Multi-robot Cooperation Based on Hierarchical Reinforcement Learning

Xiaobei Cheng, Jing Shen, Haibo Liu, and Guochang Gu

College of Computer Science and Technology, Harbin Engineering University, Harbin 150001, China
adbccxb1306@sina.com.cn, {liuhaibo, shenjing, guguochang}@hrbeu.edu.cn

Abstract. Multi-agent reinforcement learning for multi-robot systems is a challenging issue in both robotics and artificial intelligence. But multi-agent reinforcement learning is bedeviled by the curse of dimensionality. In this paper, a novel hierarchical reinforcement learning approach named MOMQ is presented for multi-robot cooperation. The performance of MOMQ is demonstrated in three-robot trash collection task.

Keywords: multi-robot, cooperation, hierarchical reinforcement learning.

1 Introduction

Multi-Robot Systems (MRSs) can often be used to fulfil the tasks that are difficult to be accomplished by an individual robot, especially in the presence of uncertainties, incomplete information, distributed control, and asynchronous computation, etc. So MRSs have received considerable attention during the last decade [1][2]. Currently, there has been a great deal of research on multi-agent reinforcement learning (MARL) in MRSs [3]. Multi-agent reinforcement learning allows participating robots to learn mapping from their states to their actions by rewards or payoffs obtained through interacting with their environment. MRSs can benefit from MARL in many aspects. Robots in MRSs are expected to coordinate their behaviors to achieve their goals. These robots can either obtain cooperative behaviors or accelerate their learning speed through learning [4]. But MARL is bedeviled by the curse of dimensionality.

Two methods for combating the curse of dimensionality are function approximation and hierarchical decomposition. Function approximation is aimed at approximating and thereby compacting a value function. Hierarchical approaches use structure in the representation to try to compact the representation and have the potential to reduce the exponential growth in the size of the state space to linear in the number of variables. Hierarchical solution involves multiple levels or stages of decision making that together solve the whole problem. Several alternative frameworks for hierarchical reinforcement learning (HRL) have been proposed [5], including Options [6], HAMs [7] and MAXQ [8]. Ghavamzadeh *et al* [9] extended MAXQ method for single agent HRL to multi-agent cooperative HRL. MAXQ has a number of notable features. MAXQ represents the value of each state in a subtask as a decomposed sum of

Y. Shi et al. (Eds.): ICCS 2007, Part III, LNCS 4489, pp. 90–97, 2007.

completion values. MAXQ allows subtask policies to be reused in different contexts. The final MAXQ feature to be highlighted is the opportunity for state abstraction [8]. State abstraction is key to reducing the storage requirements and improving the learning efficiency. State abstraction means that multiple states are aggregated and one completion value stored for the aggregate state in a subtask. But in the MAXQ framework the ability of state abstraction is limited. For example, in the multi-agent taxi domain [8] and the multi-robot trash collection task [9] the subtask *navigate* cannot be decomposed into more refined MAXQ subtask. On the other hand, the hierarchies are difficult to be discovered automatically.

In this paper, a novel multi-agent hierarchical reinforcement learning approach named MOMQ (Multi-robot Option-MaxQ) by integrating Options into MAXQ is presented for multi-robot cooperation. In the MOMQ framework, the MAXQ framework is used to introduce knowledge into reinforcement learning and the Option framework is used to construct hierarchies automatically.

2 MOMQ Framework

The MOMQ is based on the multi-agent cooperative MAXQ method in [8]. Consider sending a team of robots to pick up trash from trash cans over an extended area and accumulate it into one centralized trash bin, from where it might be sent for recycling or disposed. It is assumed that the robots are homogeneous, i.e., all robots are given the same task hierarchy. At each level of the hierarchy, the designer of the system defines cooperative subtasks to be those subtasks in which coordination among robots significantly increases the performance of the overall task. The set of all cooperative subtasks at a certain level of the hierarchy is called the cooperation set of that level. Each level of the hierarchy with non-empty cooperation set is called a cooperation level. The union of the children of the *l*th level cooperative subtasks is represented by U_l. Robots actively coordinate only while making decision at cooperative subtasks and are ignorant about the other robots at non-cooperative subtasks. Therefore, cooperative subtasks are configured to model joint-action values. An simulation experiment environment with three robots ($R1$, $R2$, and $R3$) is shown in Fig.1(a). Robots need to learn three skills here. First, how to do each subtask, such as navigate to trash cans $T1$, $T2$, or $T3$ or *Dump*, and when to perform *Pick* or *Put* action. Second, the order to do the subtasks, for instance go to $T1$ and collect trash before heading to *Dump*. Finally, how to coordinate with each other, i.e., robot $R1$ can pick up trash from $T1$ whereas robot $R2$ can service $T2$, and so on. The overall task is decomposed into a collection of primitive actions and temporally extended (non-primitive) subtasks that are important for solving the problem. The non-primitive subtasks in the trash collection task are root (the whole trash collection task), *collect* $T1$, $T2$, and $T3$, *navigate to* $T1$, $T2$, $T3$, and *Dump*. Each of these subtasks has a set of termination states, and terminates when reaches one of its termination states. Primitive actions are always executable and terminate immediately after execution. After defining subtasks, we must indicate for each subtask, which other primitive or non-primitive subtasks it should employ to reach its goal. For example, *navigate to* $T1$, $T2$, $T3$, and *Dump* use four primitive *up*, *down*, *left*, *right*. *Collect* $T1$ should use two subtasks *navigate to* $T1$ and *Dump* plus two primitive actions *Put* and *Pick*, and so on. The problem is how to

navigate efficiently in a large-scale environment. MOMQ framework can now be illustrated according to Fig.1(b). Imagine the robots start to learn the task with same MOMQ graph structure. The robots need to learn the fourth skill, i.e., how to construct the Options automatically.

The MOMQ method decomposes an MDP (Markov decision process) M into a set of subtasks M_0; $M_1...M_n$, where M_0 is the root task and solving it solves the entire MDP M. Each non-primitive subtask is a six tuple (S_i, I_i, π_i, T_i, A_i, R_i). S_i is the state space for subtask i. It is described by those state variables that are relevant to subtask i. The range of the state variables describing S_i might be a subset of their range in S, the state space of the overall task MDP M. I_i is the initiation set for subtask i. Subtask i could start only in state $s \in I_i$. π_i is the policy for subtask i can only be executed if the current state $s \in (S_i - T_i)$. The hierarchical policy is executed using a stack discipline similar to ordinary programming languages. Each subtask policy takes a state and returns the name of a primitive action to execute or the name of a subtask to invoke. T_i is the set of terminal states for subtask i. Subtask i terminates when it reaches a state in T_i. A_i is the set of actions that can be performed to achieve subtask i. These actions can either be primitive actions from A (the set of primitive actions for MDP M) or they can be other subtasks. R_i is the pseudo reward function, which specifies a *pseudo-reward* for each transition from a state $s \in (S_i - T_i)$ to a terminal state $s \in T_i$. This *pseudo-reward* tells how desirable each of the terminal states is for this particular subtask. The initial decomposition does not include the Option level. The Options are constructed and inserted into the task graph automatically during learning. Then, the Options play the same roles as other sub-tasks.

Each primitive action a is a primitive subtask in this decomposition, such that a is always executable, it terminates immediately after execution, and it's *pseudo-reward* function uniformly is zero. The projected value function V^{π} is the value of executing hierarchical policy starting in state s, and at the root of the hierarchy. The completion function ($C^{\pi}(i, s, a)$) is the expected cumulative discounted reward of completing subtask M_i after invoking the subroutine for subtask M_a in state s. The (optimal) value function $V_t(i, s)$ for doing task i in state s is calculated by decomposing it into two parts as in (1): the value of the subtask which is independent of the parent task, and the value of the completion of the task, which of course depends on the parent task.

$$V^{\pi}(i,s) = \begin{cases} \max_a Q^{\pi}(i,s,\pi_i(s)) & \text{if } i \text{ is composite} \\ \sum_{s'} P(s'|s,i)R(s'|s,i) & \text{if } i \text{ is primitive} \end{cases} \qquad (1)$$

$$Q^{\pi}(i,s,a) = V^{\pi}(a,s) + C^{\pi}(i,s,a)$$

where $Q_t(i, s, a)$ is the action value of doing subtask a in state s in the context of parent task i.

In cooperative level, The joint completion function for robot j, $C^j(i, s, a^1,..., a^{j-1}, a^{j+1},...a^n, a^j)$, is the expected discounted cumulative reward of completing cooperative subtask i after taking subtask a^j in state s while other robots performing subtasks a^k, $\forall k \in \{1,\ldots, n\}$, $k \neq j$. The reward is discounted back to the point in time where a^j begins execution. (a^1,..., a^n) is a joint-action in the action set of i. More precisely, the decomposition equations used for calculating the value function V for cooperative subtask i of robot j have the forms as in (2),

$$\begin{aligned} V^j(i,s,a^1,...,a^{j-1},a^{j+1},...a^n) &= Q^j(i,s,a^1,...,a^{j-1},a^{j+1},...,a^n,\pi_i^j(s)) \\ Q^j(i,s,a^1,...,a^{j-1},a^{j+1},...,a^n) &= V^j(a^j,s)+C^j(i,s,a^1,...,a^{j-1},a^{j+1},...,a^n) \end{aligned} . \quad (2)$$

3 MOMQ Algorithms

According to prior knowledge, the designer firstly constructs the initial MOMQ task graph manually. Then the MOMQ algorithms construct the completed MOMQ task graph and learn it automatically, which can be outlined as follows:

1) Interact with environment and learn using *MOMQ learning algorithm*.
2) If the MOMQ constructing algorithm was invoked previously, then go to 1).
3) Run *state transition graph constructing algorithm*.
4) If there is no new states being encountered for the last Y episodes (task-dependent), then go to 5), else go to 1).
5) Run *MOMQ constructing algorithm*. Go to 1).

Consider an undirected edge-weighted graph $G=(P, E, W)$, P is the set of nodes, each visited state will become a node in the graph, E is the set of edges, and W is the $(|P| - 1) \times (|P| - 1)$ upper triangular matrix of weights. In the initial phase of the MOMQ learning procedure, The *state transition graph* $G=(\{s_0\}, \Phi, [0])$, where, $s_0 \in S$ is the initial state, Φ means empty set. The *state transition graph constructing algorithm* is shown as follows:

```
For each observed transition s_i→s_j (s_i, s_j∈S, s_i≠s_j) Do
  If s_j∉P then
    P ← P ∪ {s_j}; E ← E ∪ {(s_i, s_j)}
    Extend W with [0…0 1 0…0]^T, the ith element is 1.
  Else
    If i>j then  w_ij=w_ij+1  else  w_ji=w_ji+1  End If
  End If
End For
```

The *MOMQ constructing algorithm* is proposed basing the aiNet, an artificial immune network model proposed in [10]. The aiNet can be defined as an edge-weighted graph, not necessarily fully connected, composed of a set of nodes, called cells, and sets of node pairs called edges. Each connected edge has a number assigned, called weight or connection strength. The aiNet can be used for data clustering. However, it has many drawbacks such as its high number of user-defined parameters, its computational cost per iteration $O(p^3)$ with relation to the length p of the input vectors, and the network sensitivity to the suppression threshold. In this paper, we improve the aiNet by initiating the aiNet with vaccine inoculation, i.e. the node information of the state transition graph, and leaving out the clonal procedure. The Ag-Ab affinity is measured by a topological distance metric (dissimilarity) between them. Oppositely, the Ab-Ab affinity is defined by a similarity metric between them. The *MOMQ constructing algorithm* works as follows:

1) Initiate aiNet with vaccine inoculation: set $C = P$, and $S = W$, where, C is a matrix containing all the network cells, and S is the network Ab-Ab affinity matrix.

2) For each antigen i (responding to $s_i \in P$), do:
   a) Determine Ag-Ab affinity matrix D;
   b) Select $\xi\%$ of the highest affinity cells to create a memory cell matrix M;
   c) Eliminate those cells whose affinity is inferior to threshold σ, yielding a reduction in the size of the M matrix;
   d) Calculate S,
   e) Eliminate $s_{ij} < \sigma$ (clonal suppression);
   f) Concatenate C and M, (C=[C;M]);
3) Determine S, and eliminate those cells whose $s_{ij} < \sigma$ (network suppression);
4) Replace $r\%$ of the worst cells;
5) If the network reaches a pre-defined number k of cells then go to 6) else go to 2);
6) $M \leftarrow C$;
7) Each memory cell m_j governs a cluster. The cluster is comprised of the states responding to those antigens of which m_j is their highest affinity cell. One cluster corresponds to an Option.
8) Insert constructed Options into the MOMQ task graph.

In the MOMQ framework, the Q values and the C values can be learned through a standard temporal-difference learning method, based on sample trajectories [7]. One important point to note here is that since subtasks are temporally extended in time, the update rules used here are based on the SMDP (semi-MDP) model. The pseudo code of *MOMQ learning algorithm* is shown as follows, which is similar with the *cooperative HRL algorithm* in [8].

Function MOMQ(Robots j, Task i at the lth level, State s)
  $Seq \leftarrow \{\}$
  If i is a primitive action **Then**
    Execute action i in state s
    receive reward r($s' \mid s, i$)
    observe state s'
    $V_{t+1}^{j}(i,s) \leftarrow (1-\alpha_t^j(i))V_t^j(i,s) + \alpha_t^j(i)r(s' \mid s,i)$ /* $\alpha_t^j(i)$ is the learning rate*/
    Push (*state s, actions in* $\{U_l \mid l$ *is a cooperation level*$\}$ *being performed by the other robots*) onto the front of Seq
  Else
    While i has not terminated **Do**
      If i is a cooperative subtask **Then**
        Choose action a^j according to $\pi_i^j(s, a^1, ..., a^{j-1}, a^{j+1}, ..., a^n)$
        $ChildSeq \leftarrow$ MOMQ(j, a^j, s)
        Observe result state s' and $\hat{a}^1, ..., \hat{a}^{j-1}, \hat{a}^{j+1}, ..., \hat{a}^n$
        $a^* \leftarrow \arg\max_{a' \in A_i}[C_t^j(i, s, \hat{a}^1, ...\hat{a}^{j-1}, \hat{a}^{j+1}, ...\hat{a}^n, a') + V_t^j(a', s')]$
        $N \leftarrow 0$
        For each $(s, a^1, ..., a^{j-1}, a^{j+1}, ..., a^n)$ in $ChildSeq$ from the beginning **Do**
          $N \leftarrow N + 1$

$$C_{t+1}^{j}(i,s,a^1,...,a^{j-1},a^{j+1},...,a^n,a^j) \leftarrow$$
$$(1-\alpha_t^j(i))C_t^j(i,s,a^1,...,a^{j-1},a^{j+1},...,a^n,a^j)$$
$$+\alpha_t^j(i)\gamma^N[C_t^j(i,s',\hat{a}^1,...,\hat{a}^{j-1},\hat{a}^{j+1},...,\hat{a}^n,a^*)+V_t^j(a^*,s')]$$

/* where γ is discount factor */

End For

Else /* i is not a cooperative subtask */

Choose action a^j according to $\pi_i^j(s)$

ChildSeq ← MOMQ(*j*, a^j, *s*)

Observe result state *s'*

$a^* \leftarrow \arg\max_{a' \in A_i}[C_t^j(i,s,a')+V_t^j(a',s')]$

N ← 0

For each *s* in *ChildSeq* from the beginning **Do**

N ← *N* + 1

$$C_{t+1}^{j}(i,s,a^j) \leftarrow (1-\alpha_t^j(i))C_t^j(i,s,a^j)+\alpha_t^j(i)\gamma^N[C_t^j(i,s',a^*)+V_t^j(a^*,s')]$$

End For

End If

Append *ChildSeq* onto the front of *Seq*

s = *s'*

End While

End If

Return *Seq*

End MOMQ

4 Simulation Experiments

Consider the three-robot trash collection domain shown as in Fig. 1. Each robot starts in a random location and learns the task of picking up trash from *T*1, *T*2, and *T*3 and depositing it into the *Dump*. There are six primitive actions in this domain: (a) four navigation actions that move the robot one square *Up*, *Down*, *Left*, or *Right*, (b) a *Pick* action, and (c) a *Put* action. Each action is deterministic. The goal state is reached when trash from *T*1, *T*2, and *T*3 has been deposited in *Dump*. The environment space is partitioned into 64 states by grids. The robots do not know the structure of the environment but they can sense their locations. We apply the MOMQ algorithms to this problem, and compare its performance with the *cooperative HRL algorithm* in the multi-agent MAXQ framework [8].

In the experiments, there is a reward of -1 for each action and an additional reward of +20 for successfully depositing the trash into the *Dump*. There is a reward of -10 if the robot attempts to execute the *Put* or *Pick* actions illegally. If a navigation action would cause the robot to hit a wall, the action is a no-op, and there is only the usual reward of -1. The discount factor is set to γ=0.9. The initial Q-values are set to 0 and the learning rate is a constant α=0.1. A ε-greedy exploration is used for both algorithms, with ε=0.3. The constructing algorithm is initiated if no new state is observed in the last *Y*=2 episodes. Set the values of the parameters in the *MOMQ*

constructing algorithm as follows: ξ=20, r=5, σ=0.1, and k=10, experimentally. Each experiment is repeated ten times and the results averaged.

Using MOMQ framework, we designed the initial task graph, the same as the MAXQ task graph. During learning procedure, the environment is explored and six Options (Opt1,…, Opt6) are automatically constructed for *Navigate* action. The completed MOMQ task graph is shown in Fig.1(b). According to the constructed Options, the state space of the environment is partitioned into six regions as shown in Fig.2(a). The state space of each Option is reduced to below 20% of the whole environment space. As a result, the solving space for the *Navigate* action of the three robots is reduced to about $1/10^{124}$. It is natural that the convergence is sped up in a smaller space. Such advantage is shown in Fig.2.

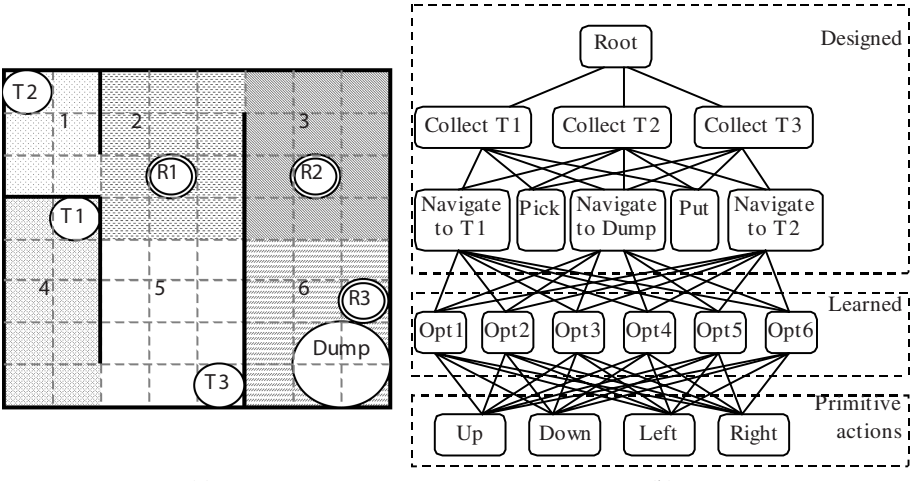

Fig. 1. The constructed results: (a) the constructed Options for *Navigate* action, and (b) the constructed MOMQ task graph

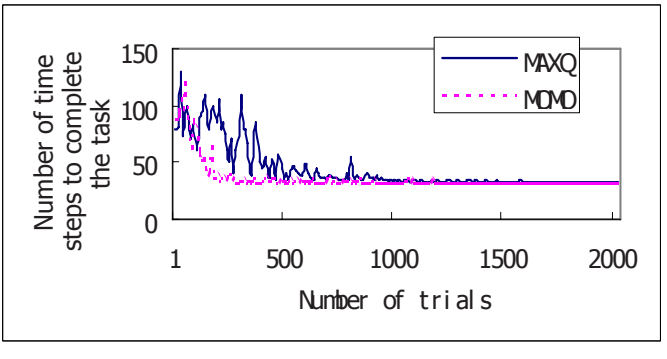

Fig. 2. Performance comparison of MOMQ with MAXQ

5 Conclusions

We described an approach, named MOMQ, for multi-robot cooperation by integrating Options into the MAXQ hierarchical reinforcement learning method. In the MOMQ framework, the MAXQ framework is used to introduce knowledge into reinforcement learning and the Option framework is used to construct hierarchies automatically. The MOMQ is more practical than MAXQ in partial known environment. The advantage performance of MOMQ is demonstrated in three-robot trash collection task and compared with MAXQ. The success of this approach depends of course on providing it with not only a good initial hierarchy but also a good learning ability.

Acknowledgments. This work is supported by the Young Researchers Foundation of Heilongjiang under grant QC06C022, the Fundamental Research Foundation of Harbin Engineering University under grant HEUFT05021, HEUFT05068 and HEUFT07022.

References

1. Cao, Y.U., Fukunaga, A.S., Kahng, A.B.: Cooperative mobile robotics: antecedents and directions. Autonomous Robots, vol.4 (1997) 1-23
2. Fernandez, F., Parker, L.E.: Learning in large cooperative multi-robot domains. International Journal of Robotics and Automation, vol.16, no.4 (2001) 217-226
3. Touzet, C.F.: Distributed lazy Q-learning for cooperative mobile robots. International Journal of Advanced Robotic Systems, vol.1, no.1 (2004) 5-13
4. Yang, E., Gu, D.: Multiagent Reinforcement Learning for Multi-Robot Systems: A Survey. Technical Report CSM-404, University of Essex (2004)
5. Barto, A.G., Mahadevan, S.: Recent advances in hierarchical reinforcement learning. Discrete Event Dynamic Systems: Theory and Applications, vol.13, no.4 (2003) 41-77
6. Sutton, R., Precup, D., Singh, S.: Between MDPs and Semi-MDPs: A framework for temporal abstraction in reinforcement learning. Artificial Intelligence, vol.112 (1999) 181-211
7. Parr, R.: Hierarchical control and learning for Markov decision processes. PhD dissertation, University of California at Berkeley (1998)
8. Dietterich, T.: Hierarchical reinforcement learning with the MAXQ value function decomposition," Journal of Artificial Intelligence Research, vol.13 (2000) 227-303
9. Ghavamzadeh, M., Mahadevan, S., Makar, R.: Hierarchical multiagent reinforcement learning. Journal of Autonomous Agents and Multi-Agent Systems, vol.13 (2006) 197-229
10. de Castro, L.N., Von Zuben, F.N.: An evolutionary immune network for data clustering. Proceedings of the IEEE Brazilian Symposium on Artificial Neural Networks, vol.1, Rio de Janeiro, Brazil (2000) 84-89