Learning to Cooperate using Hierarchical Reinforcement Learning

Mohammad Ghavamzadeh
Department of Computing Science
University of Alberta
Edmonton, AB T6G 2E8, Canada
mgh@cs.ualberta.ca

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

ABSTRACT

In this paper, we investigate the use of hierarchical reinforcement learning (HRL) to speed up the acquisition of cooperative multi-agent tasks. We introduce a hierarchical multi-agent RL framework, and present a hierarchical multi-agent RL algorithm called *Cooperative HRL*. The fundamental property of our approach is that the use of hierarchy allows agents to learn coordination faster by sharing information at the level of subtasks, rather than attempting to learn coordination at the level of primitive actions. We study the performance of the *Cooperative HRL* algorithm using a four-agent automated guided vehicle (AGV) scheduling problem. We also address the issue of rational communication behavior among autonomous agents in this paper. The goal is for agents to learn both action and communication policies that together optimize the task given a communication cost. We extend our multi-agent HRL framework to include communication decisions and present a cooperative multi-agent HRL algorithm called *COM-Cooperative HRL*. We demonstrate the efficiency of this algorithm as well as the relation between the communication cost and the learned communication policy using a multi-agent taxi problem.

1. INTRODUCTION

Multi-agent learning has been recognized to be challenging for two main reasons: the *curse of dimensionality*: the number of parameters to be learned increases dramatically with the number of agents, and *partial observability*: states and actions of the other agents which are required for an agent to make a decision are not fully observable and inter-agent communication is usually costly. In this paper, we use hierarchical reinforcement learning (HRL) to address these problems, and to accelerate learning to communicate and act in cooperative multi-agent systems. The key idea underlying our approach is that coordination skills are learned much more efficiently if agents have a hierarchical representation of the task structure. The use of hierarchy speeds up learning in cooperative multi-agent domains by making it possible to learn coordination skills at the level of subtasks instead of primitive actions.

In our hierarchical multi-agent RL framework, we assume learning is distributed, each agent is given a hierarchical decomposition of the overall task, and has only a local view of the overall state space. We define *cooperative subtasks* to be those subtasks in which coordination among agents has significant effect on the performance of the overall task. Agents cooperate at cooperative subtasks and are unaware of their teammates at the other subtasks. Cooperative subtasks are usually defined at the highest level(s) of a hierarchy. Coordination at high-level provides significant advantage over flat methods by preventing agents from getting confused by low-level details and reducing the amount of communication needed for proper coordination among agents.

These benefits can be potentially achieved using any type of HRL algorithm. However, it is necessary to generalize the HRL frameworks to make them more applicable to multi-agent learning. In this paper, initially we assume that communication is free and present a hierarchical multi-agent RL algorithm called *Cooperative HRL*. We study the performance of the *Cooperative HRL* algorithm using a complex four-agent automated guided vehicle (AGV) scheduling problem. We compare its performance and speed with selfish multi-agent HRL and single-agent HRL algorithms. We also demonstrate that the *Cooperative HRL* algorithm outperforms three widely used industrial heuristics for AGV scheduling. Later in the paper, we address the issue of optimal communication, which is important when communication is costly. We generalize the *Cooperative HRL* algorithm to include communication decisions and present a multi-agent HRL algorithm called *COM-Cooperative HRL*. We study the empirical performance of this algorithm as well as the relation between communication cost and the learned communication policy using a multi-agent taxi problem.[1]

2. HIERARCHICAL MULTI-AGENT RL

In this section, we introduce a hierarchical multi-agent RL framework in which agents are capable of learning simultaneously at multiple levels of hierarchy. This is the framework underlying the hierarchical multi-agent RL algorithms presented in this paper. The main contribution of

AAMAS'06 May 8–12 2006, Hakodate, Hokkaido, Japan.
Copyright 2006 ACM 1-59593-303-4/06/0005 ...$5.00.

[1] This paper summarizes our previous work on hierarchical multi-agent RL. For more detailed description of the models and algorithms presented in this paper, and also more experimental results please refer to [3].

this framework is that it enables agents to exploit the hierarchical structure of the task in order to learn coordination strategies more efficiently. Our hierarchical multi-agent RL framework can be viewed as extending the existing single-agent HRL methods, including hierarchies of abstract machines (HAMs) [4], options [8], and MAXQ [2], especially MAXQ, to the cooperative multi-agent setting.

We use a multi-agent taxi problem to illustrate the overall approach. Consider a 5-by-5 grid world inhabited by two taxis T1 and T2 shown in Figure 1. Passengers appear at four stations marked as B, G, R, and Y, and wish to be transported to one of the other stations chosen randomly. Taxis must go to the location of a passenger, pick her up, go to her destination station, and drop her there. This task can be parallelized among the taxis. Taxis need to learn three skills here. **1)** How to do each subtask, such as *navigate* to B or Y. **2)** The order to carry out the subtasks, for example go to a station and pickup a passenger before heading to her destination. **3)** How to coordinate with each other, for example if taxi T1 is on its way to pick up a passenger at B, taxi T2 should serve a passenger at one of the other stations.

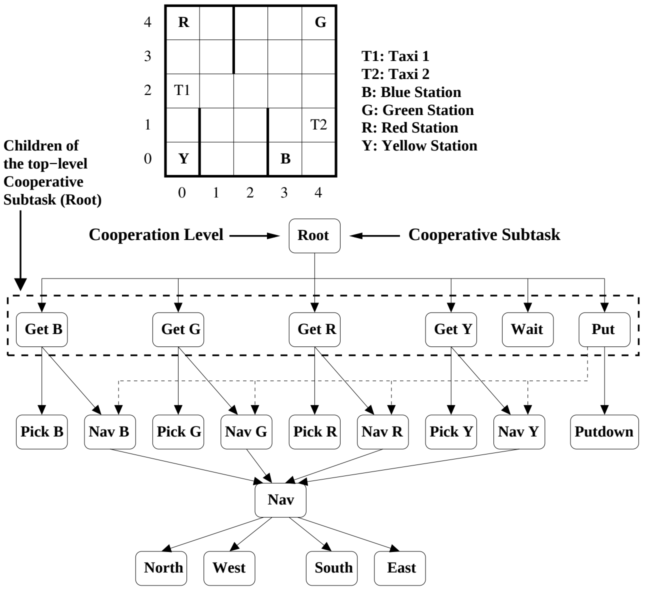

Figure 1: A multi-agent taxi problem and its task graph.

The strength of the HRL methods (when extended to multi-agent domains) is that they can serve as a substrate for efficiently learning all these three skills. In these methods, 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 taxi problem are *Root* (the overall task), *Get* G,...,*Get* Y to pick up passengers, *Put* to drop down passengers, and *Nav* B,...,*Nav* Y for navigation to stations. Each of these subtasks has a set of terminal states, and terminates when it reaches one of its terminal states. 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, navigation subtasks use four primitive actions *North*, *West*, *South*, and *East*. *Get* subtasks use navigation subtasks plus primitive action *Pick*, and so on. All of this information is summarized by a directed acyclic graph called *task graph*. The task graph for the taxi problem is shown in Figure 1. This hierarchical model is able to support *state abstraction* (while a taxi is transporting a passenger, the status of the stations is irrelevant to this process. Therefore, the variables defining the status of the stations can be removed from the state space of the navigation subtasks), and *subtask sharing* (if the system learns how to solve the *Nav* G subtask once, then the solution can be shared by both *Get* G and *Put* G subtasks).

2.1 Multi-agent SMDPs

In the HRL methods, decisions are no longer made at synchronous time steps, as is traditionally assumed in RL. Instead, agent makes decision in epochs of variable length, such as when a distinguishing state is reached (e.g., station G in *navigate* to G subtask), or a subtask is completed (e.g., passenger is dropped off at her destination). A statistical model is available to treat variable length actions: the semi-Markov decision process (SMDP) model [5]. The action duration in an SMDP can depend on the transition that is made. The state of the system may change continually between actions, unlike MDPs where state changes are only due to actions. Therefore, SMDPs have become the main mathematical model underlying the HRL methods.

We now extend the SMDP model to multi-agent domains, and introduce the *multi-agent SMDP* model. We assume agents are cooperative, i.e., maximize the same utility. The individual actions of agents interact in that the effect of one agent's action may depend on the actions taken by the others. When a group of agents perform temporally extended actions, these actions may not terminate at the same time. Therefore, unlike the multi-agent extension of MDP, the MMDP model [1], the multi-agent extension of SMDP requires extending the notion of a decision making event.

Definition 1: A multi-agent SMDP (MSMDP) consists of six components (Υ,$\mathcal{S}$,$\mathcal{A}$,$\mathcal{P}$,$\mathcal{R}$,$\mathcal{T}$) and is defined as follows:

The set Υ is a finite collection of n agents, with each agent $j \in \Upsilon$ having a finite set A^j of individual actions. An element $\vec{a} = \langle a^1, \ldots, a^n \rangle$ of the joint action space $\mathcal{A} = \prod_{j=1}^{n} A^j$ represents the concurrent execution of actions a^j by each agent j. The components $\mathcal{S}$, $\mathcal{R}$, and $\mathcal{P}$ are defined as in an SMDP, the set of states of the system being controlled, the reward function mapping $\mathcal{S} \rightarrow \mathbb{R}$, and the state and action dependent multi-step transition probability function $\mathcal{P} : \mathcal{S} \times \mathbb{N} \times \mathcal{S} \times \mathcal{A} \rightarrow [0, 1]$. Since the components of a joint action are temporally extended actions, they may not terminate at the same time. Therefore, the multi-step transition probability $\mathcal{P}$ depends on how we define decision epochs and as a result, depends on the termination scheme $\mathcal{T}$. □

Three termination strategies τ_{any}, τ_{all}, and $\tau_{continue}$ for temporally extended joint actions were introduced and analyzed in [6]. In τ_{any} termination scheme, when an agent completes an action, all other agents interrupt their actions, the next decision epoch occurs, and a new joint action is selected. In τ_{all} termination scheme, when an agent completes an action, it waits (takes the *idle* action) until all the other agents finish their current actions, the next decision epoch occurs, and all the agents choose the next joint action together. Both these termination strategies are *synchronous*, since all the agents choose actions at every decision epoch. In *synchronous* termination schemes, we need a centralized mechanism to synchronize the agents at decision epochs. In $\tau_{continue}$ termination scheme, when an agent completes an action, the agents whose activities have not completed are not interrupted, next decision epoch occurs only for the agents that completed their actions, and they select their next action given the actions being performed by the other agents. This is an *asynchronous* termination strategy, since only a subset of the agents chooses action at each decision epoch. In *asynchronous* termination schemes, there is no need for a centralized mechanism to synchronize the agents and decision making can take place in a decentralized fashion. Since our goal is to design decentralized multi-agent RL algorithms, we use the $\tau_{continue}$ termination scheme for joint action selection in the hierarchical multi-agent model and algorithms presented in this paper.

2.2 Hierarchical Task Decomposition

A task hierarchy such as the one illustrated in Figure 1 can be modeled by decomposing the overall task MDP M, into a finite set of subtasks $\{M_0, \ldots, M_{m-1}\}$, where M_0 is the *root* task and solving it solves the entire MDP M. Each primitive action is a primitive subtask which is always executable and terminates immediately after execution.

Definition 2: Each *non-primitive* subtask M_i consists of five components $\langle S_i, I_i, T_i, A_i, R_i \rangle$. S_i is the *state space*, which is described by those state variables that are relevant to subtask M_i. $I_i \subset S_i$ and $T_i \subset S_i$ are the *initiation set* and the *set of terminal states* for subtask M_i. Subtask M_i can be initiated only in states belonging to I_i and terminates when it reaches a state in T_i. A_i is the *set of actions* that can be performed to achieve subtask M_i. These actions can be either primitive actions from MDP M, or they can be other subtasks. R_i is the *reward function* of subtask M_i.

If we have a policy for each subtask in the hierarchy, we can define a *hierarchical policy* for the model.

Definition 3: A hierarchical policy π is a set of policies, one policy for each subtask in the hierarchy: $\pi = \{\pi_0, \ldots, \pi_{m-1}\}$.

A hierarchical policy is executed similar to ordinary programming languages. Each subtask policy takes a state and returns the name of a primitive action to execute or a subtask to invoke. When a subtask is invoked, its policy is executed until it enters one of its terminal states. Under a hierarchical policy π, we define a multi-step transition probability function $P_i^\pi : S_i \times \mathbb{N} \times S_i \rightarrow [0,1]$ for each subtask M_i in the hierarchy, where $P_i^\pi(s', N|s)$ denotes the probability that hierarchical policy π will cause the system to transition from state s to state s' in N primitive steps at subtask M_i.

2.3 Multi-agent Setup

In our hierarchical multi-agent framework, we assume that there are n agents in the environment, cooperating with each other to accomplish a task. The designer of the system uses her domain knowledge to recursively decompose the overall task into a collection of subtasks that she believes are important for solving the problem. We assume that agents are *homogeneous*, i.e., all agents are given the same task hierarchy. At each level of the hierarchy, the designer of the system defines *cooperative subtasks* to be those in which coordination among agents 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 lth level *cooperative subtasks* is represented by U_l. Since high-level coordination allows for increased cooperation skills as agents do not get confused by low-level details, we usually define *cooperative subtasks* at the highest level(s) of the hierarchy. Agents actively coordinate while making decision at cooperative subtasks and are ignorant about other agents at non-cooperative subtasks.

For example in the multi-agent taxi problem, it is more effective that an agent learns high-level coordination knowledge (what is the utility of taxi T2 serving the passenger at station B if taxi T1 is serving the one at station G), rather than learning its response to low-level primitive actions of the other agents (what taxi T2 should do if taxi T1 takes action *North*). Therefore, we define *Root* as a *cooperative subtask*. As a result, the top-level of the hierarchy is a *cooperation level*, *Root* is the only member of the *cooperation set* at the top-level, and U_1 consists of all subtasks located at the second level of the hierarchy, $U_1 = \{Get\ B, \ldots, Get\ Y, Wait, Put\}$.

In our hierarchical multi-agent framework, we define single-agent policies for non-cooperative subtasks and joint policies for *cooperative subtasks*.

Definition 4: Under a hierarchical policy π, each non-cooperative subtask M_i can be modeled by an SMDP consisting of components (S_i, A_i, P_i^π, R_i). □

Definition 5: Under a hierarchical policy π, each *cooperative subtask* M_i located at the lth level of the hierarchy can be modeled by an MSMDP as follows:

Υ is the set of n agents in the team. We assume that agents have only local state information and ignore the states of the other agents. Thus, the state space $\mathcal{S}_i$ is defined as the single-agent state space S_i (not joint state space). The action space is joint and is defined as $\mathcal{A}_i = A_i \times (U_l)^{n-1}$. Finally, since we are interested in decentralized control, we use the $\tau_{continue}$ termination strategy. □

In the multi-agent taxi problem, the joint action space for the cooperative subtask *Root*, $\mathcal{A}_{Root}$, is specified as the cross product of its action set A_{Root}, and U_1, $\mathcal{A}_{Root} = A_{Root} \times U_1$. The pros and cons of the presented hierarchical multi-agent framework can be summarized as:

Pros: 1) Using HRL scales learning to problems with large state spaces by using the task structure to restrict the space of policies. **2)** Cooperation among agents is faster and more efficient as agents learn joint action values only at *cooperative subtasks* usually located at the high level(s) of abstraction and do not get confused by low-level details. **3)** Since high-level tasks can take a long time to complete, communication is needed only fairly infrequently. **4)** The complexity of the problem is reduced by storing only the local state information by each agent. It is due to the fact that each agent can get a rough idea of the state of the other agents just by knowing about their high-level actions.

Cons: 1) The learned policy would not be optimal if agents need to coordinate at the subtasks that have not been defined as cooperative. This issue will be addressed in one of the AGV experiments in Section 4, by extending the joint-action model to the lower levels of the hierarchy. Although, this extension provides the cooperation required at the lower levels, it increases the complexity of the learning problem. **2)** If communication is costly, this method might not find an appropriate policy for the problem. We address this issue in Section 5 by including communication decisions in the model. **3)** Storing only local state information by agents is an approximation and may cause sub-optimality. However, it greatly simplifies the underlying multi-agent RL problem.

2.4 Value Function Decomposition

A value function decomposition decompose the value function of *Root* in terms of the value functions of all the subtasks in the hierarchy. In our hierarchical multi-agent framework, we use a value function decomposition similar to the one in MAXQ [2]. The value function of subtask M_i under a hierarchical policy π, $V^\pi(i,s)$, is the expected sum of discounted reward until subtask M_i terminates. Suppose that the policy of subtask M_i, π_i, chooses subtask $\pi_i(s)$ in state s, this subtask executes for a number of steps N and terminates in state s' according to $P_i^\pi(s',N|s,\pi_i(s))$. Now we can write $V^\pi(i,s)$ in the form of a Bellman equation:

$$V^\pi(i,s) = V^\pi(\pi_i(s),s) + \sum_{s',N} P_i^\pi(s',N|s,\pi_i(s))\gamma^N V^\pi(i,s') \quad (1)$$

Equation 1 can be restated for the action-value function as

$$Q^\pi(i,s,a) = V^\pi(a,s) + \sum_{s',N} P_i^\pi(s',N|s,a)\gamma^N Q^\pi(i,s',\pi_i(s'))$$

The right-most term in this equation is the expected discounted cumulative reward of completing subtask M_i after execution of subtask M_a in state s, and is called the *completion function*. The reward is discounted back to the time where M_a begins execution. Now, we can express the action-value function Q and the value function V as:

$$Q^\pi(i,s,a) = V^\pi(a,s) + C^\pi(i,s,a) \quad (2)$$

$$V^\pi(i,s) = \begin{cases} Q^\pi(i,s,\pi_i(s)) \\ \qquad \text{if } M_i \text{ is a non-primitive subtask} \\ \sum_{s'\in S_i} P(s'|s,i)R(s'|s,i) \\ \qquad \text{if } M_i \text{ is a primitive action} \end{cases}$$

Equations 2 are referred to as the decomposition equations for a hierarchy under a hierarchical policy π. Using these equations, we can recursively calculate all the Q values in a hierarchy in terms of the value functions and the completion functions for the subtasks. The fundamental quantities that must be stored to represent Q values are the completion function values C for non-primitive subtasks and the value functions V for primitive actions.

Now, we show how this single-agent value function decomposition can be modified to formulate the joint value function for *cooperative subtasks*. In our hierarchical multi-agent model, we configure *cooperative subtasks* to store the joint completion function values.

Definition 6: The joint completion function for agent j, $C^j(i,s,a^1,\ldots,a^{j-1},a^{j+1},\ldots,a^n,a^j)$, is the expected discounted cumulative reward of completing *cooperative subtask* M_i after executing subtask a^j in state s while other agents performing subtasks $a^k, \forall k \in \{1,\ldots,n\}, k \neq j$. The reward is discounted back to the time where a^j begins execution. □

In this definition, M_i is a *cooperative subtask* at level l of the hierarchy and $\langle a^1,\ldots,a^n\rangle$ is a joint action in its action set. Each individual action in this joint action belongs to U_l. More precisely, the decomposition equations used for calculating the value function V for *cooperative subtask* M_i of agent j have the following form:

$$V^j(i,s,a^1,\ldots,a^{j-1}, a^{j+1},\ldots,a^n) = Q^j(i,s,a^1,\ldots,a^{j-1},a^{j+1},\ldots,a^n,\pi_i^j(s)) \quad (3)$$

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

For example, the value of taxi T1 performing *Get* R in the context of *Root*, when taxi T2 is executing *Get* Y, $Q^1(\mathit{Root},s,\mathit{Get\ Y},\mathit{Get\ R})$, is decomposed into the value of subtask *Get* R, $V^1(\mathit{Get\ R},s)$, and the completion value of the remainder of the *Root* task, $C^1(\mathit{Root},s,\mathit{Get\ Y},\mathit{Get\ R})$.

One important point to note in Equation 3 is that if subtask a^j is itself a *cooperative subtask* at level $l+1$ of the hierarchy, its value function is defined as a joint value function $V^j(a^j,s,\tilde{a}^1,\ldots,\tilde{a}^{j-1},\tilde{a}^{j+1},\ldots,\tilde{a}^n)$, where $\tilde{a}^1,\ldots,\tilde{a}^{j-1},\tilde{a}^{j+1},\ldots,\tilde{a}^n$ belong to U_{l+1}. In this case, $V^j(a^j,s)$ is replaced by $V^j(a^j,s,\tilde{a}^1,\ldots,\tilde{a}^{j-1},\tilde{a}^{j+1},\ldots,\tilde{a}^n)$ in Equation 3.

3. A HIERARCHICAL MULTI-AGENT RL ALGORITHM

We now use the hierarchical multi-agent RL framework described in Section 2 and present a hierarchical multi-agent RL algorithm, called *Cooperative HRL*. The pseudo code for this algorithm is shown in Algorithm 1. In the *Cooperative HRL* algorithm, the V and C values can be learned through a standard temporal-difference (TD) learning method based on sample trajectories. In this algorithm, an agent starts at *Root* and chooses a subtask until it reaches a primitive action. It executes the primitive action and updates its value function (Line 5). Whenever a non-primitive subtask terminates, its C values are updated for all states visited during the execution of that subtask (Line 27). The V values in the update equation on Line 27 are calculated using

$$V(i,s) = \begin{cases} \max_{a\in A_i} Q(i,s,a) \\ \qquad \text{if } M_i \text{ is a non-primitive subtask} \\ \sum_{s'\in S_i} P(s'|s,i)R(s'|s,i) \\ \qquad \text{if } M_i \text{ is a primitive action} \end{cases} \quad (4)$$

Similarly, when agent j completes execution of subtask $a^j \in A_i$, the joint completion function C of *cooperative subtask* M_i located at level l of the hierarchy is updated for all the states visited during the execution of subtask a^j (Line 17). This update equation indicates that in addition to the states visited during the execution of a subtask in U_l (s and s'), an agent must store the actions in U_l being performed by all the other agents ($a^1,\ldots,a^{j-1},a^{j+1},\ldots,a^n$ in state s and $\hat{a}^1,\ldots,\hat{a}^{j-1},\hat{a}^{j+1},\ldots,\hat{a}^n$ in state s'). Sequence *Seq* is used for this purpose in Algorithm 1.

Algorithm 1 The Cooperative HRL Algorithm.

```
Function Cooperative-HRL(Agent $j$, Task $M_i$ at the $l$th level of the hierarchy, State $s$)
let $Seq = \{\}$ be the sequence of (state-visited, actions in $\bigcup_{k=1}^{L} U_k$ being executed by other agents) while executing $M_i$     /* $L$ is the number of levels in the hierarchy */
if $M_i$ is a primitive action  then
  execute action $M_i$ in state $s$, receive reward $r(s'|s,i)$ and observe state $s'$
  $V_{t+1}^j(i,s) \longleftarrow [1-\alpha_t^j(i)]V_t^j(i,s) + \alpha_t^j(i)r(s'|s,i)$
  push (state $s$, actions in $\{U_l | l$ is a cooperation level$\}$ being performed by the other agents) onto the front of Seq
else      /* $M_i$ is a non-primitive subtask */
  while $M_i$ has not terminated  do
    if $M_i$ is a cooperative subtask  then
      choose subtask $a^j$ using the current exploration policy $\pi_i^j(s, a^1, \dots, a^{j-1}, a^{j+1}, \dots, a^n)$
      let ChildSeq = Cooperative-HRL($j$, $a^j$, $s$), where ChildSeq is the sequence of (state-visited, actions in $\bigcup_{k=1}^{L} U_k$ being performed by the other agents) while executing subtask $a^j$
      observe result state $s'$ and actions in $U_l$ being performed by the other agents $\hat{a}^1, \dots, \hat{a}^{j-1}, \hat{a}^{j+1}, \dots, \hat{a}^n$
      let $a^* = \arg\max_{a' \in A_i}[V_t^j(a', s') +$
                  $C_t^j(i, s', \hat{a}^1, \dots, \hat{a}^{j-1}, \hat{a}^{j+1}, \dots, \hat{a}^n, a')]$

      let $N = 0$
      for each $(s, a^1, \dots, a^{j-1}, a^{j+1}, \dots, a^n)$ in ChildSeq from the beginning  do
        $N = N + 1$
        $C_{t+1}^j(i, s, a^1, \dots, a^{j-1}, a^{j+1}, \dots, a^n, a^j) \longleftarrow$
        $[1-\alpha_t^j(i)]C_t^j(i, s, a^1, \dots, a^{j-1}, a^{j+1}, \dots, a^n, a^j) +$
        $\alpha_t^j(i)\gamma^N[C_t^j(i, s', \hat{a}^1, \dots, \hat{a}^{j-1}, \hat{a}^{j+1}, \dots, \hat{a}^n, a^*) +$
        $V_t^j(a^*, s')]$
      end for
    else      /* $M_i$ is a non-cooperative subtask */
      choose subtask $a^j$ according to the current exploration policy $\pi_i^j(s)$
      let ChildSeq = Cooperative-HRL($j$, $a^j$, $s$), where ChildSeq is the sequence of (state-visited, actions in $\bigcup_{k=1}^{L} U_k$ being performed by the other agents) while executing subtask $a^j$
      observe result state $s'$
      let $a^* = \arg\max_{a' \in A_i}[C_t^j(i, s', a') + V_t^j(a', s')]$
      let $N = 0$
      for each $s$ in ChildSeq from the beginning  do
        $N = N + 1$
        $C_{t+1}^j(i, s, a^j) \longleftarrow [1-\alpha_t^j(i)]C_t^j(i, s, a^j) +$
                  $\alpha_t^j(i)\gamma^N[C_t^j(i, s', a^*) + V_t^j(a^*, s')]$
      end for
    end if
    append ChildSeq onto the front of Seq
    $s = s'$
  end while
end if
return Seq
end Cooperative-HRL
```

4. EXPERIMENTAL RESULTS FOR THE COOPERATIVE HRL ALGORITHM

In this section, we demonstrate the performance of the *Cooperative HRL* algorithm using a complex four-agent AGV scheduling problem, and compare it with selfish multi-agent HRL, where each agent acts independently and learns its own optimal policy, and single-agent HRL algorithms. Automated Guided Vehicles (AGVs) are used in flexible manufacturing systems (FMS) for material handling. Any FMS using AGVs faces the problem of optimally scheduling the paths of the AGVs in the system. Since this problem is analytically intractable, various heuristics are generally used to schedule AGVs. However, the heuristics perform poorly when the constraints on the movement of the AGVs are reduced. The system performance is generally measured in terms of throughput, i.e., the number of finished assemblies deposited at the unloading deck per unit time.

Figure 2 shows the layout of the AGV scheduling domain. M1 to M4 show workstations. Parts of type i have to be carried to the drop-off station at workstation i, D_i, and the assembled parts brought back from the pick-up stations of workstations, P_i's, to the warehouse. The AGV travel is unidirectional (as the arrows show). This task is decomposed using the task graph in Figure 3. Each agent uses a copy of this task graph. We define *Root* as a *cooperative subtask* and the highest level of the hierarchy as a *cooperation level*. Therefore, all subtasks at the second level of the hierarchy $(DM1, \dots, DM4, DA1, \dots, DA4)$ belong to set U_1. Coordination skills among agents are learned using joint action-values at the highest level of the hierarchy.

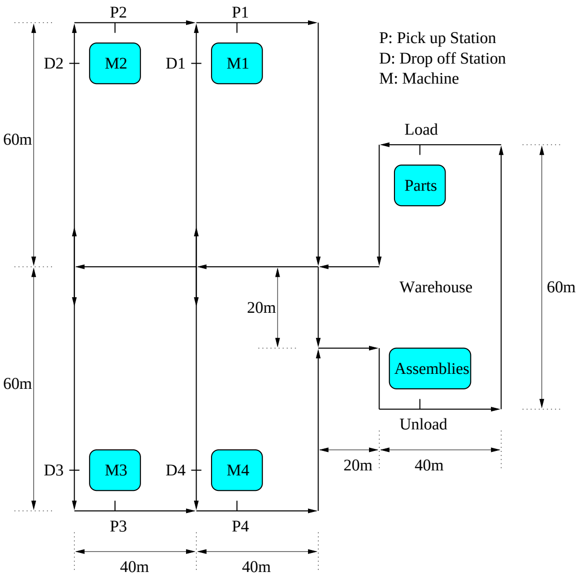

Figure 2: A multi-agent AGV scheduling domain. AGVs (not shown) carry raw materials and finished parts between machines and the warehouse.

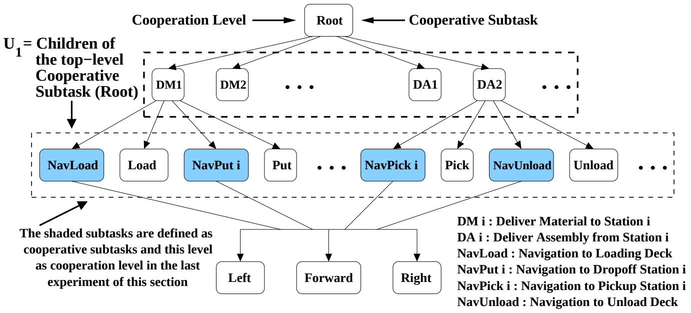
Cooperation Level
Root
Cooperative Subtask
U1 = Children of the top−level Cooperative Subtask (Root)
DM1
DM2
. . .
DA1
DA2
. . .
NavLoad
Load
NavPut i
Put
. . .
NavPick i
Pick
NavUnload
Unload
. . .
The shaded subtasks are defined as cooperative subtasks and this level as cooperation level in the last experiment of this section
Left
Forward
Right
DM i : Deliver Material to Station i
DA i : Deliver Assembly from Station i
NavLoad : Navigation to Loading Deck
NavPut i : Navigation to Dropoff Station i
NavPick i : Navigation to Pickup Station i
NavUnload : Navigation to Unload Deck

Figure 3: Task graph for the AGV scheduling task.

The state of the environment consists of the number of parts in the pick-up and drop-off stations 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 status as a part of its state space. Thus, in the flat case, the state space consists of 100 locations, 8 buffers of size 3, 9 possible states of AGV (carrying part1, ... , carrying assembly1, ... , empty), and 2 values for each part in the warehouse, i.e., $100 \times 4^8 \times 9 \times 2^4 \approx 10^9$ states. State abstraction helps in reducing the state space considerably and speeds up the algorithm. Only the relevant state variables are used while storing the completion function values.

Here we assume that there are four AGVs in the environment. 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 variance 2 sec. The execution time of primitive actions (right, left, forward, load, and unload) is normally distributed with mean 1000 μ-sec and variance 50 μ-sec. The execution time for the idle action is also normally distributed with mean 1 sec and variance 0.1 sec. In these algorithms, learning rate α is set to 0.2, and exploration starts with 0.3, remains unchanged until the performance reaches to a certain level, and then is decreased by a factor of 1.01 every 60 seconds. We use discount factors 0.9, 0.95, and 0.99 in these algorithms. Using discount factor 0.99 yielded better performance in all the algorithms. In this task, each experiment was conducted five times and the results were averaged.

Figure 4 shows that the *Cooperative HRL* algorithm achieves higher throughput than the single-agent HRL and the selfish multi-agent HRL algorithms. As seen in Figure 4, agents learn a little faster initially in the selfish multi-agent method, but after some time the algorithm results in sub-optimal performance. This is due to the fact that two or more AGVs select the same action, but once the first AGV completes the task, the other AGVs 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.

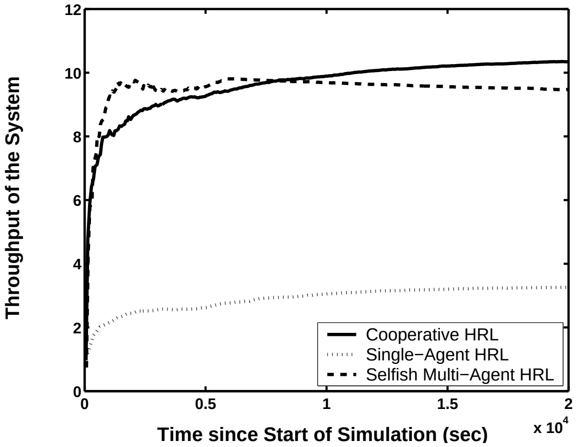

Figure 4: This figure shows that the *Cooperative HRL* algorithm outperforms both the selfish multi-agent HRL and the single-agent HRL algorithms.

Figure 5 compares the *Cooperative HRL* algorithm with several well-known AGV scheduling rules, *highest queue first*, *nearest station first*, and *first come first serve*, showing clearly the improved performance of the HRL method.

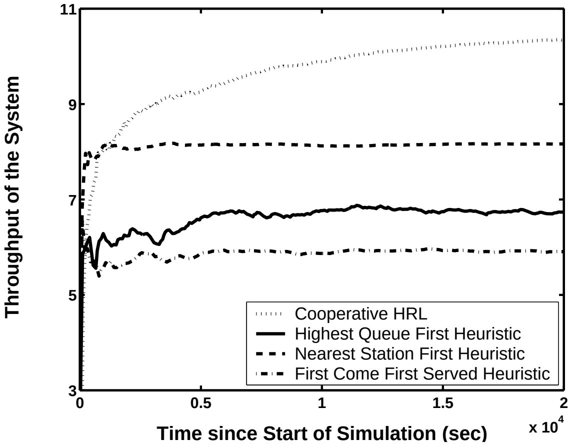

Figure 5: This plot shows that the *Cooperative HRL* algorithm outperforms three widely used industrial heuristics for AGV scheduling.

So far we have only defined *Root* as a *cooperative subtask.* Now, we also define navigation subtasks at the third level of the hierarchy as *cooperative subtasks.* As a result, the third level of the hierarchy is a *cooperation level* and its *cooperation set* contains all the navigation subtasks at that level (see Figure 3). Figure 6 compares the performance of the system in these two cases. When the navigation subtasks are also configured to represent joint actions, 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 AGV travel is unidirectional (see Figure 2), thus coordination at the navigation level does not improve the performance of the system. However, there are problems that having joint actions at multiple levels will be worthwhile, even if convergence is slower, due to better overall performance.

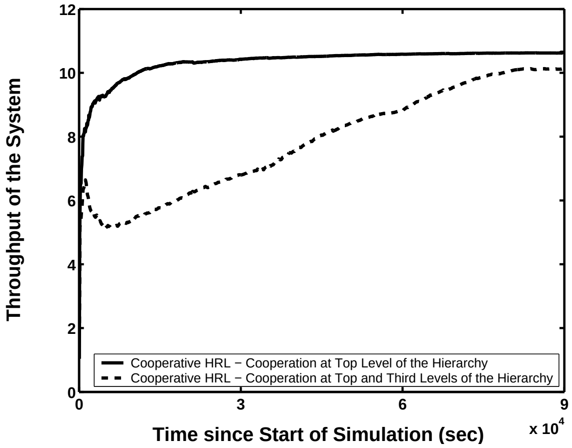

Figure 6: This plot compares the performance of the *Cooperative HRL* algorithm with cooperation at the top level of the hierarchy vs. cooperation at the top and third levels of the hierarchy.

5. HIERARCHICAL MULTI-AGENT RL WITH COMMUNICATION DECISIONS

Communication can be viewed as an action taken by an agent to obtain the local information of its teammates, which may incur a certain cost. In *Cooperative HRL* algorithm, we assume that communication is free, and thus as soon as an agent selects an action at a *cooperative subtask*, it broadcasts it to the team. When communication is not free, it is no longer optimal for a team that agents always broadcast actions taken at their *cooperative subtasks* to their teammates. Therefore, agents must learn to use communication optimally by taking into account its long term return and its immediate cost. We now extend the *Cooperative HRL* algorithm to include communication decisions and present a new algorithm called *COM-Cooperative HRL*.

In the *COM-Cooperative HRL* algorithm, we add a communication level to the task graph of the problem below each *cooperation level*, as shown in Figure 7 for the multi-agent taxi problem. In this algorithm, when an agent is going to make a decision at an lth level *cooperative subtask*, it first decides whether to communicate (takes *Communicate* action) with the other agents to acquire their actions in U_l, or do not communicate (takes *Not-Communicate* action) and selects its action without inquiring new information about its teammates. Agents decide about communication by comparing the expected value of communication $Q(Parent(Com), s, Com)$ with the expected value of not communicating with the other agents $Q(Parent(NotCom), s, NotCom)$. If agent j decides not to communicate, it chooses an action like a selfish agent by using its action-value (not joint action-value) function $Q^j(NotCom, s, a)$, where $a \in Children(NotCom)$. Upon the completion of the selected action, the expected value of not communicate $Q^j(Parent(NotCom), s, NotCom)$ is updated using the sum of all rewards received during the execution of this action. When agent j decides to communicate, it inquires the actions in U_l being performed by all other agents. Then it uses its joint action-value (not action-value) function $Q^j(Com, s, a^1, \ldots, a^{j-1}, a^{j+1}, \ldots, a^n, a)$, $a \in Children(Com)$ to select its next action in U_l. When the selected action terminates, the expected value of communication $Q^j(Parent(Com), s, Com)$ is updated using the sum of all rewards received during the execution of this action plus the communication cost.

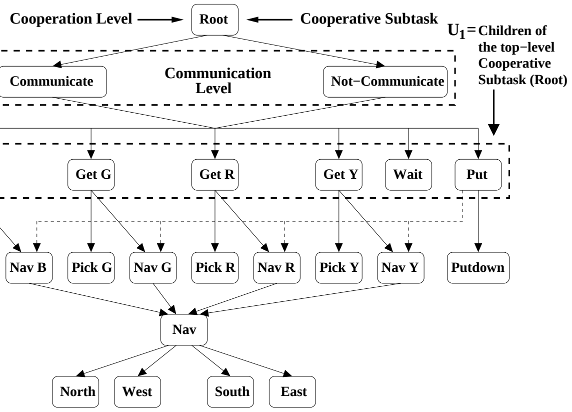

Figure 7: Task graph of the multi-agent taxi problem with communication actions.

In *COM-Cooperative HRL*, we assume that when an agent decides to communicate, it communicates with all the other agents as described above. We can make the model more complicated by making decision about communication with each individual agent. In this case, the number of communication actions would be $C^1_{n-1} + C^2_{n-1} + \ldots + C^{n-1}_{n-1}$, where C^q_p is the number of distinct combinations selecting q out of p agents. For instance, in a three-agent case, communication actions for agent 1 would be *communicate with agent 2*, *communicate with agent 3*, and *communicate with both agents 2 and 3*. It increases the number of communication actions, and therefore the number of parameters to be learned. However, there are methods to reduce the number of communication actions in real-world applications. For example, we can cluster agents based on their role in the team and assume each cluster as a single entity to communicate with. It reduces n from the number of agents to the number of clusters.

In the *COM-Cooperative HRL* algorithm, *Communicate* subtasks are configured to store joint completion function values and *Not-Communicate* subtasks are configured to store completion function values. The joint completion function for agent j, $C^j(Com, s, a^1, \ldots, a^{j-1}, a^{j+1}, \ldots, a^n, a^j)$ is defined as the expected discounted cumulative reward of completing the *cooperative subtask* $Parent(Com)$ after executing subtask a^j in state s, while other agents performing subtasks $a^i, \forall i \in \{1, \ldots, n\}$, $i \neq j$.

In the multi-agent taxi domain, if taxi $T1$ communicates with taxi $T2$, its value function decomposition would be

$$Q^1(Com, s, Get\ Y, Get\ R) = V^1(Get\ R, s) + C^1(Com, s, Get\ Y, Get\ R)$$

which represents the value of taxi $T1$ performing subtask *Get R*, when taxi $T2$ is executing subtask *Get Y*. Note that this value is decomposed into the value of subtask *Get R* and the value of completing subtask $Parent(Com)$ ($Root$) after executing subtask *Get R*. If taxi $T1$ does not communicate with taxi $T2$, its value function decomposition would be

$$Q^1(NotCom, s, Get\ R) = V^1(Get\ R, s) + C^1(NotCom, s, Get\ R)$$

which represents the value of taxi $T1$ performing subtask *Get R*, regardless of the action being executed by taxi $T2$.

In the *COM-Cooperative HRL* algorithm, the V and C values are learned similar to the *Cooperative HRL* algorithm. Completion function values for an action in U_l are updated when we take an action under a *Not-Communicate* subtask, and joint completion function values for an action in U_l are updated when it is selected under a *Communicate* subtask.

6. EXPERIMENTAL RESULTS FOR THE COM-COOPERATIVE HRL ALGORITHM

In this section, we demonstrate the performance of the *COM-Cooperative HRL* algorithm using the multi-agent taxi problem shown in Figure 1. We assume that the task is continuing, passengers appear according to a fixed passenger arrival rate[2] at these four stations and wish to be transported to one of the other stations chosen randomly. The goal here is to increase the throughput of the system, which is measured in terms of the number of passengers dropped off at their destinations per 5,000 time steps, and to reduce

[2] Passenger arrival rate 10 indicates that on average, one passenger arrives at stations every 10 time steps.

the average waiting time per passenger. The state variables in this task are locations of the taxis (25 values each), status of the taxis (5 values each, taxi is empty or transporting a passenger to one of the four stations), and status of the four stations (4 values each, station is empty or has a passenger whose destination is one of the other three stations). The *Cooperative HRL* and *COM-Cooperative HRL* algorithms use the task graphs in Figures 1 and 7 respectively. We used several discount factors, however using discount factor 0.99 yielded better performance in all the algorithms. All the experiments in this section were repeated five times and the results were averaged.

Figure 8 shows the throughput and the average waiting time per passenger for four algorithms, single-agent HRL, selfish multi-agent HRL, *Cooperative HRL*, and *COM-Cooperative HRL* when communication cost is zero. *Cooperative HRL* and *COM-Cooperative HRL* with *ComCost* = 0 have better throughput and average waiting time per passenger than selfish multi-agent HRL and single-agent HRL. The *COM-Cooperative HRL* algorithm learns slower than *Cooperative HRL*, due to more parameters to be learned in this model. However, it eventually converges to the same performance as the *Cooperative HRL* algorithm does.

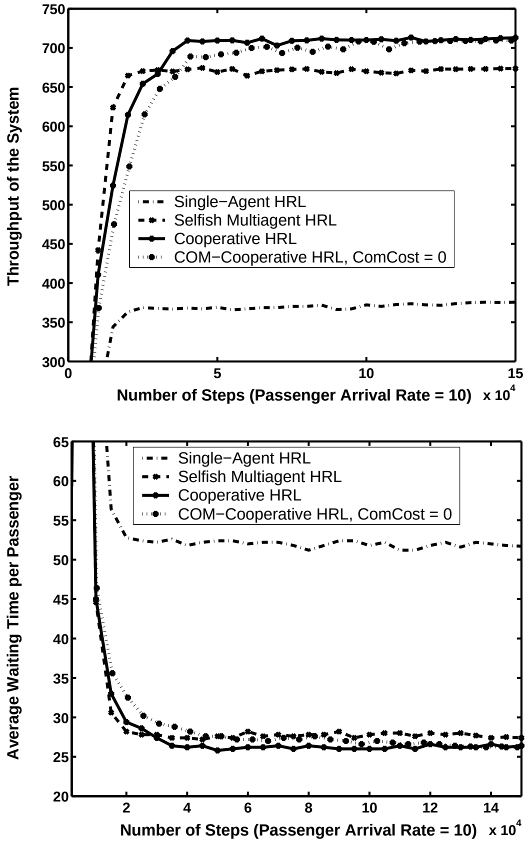

Figure 8: This figure shows that *Cooperative HRL* and *COM-Cooperative HRL* with *ComCost* = 0 have better throughput (top) and average waiting time per passenger (bottom) than selfish multi-agent HRL and single-agent HRL.

Figure 9 demonstrates the relation between the learned communication policy and the communication cost. These two figures show the throughput and the average waiting time per passenger for selfish multi-agent HRL and *COM-Cooperative HRL* when communication cost equals 0, 1, 5, and 10. In both figures, as the communication cost increases, the performance of the *COM-Cooperative HRL* algorithm becomes closer to the performance of the selfish multi-agent HRL algorithm. It indicates that when communication is expensive, agents learn not to communicate and to be selfish.

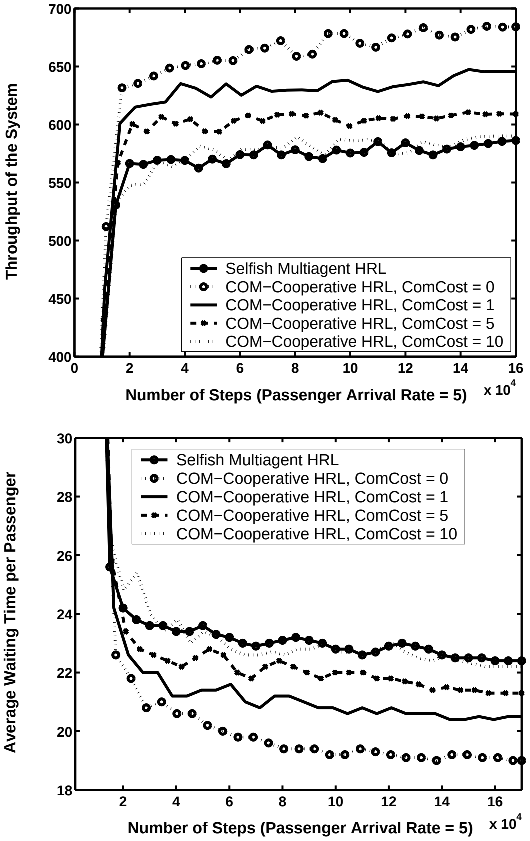

Figure 9: This figure shows that as communication cost increases, the throughput (top) and the average waiting time per passenger (bottom) of the *COM-Cooperative HRL* algorithm become closer to those for the selfish multi-agent HRL algorithm.

7. DISCUSSION AND FUTURE WORK

In this paper, we studied the use of HRL to address the *curse of dimensionality* (joint state space and joint action space problems) and *partial observability* in cooperative multi-agent systems. The key idea underlying our approach is that the use of hierarchy speeds up learning in cooperative multi-agent domains by making it possible to learn coordination skills at the level of subtasks instead of primitive actions. The hierarchical multi-agent models and algorithms presented in this paper address the joint action space problem by allowing the agents to learn joint action values only at *cooperative subtasks* usually located at the high level(s) of hierarchy. They also address the communication problem by allowing coordination at the level of subtasks instead of primitive actions. Since high-level subtasks can take a long time to complete, communication is needed only fairly infrequently. Additionally, the *COM-Cooperative HRL* algorithm presented in this paper optimizes communication by including communication decisions in the hierarchical model. We did not directly address the joint state space problem in this paper. Although the presented models do not prevent us from using joint state space for the subtasks in a hierarchy, we avoided dealing with its complexity by storing only local state information by each agent. However, using hierarchy can alleviate the joint state space problem in cooperative multi-agent systems. First, only *cooperative subtasks* can be defined as joint state space problems. It is an approximation, but it can be a good approximation if agents rarely need to cooperate at non-cooperative subtasks. Second, state abstraction in a hierarchical model can help to reduce the size of the joint state space at a *cooperative subtask*. Third, since each agent can get a rough idea of the state of the other agents just by knowing about their high-level subtasks, it would sometimes be even possible to achieve a reasonably good performance by storing only local state information at *cooperative subtasks*, as shown in the experiments of this paper.

A number of extensions would be useful, from studying the scenario where agents are heterogeneous, to recognizing the high-level subtasks being performed by the other agents using a history of observations (plan recognition and activity modeling) instead of direct communication. In the later case, we assume that each agent can observe its teammates and uses its observations to extract their high-level subtasks. Good examples for this approach are games such as soccer, football, or basketball, in which players often extract the strategy being performed by their teammates using recent observations instead of direct communication. Saria and Mahadevan presented a theoretical framework for online probabilistic plan recognition in cooperative multi-agent systems [7]. We believe that their model can be combined with the learning algorithms presented in this paper to reduce communication by learning to recognize the high-level subtasks being performed by the other agents. In the models presented in this paper, *cooperative subtasks* are predefined by the designer of the system. Another useful extension is to give the agents the ability to discover *cooperative subtasks*, or more general, the ability to decide (or learn to decide) when and with whom to cooperate.

8. REFERENCES

[1] C. Boutilier. Sequential optimality and coordination in multi-agent systems. In *Proceedings of the Sixteenth International Joint Conference on Artificial Intelligence*, pages 478–485, 1999.

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

[3] M. Ghavamzadeh, S. Mahadevan, and R. Makar. Hierarchical multi-agent reinforcement learning. *To appear in Journal of Autonomous Agents and Multi-Agent Systems*, 2006.

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

[5] M. Puterman. *Markov Decision Processes.* Wiley Interscience, 1994.

[6] K. Rohanimanesh and S. Mahadevan. Learning to take concurrent actions. In *Proceedings of the Sixteenth Annual Conference on Neural Information Processing Systems*, 2002.

[7] S. Saria and M. Mahadevan. Probabilistic plan recognition in multi-agent systems. In *Proceedings of the Fourteenth International Conference on Automated Planning and Scheduling*, pages 12–22, 2004.

[8] 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.