Dynamic Abstraction in Reinforcement Learning via Clustering

Shie Mannor SHIE@MIT.EDU
Laboratory for Information and Decision Systems, Massachusetts Institute of Technology, Cambridge, MA 02139

Ishai Menache IMENACHE@TX.TECHNION.AC.IL
Amit Hoze AMITHOZE@ALUMNI.TECHNION.AC.IL
Uri Klein URIKLEIN@ALUMNI.TECHNION.AC.IL
Faculty of Electrical Engineering, Technion, 32000 Israel

Keywords:
Reinforcement Learning, Q-Learning, Options, Hierarchical Reinforcement Learning, Clustering

Abstract

We consider a graph theoretic approach for automatic construction of options in a dynamic environment. A map of the environment is generated on-line by the learning agent, representing the topological structure of the state transitions. A clustering algorithm is then used to partition the state space to different regions. Policies for reaching the different parts of the space are separately learned and added to the model in a form of options (macro-actions). The options are used for accelerating the Q-Learning algorithm. We extend the basic algorithm and consider building a map that includes preliminary indication of the location of "interesting" regions of the state space, where the value gradient is significant and additional exploration might be beneficial. Experiments indicate significant speedups, especially in the initial learning phase.

1. Introduction

Improving the performance with experience is the hallmark of Reinforcement Learning (RL). There are quite a few successful RL algorithms that theoretically solve any problem that can be cast in the Markov Decision Process (MDP) framework. Unfortunately, many RL techniques are not able to solve moderately large problems in reasonable time. The difficulty in solving such tasks is usually a result of the combination of the size of the state space with the lack of immediate reinforcement signal (the so-called "temporal credit assignment problem"). To date, there are three principal approaches for addressing these problems. The first approach is to consider a limited space of control strategies and search this space directly (Moriarty et al., 1999), or within an actor-critic framework (see Barto et al., 1983; Baxter & Bartlett, 2001). The second approach is to apply low order approximations of the value function (e.g., Bertsekas & Tsitsiklis, 1995). The third approach is to decompose the control to a hierarchy of several simpler tasks and learn each of them independently (e.g., Dayan & Hinton, 1993, Dietterich, 2000, Sutton et al., 1999, see Barto & Mahadevan, 2003 for a detailed recent review). This decomposition simplifies the learning problem in two ways. First, the effective size of the state space is reduced since every sub-task considers only a smaller number of relevant states. Second, learning is accelerated since every separate task is easier to learn.

In this paper we consider the problem of automatically discovering subtasks and hierarchies. Since in many cases the environment is unknown (or partially known), we cannot assume that the agent has the ability to identify desired subtasks beforehand. Even if the agent gathers knowledge of the environment, the decomposition to subtasks may not be straightforward. Moreover, the collection of advantageous subtasks may change and evolve throughout the learning process. We refer the reader to Barto and Mahadevan (2003) for further discussion on subtask discovery during the learning process, termed "dynamic abstraction", and its impact on RL.

A common approach is to define subtasks in the state space context. The learning agent identifies important states, which are believed to possess some "strategic" importance and are worthwhile reaching. The agent

Appearing in *Proceedings of the 21st International Conference on Machine Learning*, Banff, Canada, 2004.

learns sub-policies for reaching those key states. One approach is to look for states with non-typical reinforcement (a high reinforcement gradient, for example, as in Digney, 1998). This approach may not prove useful in domains with delayed reinforcement. Another approach is to choose states based on their frequency of appearance (see McGovern & Barto, 2001). The motivation here is that states that have been visited often in the past are likely to be a part of the agent's optimal path. Exploration time may be saved by having local policies for reaching those states directly. McGovern and Barto (2001) added the success condition to the frequency measure—states serve as potential subgoals if they are frequently visited on successful paths but are not visited on unsuccessful ones. A problem with frequency based solutions is that the agent may need excessive exploration of the environment in order to distinguish between "important" and "regular" states, so that options are defined at relatively advanced stages of the learning process. A different approach was introduced in Menache et al. (2002) where bottleneck states were defined as states that separate well the initial and target states. Our approach deviates from these approaches by considering clusters of states as intermediate stages in the learning process, rather than unique states, which leads to a more robust and versatile learning procedure. We define each *option* as a sub-policy that allows the agent to efficiently shift from one cluster of states to the other.

Our approach is to let the agent roam around the environment and then after some, possibly inaccurate, information was collected perform a clustering algorithm, and generate options for reaching the clusters. This approach relies on the assumption that the path towards achieving a complex goal passes through intermediate stages (each consists of multiple states) that occupy different parts of the state space. If those stages are discovered, and a policy that reaches each of them is separately learned, the overall learning procedure may become simpler and faster. An additional advantage of the clustering approach is that exploration may become more efficient since the agent can quickly wander to states that would be otherwise less explored, since they may be harder to reach from the initial states. The input for the clustering algorithm consists of the agent's recorded state transitions, which may be seen as a topological representation of the learning task dynamics. We then generalize the clustering algorithm by suggesting to use not only the state-transition map as an input, but also the current value estimates. The suggested algorithm encourages creating clusters with small deviation in the value function. The idea is to encourage the agent to travel between homogeneous clusters, increasing its probability to reach clusters with "interesting" values. In essence, the process of creating clusters may be considered as bootstrapping. The clusters are formed early in the learning process, much before convergence, and are based on a rough estimate of the environment. Using this rough estimate we improve the exploration, and focus on promising areas.

The paper is organized as follows: In Section 2 we describe the RL setup, extended to use options. The clustering problem is formally defined in Section 3. We consider two types of information that can be used by the clustering procedure. First, we limit our attention to clustering using topological state transition information. We then suggest to incorporate the preliminary value function estimation as well. Three experiments are described in Section 4: a simple maze world, the car-hill problem, and a more complicated maze. Section 5 contains concluding remarks.

2. Reinforcement Learning with Options

In this section we define the setup and survey the RL with options. See McGovern et al. (1997) for further details. We will consider a discrete time MDP with a finite set of states S and a finite set of actions A. At each time step t, $t = 1, 2, \ldots$, the learning agent is in some state $s_t \in S$. The agent can choose an action a_t from the set of available actions at state s_t, $A(s_t)$, causing a state transition to $s_{t+1} \in S$. The agent observes a scalar reward r_t which is a (possibly random) function of the current state and the action performed by the agent. The agent's goal is to find a map from states to actions, called a policy, which maximizes the expected discounted reward over time, $\mathbb{E}\{\sum_{t=0}^{\infty} \gamma^t r_t\}$, where $\gamma < 1$ is the discount factor and expectation is taken with respect to the random transitions, random rewards, and possibly random policy of the agent. A popular RL algorithm is the Q-Learning algorithm (Dayan & Watkins, 1992). In Q-Learning the agent updates the *Q-function* at every time epoch. The Q-function maps every state-action pair to the expected reward for taking this action at that state, and following an optimal strategy from that point on.

We now recall the extension of Q-Learning to Macro-Q-Learning (or learning with options, see McGovern et al., 1997). An option is a sequence of (primitive) actions that are executed by the agent (governed by a "local" policy) until a termination condition is met. Formally, an option is defined by a triplet $\langle I, \pi, \beta \rangle$, where: I is the options input set, i.e., all the states from which the option can be initiated; π is the op-

tion's policy, mapping states belonging to I to a sequence of actions; β is the termination condition over states (i.e., $\beta(s)$ denotes the termination probability of the option when reaching state s). When the agent is following an option, it must follow it until it terminates. When not following an option, the agent can choose, at any given state, either a primitive action or to initiate an option, if available (we shall use the notation $A'(s_t)$ for denoting all *choices*, i.e., the collection of primitives and options available at state s_t). The update rule for an option o_t, initiated at state s_t, is:

$$\begin{aligned} Q(s_t, o_t) \quad := \quad & Q(s_t, o_t) + \alpha(n(t, s_t, o_t))\Big(\\ & \gamma^{\tau} \max_{a' \in A'(s_{t+\tau})} Q(s_{t+\tau}, a') - Q(s_t, o_t) \\ & + r_t + \gamma r_{t+1} + \ldots \gamma^{\tau-1} r_{t+\tau-1} \Big), \end{aligned}$$

where τ is the actual duration of the option o_t, $\alpha(n(t, s_t, a_t))$ is the learning rate function which depends on $n(t, s_t, o_t)$, the number of times o_t was exercised in state s_t until time t. The update rule for a primitive action is similar with $\tau = 1$.

We now add the automatic option generation procedure and show how to combine it with Macro-Q-Learning. The outline of the learning procedure is described in Fig. 1. After some initiation conditions are met a clustering algorithm is invoked and options are created. The options in the context of clusters consist of supplying the shortest-path policies from each cluster to neighboring clusters to which it is connected (one option per neighboring cluster). We now supply additional details on the different steps of the algorithm.

Repeat:

1. Interact with environment and learn using Macro Q-Learning.
2. Save state transition history.
3. If clustering conditions are met, and the clustering procedure was not invoked previously:
   (a) Translate the state transition history to a graph representation.
   (b) Run clustering algorithm.
   (c) Learn the options for reaching neighboring clusters from each cluster.
   (d) Add the new options to the agent's choices.

Figure 1. Outline of the Q-Learning with options, based on clustering.

Activating clustering conditions: The timing of activating the clustering algorithm introduces a trade-off. On the one hand, we would like the clustering algorithm to be performed early in the learning process, where the impact on the exploration would be the most significant. On the other hand, if clustering is performed too early, the information obtained may not suffice for finding meaningful clusters, and the resulting options would contribute less to the learning effort. We found that a decent activation condition for the domains we experimented with is to wait until no new states were encountered for T time steps (or N episodes), indicating a stable state-transition model, with T (or N) being a task-dependent parameter. Conditions for activating the clustering algorithm should in general be task dependent and call for further study.

Translating the graph from history: Each visited state becomes a node in the graph. Each observed transition $s \to s'$ $(s, s' \in S)$, is translated to an edge (s, s') in the graph.

Learning an option: After clusters have been chosen, a local policy for reaching neighboring clusters is learned by an Experience Replay (Lin, 1992) procedure. Dynamic Programming (DP) is performed separately on each cluster, in order to determine the shortest paths to the neighboring clusters. The inputs for the DP are as follows: All border states belonging to neighboring clusters are assigned an artificial positive reward; the recorded experience serves as an estimation for the state transition probabilities. The termination probability of the option $\beta(s)$ is set to 1 for border states and 0 for internal states (in addition, we limit the maximal number of actual time steps of an option to some $\tau_{\max} \gg 1$). The details of the clustering algorithm itself are the topic of the next section.

3. Clustering

In order to explain the use of a clustering algorithm in the context of option discovery in RL, we first briefly review the graph theoretic problem it solves. Consider a directed graph $G = (V, E)$ (V is the set of nodes and E is the set of edges). We denote by f a function measuring the quality of a cluster. The function f takes as input a subset of nodes $C \subseteq V$ and the set of corresponding edges $\{(s, s') | s, s' \in C\} \subseteq E$ and returns a real number representing the cluster's quality. Denote by $E_{C_i \to C_j}$ the set of edges that originate in C_i and terminate in C_j and by $E_{C_i \leftrightarrow C_j}$ the set $E_{C_i \to C_j} \cup E_{C_j \to C_i}$ of edges that connect C_i to C_j or vice versa. In addition, let us define the inter-cluster quality function g. The function $g(C_i, C_j, E_{C_i \leftrightarrow C_j})$ takes two sets of nodes (C_i and C_j) and the set of edges between them, and

returns a number representing the separation quality between the clusters. The clustering problem is to determine the best partition of the (encountered) states for a given graph.

$$\max_{k\geq 1, C_1, C_2, \ldots, C_k \text{ s.t.} \atop \bigcup_{C_i = V} \text{ and } C_i \cap C_j = \emptyset \text{ for } i \neq j} \sum_{i=1}^{k} f(C_i, E_{C_i \to C_i}) + \sum_{i \neq j} g(C_i, C_j, E_{C_i \leftrightarrow C_j}). \quad (1)$$

The choice of functions f and g is important to the success of the algorithm. However, it turns out that even without laboriously tuning the parameters of f and g, reasonable results were obtained.

The maximization problem posed in Eq. (1) is not trivial. In fact, the problem is often NP-hard even for "simple" f and g. We refer the reader to Jain and Dubes (1988) and Hochbaum (1996) for a detailed discussion on the complexity of various variants of clustering algorithms and to Jain et al. (1999) for a survey of the vast number of applications. Given a choice of f and g we applied the so-called agglomerative approach (Anderberg, 1973; Jain & Dubes, 1988). According to this approach one starts with more clusters than desired and then merges the clusters by selecting the pair whose merging improves Eq. (1) by most. The algorithm (which is polynomial in the number of nodes, assuming that a calculation of f and g can be done in polynomial time) is described in Fig. 2.

Begin with n clusters. We selected each initial cluster to contain one state (i.e., $n = |S|$).
Repeat until *stopping conditions* are met:

1. Go over all pairs of neighboring clusters (i.e., i, j such that $E_{C_i \leftrightarrow C_j} \neq \emptyset$).
2. Calculate the utility in Eq. (1) with merging C_i and C_j.
3. Choose the pair whose union contributes the most to the utility.

Figure 2. A greedy agglomerative clustering algorithm.

There are several possible stopping conditions for the clustering algorithm. For example, the agent can wait until the total utility in Eq. (1) reaches a predetermined value. Another possible stopping condition is to wait until the difference in utilities between two consecutive clustering iterations is less than some predetermined threshold. In the experiments reported in this paper we assumed that the number of clusters is predetermined, so that the stopping condition is simply to stop when the number of clusters reaches some predetermined k.

We now introduce and motivate two possible clustering methodologies, whose solution is likely to produce useful options. We start with clustering based on topological information in Section 3.1. We then present in Section 3.2 an extension that considers reward related information in addition to topological information.

3.1. Clustering by Topology

As the learning agent interacts with the environment, information is gathered regarding the topological structure of the environment. We consider two desired properties of the partition of states to clusters. First, the size of the clusters should be roughly the same. If a cluster is too large then reaching it might be meaningless, and the options that reach it would probably not contribute much to the learning process. On the other hand, if the cluster is too small then the options reaching it may not play a significant role in the overall exploration effort. Second, the clusters should be well separated. It is preferable that neighboring clusters are distinct and have minimal interaction. This requirement captures the bottleneck notion of McGovern and Barto (2001) and Menache et al. (2002), where it was argued that bottlenecks between well separated clusters of states make useful intermediate subgoals.

Various choices for f and g may satisfy the above properties. We explored several heuristic possibilities for f, but none proved significantly better than using a trivial $f = 0$. We used

$$g(C_i, C_j, E_{C_i \leftrightarrow C_j}) = \frac{\min(|C_i|, |C_j|) \log(\max(|C_i|, |C_j|))}{|E_{C_i \leftrightarrow C_j}|}. \quad (2)$$

In words, g is proportional to the size of the smaller cluster, log the size of the larger, and inversely proportional to the number of edges that connect the two clusters (also $g \equiv 0$ if $E_{C_i \leftrightarrow C_j} = \emptyset$). This particular choice of g is justified as follows. The general optimization procedure (Eq. (1)) tries to maximize the sum of g between clusters. As a result, a small cluster would cause small value of g and is therefore likely to be "swallowed" by some neighboring cluster. In addition, a term of the form $\log(\max(|C_i|, |C_j|))$ encourages big clusters to swallow small ones. Since the total size of the clusters $\sum |C_j|$ equals $|S|$, and does not change with the partition, the numerator encourages all the clusters to have roughly the same size. A different interpretation of the numerator in Eq. (2), is to consider the entropy of the clustering, $\sum P_i \log P_i$, (P_i is

the fraction of states in cluster i) which achieves its maximum when P is a *uniform* distribution. The denominator decreases for well separated clusters (e.g., if there is a "bottleneck" between them) so that g increases for clusters that are distinctively apart.

3.2. Clustering by Value

In addition to the topological information (state transitions) there is some reward related information, which the learning agent gathers. When the clustering process is initiated, this information may be far from being complete, however it can be used in order to look for "interesting" regions in the state space. For example, an area in the state-space with a dense concentration of distinct rewards should not be contained in a large cluster, since careful control is required to maximally exploit it. On the other hand, an area with a few rewards may be regarded as one cluster, since the only interest of the agent is to exit it and explore other areas. Thus, we consider the current estimate of the value function and use it in order to encourage separating clusters with an inhomogeneous value estimation. Specifically, we use a different g,

$$g'(C_i, C_j, E_{C_i \leftrightarrow C_j}) = g(C_i, C_j, E_{C_i \leftrightarrow C_j}) \cdot \left(\frac{\Delta(E_{C_i \leftrightarrow C_j})}{|E_{C_i \leftrightarrow C_j}|} \right)^{\nu}, \quad (3)$$

where g was specified in Eq. (2) and $\Delta(E_{C_i \leftrightarrow C_j}) = \sum_{(s,s') \in E_{C_i \leftrightarrow C_j}} (1 + |V(s) - V(s')|)$ is the sum of differences of the value function (that is $V(s) = \max_{a \in A(s)} Q(s,a)$, where the current estimates of the Q-function are considered) of the edges connecting the two clusters. The constant $\nu > 0$ is chosen by trial and error. The additional term is intended to encourage uniting clusters with a small value gradient.

4. Experiments

In this section we describe three experiments. We start with a simple maze-world with a single initial state and a single goal and compare standard Q-Learning with Q-Learning with options that are discovered using clustering based on topology. We then consider the classical car-hill problem and perform a similar comparison. We finally consider a larger maze with multiple positive and negative rewards, and show the advantage of clustering by value.

4.1. A Simple Maze

The first experiment is with a maze environment described in Fig. 3. This is a maze with approximately 1,000 states. In each state the agent can move to one of the four directions, unless there is an obstacle in that direction. The agent starts from the upper left corner and its goal is to reach the bottom right corner as quickly as possible. The immediate reward the agent obtains is 0 except for the goal state, where it is +20. Each trial starts in the upper left corner and terminates in the goal state. The discount factor was set to $\gamma = 0.9$. There is a probability of 0.1 that the agent fails to move in the desired direction. We tested both standard Q-Learning and Q-Learning with options using clustering by topology (see Section 3.1). The initial Q-values were set to 0 and the learning rate was a constant $\alpha = 0.1$. An ϵ-greedy exploration was used for both algorithms, with $\epsilon = 0.3$. The number of clusters was set in advance to 19 (there are 19 rooms). The clustering was initiated if no new state was observed in the last $N = 10$ episodes. The maze is essentially comprised of rooms that are "well separated", so the clustering algorithm worked particularly well, matching clusters to rooms in almost every run.

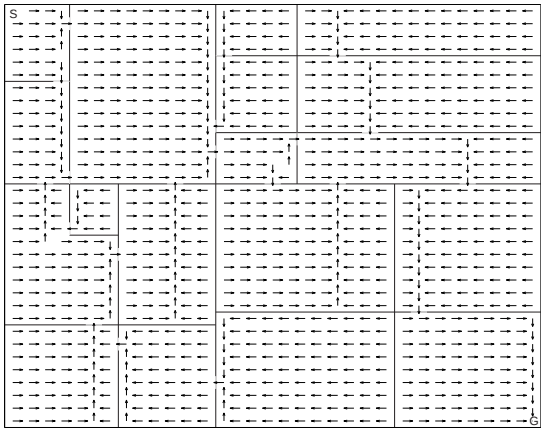

Figure 3. The simple maze containing 19 rooms and a single goal. The goal is marked as "G" in the bottom right corner. The initial state is marked "S" in the top left corner. The optimal policy is shown using arrows.

Fig. 4 presents the performance of the greedy policy derived from the current Q-function as a function of the trial number for both algorithms. This graph represents the expected number of steps to the goal (on a log scale) of the greedy policy w.r.t. to the Q-function learned at the end of the trial. It can be observed that Q-Learning with clustering approaches optimality much earlier than standard Q-Learning. The advantage of using the clustering algorithm is further demonstrated in Fig. 5. Here we show the Q-value of the initial state as a function of time, where each interaction with the environment takes one unit of time. Learning is accelerated on average by a factor of more than two, with a lower variance at the initial learning

phase (a smaller variance).

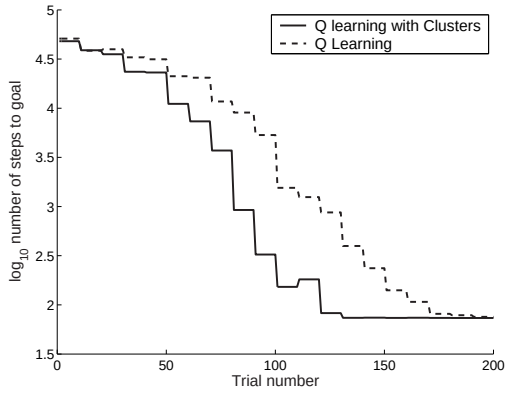

Figure 4. Simple maze performance. Number of steps to goal for the greedy policy. We compare Q-Learning and Q-Learning with clustering by topology, as a function of the trial number. The $\log_{10}$ of the average steps to goal of the greedy policy is depicted, averaged over 100 runs.

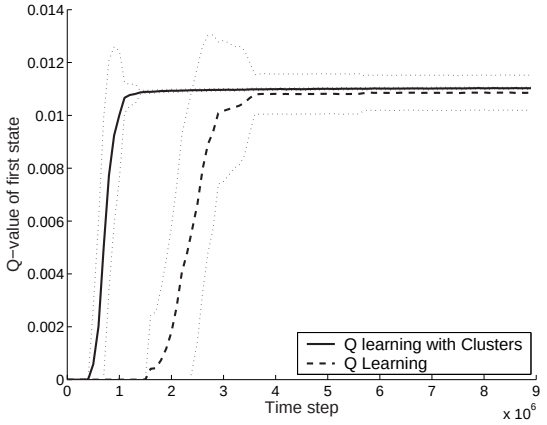

Figure 5. Simple maze performance. The Q-values of the initial state for Q-Learning and Q-Learning with clustering by topology as a function of time (number of steps). Results are averaged over 100 runs. The thin dotted lines represent plus/minus one empirical standard deviation.

4.2. The Car-Hill Problem

The second experiment considers the well known car-hill problem. In this problem a car tries to climb a one dimensional hill. In order to climb it, the car has to move in the opposite direction first, to build momentum, and then climb the hill. The state space of this problem is the position ($|p| \leq 1$) and velocity ($|v| \leq 3$). The agent tries to get to the right, i.e., to $p = 1$. The control decision is the direction of acceleration (either left or right), see Ernst et al. (2003) for the exact specifications that were used here. We considered continuous state space in discrete time, and discretized the space uniformly to a 50×50 space. The discretization was performed only with respect to the learning process - the dynamics were kept continuous (we used the closest point when referring to Q-values).

We tested standard Q-Learning and Q-Learning with clustering by topology on the car-hill domain. For the second algorithm we used $k = 14$ clusters (in practice, the number of clusters k had little effect on performance; using $k \in [10, 20]$ gave similar results). The clustering algorithm was invoked after $N = 10$ episodes where new states were not observed. The learning parameters for the two algorithms remain the same as in the previous experiment (Section 4.1), with the reward being $+1$ to get to the goal, -1 to be thrown away at the left side, and 0 otherwise. In Fig. 6 we show the clusters generated by a typical run. It can be seen that there is a single large cluster (noted by $\cdot$) and then small clusters marked by the letters "a"-"m". The optimal policy turned up to be gaining height on the opposite direction (e.g., reaching cluster "b"), and then moving to cluster "k", possibly through cluster "g". We note that a significant part of the state space was never visited (the white area), indicating efficient exploration. In Fig. 7 we compare the Q-values of the initial state ($p = -0.5$ and $v = 0$), showing a significant improvement made by the clustering approach.

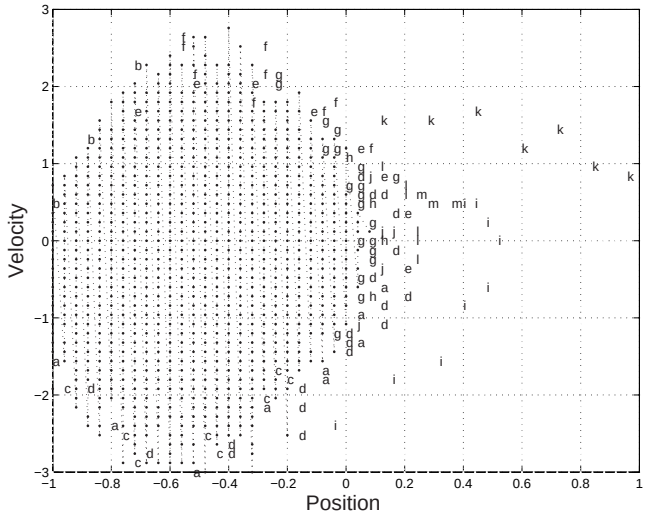

Figure 6. The clusters for the car-hill problem. The $\cdot$ sign represents the biggest cluster, the rest of the clusters are denoted by a letter ("a" to "m"). The target is to reach a position of $p = 1$, which is achieved within cluster "k".

4.3. A Multi-Reward Maze

We now consider a larger (50×50 states) and more complicated maze with multiple positive ($+20$) and negative (-20) rewards (see Fig. 8 for the maze map).

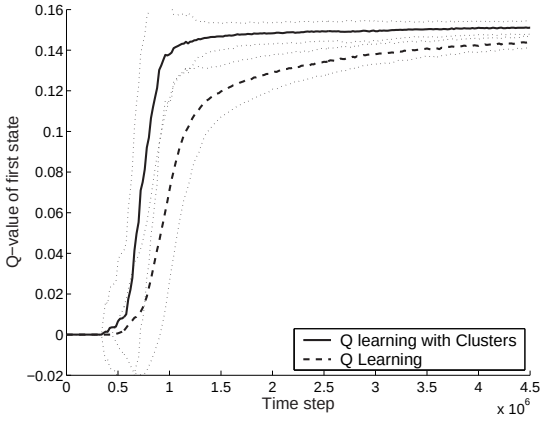

Figure 7. Car-hill performance. The Q-value of state $(p, v) = (-0.5, 0)$ (the initial state) for Q-Learning and Q-Learning with clustering by topology as a function of time (number of steps). Results are averaged over 100 runs. The thin dotted lines represent plus/minus one empirical standard deviation.

The maze follows similar rules to the simpler maze considered in Section 4.1, however multiple starting states are possible (the agent starts from one of the possible starting states, at random). We ran Q-Learning, Q-Learning with clustering by topology (utility function of Eq. (2) with $k = 40$ clusters), and Q-Learning with clustering by value (utility function of Eq. (3) with $\nu = 10$ and $k = 40$ clusters). The rest of the learning parameters remained as in Section 4.1. In Fig. 9 we compare the average Q-value of the initial states using the three algorithms. Clearly, our clustering approach has a significant advantage over standard Q-Learning. Moreover, the performance of clustering by value is better than the performance of clustering only by topology, as the steep rise in performance (the "knee") starts earlier. In Fig. 8 we present the different clusters generated by the two clustering algorithms (clusters boundaries are marked with a dotted line). Evidently, the clustering by value concentrates more on the "interesting" region (with high reward density) in the center of the maze.

5. Concluding Remarks

In this paper we explored a clustering approach to automatically generating options in RL. We considered two alternative clustering heuristics and showed that the concept works well in experiments. Our learning algorithm has the ability to define useful clusters, which can not be easily figured out, even when the model of the environment is known. As a consequence, learning is enhanced in tasks, which do not possess a natural hierarchical structure (e.g., the car-hill problem).

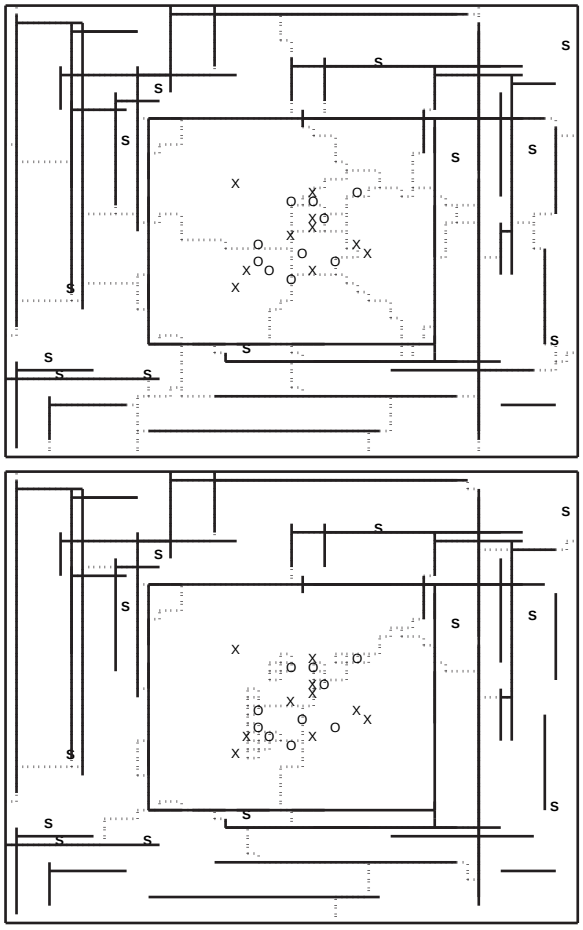

Figure 8. The multi-reward maze with the clusters generated by the two clustering algorithms (one map per algorithm). Each possible starting state is marked with an "S", states with positive reward are marked with "O" and states with negative reward are marked with "X". The upper graph is the map generated by clustering by topology and the lower graph is the map generated by clustering by value. Cluster boundaries are presented by a dotted line.

The clustering approach is not a panacea for finding options in all types of MDPs. We expect it to enhance learning when the state space is well separated. The identification of conditions under which the generated options are useful requires further study.

The most important research direction, in our opinion, involves scaling up the method. The examples presented in this paper are of moderate size, and while the performance of the clustering method is encouraging, some effort should be devoted to scaling it up to real-world applications. For example, when the state space is continuous the construction of the graph itself is an issue as multiple states should probably be

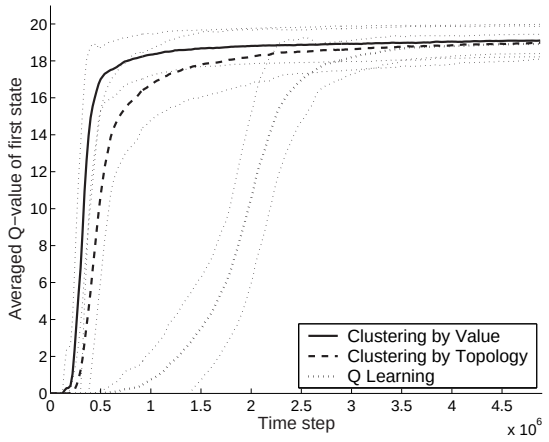

Figure 9. Multi-reward maze performance. The Q-value of the initial states (averaged over the 12 possible initial states) for Q-Learning, Q-Learning with clustering by topology, and Q-Learning with clustering by value. Results are averaged over 100 runs. The thin dotted lines represent plus/minus one empirical standard deviation.

aggregated before using the clustering algorithm. In addition, the condition for activating the clustering algorithm should be modified.

Additional directions, which we believe are promising include: performing the clustering multiple times, modifying the available options during the learning process as more information becomes available; "forgetting" options that are not useful or even damaging over time; using multiple alternative clustering functions, and learning to use the functions that produced the most useful options (i.e., meta-learning the clustering objective function); and finally an extension of the ideas presented here to the partial observed domains (cf. Theocharous & Kaelbling, 2003).

Acknowledgements

We are grateful for the helpful suggestions made by three anonymous reviewers. The work of S.M. was partially supported by the National Science Foundation under grant ECS-0312921.

References

Anderberg, M. (1973). *Cluster analysis for applications.* Academic Press.

Barto, A., & Mahadevan, S. (2003). Recent advances in hierarchical reinforcement learning. *Discrete Event Systems Journal*, *13*, 41–77.

Barto, A., Sutton, R., & Anderson, C. (1983). Neuronlike adaptive elements that can solve difficult learning control problems. *IEEE Transactions on Systems, Man, and Cybernetics*, *13*, 834–846.

Baxter, J., & Bartlett, P. L. (2001). Infinite-horizon policy-gradient estimation. *Journal of Artificial Intelligence Research*, *15*, 319–350.

Bertsekas, D. P., & Tsitsiklis, J. N. (1995). *Neuro-dynamic programming.* Athena Scientific.

Dayan, P., & Hinton, G. E. (1993). Feudal reinforcement learning. *Advances in Neural Information Processing Systems 5.* Morgan Kaufmann.

Dayan, P., & Watkins, C. (1992). Q-learning. *Machine Learning*, *8*, 279–292.

Dietterich, T. G. (2000). Hierarchical reinforcement learning with the MAXQ value function decomposition. *Journal of Artificial Intelligence Research*, *13*, 227–303.

Digney, B. (1998). Learning hierarchical control structure for multiple tasks and changing environments. *Proceedings of the Fifth Conference on the Simulation of Adaptive Behavior.*

Ernst, D., Geurts, P., & Wehenkel, L. (2003). Iteratively extending time horizon reinforcement learning. *Proceedings of the 14th European Conference on Machine Learning* (pp. 96–107).

Hochbaum, D. (1996). *Approximation algorithms for NP-hard problems.* Brooks/Cole Publishing Co.

Jain, A., & Dubes, R. (1988). *Algorithms for clustering data.* Prentice Hall.

Jain, A., Murty, M., & Flynn, P. (1999). Data clustering: A review. *ACM Computing Surveys*, *31*, 264–323.

Lin, L. G. (1992). Self-improving reactive agents based on reinforcement learning, planning and teaching. *Machine Learning*, *8*, 293–321.

McGovern, A., & Barto, A. G. (2001). Automatic discovery of subgoals in reinforcement learning using diverse density. *Proceedings of the 18th International Conference on Machine Learning* (pp. 361–368). Morgan Kaufmann.

McGovern, A., Sutton, R. S., & Fagg, A. H. (1997). Roles of macro-actions in accelerating reinforcement learning. *Proceedings of the 1997 Grace Hopper Celebration of Women in Computing* (pp. 13–18).

Menache, I., Mannor, S., & Shimkin, N. (2002). Q-Cut - dynamic discovery of sub-goals in reinforcement learning. *Proceedings of the 13th European Conference on Machine Learning* (pp. 295–306). Morgan Kaufmann.

Moriarty, D., Schultz, A., & Grefenstette, J. (1999). Evolutionary algorithms for reinforcement learning. *Journal of Artificial Intelligence Research*, *11*, 199–229.

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

Theocharous, G., & Kaelbling, L. P. (2003). Approximate planning in POMDPs with macro-actions. To appear in Advances in Neural Processing Information Systems 17.