# K-Cluster Algorithm for Automatic Discovery of Subgoals in Reinforcement Learning

Ben-Nian Wang[1,2], Yang Gao[1], Zhao-Qian Chen[1], Jun-Yuan Xie[1], Shi-Fu Chen[1]
[1]*National Laboratory for Novel Software Technology, Nanjing University, China*
[2]*Computer Science and Technology Department, Tongling College, China*
*bnwang@mail.tl.ah163.net, {gaoy, chenzq, jyxie,chensf}@nju.edu.cn*

## Abstract

*Options have proven to be useful to accelerate agent's learning in many reinforcement learning tasks. determining useful subgoals is a key step for agent to create options. A K-cluster algorithm for automatic discovery of subgoals is presented in this paper. This algorithm can extract subgoals from the trajectories collected online in clustering way. The experiments show that the K-cluster algorithm can find subgoals more efficiently than the Diverse Density algorithm, and that the reinforcement learning with this algorithm outperforms the one with the Diverse Density algorithm and flat Q-learning.*

## 1. Introduction

Hierarchical reinforcement learning (HRL) is a new way to solve the problem of the curse of dimensionality in reinforcement learning (RL), which constructs the hierarchical control mechanism by introducing temporal abstraction into RL. Three approaches to temporal abstraction, which are Options [1], HAMs [2] and MAXQ [3], have been developed recently from action, policy and task perspectives respectively. In this paper, we only consider the option approach.

An option, formalized by Sutton, Percup and Singh, is the generalization of primitive action to include temporally extended course of action. The simplest kind of option consists of three components: a policy $\pi : S \times \cup_{s \in S} A_s \rightarrow [0,1]$ , a termination condition $\beta : S - [0,1]$, and an input set $I \subseteq S$, where $S$ is a finite set of states that an agent observes in its environment, and $A_s$ is a finite set of admissible actions in state $s$. An option $< I, \pi, \beta >$ is available in state $s$ if and only if $s \in I$. If the option is taken, then actions are selected according to $\pi$ until the option terminates stochastically according to $\beta$. For example, if the current state is $s$, the next action is $a$ with probability $\pi(s,a)$, and then the environment makes a transition to state $s'$, where the option either terminates with probability $\beta(s')$ or else continues, determining the next action $a'$ with probability $\pi(s',a')$, and so on. When the option terminates, the agent has the opportunity to select another option.

For an option approach, each option has its own policy that makes agent achieve the subgoal related to this option. A natural issue arises here. That is, how are useful subgoals determined? In this paper, such an issue is investigated and a K-cluster algorithm for subgoal discovery is proposed. By clustering the trajectories that are collected online, this algorithm can extract subgoals from them. In the experiment of a six-room grid-world, this algorithm can find all doorways quite efficiently.

The rest of this paper is organized as follows. Section 2 presents our subgoal discovery algorithm based on K-cluster, Section 3 discusses related work, Section 4 reports on experiments, and Section 5 summaries the main contribution of this paper.

## 2. Automatic Discovery of Subgoals

Stolle and Precup [4] view subgoals as bottlenecks in the state space, and define bottlenecks as states that are frequently visited on system trajectories, where trajectory is a sequence of states that agent visits during an episode. The word " frequently", however, cannot express the concept of bottleneck sufficiently. As bottlenecks, the states must satisfy two conditions: (1) the number of them is small, and (2) a trajectory between starting state and goal state passes through one of them at least. Under the two conditions, we propose a new subgoal discovery method based on K-cluster algorithm.

Proceedings of the 2005 International Conference on Computational Intelligence for Modelling, Control and Automation, and International Confe
Intelligent Agents, Web Technologies and Internet Commerce (CIMCA-IAWTIC'05)
0-7695-2504-0/05 $20.00 © 2005 IEEE

```
Function k_Cluster (TrajSet, K)
N ← the number of trajectories in TrajSet;
SubgoalList ← φ ;
If N ≥ K
  CandidateSet ← {s: Σ_{s∈T∈TrajSet} 1 ≥ N/K };
  Clear redundancy from CandidateSet;
  For each s∈ CandidateSet
    Cluster={T: s∉ T∈ TrajSet};
    If Cluster is empty
      Append s to SubgoalList;
    Else If K-1 ≥ 1
      S ← K_Cluster(Cluster,K-1);
      Append S to SubgoalList;
    End If
  End For
End If
Output SubgoalList;
```

**Figure 1**. K-cluster algorithm for discovery of subgoals. The input parameter TrajSet is a set of the successful trajectories, and *K* is the maximum number of clusters. The output result SubgoalList lists the subgoals found by this algorithm.

```
Procedure HRL_framework
For each episode
  Learn using RL;
  Save successful trajectories into TrajSet;
  If the number of the trajectories in TrajSet
  reaches a threshold Δ
    SubgoalList ← K_Cluster(TrajSet, K);
    For each g∈ SubgoalList
      Create an option o_g =< I, π, β >;
      Initialize I by examining trajectories
      in TrajSet;
      Set β(g) = 1 , β(S - I) = 1 , β(·) = 0 else;
      Initialize policy π using experience replay;
    End For
  End If
End For
```

**Figure 2**. HRL framework with K-cluster algorithm. In this framework, subgoals are extracted using K-cluster algorithm, and options are created based on the subgoals.

### 2.1. K-Cluster Algorithm for Subgoal Discovery

K-Cluster algorithm implies the following idea. We cluster the trajectories collected online, that is, put all trajectories that pass through a certain state called cluster point into a class. Assume that there are $K$ classes generated in this way, which means that there are $K$ cluster points. If the $K$ cluster points are potential bottlenecks, then $K$ should be small by the definition of bottleneck. Let $N$ is the total number of the trajectories. Among the $K$ cluster points, obviously, there must exist one point whose visitation count isn't less than $N/K$. For given $K$, therefore, we can filter out all these states whose visitation counts aren't less than $N/K$. For each of these states , all trajectories fall into two classes, i.e., the trajectories that pass through it are put into one class, and the rest into another one. We continue this procedure with the second class if it isn't empty. If the number of classes isn't more than $K$ eventually, then all cluster points corresponding to these classes can be regarded as potential bottlenecks or potential subgoals. The pseudo-code of K-cluster algorithm is shown in figure 1.

For the algorithm shown in figure 1, further explanation is provided:

(1) We call a trajectory that end at the goal successful trajectories. The parameter TrajSet is a set of the successful trajectories, in which trajectories can be allowed to contain cycles of states and to start from a variety of starting states or to end at a variety of goal states, so the knowledge acquired with K-cluster algorithm can be reused in the same or similar tasks.

(2) The parameter $K$ is the number of classes that the trajectories in TrajSet will be clustered into. Generally, we set $K$ =2.

(3) In the set CandidateSet computed in the line 5 in figure 1, perhaps there are some states that are adjacent on the trajectories in TrajSet, so it is necessary to clear the redundant adjacencies. This procedure is described as follows: for each adjacent state string $s_1 \cdots s_n$ on the trajectories in TrajSet, if all these states belong to CandidateSet, the middle state in the adjacent state string $s_1 \cdots s_n$ is kept in CandidateSet, and the others are cleared from CandidateSet.

(4) K-cluster algorithm is a recursive procedure that has a strongly linear character, in which there doesn't exist any convergence problem due to the limitation of $K$. Its time complexity is not more than $4NL$, where $L$ is the average length of the trajectories in TrajSet. Since $K$ and $N$ are very small in practical tasks, the computing expense of K-cluster algorithm is very low.

### 2.2. Creating Options

For each subgoal $g$ in SubgoalList, we can create an option: $o_g =< I, \pi, \beta >$. The option's input set, $I$, can be initialized through searching the trajectories in TrajSet. If a state $s$ is before $g$ and after $g'$ on a trajec-

tory if there exists $g'$, where $g' \neq g$ and $g' \in$ SubgoalList, then $s$ is added to $I$. The option's policy, $\pi$, is initialized by creating a new value function that uses the same state space as the overall task. The reward function used for the option is to give a reward of 0 on each step and 1 when the option achieves its subgoal. The option's value function is learned using experience replay [5] with the trajectories in TrajSet. The termination condition, $\beta$, is set as follows: $\beta(g)$ =1 and $\beta(S-I)$ =1, and $\beta(\cdot)$ =0 otherwise.

Figure 2 shows a whole HRL framework, in which subgoals are extracted using K-cluster algorithm and then options are created based on these subgoals. For some complex tasks, this framework can also be executed at a local level.

## 3. Related Work

There are several algorithms that deal with subgoal discovery in reinforcement learning. The algorithms that are related to our algorithm in this paper are a kind of state visitation counts-based subgoal discovery ones.

Stolle and Precup [4] present an algorithm for subgoal discovery that seeks bottlenecks in the state space. At first, the algorithm selects at random a number of starting states and target states that will be used to generate random tasks for the agent. Then, for each pair of starting state and goal state, there are two phases, in the first one, the agent performs Q-learning to learn a policy for going from the starting state to the goal state, in the second one, the agent performs episodes using the greedy policy learned, and counts the total number of times that each state is visited. Finally, the states that have the highest visitation counts are considered to be subgoals. Obviously, the algorithm is a typical off-line one.

Another method for subgoal discovery is the work of McGovern and Barto [6]. Their approach is to treat the problem of finding bottleneck regions as a multiple-instance learning problem. They label a trajectory as a positive bag, if it can lead the agent to the goal state successfully, or as a negative bag if it fails to lead the agent to the goal state. After that, they use the concept of Diverse Density [7] to detect the bottleneck regions, that is, the concepts that have the largest diverse density values are considered to be potential subgoal states. Finally, a predefined filter is applied to eliminate certain states from the consideration as subgoals.

Kretchmar *et. al* [8] analyzed the drawbacks of the Diverse Density work of McGovern and Barto, including the definition of a negative bag, the limitation of a physical distance metric, the sensitivity to various parameters, and the need for a predefine filter. In response to these issues, they develop an improved subgoal discovery algorithm called the FD Algorithm. This algorithm only uses the successful trajectories and discards the unsuccessful trajectories. Subgoal candidateship can be computed in accordance with the product of a frequency metric F and a temporal distance metric D. For each state, F is the ratio of the number of trajectories containing the state to the total number of trajectories. D is computed based on the temporal distance of the state from undesirable subgoal locations. A state receives a higher score if, temporally, it tends to be in the middle of the trajectory rather than towards the beginning or the end of the trajectory. Finally, the values of the two metrics are multiplied together and the state with the highest combined value is considered to be a subgoal.

Kretchmar *et. al* improved the work of McGovern and Barto. However, they also introduced some problems into their algorithm. Although the temporal distance metric D can restrain the states that are adjacent to the starting state and goal state from being subgoals, it causes the problem that the nearer to the middle position on a trajectory the state locates, the greater opportunity it has. In order to avoid the influence of noises on the performance of the algorithm, their experimentation with the FD algorithm used trajectories from an agent that had already run 150 episodes of Sarsa(0). Since one of the purposes of finding subgoals is to accelerate an agent's learning, the FD method becomes very impractical as the state space grows.

Compared to the above algorithms, our algorithm hasn't any metrics to compute, doesn't need any handcrafted filter, and has fewer parameters, so it is very simple and robust. Particularly, the above algorithms can't judge whether there exists subgoal or not in the ongoing task, and they implicitly assume that subgoal exists in the ongoing task, but our algorithm can get such knowledge by checking whether the SubgoalList returned from the function of k_cluster is empty or not. Moreover, the above algorithms find only one subgoal per run, while ours can extract multiple subgoals.

## 4. Experiment Results

We test the K-cluster algorithm by applying it to a six-room grid-world task shown in figure 3. There are two purposes for this experiment, one is to verify whether the K-cluster algorithm can find useful subgoals efficiently, and the other is to estimate whether the options created based on the subgoals extracted by K-cluster algorithm can accelerate agent's learning. We compare the performance of our algorithm to that of the Diverse Density algorithm and of flat Q-learning with the same settings.

For the convenience of description, we index the position of an agent in the grid-world from top to bottom and from left to right as numbers 1-672, in which

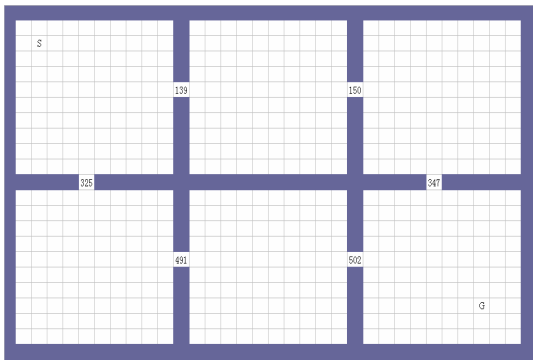


**Figure 3**. Six-room grid-world environment for the experiment.

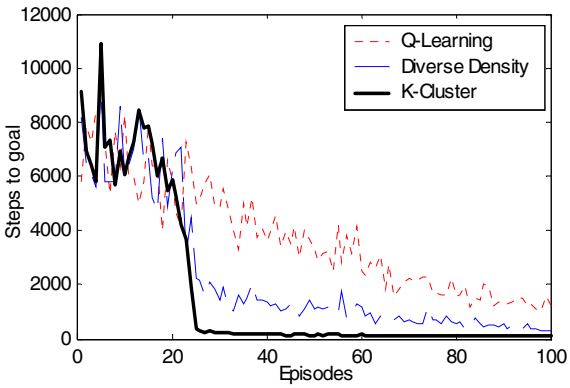


**Figure 4**. Performance comparison of HRL with K-Cluster Algorithm to HRL with the Diverse Density Algorithm and flat Q-Learning on the six-room gird-world. The results are averaged over 30 runs.

state 139, 150, 325, 347, 491 and 502 are six doorways. The starting state $s$ is placed in the upper left-hand room, and the goal state $G$ in the lower right-hand corner. An episode terminates when the agent reaches the goal G. The primitive actions available to the agent are up, down, right, and left. Each action succeeds in moving the agent in the chosen direction with probability 0.9 and in a uniform random direction with probability 0.1. The agent receives a reward of 1 for reaching the goal and 0 otherwise. The discount factor $\gamma$=0.9, the learning rate $\alpha$=0.1, the threshold $\Delta$=20, and the maximum number of clusters $K$=2. The agent explores using the $\varepsilon$-greedy method where the greedy action is selected with probability (1-$\varepsilon$), $\varepsilon$=0.1.

We run the HRL framework with the K-cluster algorithm and that with the Diverse Density algorithm and flat Q-learning 30 times respectively. For the former two algorithms, agents execute flat Q-learning in the early runs, and once they create options, they switch their learning algorithms from flat Q-learning to SMDP Q-learning. In every run, the K-cluster algorithm can find all subgoals (doorways) in the gird-world task, while the Diverse Density algorithm locates subgoals to the states near the doorways. Figure 4 compares the performance of K-cluster algorithm against the Diverse Density algorithm and flat Q-learning. The results in this graph are averaged over 30 runs, which show that the two HRL algorithms have considerably accelerated learning compared to Q-learning after 20 trials due to options being created, and the HRL with K-cluster algorithm has better performance than the one with the Diverse Density algorithm because the former can locate all subgoals more exactly than the latter.

## 5. Conclusions

In this paper, we present a K-cluster algorithm for automatic discovery of subgoals. By using a method of clustering, this algorithm can extract subgoals from the set of the trajectories collected online. Compared to the Diverse Density and FD algorithms, it has fewer parameters to set and is more suitable to practical applications. The experiments reported in this paper show that the algorithm can find subgoals quite efficiently, and that the SMDP Q-learning with this algorithm outperforms the one with the Diverse Density algorithm and the MDP Q-learning.

Options have proven to be useful to accelerate agent's learning in many RL tasks. Finding useful subgoals and creating options are the key steps in HRL. An efficient subgoal discovery algorithm should be able to mine the agent's experience sufficiently, and to identify all subgoals precisely. Our research is focusing on this point. However, the algorithm presented in this paper is not useful for every environment. Developing a more general method will be our further work.

## Acknowledgment

This work was supported by the National Natural Science Foundation of China under Grant No. 60475026, the National Grand Fundamental Research 973 Program of China under Grant No. 2002CB-312002, the Natural Science Foundation of Jiangsu Province of China under Grant No. BK2004079, and the Creative Talent Foundation of Jiangsu Province of China under Grant No. BK2003409.

## References

[1] 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, pp.181–211, 1999.

[2] R. Parr, S. Russell, "Reinforcement Learning with Hierarchies of Machines," In: Proceedings of Advances in Neural Information Processing Systems 10. Cambridge, MA: MIT Press, 1998, pp. 1043-1049.

[3] T.G. Dietterich, "Hierarchical Reinforcement Learning with the MAXQ Value Function Decomposition," Journal of Artificial Intelligence Research, vol. 13, pp. 227-303, 2000.

[4] M. Stolle, D. Precup, "Learning Options in Reinforcement Learning," In: Proceedings of the 5th International Symposium on Abstraction, Reformulation and Approximation, Kananaskis, Alberta, Canada, 2002, pp. 212–223.

[5] L. Lin, "Self-improving Agents Based on Reinforcement Learning, Planning and Teaching," Machine Learning, vol.3, pp. 293–321, 1992.

[6] A. McGovern, A. Barto, "Automatic Discovery of Subgoals in Reinforcement Learning Using Diverse Density," In: Proceedings of the Eighteenth International Conference on Machine Learning, San Francisco, CA: Morgan Kaufmann, 2001, pp. 361-368.

[7] O. Maron, P.T. Lozano, "A Framework for Multiple-instance Learning," In: Proceedings of Advances in Neural Information Processing Systems 10, Cambridge, MA: MIT Press, 1998, pp. 570–576.

[8] R.M. Kretchmar, T. Feil, and R. Bansal, "Improved Automatic Discovery of Subgoals for Options in Hierarchical Reinforcement Learning," Journal of Computer Science and Technology, vol. 2, pp. 9–14, 2003.