inferred options can then be used to improve learning and planning.**

***Index Terms*—hierarchical reinforcement learning, inverse reinforcement learning, options discovery, skill acquisition**

I. Introduction

Humans often execute tasks in an autonomous manner, without actually focusing on the individual actions they perform. Consider the act of driving a car—the person will enter the car, drive and then arrive at their destination, but while driving they do not contemplate each decision, such as turning, braking or changing gear. Instead, the person only checks whether or not they have arrived at their destination.

Hierarchical reinforcement learning encapsulates this idea for agents acting in an environment. This is achieved through action abstraction, transforming low-level actions into higher-level skills. A skill may be composed of many different actions and consists of a particular endpoint or termination state. Using this approach, an agent executing a specific skill only has to check whether it has reached such a termination state, instead of considering which action to take at each time step.

Before action abstraction can be performed, an agent must first learn which low-level actions to take. One method which has proved successful for teaching agents in complex domains is that of Learning from Demonstration (LfD), which provides the agent with demonstrations created by a human expert from which to learn. The agent then makes use of a process known as inverse reinforcement learning (IRL) [1] to learn a policy that best describes the expert's demonstration.

[†]Authors contributed equally.

This work is based on the research supported in part by the National Research Foundation of South Africa (Grant Number: 17808).

One issue with this approach is that the learned policy describes the entire demonstration and has no context or transferability outside of the original problem domain. Recent work has attempted to decompose into sets of skills. Most of this work has focused on representing skills as methods which attempt to reach a subgoal state [8], but these approaches often struggle to encapsulate more complicated reward functions.

One method which overcomes this is nonparametric Bayesian reward segmentation (NPBRS) [12]. This approach uses nonparametric Bayesian statistics to propose segmentations and maximum entropy IRL [19] to learn the reward functions associated with these segmentations. Another benefit of this method is that it does not require the number of skills to be known, but rather is able to infer them from the data.

While this method has proven to be successful in segmenting skills from demonstrations, the policies and trajectories generated by NPBRS are not leveraged in any way. We propose a method which uses these policies and trajectories so that high-level skills can be autonomously acquired by an agent and leveraged to improve learning.

To transform low-level actions into higher-level skills we make use of the options framework [15]. An option is a temporally extended action, so it may take more than one time step to complete. Each option consists of an initiation set, a policy and a termination condition. To learn options, our method makes use of the one-class support vector machine clustering algorithm [13], by classifying the start and endpoints of the trajectories segmented by NPBRS. These classifications provide an initiation set and a termination set, from which a termination condition can be generated. These are combined with the policies produced by NPBRS to create options.

The method is tested in the four rooms domain used in the original presentation of the options framework [15]. We show that an agent in this domain is able to autonomously discover usable options from human demonstration, and is able to use these options to accelerate learning dramatically. We explain the benefits of this method with regards to planning, in that it greatly reduces the size of an agent's decision tree.

II. Background

A. Reinforcement Learning

Reinforcement learning defines a set of algorithms which aim to solve problems by maximising the reward obtained in a particular situation. These problems are usually modelled as a Markov Decision Process (MDP) [14]. An MDP is a tuple defined as $M = (S, A, P, R, \gamma)$, where S is a set of states, A is a set of actions, $P(s'|s, a)$ is the probability that a transition to state s' will result from taking action a in state s, $R(s, a, s')$ is the reward obtained when taking action a in state s and transitioning to state s', and $\gamma \in [0, 1]$ is the discount factor.

The goal of reinforcement learning is to find an action policy that maximises the cumulative expected reward. An action policy is a function which gives the probability of an agent in state s choosing to take action a, and is denoted by $\pi(a|s)$. Due to the Markov Property, a transition from one state to the next only depends on the action taken in the current state.

B. Options Framework

While any task can theoretically be solved using only low-level actions, in practice many learning tasks are intractable in this space. Action abstraction, which combines low-level actions into higher-level skills, can be used to reduce the learning time. A commonly-used approach for this is the options framework [15]. An option is a temporally-extended action made up of the original low-level actions.

An option o is formally defined as a tuple (I_o, π_o, β_o), where $I_o \subseteq S$ is an initiation set of all states from which option o can be initiated, $\pi_o : S \times A \to [0, 1]$ is a policy giving the probability of option o executing each action in each state, and $\beta_o : S \to [0, 1]$ is the termination condition, which gives the probability of the option terminating in a particular state.

A method for creating a new option needs to determine how to learn its policy and how to define its initiation set and termination condition. This is achieved by identifying states in which the option will terminate once reached, and by defining the initiation set as the set of states from which that option would be useful to execute.

C. Inverse Reinforcement Learning

Inverse reinforcement learning (IRL) [1] defines a set of algorithms used to determine a reward function from a given demonstration. IRL algorithms consider all of the information from an MDP excluding the reward function, defined as an MDP$\backslash R$. Together with this they also require a set of demonstration trajectories, defined as $\zeta = \{(s_1, a_1), (s_2, a_2), ..., (s_n, a_n)\}$, where each pair (s_i, a_i) indicates the action a_i taken by the demonstrator while in state s_i. The algorithm then attempts to find the reward function R that is most likely to have produced ζ by an agent attempting to solve the given MDP (S, A, P, R, γ).

We consider the maximum entropy inverse reinforcement learning algorithm [19]. This algorithm builds upon the initial approach to IRL [1], which maps rewards to features within the states so as to reflect the importance of those features to the expert presenting the demonstration. The problem is that IRL is ill-posed, as it is possible for many reward functions to map to the same feature counts.

Maximum entropy IRL instead focuses on the entire distribution of possible behaviours and considers a set of trajectories, ζ. Using MDPs as previously defined, maximum entropy IRL additionally defines $\mathbf{f_s} \in \mathbf{R^k}$ as the feature vector of the state s, and $\theta \in \mathbf{R^k}$ as the reward weights [19].

The algorithm uses the principle of maximum entropy [7] to assert that the most likely distribution is one that does not display any preferences not implied by the feature counts. This is achieved by the relation $P(\zeta) \propto e^{R(\zeta)}$, where $P(\zeta)$ is the probability of trajectory ζ occurring. In this model, paths with higher rewards are exponentially preferred over paths with lower rewards. This is used to generate a distribution over paths, given by $P(\zeta|\theta, P)$, where P is the transition distribution. By maximising the entropy of the distribution over paths a convex function is obtained, for which its maxima can be calculated using gradient-based optimization.

At the maxima, the gradient is 0, and so the feature counts match. This guarantees that the agent performs equivalently to the expert's demonstration.

D. Nonparametric Bayesian Reward Segmentation

Bayesian statistical inference is the process of calculating a posterior distribution by using a given prior probability distribution for an unknown parameter and a calculated likelihood for that distribution. Nonparametric models [6] are probability models that may have infinitely many parameters, while stochastic processes, such as the Beta process, are examples of prior distributions when dealing with a nonparametric Bayesian model.

Nonparametric Bayesian reward segmentation (NPBRS) [12] is a method proposed for combining nonparametric Bayesian statistics for segmentation and IRL for policy learning. The maximum entropy algorithm is used to find the option policies that represent skills and the Beta-Process Autoregressive Hidden Markov Model (BP-AR-HMM) [4] is used to determine how to segment the demonstration for skill extraction.

A Hidden Markov Model is an MDP with hidden states, known as modes. The benefit of using the BP-AR-HMM is that the number of hidden modes does not need to be known as a beta process prior is placed on the sequence of modes. Instead of specifying the appropriate number of modes, they can then be inferred directly from the data. The model is also autoregressive, meaning that for continuous observations a mode-specific Vector Autoregressive (VAR) process can be used to describe temporal dependencies. The generative model for the BP-AR-HMM [4] is given as follows:

$$\begin{aligned}
B \mid B_0 &\sim BP(1, \beta) \\
X_i \mid B &\sim BeP(B) \\
\pi_j^{(i)} | f_i, \gamma, \kappa &\sim Dir([\gamma, ..., \gamma + \kappa, \gamma, ...] \otimes f_i) \\
z_t^{(i)} &\sim \pi^{(i)}_{z_{t-1}^{(i)}} \\
y_t^{(i)} &= \sum_{j=1}^{r} A_{j, z_t^{(i)}} y_{t-j}^{(i)} + e_t^{(i)}(z_t^{(i)})
\end{aligned}$$

B is drawn from a Beta Process (BP) and provides a set of global probabilities for each skill. This probabilities are used to produce a Bernoulli Process (BeP), from which X_i is drawn. Considering the ith trajectory, distribution X_i is used to construct a binary feature vector f_i, indicating which skills are present for this trajectory. Note that B encourages sharing of skills among the demonstration trajectories. Next, for mode j we define the transition probabilities for time series i using the transition probability vector $\pi_j^{(i)}$, which is drawn from a Dirichlet distribution.

Hyperparameter κ is used to place extra expected mass on the jth component, making the selection "sticky" due to the fact that skills are likely to be employed for multiple sequential time steps. For each time step t, a mode $z_t^{(i)}$ is drawn from the transition distribution at time step $t-1$. The observation for the ith time series, at time t, is represented by $y_t^{(i)}$. If the order of the model is r, then $y_t^{(i)}$ is computed as a sum of mode-dependent linear transformations using the previous r observations, as well as the model-dependent noise term $e_t^{(i)}$.

Sampling of the mode sequence $z_t^{(i)}$ is achieved using the Markov Chain Monte Carlo sampler [5] developed for the BP-AR-HMM. This sampler proposes skill birth and death moves based on their likelihoods, thereby adding or removing features from the global feature set.

The BP-AR-HMM emissions are modelled as VAR processes by *Fox et al.* [4]. To use this model in conjunction with inverse reinforcement learning, we require the emissions to be modelled as MDPs. *Ranchod et al.* [12] propose the following model to achieve this:

$$P(a)|z_t^{(i)} = \frac{e^{\tau Q^{z_t^{(i)}}(y_{t-1}^{(i)},a)}}{\sum_a e^{\tau Q^{z_t^{(i)}}(y_{t-1}^{(i)},a)}}$$
$$a_t^{(i)} \sim P(a)|z_t^{(i)}$$
$$y_t \sim T(y_{t-1}, a_t^{(i)})$$

This model removes the step of sampling of parameters A and e for the BP-AR-HMM. Instead the dynamics of the environment are used in conjunction with IRL to calculate the transition probabilities. The action-value function associated with each skill is represented by $Q^{z_t^{(i)}}$. This function is learned by grouping sub-trajectories in terms of skills and performing the maximum entropy IRL algorithm on each skill. This produces reward functions associated with each skill. The optimal policy for $Q^{z_t^{(i)}}$ is then determined by using value iteration, which gives the likelihood of each of the demonstration trajectories, and selecting the set with the highest likelihood [12].

III. Research Methodology

While previous work using the NPBRS framework [12] is able to successfully segment skills from demonstrations, the resulting policies are not used in any way. In this section, we show how to leverage these approaches to learn option models, consisting of initiation sets, policies and termination conditions. This requires creating a set of demonstration trajectories, segmenting these trajectories using NPBRS, and then estimating the initialisation set and termination condition from the segmented trajectories produced by NPBRS.

A. Generating Trajectories

Due to the deterministic nature of the domain in which this approach was tested, the use of human expert demonstrations was not required. Instead, Q-learning [18] was used to learn an optimal policy for the domain and trajectories were generated by following this policy to and from predefined start and endpoints. These trajectories are then sent to the NPBRS method to be segmented.

B. Learning Options from Demonstration

Given the set of trajectories produced by NPBRS, we wish to generate estimates for the initialisation and termination sets for an option. We phrase this as a classification problem in which we wish to classify the final trajectory state as a termination state (positive) or a non-termination state (negative), and similarly classify start states as to whether they are initiation states or not. It is important to note here that while we are classifying termination states, options require a termination condition. Thus, we will use our set of termination states to generate a probability of being a termination state for any particular state and use this as our termination condition.

A commonly used algorithm for supervised classification problems is the Support Vector Machine (SVM) [16]. The SVM aims to split a given dataset into two separate classes by finding a hyperplane with the maximum margin between the two classes. SVMs require the training data to be labelled into two distinct classes, but in our case there are no negative samples available for learning the initiation set and termination condition of the options.

An alternate approach is to use an anomaly detection method, in this case the one-class SVM (OC-SVM) [13]. The OC-SVM determines a hypersphere that bounds as much of the training data as possible, while attempting to minimise its volume.

Given a set of training data $X_1, ..., X_n \in d$, the OC-SVM first uses a mapping $\Phi : \mathcal{R}^d \to \mathcal{F}$ to project this data into a higher-dimensional space. The hypersphere in this space is then parameterised by a centre c and a radius r. These are computed by minimising the equation:

$$\min_{r,c,\xi} r^2 + \tfrac{1}{vn}\textstyle\sum_i \xi_i$$

$$\text{s.t. } \left\|\Phi\left(X_i\right) - c\right\|^2 \leq r^2 + \xi_i, \quad \xi_i \geq 0, \quad i = 1, \ldots, n,$$

where $v \in (0, 1)$ is a trade-off parameter between the radius and the number of training data points that fall inside the hypersphere, and ξ_i are slack variables which determine how far away outliers lie from the hypersphere surface.

Thus, given a set of trajectories, we can use the OC-SVM to define separate hyperspheres, using the trajectory start and end states as our training data. We will then consider any state which lies within our hypersphere trained on the start states as an initiation state and any state which lies within our hypersphere trained on the end states as a termination state.

IV. Experimentation

A. Domain and Setup

The domain used to test our method for learning options is the four rooms domain discussed in the original options framework paper [15]. The domain consists of four rooms, each connected by a hallway. *Sutton et al.* [15] investigates the usefulness of options by handcrafting options for each room that take the agent to the adjoining hallways. In this case the initiation set is any state within the room, the policy is the set of actions taking the shortest path to the hallway and the termination condition is 1 for a hallway state and 0 elsewhere.

For each room, there are two different options available, one which takes the agent from its current position to the hallway encountered when travelling clockwise, and the other which takes the agent to the corresponding anti-clockwise hallway. We wish to infer similar options from a set of given demonstrations.

First, we use Q-learning to generate a set of 5000 trajectories, each with random start and goal states. The reward function for learning in this domain was defined as 10 at the goal state and -1 elsewhere. These trajectories were then used as input for the NPBRS segmentation, with the sampler left to run for 60 minutes. This produced a total of four segmented skills, each with their own policy, reward function and set of inferred trajectories.

Due to the fact that a large number of trajectories were used to generate our skills, the number of inferred trajectories returned by NPBRS for each skill is also large. We wish to only use trajectories in which the termination state has a high likelihood of occurring. To achieve this we apply a threshold of 2% of the total states on the occurrence of termination states and discard any trajectory with a termination state occurring less than this threshold.

Using the trajectories resulting from this thresholding we can then apply our OC-SVM to obtain sets of initiation and termination states. We use the *Scikit-learn* one-class SVM for Python, with the radial basis function kernel, $\nu = 0.1$ and $\gamma = 0.5$, where ν is an upper bound on the fraction of training errors and γ is the kernel coefficient.

B. Results and Discussion

The set of initiation states obtained from applying the OC-SVM to the trajectory sets can be seen in Fig. 1 and similarly the set termination states can be seen in Fig. 2. We define our termination condition here as 1 for any state contained in the set of termination states and 0 elsewhere. Finally, to obtain the policies of our options, we take the policy generated for each skill by NPBRS and restrict it to only include states that the policy leads to from any of our initiation states. These policies can be seen in Fig. 3, where green states indicate our initiation states and purple states indicate termination states.

Looking at the resulting options that we have obtained, we put particular focus on the termination condition. This is because it is these termination conditions which assist in greatly reducing the complexity of the agent's decision tree. Based on how we have defined our termination condition, the agent no longer has to check which action to take at each time step once initiating an option. Instead it is only required to check whether it is in a termination state or not.

From the termination states detected in Fig. 2 we observe that the OC-SVM identifies hallway areas as the termination states. We consider the three states surrounding a particular hallway as states contributing to a *hallway zone*. This is because these states act as chokepoints for any trajectory moving between rooms, as the agent will have to pass through all three of these states. All of the termination states that we obtain are within 1 step of these *hallway zone* states.

It is easiest to consider the initiation states in Fig. 1 in conjunction with the policies beginning from those states in Fig. 3. From this we can see that the initiation states typically lie on a path that takes the agent between hallways. Hence the options that we form by combining our policy, initiation set and termination condition allow for the agent to travel between hallways. The options that we are able to autonomously detect are very similar to those defined originally for the four rooms domain by *Sutton et al.* [15].

The only notable significance between the sets of options is that the original initiation set contains all states for a particular room, while our initiation set only lies on the path between hallways, restricting us to inter-hallway travel. This aside, we can conclude that our method was successful in learning options from demonstration trajectories for the four rooms domain.

C. Learning With Options

To show the effectiveness of our learned options, we use them to find optimal policies for two different goal states. This process mimics the learning performed in the original options work [15], with the goal states defined as G1 and G2 as seen in Fig. 4.

We combine our options with the standard MDP for the four rooms domain to form a semi-Markov decision processes (SMDP) [2], which allows us to switch between executing option and using single actions.

We apply Q-learning [18] to our domain to learn the optimal policies, recording the average number of steps taken for each episode. We then compare this to the average number of steps using Q-Learning without options, and using the handcrafted clockwise and anti-clockwise options described previously. For all cases, Q-learning is performed with ϵ-greedy exploration ($\epsilon = 0.1$), learning rate $\alpha = 0.5$, and discount factor $\gamma = 0.9$. Results are averaged over 25 runs.

The results are shown in Fig. 5, where our options display similar performance to that of the handcrafted options. Both SMDPs begin learning at a lower average number of steps and converge faster than the MDP without options.

The difference between our options and the handcrafted options can be seen in the first few episodes, where our options have a higher average number of steps due to the limited initiation sets in comparison to the handcrafted initiation sets which each consist of the entire room.

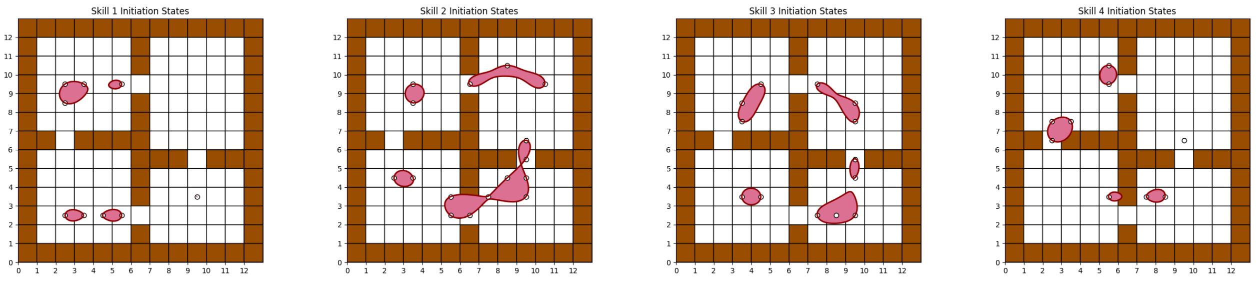

Fig. 1: Initiation states learned by OC-SVM

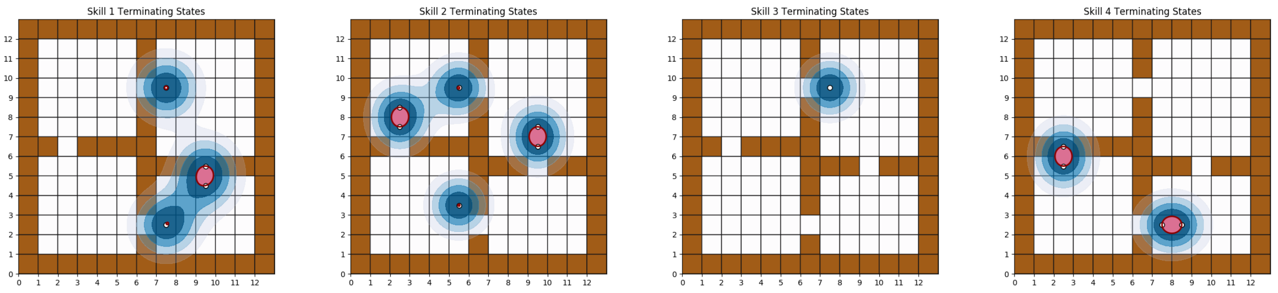

Fig. 2: Termination states learned by OC-SVM

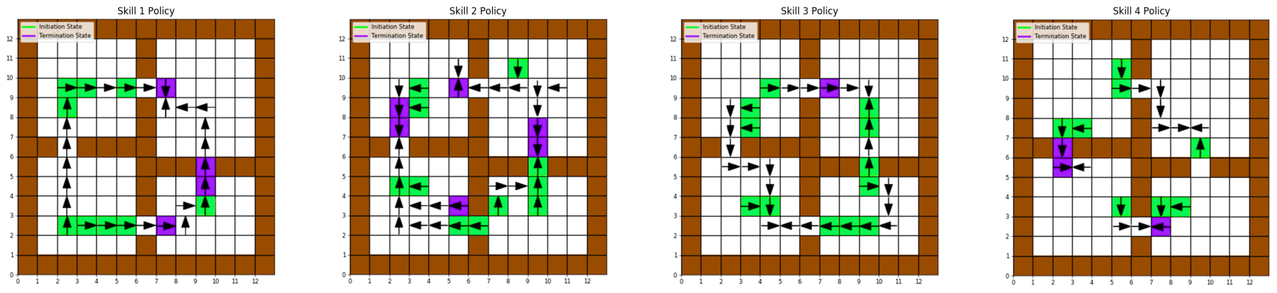

Fig. 3: Policies for each skill starting at initiation states

D. *Limitations and Future Work*

One of the main limitations with using the OC-SVM is in generating the termination condition. This is due to the fact that the OC-SVM is used for anomaly detection and thus only returns a classification of -1 or 1 depending on whether a given point is predicted to be an anomaly or not. This does not prove to be an issue in our study because we define our termination condition to be 1 for a state in the set of termination states and 0 elsewhere, but generally we want to be able to estimate the probability of terminating at each state. Potential adaptations to achieve this include running the output through logistic regression or using a kernel density estimator [17].

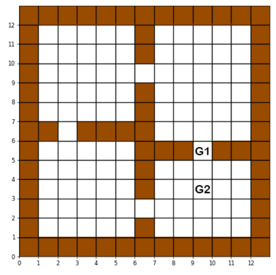

Fig. 4: Four rooms domain with goals defined at G1 and G2

Another limitation is in the number of trajectories we have generated. NBPRS should generally perform better given more trajectories, but it is possible that too many trajectories can result in noise occurring in the skills that are proposed. Ideally we would like to generate multiple sets of trajectories of different sizes and use the set for which the segmentation of the demonstration has highest log-likelihood.

Finally, we must note that the NPBRS framework looks to find the longest trajectory possible to explain a single skill. In this case the domain is not very large and so smaller trajectories might explain the skill better. Despite this, both of the estimated policies will still be the same and so this should not have too great of an impact on the final options that are learned.

One potential extension of the work in this area would be to incorporate the learned termination states into the NPBRS framework. These termination states can then be used to cap

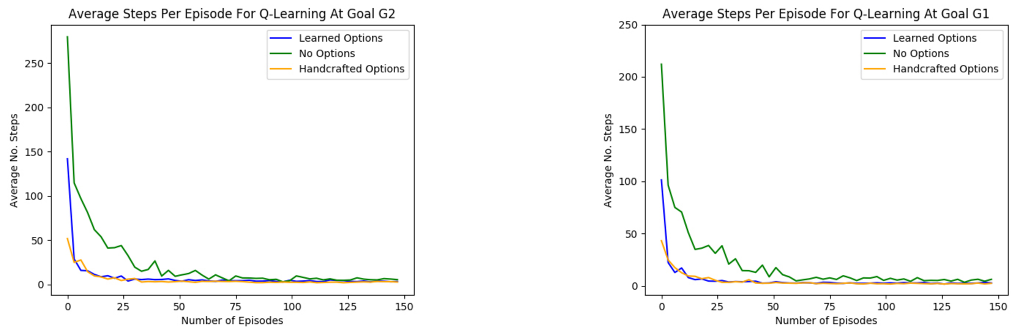

Fig. 5: Performance comparison between the standard MDP without options, our learned options, and the handcrafted options from the original framework using Q-learning in the four rooms domain.

the length of the proposed trajectories. Another potential area of future work is the adaptation of this method to a continuous domain, and further into real-world domains for robot tasks.

V. Related Work

Niekum et al. [11] also make use of the BP-AR-HMM for segmenting demonstration trajectories, but model the extracted skill policies as Dynamic Movement Primitives rather than options. The use of a sequence of primitives allows for better generalisation and the application of policy improvement on the skill policies, but is limited to single demonstration segments.

Support vector machines have been used in conjunction with the options framework before to learn sets of parameterised skills [3]. This method requires the policies for these skills to be learned from experience rather than segmented from a set demonstration trajectories though.

An alternate approach to learning options from sets of demonstration trajectories has been through the use of clustering algorithms. This includes K-means clustering and spectral clustering methods such as Perron Cluster Analysis [9], [10]. These methods group regions of the trajectory into abstract states and options are then used to define transitions between these states.

VI. Conclusion

In this paper, we presented a method that allows an agent to autonomously discover usable options from human demonstrations. We detailed the method of discovering these skills using the one-class SVM and tested it the four rooms domain, showing the benefit that the discovered options provide for learning. Finally, we outlined areas of improvement within the method and discussed the potential to incorporate the method within the NPBRS framework. While the method has proved successful in a common reinforcement learning domain, there is also potential for it to be adapted to handling more complex, real-world problems.

References

[1] P. Abbeel and A. Y. Ng, “Apprenticeship learning via inverse reinforcement learning,” in *Proceedings of the twenty-first international conference on Machine learning*. ACM, 2004, p. 1.

[2] S. J. Bradtke and M. O. Duff, “Reinforcement learning methods for continuous-time markov decision problems,” in *Advances in neural information processing systems*, 1995, pp. 393–400.

[3] B. C. Da Silva, G. Konidaris, and A. G. Barto, “Learning parameterized skills,” in *Proceedings of the 29th International Coference on International Conference on Machine Learning*. Omnipress, 2012, pp. 1443–1450.

[4] E. B. Fox, E. B. Sudderth, M. I. Jordan, and A. S. Willsky, “Joint modeling of multiple related time series via the beta process,” *The Annals of Applied Statistics*, vol. 8, no. 3, pp. 1281–1313, 2014.

[5] E. B. Fox, “Bayesian nonparametric learning of complex dynamical phenomena,” Ph.D. dissertation, Massachusetts Institute of Technology, 2009.

[6] M. Hollander, D. A. Wolfe, and E. Chicken, *Nonparametric statistical methods*. John Wiley & Sons, 2013, vol. 751.

[7] E. T. Jaynes, “Information theory and statistical mechanics,” *Physical review*, vol. 106, no. 4, p. 620, 1957.

[8] G. Konidaris, S. Kuindersma, R. Grupen, and A. Barto, “Robot learning from demonstration by constructing skill trees,” *The International Journal of Robotics Research*, vol. 31, no. 3, pp. 360–375, 2012.

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

[10] A. Y. Ng, M. I. Jordan, and Y. Weiss, “On spectral clustering: Analysis and an algorithm,” in *Advances in neural information processing systems*, 2002, pp. 849–856.

[11] S. Niekum, S. Osentoski, G. Konidaris, and A. G. Barto, “Learning and generalization of complex tasks from unstructured demonstrations,” in *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*, Oct 2012, pp. 5239–5246.

[12] P. Ranchod, B. Rosman, and G. Konidaris, “Nonparametric bayesian reward segmentation for skill discovery using inverse reinforcement learning,” in *2015 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, Sep. 2015, pp. 471–477.

[13] B. Schölkopf, J. C. Platt, J. Shawe-Taylor, A. J. Smola, and R. C. Williamson, “Estimating the support of a high-dimensional distribution,” *Neural computation*, vol. 13, no. 7, pp. 1443–1471, 2001.

[14] R. S. Sutton, A. G. Barto, *et al.*, *Introduction to reinforcement learning*. MIT press Cambridge, 1998, vol. 135.

[15] R. S. Sutton, D. Precup, and S. Singh, “Between mdps and semi-mdps: A framework for temporal abstraction in reinforcement learning,” *Artificial intelligence*, vol. 112, no. 1-2, pp. 181–211, 1999.

[16] V. Vapnik, *The nature of statistical learning theory*. Springer science & business media, 2013.

[17] M. P. Wand and M. C. Jones, *Kernel smoothing*. Chapman and Hall/CRC, 1994.

[18] C. J. Watkins and P. Dayan, “Q-learning,” *Machine learning*, vol. 8, no. 3-4, pp. 279–292, 1992.

[19] B. D. Ziebart, A. L. Maas, J. A. Bagnell, and A. K. Dey, “Maximum entropy inverse reinforcement learning.” in *Aaai*, vol. 8. Chicago, IL, USA, 2008, pp. 1433–1438.