STUDENT PAPER: A Multiagent Reinforcement Learning Algorithm by Dynamically Merging Markov Decision Processes

Mohammad Ghavamzadeh
Department of Computer Science
University of Massachusetts Amherst
Amherst, MA 01003
mgh@cs.umass.edu

Sridhar Mahadevan
Department of Computer Science
University of Massachusetts Amherst
Amherst, MA 01003
mahadeva@cs.umass.edu

PAPER ID: 501

ABSTRACT

One general strategy for accelerating the learning of cooperative multiagent tasks is to reuse (good or optimal) solutions to the task when each agent is acting alone. In this paper, we formalize this approach as dynamically merging solutions to multiple Markov decision processes (MDPs), each representing an individual agent's solution when acting alone, to obtain (good or optimal) solutions to the overall multiagent MDP when all the agents act together. We present a new temporal-difference learning algorithm called MAPLE (MultiAgent Policy LEarning) that uses Q-learning and dynamic merging to efficiently construct global solutions to the overall multiagent problem from solutions to the individual MDPs. We illustrate the efficiency of MAPLE by comparing its performance with standard Q-learning applied to the overall multiagent MDP. We also describe a corresponding planning algorithm that, given complete knowledge of the underlying single agent MDPs, uses dynamic merging to efficiently solve the multiagent MDP. We also illustrate how the dynamic merging framework can be extended to the case when agents use temporally extended actions, by using semi-Markov decision processes (SMDPs) to represent variable-length decision epochs.

1. INTRODUCTION

In this paper, we are interested in planning and learning in cooperative multiagent systems, where 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 agents have already learned the task individually, and use this knowledge to find an efficient multiagent solution.

Consider sending a team of robots to carry out a task such as inspecting an indoor environment for intruders. This problem is naturally viewed as a multiagent task [15]. The most effective strategy will require coordination among individual robots. We are faced with similar teamwork problems in our every day life. Frequently, we know how to perform a task in isolation and we reuse the information we have about the individual execution of the task and combine it with the information of our teammates to efficiently find an optimal solution for doing the task cooperatively.

In this paper, we formulate this problem as that of dynamically merging multiple single agent tasks and present algorithms for efficient planning and learning of multiagent tasks by reusing agents' knowledge about the execution of the same task when acting alone.

We adopt the framework of Markov decision processes [10], which has been well-studied in both single agent and multiagent domains. Planning and learning in multiagent systems is much more challenging since the number of states, actions and parameters to be learned increases exponentially 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 [8]. We do not address these aspects of multiagent planning and learning in this paper. In this paper, we assume that decision-making is synchronous meaning all agents make decisions at the same time.

We model the single agent task as an MDP and its multiagent version as a multiagent MDP. We illustrate how the whole problem could be formulated as that of dynamically merging multiple single agent MDPs into a multiagent MDP. Then, we present an algorithm similar to value iteration, and a new TD learning algorithm called MAPLE (MultiAgent Policy LEarning) for efficient planning and learning in the composite multiagent domains. MAPLE uses Q-learning and dynamic merging to efficiently construct global solutions to the overall multiagent problem from solutions to the individual MDPs.

We can scale up planning and learning and make them more efficient by using high-level (temporally extended) actions instead of low-level (primitive) actions. In this paper, we show that a multiagent system in which agents may choose temporally extended actions can be formulated as a multiagent semi-Markov decision process (multiagent SMDP), if the underlying temporally extended actions executed by all agents are restricted to Markov options. We also illustrate how the multiagent dynamic merging framework and the multiagent learning algorithm proposed in this paper (MAPLE) can be extended to the case when agents use temporally extended actions.

The rest of this paper is organized as follows. We introduce MDPs, multiagent MDPs, and multiagent SMDPs in Section 2. Section 3 introduces the concept of dynamic merging in multiagent systems and describes MAPLE and corresponding planning algorithm for efficient planning and learning through dynamic merging in multiagent domains by exploiting the knowledge of individual agents about the single agent execution of the same task. Section 4 presents experimental results of using the proposed algorithms in a multiagent taxi problem and demonstrate their effectiveness over standard value iteration and Q-learning. Section 5 illustrates how the dynamic merging framework and the proposed algorithms can be extended to the case when agents use temporally extended actions, by using semi-Markov decision processes (SMDPs) to represent variable-length decision epochs. Finally, section 6 summarizes the paper and discusses some directions for future work.

2. SEQUENTIAL DECISION-MAKING FRAMEWORK

Many sequential decision making tasks are naturally formulated as Markov Decision Processes (MDPs). In this section, we begin by presenting the standard (single agent) MDP framework and describe planning and learning using this model. Then we illustrate the case when a collection of agents is controlling the process and briefly present multiagent MDP. We also address the case when agents use temporally extended actions and show how these systems can be modeled by multiagent SMDPs.

2.1 Markov Decision Processes (MDPs)

An MDP is defined as a 4-tuple (S, A, P, R), in which S is a finite set of states, A is a finite set of actions, P is the transition probability function, and R is the reward function. Dynamics of the environment is defined by a transition probability function $P : S \times A \times S \longrightarrow [0, 1]$, where $P^a_{ss'}$ denotes the probability that action a, when executed in state s, transfers the system to state s'. The reward function is defined as a real valued bounded function $R : S \times A \times S \longrightarrow \Re$, where $R^a_{ss'}$ is the reward of taking action a in state s and observing state s' as the next state. In this model, a deterministic policy is defined as a function that assigns an action to each state of the MDP. The value of a state under a policy π, $V^\pi(s)$, is the expected discounted sum of rewards obtained by following policy π starting in that state. The action value of a state under a policy π, $Q^\pi(s, a)$, is the expected discounted sum of reward obtained by taking action a at that state and following policy π afterward. The objective is to find an optimal policy π^* that maximizes the value of every state.

When the model of an MDP is known, the optimal value function is the solution to the Bellman optimality equations: for all $s \in S$,

$$V(s) = \max_{a \in A}(\sum_{s'} P^a_{ss'}[R^a_{ss'} + \gamma V(s')]) \tag{1}$$

The Bellman optimality equations can be solved using iterative algorithms such as value iteration [2, 13]. In this algorithm, we start with an initial guess V_0 and iterate for every state s

$$V_{k+1}(s) = \max_{a \in A}(\sum_{s' \in S} P^a_{ss'}[R^a_{ss'} + \gamma V_k(s')])$$

As k goes to infinity, V_k converges to the value of optimal policy, V^*.

Alternatively, if the model of environment's dynamics is unknown, the optimal action value function can be learned using Q-learning [13].

2.2 Multiagent MDPs

In the last section, we showed that we can find the optimal policy for a single agent task represented by an MDP. However, if multiple similar agents work together in an environment, each is modeled by its own MDP, optimal behavior is not automatically defined. We assume that the total reward of the system is the summation of the rewards of the individual MDPs. Therefore, the goal of the system is to select actions for each MDP at every time step so as to maximize the expected discounted total reward over time. If each agent can choose its action independent of the other agents' actions, then the solution to the composite multiagent MDP is the combination of optimal action for each agent. But the individual actions of agents usually interact with each other, the effect of one agent's action may depend on the actions taken by others or choosing an action by an agent may constrain actions that can be chosen by others.

We model these composite systems as a multiagent MDP [3], which is similar to an MDP, except that actions are distributed among multiple agents. A multiagent MDP is a 5-tuple (α, S, A, P, R), in which α is a finite collection of N agents, $S = S^1 \times ... \times S^N$ is the joint state set and $A = A^1 \times ... \times A^N$ is the joint action set. A joint action $(a^1, ..., a^N)$ represents the concurrent execution of the actions a^i by each agent i. In this model the transition probabilities and rewards are as in the standard MDP, except that they are defined over joint states and joint actions.

Taking the joint state space and joint action space to be the set of basic states and actions, a multiagent MDP can be viewed as a standard MDP. In this model, the optimal policy is defined as optimal joint action for each joint state and can be computed by solving the multiagent MDP using an algorithm like value iteration or can be learned by an algorithm like Q-learning. Since multiagent MDPs usually have huge state and action spaces (because they grow exponentially with the number of agents) and complexity of these algorithms depends on the number of states and actions, thus applying these algorithms directly to solve multiagent MDP would not be very efficient or likely to scale.

2.3 Multiagent Semi-Markov Decision Processes

Hierarchical methods constitute a general framework for scaling planning and learning to large domains by using the task structure to restrict the space of policies. In hierarchical methods, actions are a generalization of primitive actions that include temporally extended courses of action. Temporally extended actions are closed-loop policies for taking action over a period of time, such as picking up an object, getting out of the room, and traveling to a distant city.

Hierarchical methods have been used in cooperative multiagent systems to speed up planning and learning by using explicit task structure [6, 7]. A further advantage of the use of hierarchy in multiagent planning and learning is that it makes it possible to plan and learn coordination skills at the level of high-level (temporally extended) actions, which is faster and more efficient.

In order to use hierarchical methods in multiagent systems, we define the action set of each individual agent as a collection of primitive and high-level actions. In this paper, we model high-level actions using the options framework [14], because this framework defines both temporally extended and primitive actions in a general way. Options consist of three components: a policy $\pi : S \times A \longrightarrow [0, 1]$, a termination condition $\beta : S \longrightarrow [0, 1]$, and an initiation set $I \subseteq S$, where I denotes the set of states $s \in S$ in which the option can be initiated. For any state s, if option π is taken, then primitive actions are selected based on π until it terminates according to β. An option o is a *Markov option* if its policy, initiation set and termination condition depend stochastically only on the current state $s \in S$. It has been shown [14] that a set of temporally extended actions defined over an MDP constitutes a semi-Markov decision process (SMDP), and the theory of SMDPs provides the foundation for the theory of temporally extended actions. Therefore, each single agent task in which the action set is a collection of options can be formulated as an SMDP.

Now consider a multiagent system, in which the action set of each component agent is a collection of primitive and high-level actions modeled by options. This multiagent system with temporally extended actions can be defined similar to the multiagent MDP explained in the last section, except the joint action set is replaced by joint option set, $O = O^1 \times ... \times O^N$. In this model the transition probabilities and rewards are defined over joint states and joint options. Here we assume that decision-making is synchronous, where all agents take actions concurrently at each decision epoch. We do not address asynchronous decision-making in this paper, where agents take actions at individual decision epochs.

In a multiagent system with temporally extended actions, a joint option $(o^1, ..., o^N)$ represents the concurrent execution of options o^i each agent i. We also need to define the event of termination of a joint option to determine the next decision epoch of the multiagent system. When a joint option $o = (o^1, ..., o^N)$ is executed in state s, a set of N options $o^i \in o$ are initiated, each by an agent. Each option will terminate at some random time t_{o^i}. We can define the event of termination for a joint option based on either of the following events: (1) when any of the options $o^i \in o$ executed by an agent terminates, joint option o is declared terminated and the rest of the options being executed by agents that are not terminated at that point in time, are interrupted, or (2) when all of the options are terminated. Rohanimanesh and Mahadevan showed that the set of decision epochs for concurrent options defines a semi-Markov decision process, if the underlying temporally extended actions being parallelized are restricted to Markov options [11]. Similarly, we can show that the set of decision epochs for joint options in a multiagent system with temporally extended actions, with either of the termination conditions presented above, defines a semi-Markov decision process, if the underlying temporally extended actions executed by all agents are restricted to Markov options. This property allows us to use SMDP planning and learning algorithms for planning and learning over joint options.

3. MULTIAGENT DYNAMIC MERGING ALGORITHMS

Given a task, and a collection of agents, each of which has already computed/learned [1] the optimal solution of this task individually, we would like to efficiently compute/learn the optimal solution for the same task when it is being executed by all the agents at the same time. This is the situation where a group of agents that already computed/learned the optimal solution of a specific task in isolation (single agent task), and now decide to do the same task together (multiagent task). The agents want to exploit their prior knowledge about the single agent task to efficiently compute/learn the optimal solution for the multiagent task. More formally, we would like to compute/learn the optimal policy for the multiagent MDP given the optimal solutions for single agent MDPs. We can also relax the problem and say we would like to compute/learn the optimal policy for multiagent task given only upper and lower bounds on the value functions or action value functions of the single agent tasks. Given bounds on the value functions or action value functions of the single agent MDPs, several heuristics can be used to find a policy for the multiagent MDP, however, these policies are usually sub-optimal.

The benefit of using dynamic merging in multiagent systems is to speed up computing/learning the optimal policy in a multiagent MDP by exploiting the prior knowledge about single agent tasks. Moreover, it could be useful to reduce the amount of communication required for coordination in multiagent planning, when the model of the environment is known.

Consider a collection of N agents, each of which has already computed/ learned the optimal solution (or upper and lower bounds on the optimal solution) for a specific task, modeled by a standard MDP. Now, these N agents decide to perform this task together and would like to compute/learn the joint optimal policy of this multiagent task, modeled by a multiagent MDP. The MDP for agent i is defined as $MDP^i = (S^i, A^i, P^i, R^i)$. In the multiagent MDP of the composite multiagent task, the state space is a subset of the joint state space of N agents (because some state variables are independent of the agents), the action space is the joint

[1] These two cases correspond to when the model of the environment's dynamics is known and when the model of the environment's dynamics is unknown.

action space of N agents, the transition probabilities and rewards are factorial and defined for joint states and joint actions as: for all states $s, s' \in S$ and for all actions $a \in A$, $P^a_{ss'} = \prod_{i=1}^N P^{a^i}_{s^i s^{i'}}$ and $R^a_{ss'} = \sum_{i=1}^N R^{a^i}_{s^i s^{i'}}$. In the algorithm proposed in this section, we assume that rewards are non-negative for all transitions. We also assume that the system is fully observable to each agent.

One approach to computing/learning the optimal policy for the multiagent task would be to directly perform value iteration/ Q-learning to solve/learn the multiagent MDP. A more efficient approach would make use of the existing solutions (or upper and lower bounds on the solutions) of the corresponding single agent tasks to solve/learn the composite multiagent task. In the following two subsections, we introduce planning and learning algorithms for doing this.

3.1 Planning Algorithm

In planning, each agent knows the complete and accurate model of the environment's dynamics, the transition probabilities and rewards of single agent MDPs and therefore, the corresponding multiagent MDP. The goal is to compute the optimal policy for the multiagent MDP.

In this case, we can use an algorithm based on one proposed by Singh and Cohn [12] to find the optimal joint policy for the multiagent system. Singh and Cohn assume that an agent knows the optimal solutions (or more general, upper and lower bounds on the optimal value functions) for a collection of sub-tasks (MDPs) and would like to compute the optimal solution for the composite task of executing these sub-tasks simultaneously. They propose a dynamic programming algorithm to exploit the agent's knowledge about the sub-tasks to efficiently compute the optimal solution for the composite task. Pseudo-code for this algorithm is shown in Algorithm 1.

Algorithm 1 Modified Value Iteration Algorithm

1: Initialize $(\forall s \in S)$
2: $L_v(s) = max_{i=1}^N V^{*,i}(s^i, a^i)$
3: $U_v(s) = \sum_{i=1}^N V^{*,i}(s^i, a^i)$
4: $A(s) = A$
5: Initialize s
6: **repeat**
7: $\quad L_v(s) = max_{a \in A(s)}(\sum_{s'} P^a_{ss'}[R^a_{ss'} + \gamma L_v(s')])$
8: $\quad U_v(s) = max_{a \in A(s)}(\sum_{s'} P^a_{ss'}[R^a_{ss'} + \gamma U_v(s')])$
9: $\quad A(s) = \bigcup a \in A(s) \wedge \sum_{s'} P^a_{ss'}[R^a_{ss'} + \gamma U_v(s')]$
$\qquad \geq max_{b \in A(s)} \sum_{s'} P^a_{ss'}[R^a_{ss'} + \gamma L_v(s')]$
10: $\quad s = s' \in S$ such that $\exists a \in A(s), P^a_{ss'} > 0$
11: **until** algorithm converges

This algorithm terminates when only one joint action remains for each joint state, or when the range of all joint actions for any state are bounded by a parameter ϵ.

In the first 4 steps of this algorithm, we define upper and lower bounds on the value function of the multiagent MDP denoted as U_v and L_v respectively, and initialize them as $L_v(s) = max_{i=1}^N L_v^i(s^i)$, and $U_v(s) = \sum_{i=1}^N U_v^i(s^i)$ (U_v^i and L_v^i are the upper and lower bounds on the value function of MDP^i). We also create the initial set of possible joint actions for every joint state. In steps 7 and 8, this algorithm backs up the upper and lower bounds on the value of composite states using a form of equation 1. As successive backups narrow the upper and lower bounds on states, the set of actions that are considered in each state are pruned by eliminating any action whose upper bound value is dominated by the best lower bound value of some other action in the action set of that state (step 9 of the algorithm). One important aspect of this algorithm is that it focuses the backups on states that are reachable on currently available joint actions from the last state (step 10 of the algorithm). It is similar to Real Time Dynamic Programming (RTDP) method [1], which updates states on trajectories through the state space. This property can lead to significant computational savings. Ideas based on prioritized sweeping [9] could also be combined into this algorithm to increase its efficiency.

3.2 Learning Algorithm (MAPLE)

In learning, the model of the environment's dynamics, the transition probabilities and rewards of single agent MDPs and therefore, the corresponding multiagent MDP is unknown to the agents. Agents learn the optimal solution of the problem (modeled by MDP or multiagent MDP) directly from interaction with the environment.

In this case, we assume that there is a set of N agents that already learned the optimal solution (or more generally, upper and lower bounds on the optimal action value function) for a specific single agent task and would like to exploit it to efficiently learn the optimal policy when they all are doing the same task together (the multiagent version of the same task). Therefore, the task is to learn the optimal policy for the multiagent task modeled by a multiagent MDP. If each agent can choose its action independent of the actions of other agents, then the solution to the multiagent MDP is the combination of optimal action for each agent. But usually the effect of one agent's action depends on the actions taken by others or choosing an action by an agent may constrain actions that can be chosen by others. Therefore, each agent should change its local learned policy in order to achieve a multiagent global optimal policy.

The new learning algorithm that we present in this section is called MAPLE (MultiAgent Policy LEarning). It uses Q-learning and dynamic merging to efficiently construct global solutions to the overall multiagent problem from solutions to the individual MDPs.

In MAPLE, we use symbols L_Q and U_Q for the lower and upper bounds on the action value function of the multiagent MDP. We use L_Q^i and U_Q^i as symbols for the lower and upper bounds on the action value function of MDP^i. If the optimal action value function for the i^{th} agent's MDP is available, then $L_Q^i = U_Q^i = Q^{*,i}$.

This algorithm uses upper and lower bounds on action values of single agent MDPs to initialize upper and lower bounds on the action values of the multiagent MDP, and then incrementally updates and narrows these bounds during interaction with the environment using a form of Q-learning that allows pruning the actions whose bounded values are dominated by the bounded value of some other action. Pseudo-code for this algorithm is shown in Algorithm 2.

Algorithm 2 MAPLE

```
1: Initialize (∀s ∈ S, ∀a ∈ A)
2: L_Q(s,a) = max_{i=1}^{N} Q^{*,i}(s^i, a^i)
3: U_Q(s,a) = Σ_{i=1}^{N} Q^{*,i}(s^i, a^i)
4: A(s) = A
5: repeat
6:    Initialize s
7:    repeat
8:       Choose a from A(s) using policy derived from U_Q
         (e.g., ε-greedy)
9:       Take action a, observe reward r and state s'
10:      L_Q(s,a) = (1−α)L_Q(s,a)+α[r+γ max_{a'} L_Q(s',a')]
11:      U_Q(s,a) = (1−α)U_Q(s,a)+α[r+γ max_{a'} U_Q(s',a')]
12:      A(s) = ⋃ a ∈ A(s) ∧ U_Q(s,a)
                         ≥ max_{b∈A(s)} L_Q(s,b)
13:      s ⟵ s'
14:   until s is terminal
15: until algorithm converges
```

This algorithm first computes the initial upper and lower bounds on the action value function in steps 2 and 3. It also initializes the set of possible actions for each state $s \in S$ to the action set of the multiagent MDP, A, in step 4.

During the agents' interaction with the environment, when the algorithm visits a state $s \in S$, it chooses a joint action a from the current set of available actions for state s, $A(s)$, using a policy derived from U_Q and backs up the upper and lower bounds on the value of the chosen action at the current state, using a form of Q-learning backup equation (steps 10 and 11). The set of possible actions in state s is pruned by eliminating any action whose upper bound is worse than the best lower bound (step 12). The algorithm terminates when only one joint action remains for each state, or when the range of all available joint actions for any state are bounded by a predefined parameter ϵ.

An important aspect of this algorithm is that in each state, only remaining actions are backed up and system transfers to states resulting from these remaining actions. Since after each visit of a state, usually many actions are eliminated from the set of possible actions for that state, this property significantly reduces the computational cost and speeds up the learning algorithm.

This algorithm has also a desirable anytime characteristic. If we have to pick an action in state s before the algorithm converges (while multiple joint actions remain in the action set for state s), we pick the action with the highest lower bound. If a new agent (single agent MDP) arrives before the multiagent algorithm converges, it can be accommodated dynamically using whatever lower and upper bounds exist at the time it arrives.

3.3 Analysis of the Learning Algorithm

In this section we analyze various aspects of the proposed learning algorithm, including the upper and lower bound initialization, action selection, pruning actions, and convergence.

Upper Bound Initialization: For any joint state $s = (s^1, ..., s^N)$ and any joint action $a = (a^1, ..., a^N)$, the algorithm initializes the upper bound on the multiagent MDP action value function to the sum of the optimal action value functions of the single agent MDPs,

$$Q^*(s = s^1, ..., s^N, a = a^1, ..., a^N) \leq \sum_{i=1}^{N} Q^{*,i}(s^i, a^i)$$

Since the total reward of the system is the summation of the rewards of single agent MDPs, if there were no constraints among the actions of the single agent MDPs, then $Q^*(s,a)$ would equal $\sum_{i=1}^{N} Q^{*,i}(s^i, a^i)$. The presence of constraints implies that the sum is just an upper bound.

Lower Bound Initialization: For any joint state $s = (s^1, ..., s^N)$ and any joint action $a = (a^1, ..., a^N)$, the algorithm initializes the lower bound on the multiagent MDP action value function to the maximum of the optimal action value functions of the single agent MDPs,

$$Q^*(s = s^1, ..., s^N, a = a^1, ..., a^N) \geq max_{i=1}^{N} Q^{*,i}(s^i, a^i)$$

Again, since the total reward of the system is the summation of the rewards of single agent MDPs, and since rewards are all non-negative, if the agent with the largest optimal action value for state s always chooses its optimal action first and then other agents take their own actions, then the value of the joint action will be equal at least to the optimal action value of that agent, $max_{i=1}^{N} Q^{*,i}(s^i, a^i)$. Therefore, the maximum of the action value functions of the single agent MDPs is the lower bound on the action value function of the multiagent MDP.

Action Selection: At each step, MAPLE chooses action from the set of possible actions for current state, using policy derived from U_Q such as ϵ-greedy.

The MAPLE algorithm prunes the set of possible actions for current state by eliminating any action whose upper bound is worse than the best lower bound. Therefore, after the first visit of a state s, the upper bound on the value of each remained action in the set of possible actions for state s, is not less than the best lower bound. Thus, actions greedy with respect to U_Q would be greedy with respect to both L_Q and U_Q. The initialization of this algorithm guaranties this property even at the first visit of each state.

Action Pruning: For any joint state $s = (s^1, ..., s^N)$, if the upper bound on any joint action, a, is lower than the lower bound on some other joint action, then joint action a cannot be optimal and can safely be discarded from the possible action set of state s and never needs to be reconsidered. This is exactly the criterion used by our algorithm to prune actions. Our algorithm also maintains the upper and lower bound status as it updates them to satisfy the above condition.

Convergence: Given enough time the proposed algorithm converges to the optimal policy and optimal action value function for the multiagent MDP.

If every state action pair is updated infinitely often, Q-learning converges to the optimal policy for the multiagent MDP independent of the initial guess Q_0. The difference between standard Q-learning and our algorithm is that we

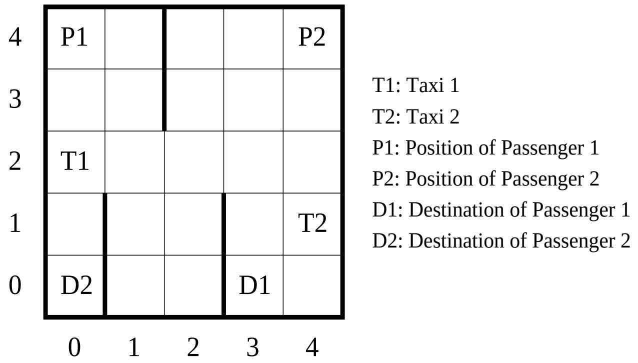

Figure 1: The Multiagent Taxi Domain

discard actions from the set of possible actions for each state. The actions we discard in a state are guaranteed not to be optimal and cannot have any effect on the action value of that state.

4. EXPERIMENTAL RESULTS

We now apply the proposed planning and learning algorithms to a modified version of the well-known taxi problem [5]. The single agent taxi problem used in this paper is a 5-by-5 grid world inhabited by a taxi agent as shown in figure 1. The taxi problem is episodic. In each episode, the taxi starts in a randomly chosen location. There are two specially designated locations in this world for passenger locations (source), $P1$ and $P2$, and two for passenger destinations, $D1$ and $D2$. The passenger in location $P1$ wishes to be transported to location $D1$ and passenger in location $P2$ wishes to be transported to location $D2$. The taxi must go to the location of a passenger, pickup up the passenger, go to its destination location, and deliver the passenger there. The episode ends when passengers are deposited at their destination locations.

There are 75 states (25 locations and whether taxi is empty, carrying passenger1 or carrying passenger2) and 6 actions (move one square in four directions, North, West, South and East, Pickup and Putdown passenger) in this problem. Each action is deterministic. There is a reward of -1 for each action and an additional reward +20 for successfully delivering the passenger. There is a reward of -10 if the taxi attempts to execute the *Putdown* or *Pickup* actions illegally. If a navigation action would cause the taxi to hit the wall, the action is a no-op, and there is only the usual reward of -1. The optimal policy in the single agent task is to find the closer passenger and transport it first.

The multiagent version of the taxi problem is exactly the same as the single agent problem, except now there are two taxis in the environment. The multiagent MDP of the multiagent task has 5625 (75×75) states and 36 (6×6) actions. It is easy to show that the optimal policy for this multiagent task is not the joint optimal policy of individual single agent tasks. Consider the case that both taxis are closer to one passenger than the other, then the joint optimal policy of single agent tasks says that both taxis should move to the location of that passenger to pick it up. But, obviously in this case, only one of the taxis will be able to pickup the passenger and the other one fails and should re-plan to move and pickup the other passenger.

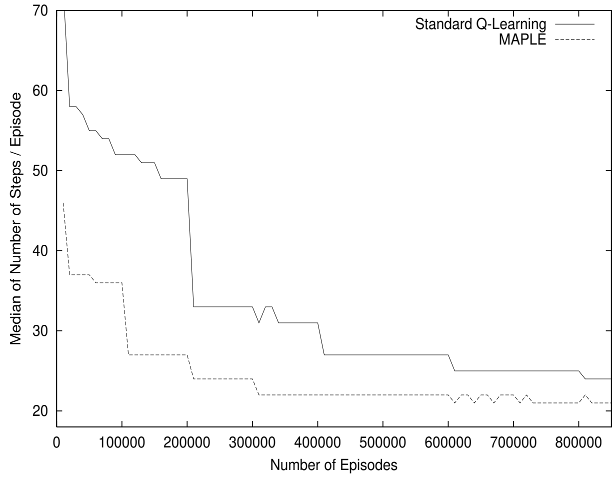

Figure 2: This figure shows that the MAPLE algorithm proposed in this paper learns faster than standard Q-learning in a multiagent taxi problem.

If we apply value iteration directly to the multiagent taxi problem, it converges and computes the optimal policy and optimal value function for the corresponding multiagent MDP after 12 sweeps [2], which is about 67500 ($75 \times 75 \times 12$) backups. Whereas, the modified value iteration algorithm proposed in this paper, initialized with the optimal solution of single agent tasks, solve the multiagent taxi problem using less than 30000 backups. The value iteration algorithm using dynamic merging reduces the number of backups by pruning actions from the set of possible action set of each state and updating states on trajectories through the state space.

Figure 2 compares the performance of MAPLE with standard Q-learning and shows that MAPLE learns faster than standard Q-learning in the multiagent taxi problem. In this figure, the horizontal axis represents the number of episodes, and the vertical axis shows the median of number of steps until success across episodes. For each episode, the median is the median of last 10000 episodes, calculated in terms of the number of actions per episode. In this experiment, we use the optimal action value functions of single agent tasks to initialize the upper and lower bounds on action value function in MAPLE.

In the second experiment, we terminate learning of the single agent taxi problem after a fixed number of episodes (before it converges) and use its non-optimal action value function to initialize the upper and lower bounds on the action value function in MAPLE. Figure 3 compares the performance of MAPLE initialized with non-optimal action value function (non-optimal action value function of single agent tasks when learning is stopped after 200 and 400 episodes) with standard Q-learning in multiagent taxi problem. This figure

[2] A sweep consists of applying a backup operation to each state in the state space.

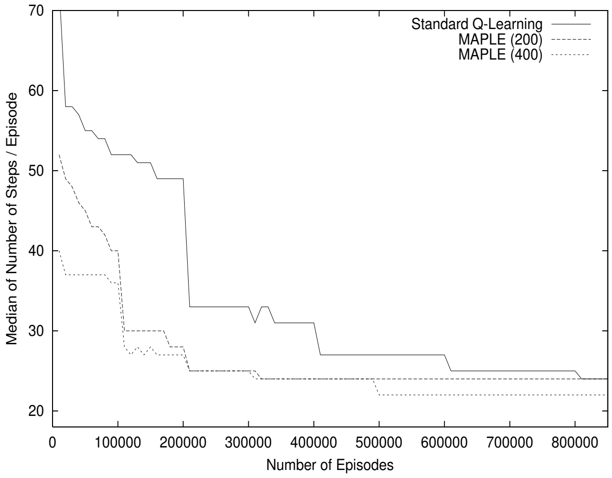

Figure 3: This figure shows that the MAPLE algorithm proposed in this paper learns faster than standard Q-learning in multiagent taxi problem, even when it is initialized with non-optimal action value function on single agent tasks. MAPLE (200) and (400) denote MAPLE initialized with non-optimal action value functions when we terminate learning of single agent tasks after 200 and 400 episodes, respectively.

shows that despite non-optimal initialization of MAPLE, it still learns faster than standard Q-learning.

5. EXTENSION TO MULTIAGENT SMDPS

In section 2.3, we showed that a multiagent system in which agents may choose high-level (temporally extended) actions, can be formulated as a multiagent semi-Markov decision process (multiagent SMDP), if the underlying temporally extended actions executed by all agents are restricted to Markov options. It allows us to apply SMDP learning algorithms such as SMDP-based Q-learning to these multiagent systems. Each joint option is viewed as an indivisible, opaque unit of action. When joint option o is initiated in state s, it transitions to state s' in which joint option o terminates according to one of the termination conditions defined for joint options in section 2.3. We can use the SMDP Q-learning method [4, 14] to update the joint option value function $Q(s, o)$ after each decision epoch where the joint option o is taken in some state s and terminates in s':

$$Q(s,o) \longleftarrow Q(s,o) + \alpha[r + \gamma^k \max_{o' \in O(s')} Q(s',o') - Q(s,o)]$$

where k denotes the number of time steps since initiation of the joint option o at state s and its termination at state s', and r denotes the cumulative discounted reward over this period.

Therefore, we can apply an algorithm similar to MAPLE to accelerate learning of cooperative multiagent tasks with temporally extended actions by reusing (good or optimal) solutions to the task when each agent is acting alone (using temporally extended actions). The algorithm would be similar to original MAPLE, except in step 9 we take joint option o in state s and after k number of time steps it terminates at state s' and we receive reward r, which is the cumulative discounted reward over this period. We should also replace steps 10 and 11 in MAPLE with

$$L_Q(s,o) = (1-\alpha)L_Q(s,o) + \alpha[r + \gamma^k max_{o'} L_Q(s',o')]$$

$$U_Q(s,o) = (1-\alpha)U_Q(s,o) + \alpha[r + \gamma^k max_{o'} U_Q(s',o')]$$

The analysis of the algorithm including upper and lower bound initialization, option selection, option pruning, and convergence remains the same as the original MAPLE algorithm, except now our algorithm converges to the hierarchical optimal policy instead of global optimal policy. Hierarchical optimal policy is a policy that achieves the highest cumulative reward among all policies consistent with the given hierarchy (policy of individual options). If the policy defined for each individual option is optimal, then hierarchical optimal policy will be the same as global optimal policy.

6. CONCLUSIONS AND FUTURE WORK

This paper addresses the problem of efficiently performing a decision-making task using a group of agents as a team, when each agent has already trained for this task in isolation. We formulate the multiagent problem, modeled by a multiagent MDP, as that of dynamically merging single agent tasks, each modeled by a standard MDP. Then, we present a modified Q-learning algorithm called MAPLE (MultiAgent Policy LEarning) for the multiagent MDP resulting from dynamically merging single agent MDPs, analyze its different aspects, prove its convergence, and illustrate its efficiency by comparing its performance with standard Q-learning in the multiagent taxi problem.

Since this algorithm dynamically prunes the set of possible actions for each state, only remaining actions are backed up and system transfers to states resulting from these remaining actions. This property significantly reduces the computational cost and speeds up the learning algorithm, because after each visit to a state, usually many actions are eliminated from the set of possible actions for that state. This algorithm has also a desirable anytime characteristic. If we have to pick an action in state s before the algorithm converges (while multiple joint actions remain in the action set for state s), we pick the action with the highest lower bound. If a new agent (single agent MDP) arrives before the multiagent algorithm converges, it can be accommodated dynamically using whatever lower and upper bounds exist at the time it arrives.

We also described a corresponding planning algorithm similar to value iteration, to solve the multiagent MDP resulting from dynamically merging single agent MDPs, and show its benefits by comparing its computational cost and speed with standard value iteration in multiagent taxi problem.

The key benefit of using the proposed algorithms for planning and learning using dynamic merging in multiagent systems is to speed up computing/learning the optimal policy in a multiagent MDP by exploiting the prior knowledge about single agent tasks. Moreover, the idea of dynamic merging could be useful to reduce the amount of communication required for coordination in multiagent planning, when the model of the environment is known. Assume the case that in a multiagent task, each agent knows only its own state and action and the fact that other agents are similar to it and have already computed the optimal solution of this task individually. In this system, all agents can use the modified value iteration algorithm introduced in section 3.1 to efficiently compute the optimal joint action and find the current state and current action taken by the other agents (of course, agents need to use a protocol to handle cases where more than one optimal joint action is available in a state [3]). Therefore, each agent needs to communicate with others only when it moves to a state, which was unlikely to be the destination of its current action and it happens only in stochastic environments.

In this paper, we also address the issue of using high-level (temporally extended) actions in multiagent systems for more efficient planning and learning. We illustrate that a multiagent system in which agents may choose temporally extended actions can be formulated as a multiagent semi-Markov decision process (multiagent SMDP), if the underlying temporally extended actions executed by all agents are restricted to Markov options. We also describe how the multiagent dynamic merging framework and the multiagent learning algorithm proposed in this paper (MAPLE) can be extended to the case when agents use temporally extended actions.

In the algorithms proposed in this paper, we assume that agents are able to observe the current states and current actions of each other. We believe that the proposed algorithms can be extended to the case when the system is not fully observable by each agent. As future work, we intend to apply the proposed merging algorithms to a complex multiagent task such as AGV (Automated Guided Vehicle) scheduling [7] and investigate their effectiveness in both observable and non-observable systems. Also, extending the concept of dynamic merging into the framework of partially observable Markov decision processes (POMDPs) is particularly interesting and has many applications in both single agent and multiagent domains.

7. REFERENCES

[1] A. G. Barto, S. J. Bradtke, and S. Singh. Learning to act using real-time dynamic programming. *Artificial Intelligence*, Special Volume: Computational Research on Interaction and Agency(72):81–138, November 1995.

[2] D. P. Bertsekas and J. N. Tsitsiklis. *Neuro-Dynamic Programming*. Athena Scientific, Belmont, MA, 1996.

[3] C. Boutilier. Sequential optimality and coordination in multiagent systems. In *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI)*, March 1999.

[4] S. Bradtke and M. Duff. Reinforcement learning methods for continuous-time markov decision problems. *Advances in Neural Information Processing Systems*, 7:393–400, 1995.

[5] T. G. Dietterich. Hierarchical reinforcement learning with the MAXQ value function decomposition. *Journal of Artificial Intelligence Research*, 13:227–303, November 2000.

[6] M. Ghavamzadeh and S. Mahadevan. Continuous-time hierarchical reinforcement learning. In *Proceedings of the Eighteenth International Conference on Machine Learning (ICML)*, pages 186–193, 2001.

[7] R. Makar, S. Mahadevan, and M. Ghavamzadeh. Hierarchical multiagent reinforcement learning. In *Proceedings of the Fifth International Conference on Autonomous Agents*, pages 246–253, 2001.

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

[9] A. W. Moore and C. G. Atkeson. Prioritized sweeping: Reinforcement learning with less data and less real time. *Journal of Machine Learning*, 13(1):1–39, October 1993.

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

[11] K. Rohanimanesh and S. Mahadevan. Decision-theoretic planning with concurrent temporally extended actions. In *Proceedings of the Seventeenth International Conference on Uncertainty in Artificial Intelligence (UAI)*, pages 100–107, August 2001.

[12] S. Singh and D. Cohn. How to dynamically merge markov decision processes. In *Proceedings of the Eleventh International Conference on Neural Information Processing Systems*, March 1999.

[13] R. S. Sutton and A. G. Barto. *Reinforcement Learning An Introduction*. The MIT Press, Cambridge, MA, 1998.

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

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