Hierarchical Multi-Agent Reinforcement Learning

Rajbala Makar*
Department of Computer Science
Michigan State University
East Lansing, MI 48824
makarraj@cse.msu.edu

Sridhar Mahadevan
Department of Computer Science
Michigan State University
East Lansing, MI 48824
mahadeva@cse.msu.edu

Mohammad Ghavamzadeh
Department of Computer Science
Michigan State University
East Lansing, MI 48824
ghavamza@cse.msu.edu

ABSTRACT

In this paper we investigate the use of hierarchical reinforcement learning to speed up the acquisition of cooperative multi-agent tasks. We extend the MAXQ framework to the multi-agent case. Each agent uses the same MAXQ hierarchy to decompose a task into sub-tasks. Learning is decentralized, with each agent learning three interrelated skills: how to perform subtasks, which order to do them in, and how to coordinate with other agents. Coordination skills among agents are learned by using joint actions at the highest level(s) of the hierarchy. The Q nodes at the highest level(s) of the hierarchy are configured to represent the joint task-action space among multiple agents. In this approach, each agent only knows what other agents are doing at the level of sub-tasks, and is unaware of lower level (primitive) actions. This hierarchical approach allows agents to learn coordination faster by sharing information at the level of sub-tasks, rather than attempting to learn coordination taking into account primitive joint state-action values. We apply this hierarchical multi-agent reinforcement learning algorithm to a complex AGV scheduling task and compare its performance and speed with other learning approaches, including flat multi-agent, single agent using MAXQ, selfish multiple agents using MAXQ (where each agent acts independently without communicating with the other agents), as well as several well-known AGV heuristics like "first come first serve", "highest queue first" and "nearest station first". We also compare the tradeoffs in learning speed vs. performance of modeling joint action values at multiple levels in the MAXQ hierarchy.

1. INTRODUCTION

Consider sending a team of robots to carry out reconnaissance of an indoor environment to check for intruders. This problem is naturally viewed as a multi-agent task [19]. The most effective strategy will require coordination among the individual robots. A natural decomposition of this task would be to assign different parts of the environments, for example rooms, to different robots. In this paper, we are interested in learning algorithms for such *cooperative* multi-agent tasks, where the agents learn the coordination skills by trial and error. The main point of the paper is simply that coordination skills are learned much more efficiently if the robots have a hierarchical representation of the task structure [13]. In particular, rather than each robot learning its response to low-level primitive actions of the other robots (for instance if robot-1 goes forward, what should robot-2 do), it learns high-level coordination knowledge (what is the utility of robot-2 searching room-2 if robot-1 is searching room-1, and so on).

We adopt the framework of reinforcement learning[14], which has been well-studied in both single agent and multi-agent domains. Multi-agent reinforcement learning has been recognized to be much more challenging, since the number of parameters to be learned increases dramatically with the number of agents. In addition, since agents carry out actions in parallel, the environment is usually non-stationary and often non-Markovian as well [9]. We do not address the non-stationary aspect of multi-agent learning in this paper. One approach that has been successful in the past is to have agents learn policies that are parameterized by the modes of interaction [18].

Prior work in multi-agent reinforcement learning can be decomposed into work on competitive models vs. cooperative models. Littman [8], and Hu and Wellman [5], among others, have studied the framework of Markov games for competitive multi-agent learning. Here, we are primarily interested in the cooperative case. The work on cooperative learning can be further separated based on the extent to which agents need to communicate with each other. On the one hand are studies such as Tan [17], which extend flat Q-learning to multi-agent learning by using joint state-action values. This approach requires communication of states and actions at every step. On the other hand are approaches such as Crites and Barto [3], where the agents share a common state description and a global reinforcement signal, but do not model joint actions. There are also studies of multi-agent learning which do not model joint states or actions explicitly, such as by Balch [2] and Mataric [9], among others. In such behavior-based systems, each robot maintains

*Currently at Agilent Technologies, CA.

AGENTS'01, May 28-June 1, 2001, Montréal, Quebec, Canada.

its position in the formation depending on the locations of the other robots, so there is some implicit communication or sensing of states and actions of other agents. There has also been work on reducing the parameters needed for Q-learning in multi-agent domains, by learning action values over a set of derived features[12]. These derived features are domain-specific, and have to be encoded by hand, or constructed by a supervised learning algorithm.

Our approach differs from all the above in one key respect, namely the use of explicit task structure to speed up cooperative multi-agent reinforcement learning. Hierarchical methods constitute a general framework for scaling reinforcement learning to large domains by using the task structure to restrict the space of policies. Several alternative frameworks for hierarchical reinforcement learning have been proposed, including options [15], HAMs [10] and MAXQ [4]. We assume each agent is given an initial hierarchical decomposition of the overall task (as described below, we adopt the MAXQ hierarchical framework). However, the learning is distributed since each agent has only a local view of the overall state space. Furthermore, each agent learns joint abstract action-values by communicating with each other only the high-level subtasks that they are doing. Since high-level tasks can take a long time to complete, communication is needed only fairly infrequently (this is another significant advantage over flat methods).

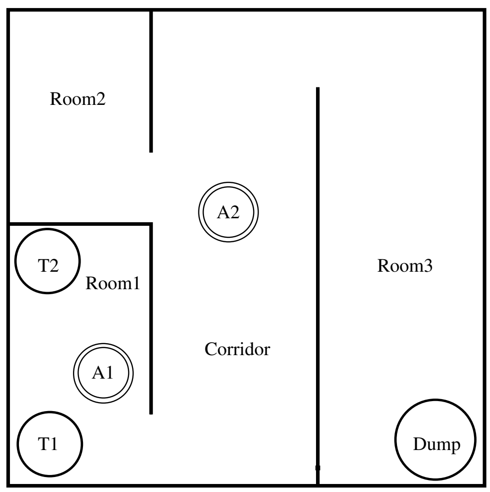
A further advantage of the use of hierarchy in multi-agent learning is that it makes it possible to learn co-ordination skills at the level of abstract actions. The agents learn joint action values only at the highest level(s) of abstraction in the proposed framework. This allows for increased co-operation skills as agents do not get confused by low level details. In addition, each agent has only local state information, and is ignorant about the other agent's location. This is based on the idea that in many cases, an agent can get a rough idea of what state the other agent might be in just by knowing about the high level action being performed by the other agent. Also, keeping track of just this information greatly simplifies the underlying reinforcement learning problem.

These benefits can potentially accrue with using any type of hierarchical learning algorithm, though in this paper we only describe results using the MAXQ framework. The reason that we decided to adopt the MAXQ framework as a basis for our multi-agent algorithm is the fact that the MAXQ method stores the value function in a distributed way in all nodes in the subtask graph. The value function is propagated upwards from the lower level nodes whenever a high level node needs to be evaluated. This propagation enables the agent to simultaneously learn subtasks and high level tasks. Thus, by using this method, agents learn the coordination skills and the individual low level tasks and subtasks all at once.

However, it is necessary to generalize the MAXQ framework to make it more applicable to multi-agent learning. A broad class of multi-agent optimization tasks, such as AGV scheduling, can be viewed as discrete-event dynamic systems. For such tasks, the termination predicate used in MAXQ has to be redefined to take care of the fact that the completion of certain subtasks might depend on the occurrence of an event rather than just a state of the environment. We extended the MAXQ framework to continuous-time MDP models, although we will not discuss this extension in this paper.

2. THE MAXQ FRAMEWORK

The multi-agent reinforcement learning algorithm introduced in this paper is an extension of the MAXQ method for single agent hierarchical learning [4]. This approach involves the use of a graph to store a distributed value function. The overall task is first decomposed into subtasks up to the desired level of detail, and the task graph is constructed. We illustrate the idea using a simple two-robot search task shown in Figure 1. Consider the case where a robot is assigned the task of picking up trash from trash cans over an extended area and accumulating it into one centralized trash bin, from where it might be sent for recycling or disposed. This is a task which can be parallelized, if we have more than one agent working on it. An office (rooms and connecting corridors) type environment is shown in figure. A1 and A2 represent the two agents in the figure. Note the agents need to learn three skills here. First, how to do each subtask, such as navigating to $T1$ or $T2$ or $Dump$, and when to perform $Pickup$ or $Putdown$ action. Second, the agents also need to learn the order to do subtasks (for instance go to $T1$ and collect trash before heading to the $Dump$). Finally, the agents also need to learn how to coordinate with other agents (i.e. $Agent1$ can pick up trash from $T1$ whereas $Agent2$ can service $T2$). The strength of the MAXQ framework (when extended to the multi-agent case) is that it can serve as a substrate for learning all these three types of skills.



T1: Location of one trash can.
T2: Location of another trash can.
Dump: Final destination location for depositing all trash.

Figure 1: A (simulated) multi-agent robot trash collection task.

This trash collection task can be decomposed into subtasks and the resulting task graph is shown in figure 2. The task graph is then converted to the MAXQ graph, which is shown in figure 3. The MAXQ graph has two types of nodes: MAX nodes (triangles) and Q nodes (rectangles), which represent the different actions that can be done under their parents. Note that MAXQ allows learning of shared subtasks. For example, the navigation task Nav is common to several parent tasks.

The multi-agent learning scenario under investigation in this paper can now be illustrated. Imagine the two robots

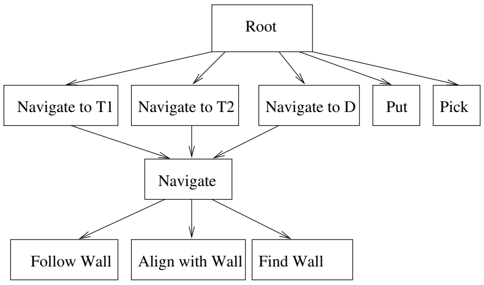


Figure 2: The task graph for the trash collection task.

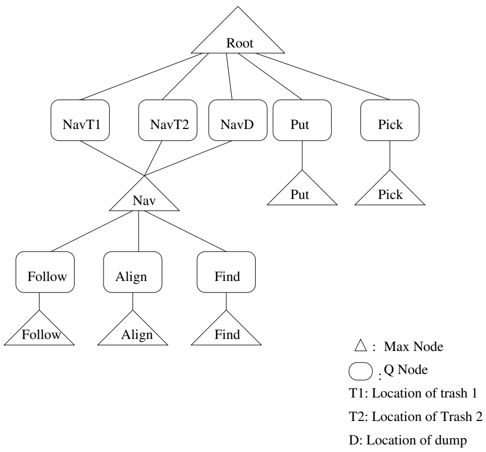


Figure 3: The MAXQ graph for the trash collection task.

start to learn this task with the same MAXQ graph structure. We can distinguish between two learning approaches. In the *selfish* case, the two robots learn with the given MAXQ structure, but make no attempt to communicate with each other. In the *cooperative* case, the MAXQ structure is modified such that the Q nodes at the level(s) immediately under the root task include the joint action done by both robots. For instance, each robot learns the joint Q-value of navigating to trash $T1$ when the other robot is either navigating to $T1$ or $T2$ or $Dump$ or doing a Put or $Pick$ action. As we will show in a more complex domain below, cooperation among the agents results in superior learned performance than in the selfish case, or indeed the flat case when the agents do not use a task hierarchy at all.

More formally, the MAXQ method decomposes an MDP M into a set of subtasks $M_0, M_1 ... M_n$. Each subtask is a three tuple $(T_i, A_i, \overline{R_i})$ defined as:

- $T_i(s_i)$ is a termination predicate which partitions the state space S into a set of active states S_i, and a set of terminal states T_i. The policy for subtask M_i can only be executed if the current state $s \in S_i$.

- A_i is a set of actions that can be performed to achieve subtask M_i. These actions can either be primitive actions from A, the set of primitive actions for the MDP, or they can be other subtasks.

- $\overline{R_i}(s^{'}|s,a)$ is the pseudo reward function, which specifies a *pseudo-reward* for each transition from a state $s \in S_i$ to a terminal state $s^{'} \in T_i$. This *pseudo-reward* tells how desirable each of the terminal states is for this particular subtask.

Each primitive action a is a primitive subtask in the MAXQ decomposition, such that a is always executable, it terminates immediately after execution, and it's *pseudo-reward* function is uniformly zero. The projected value function V^{π} is the value of executing hierarchical policy π 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: 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.

$$V_t(i,s) = \begin{cases} max_a Q_t(i,s,a) & if\ i\ is\ composite \\ \sum_{s'} P(s' \mid s,i) R(s' \mid s,i) & if\ i\ is\ primitive \end{cases}$$

$$Q_t(i,s,a) = V_t(a,s) + C_t(i,s,a) \tag{1}$$

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

The Q values and the C values can be learned through a standard temporal-difference learning method, based on sample trajectories (see [4] for details). One important point to note here is that since subtasks are temporally extended in time, the update rules used here are based on the semi-Markov decision process (SMDP) model [11].

Let us assume that an agent is at state s while doing task i, and chooses subtask j to execute. Let this subtask terminate after N steps and result in state s'. Then, the SMDP Q-learning rule used to update the completion function is given by

$$C_{t+1}(i,s,j) \leftarrow (1-\alpha_t)C(i,s,j) + \alpha_t \gamma^N (\max_{a'} V(a',s') + C_t(i,s',a'))$$

A *hierarchical* policy π is a set containing a policy for each of the subtasks in the problem: $\pi = \{\pi_0 \ldots \pi_n\}$. The projected value function in the hierarchical case, denoted by $V^{\pi}(s)$, is the value of executing hierarchical policy π starting in state s and starting at the root of the task hierarchy. A *recursively optimal* policy for MDP M with MAXQ decomposition $\{M_0 \ldots M_n\}$ is a hierarchical policy $\pi = \{\pi_0 \ldots \pi_n\}$ such that for each subtask M_i the corresponding policy π_i is optimal for the SMDP defined by the set of states S_i, the set of actions A_i, the state transition probability function $P^{\pi}(s', N|s,a)$, and the reward function given by the sum of the original reward function $R(s'|s,a)$ and the pseudo-reward function $\overline{R}_i(s')$. The MAXQ learning algorithm has been proven to converge to π_r^*, the unique recursively optimal policy for MDP M and MAXQ graph H, where M = (S,A,P,R,P_0) is a discounted infinite horizon MDP with discount factor γ, and H is a MAXQ graph defined over subtasks $\{M_0 \ldots M_n\}$.

3. MULTI-AGENT MAXQ ALGORITHM

The MAXQ decomposition of the Q-function relies on a key principle: the reward function for the parent task is the value function of the child task (see Equation 1). We show how this idea can be extended to joint-action values. The most salient feature of the extended MAXQ algorithm, which is proposed in this paper, is that the top level(s) (the level immediately below the root, and perhaps lower levels) of the hierarchy is (are) configured to store the completion function (C) values for joint (abstract) actions of all agents. The completion function $C^j(i, s, a^1, a^2 ... a^j ... a^n)$ is defined as the expected discounted reward of completion of subtask a^j by agent j in the context of the other agents performing subtasks $a^i, \forall i \in \{1, ..., n\}, i \neq j$.

More precisely, the decomposition equations used for calculating the projected value function V have the following form (for agent j)

$$V_t^j(i,s,a^1,\ldots,a^{j-1},a^{j+1},\ldots,a^n) =$$

$$\begin{cases} max_{a^j} Q_t^j(i,s,a^1 \ldots a^j \ldots a^n) \ if \ composite(i) \\ \sum_{s'} P(s' \mid s,i) R(s' \mid s,i) \ if \ primitive(i) \end{cases}$$

$$Q_t^j(i,s,a^1 \ldots a^j \ldots a^n) = V_t^j(a^j,s) + C_t^j(i,s,a^1 \ldots a^j \ldots a^n) \quad (2)$$

at the highest (or lower than the highest as needed) level(s) of the hierarchy, where joint action values are being modeled, and a^j is the action being performed by agent j. Compare the decomposition in Equation 1 with Equation 2. Given a MAXQ hierarchy M for any given task, we need to find the highest level at which this equation provides a sufficiently good approximation of the true value. For both the AGV and the trash collection domain, the subtasks immediately below the root seem to be a good compromise between good performance and reducing the number of joint state action values that need to be learned.

To illustrate the multi-agent MAXQ algorithm, for the two-robot trash collection task, if we set up the joint action-values at only the highest level of the MAXQ graph, we get the following value function decomposition for $Agent1$:

$$Q_t^1(Root, s, NavT1, NavT2) = V_t^1(NavT1, s) + C_t^1(Root, s, NavT1, NavT2)$$

which represents the value of $Agent1$ doing task $NavT1$ in the context of the overall $Root$ task, when $Agent2$ is doing task $NavT2$. Note that this value is decomposed into the value of the $NavT1$ subtask itself and the completion cost of the remainder of the overall task. In this example, the multi-agent MAXQ decomposition embodies the heuristic that the value of $Agent1$ doing the subtask $NavT1$ is independent of whatever $Agent2$ is doing.

A recursive algorithm is used for learning the C values. Thus, an agent starts from the root task and chooses a subtask till it gets to a primitive action. The primitive action is executed, the reward observed, and the leaf V values updated. Whenever any subtask terminates, the $C(i,s,a)$ values are updated for all states visited during the execution of that subtask. Similarly, when one of the tasks at the level just below the root task terminates, the $C(i,s,a^1,\ldots,a^n)$ values are updated according to the MAXQ learning algorithm.

4. THE AGV SCHEDULING TASK

Automated Guided Vehicles (AGVs) are used in flexible manufacturing systems (FMS) for material handling [1]. They are typically used to pick up parts from one location, and drop them off at another location for further processing. Locations correspond to workstations or storage locations. Loads which are released at the dropoff point of a workstation wait at its pick up point after the processing is over so the AGV is able to take it to the warehouse or some other locations. The pickup point is the machine or workstation's output buffer. Any FMS system using AGVs faces the problem of optimally scheduling the paths of AGVs in the system[7]. For example, a move request occurs when a part finishes at a workstation. If more than one vehicle is empty, the vehicle which would service this request needs to be selected. Also, when a vehicle becomes available, and multiple move requests are queued, a decision needs to be made as to which request should be serviced by that vehicle. These schedules obey a set of constraints that reflect the temporal relationships between activities and the capacity limitations of a set of shared resources.

The uncertain and ever changing nature of the manufacturing environment makes it virtually impossible to plan moves ahead of time. Hence, AGV scheduling requires dynamic dispatching rules, which are dependent on the state of the system like the number of parts in each buffer, the state of the AGV and the processing going on at the workstations. The system performance is generally measured in terms of the throughput, the online inventory, the AGV travel time and the flow time, but the throughput is by far the most important factor. In this case, the throughput is measured in terms of the number of finished assemblies deposited at

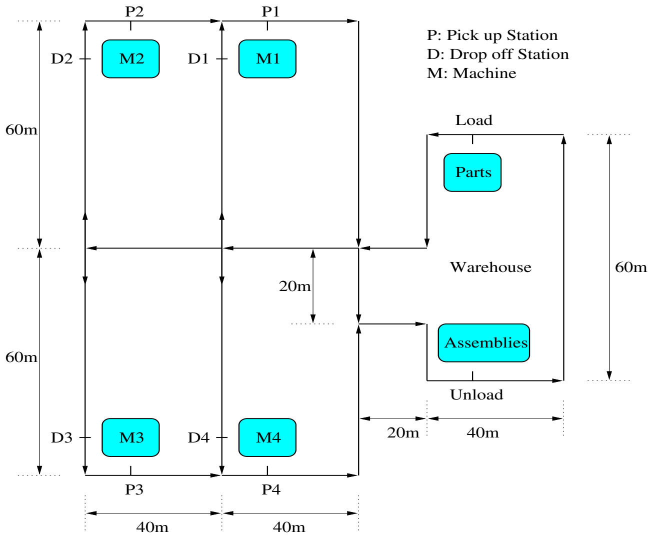
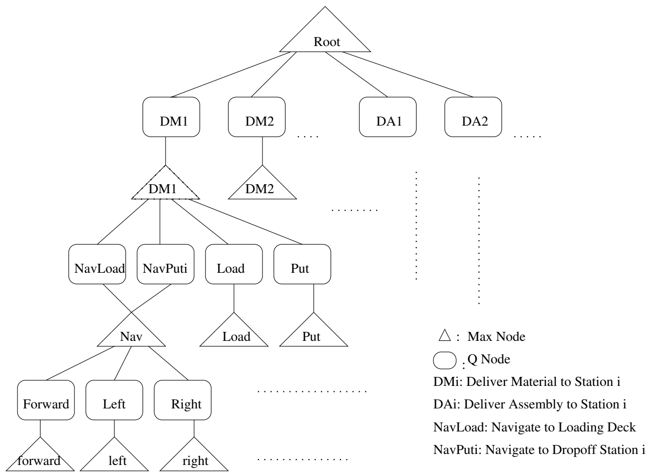


Figure 4: A multiple automatic guided vehicle (AGV) optimization task. There are four AGV agents (not shown) which carry raw materials and finished parts between the machines and the warehouse.



(b)

Figure 5: MAXQ graph for the AGV scheduling task.

the unloading deck per unit time. Since this problem is analytically intractable, various heuristics and their combinations are generally used to schedule AGVs[6, 7]. However, the heuristics perform poorly when the constraints on the movement of the AGVs are reduced.

Previously, Tadepalli and Ok [16] studied a single-agent AGV scheduling task using "flat" average-reward reinforcement learning. However, the multi-agent AGV task we study is more complex. Figure 4 shows the layout of the system used for experimental purposes. Parts of type i have to be carried to drop off station at workstation i and the assembled parts brought back into the warehouse. The AGV travel is unidirectional (as the arrows show).

The termination predicate has been redefined to take care of the fact that the completion of certain tasks might depend on the occurrence of an event rather than just a state of the environment. For example, if we consider the $DM1$ subtask in the AGV problem (see Figure 5), the state of the system at the beginning of the subtask might be the same as that at the end, as the system is very dynamic. New parts continuously arrive at the warehouse, and the machines start and end work on parts at random intervals. Also, the actions of a number of agents affects the environment. This kind of discrete event model makes it necessary to have termination of subtasks to be defined in terms of events. Hence, a subtask terminates when the event associated with that subtask is triggered by the robot performing the subtask, for example DM1 subtask terminates when the *"unload of material 1 at drop off station of machine 1"* event occurs.

4.1 State Abstraction

The state of the environment consists of the number of parts in the pickup station and in the dropoff station of each machine, and whether the warehouse contains parts of each of the four types. In addition, each agent keeps track of its own location and state as a part of the state space. Thus, in the flat case, the size of the state space is ≈ 100 locations, 3 parts in each buffer, 9 possible states of the AGV (carrying Part1, ..., carrying Assembly1, ..., Empty), and 2 values for each part in the warehouse, i.e. $100 \times 4^8 \times 9 \times 2^4 \approx 2^{30}$, which is enormous. The MAXQ state abstraction helps in reducing the state space considerably. Only the relevant state variables are used while storing the completion functions in each node of the task graph. For example, for the *Navigate* subtask, only the location state variable is relevant, and this subtask can be learned with 100 values. Hence, for the highest level with 8 actions, i.e. DM1, ..., DM4, and DA1, ..., DA4, the relevant state variables would be $100 \times 9 \times 4 \times 2 \approx 2^{13}$. For the lower level state space, the action with the largest state space is Navigate with 100 values. This state abstraction gives us a compact way of representing the C functions, and speeds up the algorithm.

5. EXPERIMENTAL RESULTS

We first describe experiments in the simple two-robot trash collection problem, and then we will turn to the more complex AGV task.

5.1 Trash Collection Task

We first provide more details of how we implemented the trash collection task. In the single agent scenario, one robot starts in the middle of Room 1 and learns the task of picking up trash from T1 and T2 and depositing it into the Dump. The goal state is reached when trash from both T1 and T2 has been deposited in Dump. The state space here is the orientation of the robot (N,S,W,E), and another component based on its percept. We assume that a ring of 16 sonars would enable the robot to find out whether it is in a corner, (with two walls perpendicular to each other on two sides of the robot), near a wall (with wall only on one side), near a door (wall on either side of an opening), in a corridor (parallel walls on either side) or in an open area (the middle of the room). Thus, each room is divided into 9 states, and the corridor into 4 states. Thus, we have $((9 \times 3) + 4) \times 4$, or 124 locations for a robot. Also, the trash object from trash basket $T1$ can be at $T1$, carried with robot, or at *Dump*, and the trash object from trash basket $T2$ can be at $T2$, carried by robot, or at *Dump*. Thus the total number of environment states is $124 \times 3 \times 3$, or 1116 for the single agent case. Going to the two-agent case would mean that the trash can be at either $T1$ or $T2$, *Dump*, or carried by one of the two robots. Thus, in the flat case, the size of the state space would grow to $124 \times 124 \times 4 \times 4$, or $\approx 24 \times 10^4$.

The environment is fully observable given this state decomposition, as the direction which the robot is facing, in combination with the percept (which includes the room the agent is in) gives a unique value for each location. The primitive actions considered here are behaviors to find a wall in one of four directions, align with the wall on left or right side, follow wall, enter or exit door, align south or north in the corridor, or move in the corridor.

In the two-robot trash collection task, examination of the learned policy in Figure 6 reveals that the robots have nicely learned all three skills: how to achieve a subtask, what order to do them in, and how to coordinate with other agents. In addition, as Figure 7 confirms, the number of steps needed to do the trash collection task is greatly reduced when the two agents coordinate to do the task, compared to when a single agent attempts to carry out the whole task.

Learned Policy for Agent 1

root
navigate to trash 1
go to location of trash 1 in room 1
pick trash 1
navigate to bin
exit room 1
enter room 3
go to location of dump in room 3
put trash 1 in dump
end

Learned Policy for Agent 2

root
navigate to trash 2
go to location of trash 2 in room 1
pick trash 2
navigate to bin
exit room 1
enter room 3
go to location of dump in room 3
put trash 2 in dump
end

Figure 6: This figure shows the policy learned by the cooperative multi-agent MAXQ algorithm in the trash collection task.

5.2 AGV Domain

We now present detailed experimental results on the AGV scheduling task, comparing several learning agents, includ-

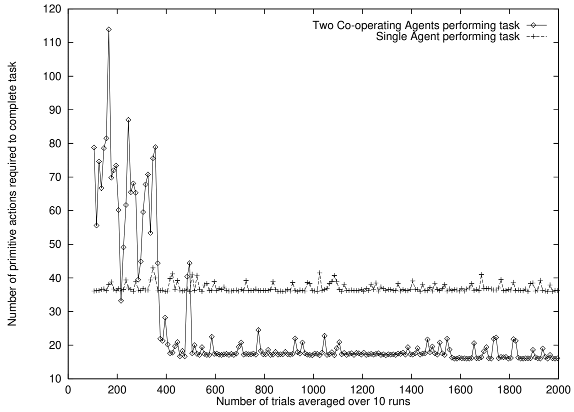


Figure 7: Number of actions needed to complete the trash collection task.

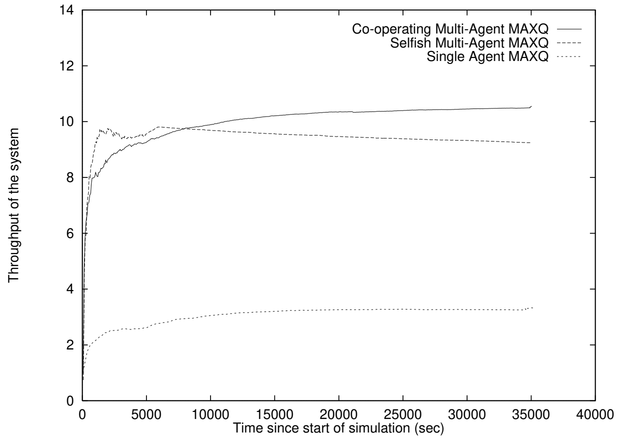


Figure 8: This figure shows that the cooperative multi-agent MAXQ approach outperforms both the selfish (non-cooperative) and single-agent MAXQ approaches when the AGV travel time is very much less compared to the assembly time.

ing a single agent using MAXQ, selfish multiple agents using MAXQ (where each agent acts independently and learns its own optimal policy), and the new co-operative multi-agent MAXQ approach. In this domain, there are four agents (each AGV is an agent).

The experimental results were generated with the following model parameters. The inter-arrival time for parts at the warehouse is uniformly distributed with a mean of 4 sec and variance of 1 sec. The percentage of Part1, Part2, Part3 and Part4 in the part arrival process are 20, 28, 22 and 30 respectively. The time required for assembling the various parts is normally distributed with means 15, 24, 24 and 30 sec for Part1, Part2, Part3 and Part4 respectively, and the variance 2 sec. Each experiment was conducted five times and the results averaged.

Figure 8 shows the throughput of the system for the three types of approaches. As seen in Figure 8, the agents learn a little faster initially in the selfish multi-agent method, but after some time, undulations are seen in the graph showing not only that the algorithm does not stabilize, but also that it results in sub-optimal performance. This is due to the fact

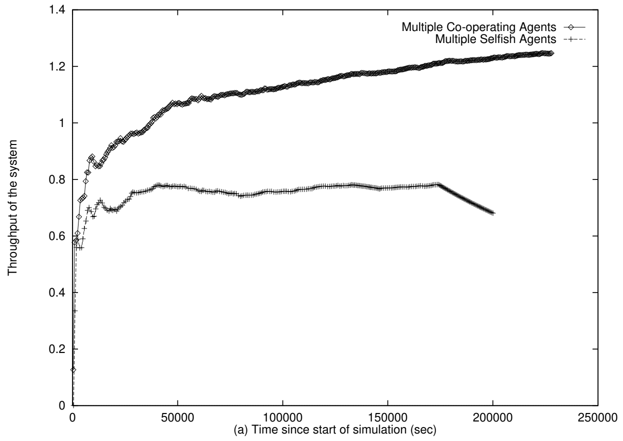


Figure 9: This figure compares the cooperative multi-agent MAXQ approach with the selfish (non-cooperative) MAXQ approach, when the AGV travel time and load/unload time is $\frac{1}{10^{th}}$ the average assembly time.

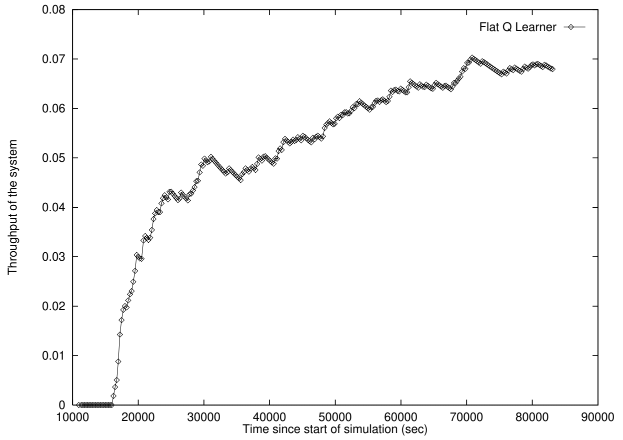


Figure 10: A flat Q-learner learns the AGV domain extremely slowly, showing the need for using a hierarchical task structure.

that two or more agents select the same action, but once the first agent completes the task, the other agents might have to wait for a long time to complete the task, due to the constraints on the number of parts that can be stored at a particular place. The system throughput achieved using the new cooperative multi-agent MAXQ method is significantly higher than the single agent or selfish multi-agent case. This difference is even more significant in figure 9, as when the agents have a longer travel time, the cost of making a mistake is greater.

Figure 10 shows results from an implementation of a single flat Q-Learning agent with the buffer capacity at each station set at 1. As can be seen from the plot on the left, the flat algorithm converges extremely slowly. The throughput at 70,000 sec has gone up to only 0.07, compared with 2.6 for the hierarchical single agent case. Figure 11 compares the cooperative multi-agent MAXQ algorithm with several well-known AGV scheduling rules, showing clearly the improved performance of the reinforcement learning method.

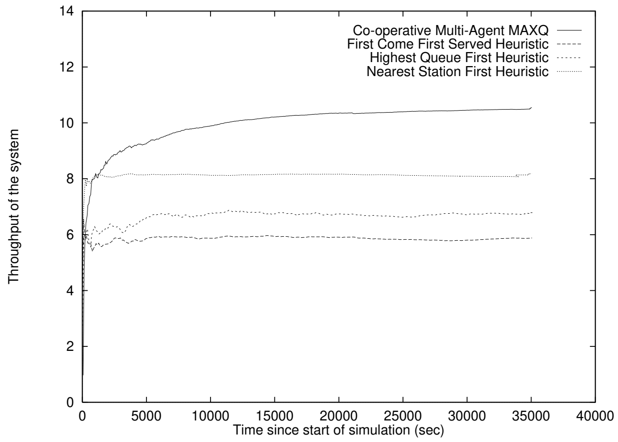


Figure 11: This plot shows the multi-agent MAXQ outperforms three well-known widely used (industrial) heuristics for AGV scheduling.

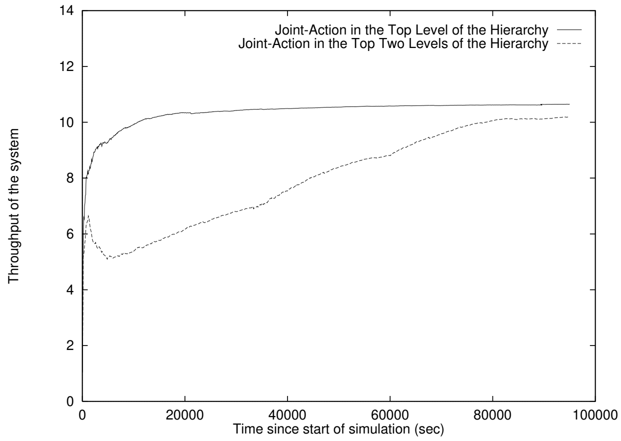


Figure 12: This plot compares the performance of the multi-agent MAXQ algorithm with joint actions at the top level vs. joint actions at the top two levels.

Finally, Figure 12 shows that when the Q-nodes at the top two levels of the hierarchy are configured to represent joint action-values, learning is considerably slower (since the number of parameters is increased significantly), and the overall performance is not better. The lack of improvement is due in part to the fact that the second layer of the MAXQ hierarchy is concerned with navigation. Adding joint actions does not help improve navigation because coordination is not necessary in this environment. However, it might turn out that adding joint actions in multiple layers will be worthwhile, even if convergence is slower, due to better overall task performance.

6. CONCLUSIONS AND FUTURE WORK

We described an approach for scaling multi-agent reinforcement learning by extending the MAXQ hierarchical reinforcement learning method to use joint action values. This decomposition relies on a key principle: the value of a parent task can be factored into the value of a subtask (which is independent of joint action values) and the completion cost (which does depend on joint action values). The effectiveness of this decomposition is most apparent in tasks where agents rarely interact in carrying out cooperative tasks (for example three robots that service a large building may rarely need to exit through the same door at the same time). Since interaction is modeled at an abstract level, coordination skills are learned rapidly. This approach can be easily adapted to constrained environments where agents are constantly running into one another (for example 10 robots in a small room all trying to leave the room at the same time) by using joint action-values at all levels of the hierarchy. However, this will result in a much larger set of action values that need to be learned, and consequently learning will be much slower.

We presented detailed experimental results from a complex AGV scheduling task, which show that the proposed hierarchical cooperative multi-agent MAXQ approach performed better than either the single agent or selfish (non-cooperative) multi-agent MAXQ methods. This novel approach of utilizing hierarchy for learning co-operation skills shows considerable promise as an approach that can be applied to other complex multi-agent domains. We primarily explored the use of the MAXQ hierarchical framework in our study, but we believe that other hierarchical methods could also be adapted to speed up multi-agent learning. The success of this approach depends of course on providing it with a good initial hierarchy.

7. ACKNOWLEDGMENTS

This work is supported by the Defense Advanced Research Projects Agency, DARPA contact No. DAANO2-98-C-4025.

8. REFERENCES

[1] R. Askin and C. Standridge. *Modeling and Analysis of Manufacturing Systems*. John Wiley and Sons, 1993.

[2] T. Balch and R. Arkin. Behavior-based formation control for multi-robot teams. *IEEE Transactions on Robotics and Automation*, 14(6):1–15, 1998.

[3] R. Crites and A. Barto. Elevator group control using multiple reinforcement learning agents. *Machine Learning*, 33:235–262, 1998.

[4] T. Dietterich. Hierarchical reinforcement learning with the MAXQ value function decomposition. *Journal of Artificial Intelligence Research*, volume 13. pages 227–303, 2000.

[5] J. Hu and M. Wellman. Multiagent reinforcement learning: Theoretical framework and an algorithm. In *Fifteenth International Conference on Machine Learning*, pages 242–250, 1998.

[6] C. Klein and J. Kim. Agv dispatching. *International Journal of Production Research*, 34(1):95–110, 1996.

[7] J. Lee. Composite dispatching rules for multiple-vehicle AGV systems. *SIMULATION*, 66(2):121–130, 1996.

[8] M. Littman. Markov games as a framework for multi-agent reinforcement learning. In *Proceedings of the Eleventh International Conference on Machine Learning*, pages 157–163, 1994.

[9] M. Mataric. Reinforcement learning in the multi-robot domain. *Autonomous Robots*, 4(1):73–83, 1997.

[10] R. Parr. *Hierarchical Control and Learning for Markov Decision Processes*. PhD Thesis, University of California, Berkeley, 1998.

[11] M. L. Puterman. *Markov Decision Processes*. Wiley Interscience, New York, USA, 1994.

[12] P. Stone and M. Veloso. Team-partitioned, opaque-transition reinforcement learning. *Third International Conference on Autonomous Agents*, pages 86–91, 1999.

[13] T. Sugawara and V. Lesser. Learning to improve coordinated actions in cooperative distributed problem-solving environments. *Machine Learning*, 33:129–154, 1998.

[14] R. Sutton and A. Barto. *An Introduction to Reinforcement Learning*. MIT Press, Cambridge, MA., 1998.

[15] R. Sutton, D. Precup, and S. Singh. Between MDPs and Semi-MDPs: A framework for temporal abstraction in reinforcement learning. *Artificial Intelligence*, 112:181–211, 1999.

[16] P. Tadepalli and D. Ok. Scaling up average reward reinforcement learning by approximating the domain models and the value function. In *Proceedings of International Machine Learning Conference*, 1996.

[17] M. Tan. Multi-agent reinforcement learning: Independent vs. cooperative agents. In *Proceedings of the Tenth International Conference on Machine Learning*, pages 330–337, 1993.

[18] G. Wang and S. Mahadevan. Hierarchical optimization of policy-coupled semi-markov decision processes. In *Proceedings of the Sixteenth International Conference on Machine Learning*, 1999.

[19] G. Weiss. *Multiagent Systems: A Modern Approach to Distributed Artificial Intelligence*. MIT Press, Cambridge, MA., 1999.