Learning to Communicate and Act using Hierarchical Reinforcement Learning

Mohammad Ghavamzadeh & Sridhar Mahadevan
Department of Computer Science, University of Massachusetts Amherst, MA 01003-4610, USA
mgh@cs.umass.edu & mahadeva@cs.umass.edu

Abstract

In this paper, we address the issue of rational communication behavior among autonomous agents. The goal is for agents to learn a policy to optimize the communication needed for proper coordination, given the communication cost. We extend our previously reported cooperative hierarchical reinforcement learning (HRL) algorithm to include communication decisions and propose a new multiagent HRL algorithm, called COM-Cooperative HRL. *In this algorithm, we define* cooperative subtasks *to be those subtasks in which coordination among agents significantly improves the performance of the overall task. Those levels of the hierarchy which include* cooperative subtasks *are called* cooperation levels. *Coordination skills among agents are learned faster by sharing information at the* cooperation levels*, rather than the level of primitive actions. We add a communication level to the hierarchical decomposition of the problem below each* cooperation level. *Before making a decision at a* cooperative subtask, *agents decide if it is worthwhile to perform a communication action. A communication action has a certain cost and provides each agent at a certain* cooperation level *with the actions selected by the other agents at the same level. We demonstrate the efficacy of the* COM-Cooperative HRL *algorithm as well as the relation between the communication cost and the learned communication policy using a multiagent taxi domain.*

1. Introduction

Cooperative multiagent learning studies algorithms for multiple agents coexisting in the same environment to learn how to interact effectively to accomplish a task. The reinforcement learning (RL) framework has been well-studied in cooperative multiagent domains [1, 2, 4, 15]. Multiagent RL has been recognized to be more challenging than single-agent RL for two main reasons: **1)** *curse of dimensionality*: the number of parameters to be learned increases dramatically with the number of agents, and **2)** *partial observability*: states and actions of the other agents which are required for an agent to make decision are not fully observable. Prior work in multiagent RL has addressed the *curse of dimensionality* in different ways. One natural approach is to restrict the amount of information available to each agent and maximize the global payoff by solving local optimization problems [10, 13]. Another approach is to exploit the structure in the multiagent problem using factored value functions [7]. This approach approximates the joint value function as a linear combination of local value functions, each of which relates only to the parts of the system controlled by a small number of agents. Factored value functions allow the agents to find a globally optimal joint-action using a message passing scheme. However, these approaches do not address the communication cost in their message passing strategy.

All the above methods ignore the fact that agents might not have free access to the other agents' information which are required for their decision making. In general, the world is *partially observable* for the agents in distributed multiagent domains. One way to address partial observability in these domains is to use communication to exchange information among agents. However, since communication can be costly, in addition to its normal actions, each agent needs to decide about communicating with other agents [11, 16]. The trade-off between the quality of solution and the communication cost is currently a very active area of research in multiagent learning and planning.

In our previous work [9], we introduced a different approach to address *curse of dimensionality* and *partial observability* in cooperative multiagent systems. The key idea underlying the approach is that coordination skills are learned much more efficiently if the agents have a hierarchical representation of the task structure. Agents have only a local view of the overall state space, and learn joint abstract action-values by communicating with each other only the high-level subtasks that they are doing. It reduces the number of parameters to be learned. Furthermore, since high-level subtasks can take a long time to complete, communication is needed only fairly infrequently and this is a significant advantage over flat techniques. Although the hierarchical RL (HRL) algorithm proposed in that work reduces the amount of communication required for coordination among agents, it does not address the issue of optimal communication, which is important when communication is costly. In this paper, we generalize our previous algorithm to include communication decisions and propose a new multiagent HRL algorithm, called *COM-Cooperative HRL*. The goal is to derive both action and communication policies that together optimize the task given the communication cost. In this algorithm, we define *cooperative subtasks* to be those subtasks in which coordination among agents has significant effect on the performance of the overall task. Those levels of the hierarchy on which *cooperative subtasks* lie are called *cooperation levels*. Agents learn coordination skills by sharing information at *cooperation levels*, rather than the level of primitive actions. We add a communication level to the hierarchical decomposition of the problem below each *cooperation level*. A communication action has a certain cost and provides each agent at a certain *cooperation level* with the actions selected by the other agents at the same level. We demonstrate the efficacy of the *COM-Cooperative HRL* algorithm as well as the relation between communication cost and communication policy in a multiagent taxi domain.

2. Hierarchical Multiagent RL Framework

In this section, we illustrate the hierarchical multiagent RL framework underlying the *COM-Cooperative HRL* algorithm proposed in this paper. Our HRL framework builds upon the MAXQ value function decomposition [5], and the options model [14].

2.1. Hierarchical Task Decomposition

To illustrate our hierarchical multiagent RL framework and algorithm, we present a multiagent taxi problem, which will also be used in the experiments of this paper. Consider a 5-by-5 grid world inhabited by two taxis ($T1$ and $T2$) shown in Figure 1. There are four specially designated locations in this domain, marked as B(lue), G(reen), R(ed) and Y(ellow). The task is continuing, passengers appear according to a fixed passenger arrival rate[1] at these four locations and wish to be transported to one of the other locations chosen randomly. Taxis must go to the location of a passenger, pick up the passenger, go to its destination location, and drop the passenger there. 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 5000 time steps, and to reduce the average waiting time per passenger.

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

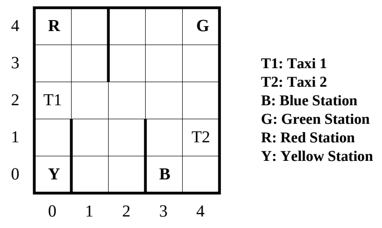


Figure 1. A multiagent taxi domain.

Hierarchical RL methods provide a general framework for scaling RL to problems with large state spaces by using the task structure to restrict the space of policies. In these methods, the overall task is decomposed into a collection of subtasks that are important for solving the problem. Each of these subtasks has a set of termination states, and terminates when one of its termination states is reached. Each primitive action (*North*, *West*, *South*, *East*, *Pickup* and *Putdown*) is a primitive subtask in this decomposition, such that it is always executable and it terminates immediately after execution. On the other hand, non-primitive subtasks such as *root* (the whole taxi problem), *Put*, *Get* B, G, R and Y, *Navigate to* B, G, R and Y, might take more than one time step to complete. 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*. *Put* uses four navigation subtasks plus one primitive action *Putdown*, and so on. All of this information is summarized by the task graph shown in Figure 2.

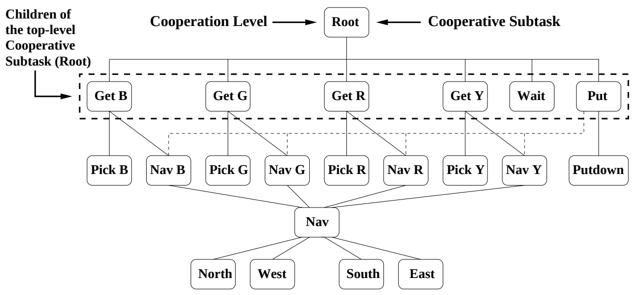


Figure 2. The task graph of the multiagent taxi domain.

2.2. Temporal Abstraction using SMDP

Hierarchical RL studies how lower level policies over subtasks or primitive actions can themselves be composed into higher level policies. Policies over primitive actions are semi-Markov when composed at the next level up, because they can take variable, stochastic amount of time to complete. Thus, semi-Markov decision processes (SMDPs) [8] have become the preferred language for modeling temporally extended actions. SMDPs extend the MDP model in several aspects. Decisions are only made at discrete points in time. State of the system may change continually between decisions, unlike MDPs where state changes are only due to the actions. Thus, the time between transitions may be several time units and can depend on the transition that is made. These transitions are at decision epochs only. Basically, the SMDP represents snapshots of the system at decision points, whereas the so-called *natural process* describes the evolution of the system over all times.

In this section, we extend the SMDP model to multiagent domains, when a team of agents controls the process, and introduce the multiagent SMDP (MSMDP) model. We assume agents are cooperative, i.e., maximize the same utility over an extended period of time. 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 multiagent extension of MDP (MMDP model [1]), the multiagent extension of SMDP is not straight forward.

Definition 1: A MSMDP consists of six components (α,$\mathcal{S}$,$\mathcal{A}$,P,R,τ) and is defined as follows:

The set α is a finite collection of n agents, with each agent $j \in \alpha$ having a finite set A^j of individual actions. An element $\vec{a} = \langle a^1, \dots, a^n \rangle$ 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 $\mathcal{S}$, R and P are as in an SMDP, the set of states of the system being controlled, the reward function mapping $\mathcal{S} \rightarrow \Re$, and the state and action dependent multi-step transition probability function $P : \mathcal{S} \times \mathcal{N} \times \mathcal{S} \times \mathcal{A} \rightarrow [0, 1]$ (where $\mathcal{N}$ is the set of natural numbers). Since individual actions in a joint-action are temporally extended, they may not terminate at the same time. Therefore, the multi-step transition probability function P depends on how we define decision epochs, and as a result, depends on the termination scheme τ that is used in the MSMDP model. Three termination strategies τ_{any}, τ_{all} and τ_{cont} for temporally extended joint-actions were investigated in [12]. In τ_{any} termination scheme, the next decision epoch is when the first action within the joint-action currently being executed terminates, where the rest of the actions that did not terminate are interrupted. When an agent finishes its action, all other agents interrupt their actions, the next decision epoch occurs and a new joint-action is selected. In τ_{all} termination scheme, the next decision epoch is the earliest time at which all the actions within the joint-action currently being executed have terminated. When an agent completes an action, it waits (takes the *idle* action) until all other agents complete their current actions. Then, the next decision epoch occurs and agents choose the next joint-action together. In both these termination strategies, all agents make a decision at every decision epoch. τ_{cont} termination scheme is similar to τ_{any} in the sense that the next decision epoch is when the first action within the joint-action currently being executed terminates. However, the other agents are not interrupted and only terminated agents select new actions. In this termination strategy, only a subset of agents choose action at each decision epoch. When an agent finishes an action , the next decision epoch occurs only for that agent and it selects its next action given the actions being performed by the other agents. $\Box$

The three termination strategies described above are the most common, but not the only termination schemes in cooperative multiagent activities. A wide range of termination strategies can be defined based on them. Of course, all these strategies are not appropriate for every multiagent task. We categorize termination strategies as *synchronous* and *asynchronous*. In *synchronous* schemes, such as τ_{any} and τ_{all}, all agents make a decision at every decision epoch and therefore we need a centralized mechanism to synchronize agents at decision epochs. In *asynchronous* strategies, such as τ_{cont}, only a subset of agents make decision at each decision epoch. In this case, there is no need for a centralized mechanism to synchronize agents and decision making can take place in a decentralized fashion.

While SMDP theory provides the theoretical underpinnings of temporal abstraction by allowing for actions that take varying amounts of time, the SMDP model provides little in the way of concrete representational guidance which is critical from a computational point of view. In particular, the SMDP model does not specify how tasks can be broken up into subtasks, how to define policy for subtasks, how to decompose value function etc. We examine these issues in the rest of this section.

Mathematically, a task hierarchy such as the one in Figure 2 can be modeled by decomposing the overall task MDP M, into a finite set of subtasks $\{M_0, \dots, M_n\}$, where M_0 is the *root* task and solving it solves the MDP M.

Definition 2: Each *non-primitive* subtask i consists of five components $(S_i, I_i, T_i, A_i, R_i)$:

- S_i is the state space for subtask i and is described by those state variables that are relevant to subtask i. The range of the state variables describing S_i might be a subset of their range in S (the state space of the overall task MDP M).
- I_i is the initiation set for subtask i. Subtask i could start only in states belong to I_i.

- T_i is the set of terminal states for subtask i. Subtask i terminates when it reaches a state in T_i.
- A_i is the set of actions that can be performed to achieve subtask i. These actions can either be primitive actions from A (the set of primitive actions for MDP M), or they can be other subtasks.
- R_i is the reward function of subtask i. □

The goal is to learn a policy for every subtask in the hierarchy. It gives us a policy for the overall task. This collection of policies is called a *hierarchical policy*.

Definition 3: A hierarchical policy π is a set with a policy for each of the subtasks in the hierarchy: $\pi = \{\pi_0, \dots, \pi_n\}$.

The hierarchical policy is executed using a stack discipline, similar to ordinary programming languages. Each subtask policy takes a state and returns the name of a primitive action to execute or a subtask to invoke. When a subtask is invoked, its name is pushed onto the stack and its policy is executed until it enters one of its terminal states. When a subtask terminates, its name is popped off the stack. Under a hierarchical policy π, we define a multi-step transition probability P_i^π for each subtask i in the hierarchy. $P_i^\pi(s', N|s)$ denotes the probability that action $\pi_i(s)$ will cause the system to transition from state s to state s' in N primitive steps.

The action-value function of executing subtask M_a under hierarchical policy π in state s in the context of parent task M_i, $Q^\pi(i, s, a)$, is decomposed into two parts: the value of subtask M_a in state s, $V^\pi(a, s)$, and the value of completing parent task M_i after invoking subtask M_a in state s, which is called the completion function $C^\pi(i, s, a)$ [5, 6]. The value function decomposition is recursively defined as:

$$Q^\pi(i, s, a) = V^\pi(a, s) + C^\pi(i, s, a) \qquad (1)$$

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

2.3. Multiagent Setup

In our hierarchical multiagent model, we assume that there are n agents in the environment, cooperating with each other to accomplish a task. The task is decomposed by the designer of the system and its task graph is built, as described in Section 2.1. We also assume that agents are *homogeneous*, i.e., all agents are given the same task hierarchy.[2] At each level of the hierarchy, we define *cooperative subtasks* to be those subtasks in which coordination among agents has significant effect on 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 a non-empty *cooperation set* is called a *cooperation level*. We usually define *cooperative subtasks* at highest level(s) of the hierarchy. Coordination at high-level has two main advantages. First, it increases cooperation skills as agents do not get confused by low level details. Second, since high-level subtasks can take a long time to complete, communication among agents is needed only fairly infrequently. In this model, we specify policies for non-cooperative subtasks as single-agent policies, and policies for *cooperative subtasks* as joint policies.

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

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

α is the set of n agents in the team. We assume that agents have only local state information and ignore state of the other agents. Therefore, the state space $\mathcal{S}_i$ is defined as the single-agent state space S_i (not joint state space). This is certainly an approximation but greatly simplifies the underlying multiagent RL problem. This approximation is based on the fact that an agent can get a rough idea of what state the other agents might be in just by knowing the high-level actions being performed by them. The action space is joint and is defined as $\mathcal{A}_i = A_i \times (U_l)^{n-1}$, where $U_l = \bigcup_{k=1}^{m} A_k$ is the union of the action sets of all the lth level *cooperative subtasks*, and m is the cardinality of the lth level *cooperation set*. In the taxi domain, *root* is defined as a *cooperative subtask*, and the highest level of the hierarchy as a *cooperation level* (see Figure 2). Thus, *root* is the only member of the *cooperation set* at that level and $U_{root} = A_{root} = \{GetB,\ GetG,\ GetR,\ GetY,\ Wait,\ Put\}$. The joint-action space for *root*, $\mathcal{A}_{root}$, is specified as the cross product of the *root* action set, A_{root}, and U_{root}. Finally, since our goal is to design a decentralized multiagent RL algorithm, we use the τ_{cont} termination scheme for joint-action selection. □

2 Studying the heterogeneous case where agents are given dissimilar decompositions of the overall task would be more challenging and beyond the scope of this paper.

2.4. Incorporating Communication in the Model

Communication is used by each agent to obtain the local information of its teammates by paying a certain cost. The *Cooperative HRL* algorithm described in our previous paper [9] works under three important assumptions, free, reliable, and instantaneous communication, i.e., communication cost is zero, no message is lost in the environment, and each agent has enough time to receive information about its teammates before taking its next action. Since communication is free, as soon as an agent selects an action at a *cooperative subtask*, it broadcasts it to the team. Using this simple rule, and the fact that communication is reliable and instantaneous, whenever an agent is about to choose an action at a lth level *cooperative subtask*, it knows the subtasks in U_l being performed by all its teammates.

However, communication can be costly and unreliable in real-world problems. 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 optimally use communication by taking into account its long term return and its immediate cost. In this paper, we examine the case that communication is not free, but still assume that it is reliable and instantaneous. We extend the *Cooperative HRL* algorithm to include communication decisions and propose a new algorithm, called *COM-Cooperative HRL*. In the *COM-Cooperative HRL*, we add a communication level to the task graph of the problem below each *cooperation level*, as shown in Figure 3 for the taxi domain. When an agent is going to select an action at a *cooperative subtask* located at the lth level of the hierarchy, it first decides whether to communicate (takes *communicate* action) with the other agents to acquire their selected actions in U_l, or takes *not-communicate* action and selects its action without new information about its teammates. The goal of our algorithm is to learn a hierarchical policy (a set of policies for all subtasks including the communication subtasks) to maximize the team utility given the communication cost. We illustrate the algorithm in more detail in the next section.

3. Cooperative HRL Algorithm with Communication (COM-Cooperative HRL)

In the *COM-Cooperative HRL*, agents decide about communication by comparing the expected value of communication plus the communication cost ($Q(Parent(Com), s, Com) + ComCost$), 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 action like a selfish agent by using its action-value function $Q^j(NotCom, s, a)$, where $a \in Children(NotCom)$. When it decides to communicate, it acquires the actions being executed by all the other agents in U_l and then uses its joint-action-value function $Q^j(Com, s, a^1, \dots, a^{j-1}, a^{j+1}, \dots, a^n, a)$ to select its next action, where $a \in Children(Com)$. For instance, in the taxi domain, when taxi $T1$ drops off a passenger and is going to pick a new one, it should first decide whether to communicate with taxi $T2$ in order to acquire its action in U_{root}. To make communication decisions, $T1$ compares $Q^1(Root, s, NotCom)$ with $Q^1(Root, s, Com) + ComCost$. If it chooses not to communicate, it selects its action using $Q^1(NotCom, s, a)$, where $a \in U_{root}$. If it decides to communicate, after acquiring the $T2$'s action in U_{root}, a^{T2}, it selects its action using $Q^1(Com, s, a^{T2}, a)$, where $a \in U_{root}$. 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 instance, 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. The joint completion function for agent j, $C^j(Com, s, a^1, \dots, a^{j-1}, a^{j+1}, \dots, a^n, a^j)$ is defined as the expected discounted reward of completing subtask a^j by agent j in the context of the parent task Com when other

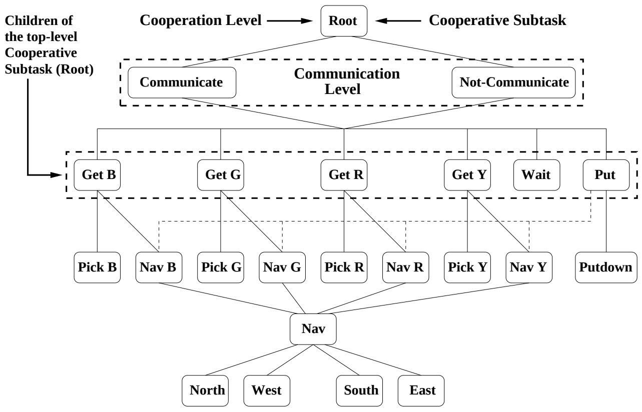


Figure 3. The task graph of the multiagent taxi domain with communication subtasks.

agents performing subtasks $a^i, \forall i \in \{1, \ldots, n\}, i \neq j$. In the taxi domain, if taxi $T1$ communicates with taxi $T2$, its value function decomposition would be

$$Q^1(Com, s, GetR, GetB) = V^1(GetB, s) + C^1(Com, s, GetR, GetB)$$

which represents the value of $T1$ performing subtask *GetB*, when $T2$ is executing subtask *GetR*. Note that this value is decomposed into the value of subtask *GetB* and the value of completing subtask $Parent(Com)$ (here $root$ is the parent of subtask Com) after executing subtask *GetB*. If $T1$ does not communicate with $T2$, its value function decomposition would be

$$Q^1(NotCom, s, GetB) = V^1(GetB, s) + C^1(NotCom, s, GetB)$$

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

The V and C values are learned through a standard temporal-difference learning method based on sample trajectories. Since subtasks are temporally extended in time, the update rules are based on the SMDP model (see [6] for details). Completion function and joint completion function values for an action in U_l are updated when the action is taken under *Not-Communicate* and *Communicate* subtasks respectively. In the later case, the actions selected in U_l by other agents are known as a result of communication and are used to update the joint completion function values.

4. Experimental Results

In this section, we demonstrate the performance of the *COM-Cooperative HRL* algorithm using the multiagent taxi problem described in Section 2.1. We also investigate the relation between communication policy and communication cost in this domain.

The state variables in this task are locations of taxis $T1$ and $T2$ (25 values each), status of taxis (2 values each, full or empty), status of stations B, G, R, Y (2 values each, full or empty), destination of stations (4 values each, one of the other three stations or without destination, which happens when the station is empty), destination of taxis (5 values each, one of the four stations or without destination, which is when taxi is empty). Thus, in the multiagent flat case, the size of the state space would grow to 256×10^6. The size of the Q table is this number multiplied by 10, the number of primitive actions (256×10^7). In the hierarchical selfish case (where each agent acts independently without communicating with other agents), using state abstraction and the fact that each agent stores only its own state variables, the number of C and V values to be learned is reduced to $2 \times 135,895 = 271,790$, which is 135,895 values for each agent. In the hierarchical cooperative without communication action, this number would be $2 \times 729,815 = 1,459,630$, and finally in the hierarchical cooperative with communication action, it is $2 \times 934,615 = 1,869,230$. All the experiments in this section were repeated five times and the results averaged.

Figures 4 and 5 show the throughput of the system and the average waiting time per passenger for four algorithms, single-agent HRL, selfish multiagent HRL, *Cooperative HRL* and *COM-Cooperative HRL* when communication cost is zero. The *Cooperative HRL* and *COM-Cooperative HRL* algorithms use the task graphs in Figures 2 and 3 respectively. As seen in Figures 4 and 5, *Cooperative HRL* and *COM-Cooperative HRL* with $ComCost = 0$ have better throughput and average waiting time per passenger than selfish multiagent HRL and single-agent HRL. The *COM-Cooperative HRL* learns slower than the *Cooperative HRL*, due to the more parameters to be learned in this model. However, it eventually converges to the same performance as the *Cooperative HRL*.

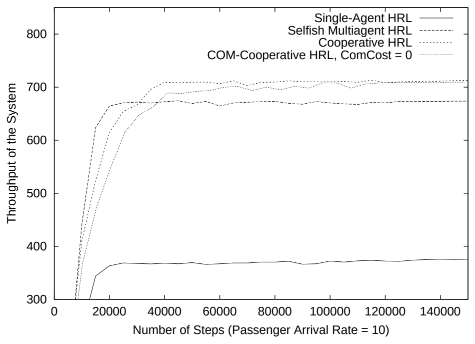


Figure 4. This figure shows that the *Cooperative HRL* and the *COM-Cooperative HRL* with $ComCost = 0$ have better throughput than the selfish multiagent HRL and the single-agent HRL.

Figure 6 compares the average waiting time per passenger for the multiagent selfish HRL and the *COM-Cooperative HRL* with $ComCost = 0$, for three different passenger arrival rates (5, 10 and 20). It demonstrates that as the passenger arrival rate becomes smaller, the coordination among taxis becomes more important. When taxis do not coordinate, there is a possibility that both taxis go to the same station. In this case, the first taxi picks up the passenger and the other one returns empty. This case can be avoided by incorporating coordination in the system. However, when the passenger arrival rate is high, there is a chance that a new passenger arrives after the first taxi picked up the pre-

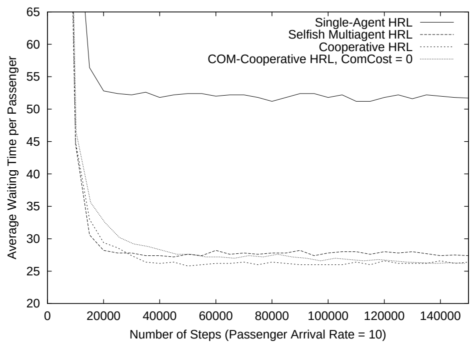
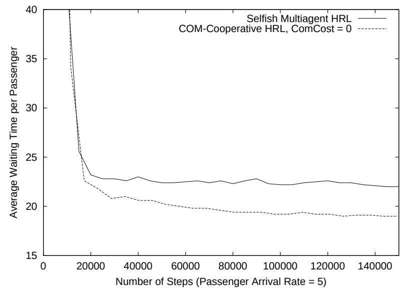
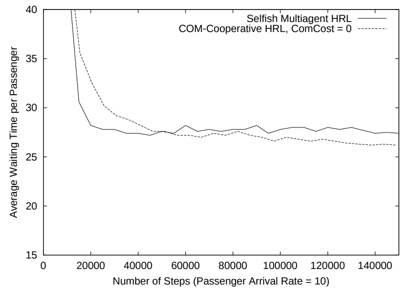


Figure 5. This figure shows that the average waiting time per passenger in the *Cooperative HRL* and the *COM-Cooperative HRL* with $ComCost = 0$, is less than the selfish multiagent HRL and the single-agent HRL.

vious passenger and before the second taxi reaches the station. This passenger will be picked up by the second taxi. In this case, coordination would not be as crucial as the case when the passenger arrival rate is low.

Figure 7 demonstrates the relation between the communication policy and the communication cost. These two figures show the throughput and the average waiting time per passenger for the selfish multiagent HRL and the *COM-Cooperative HRL* when communication cost equals 0, 1, 5, 10. In both figures, as the communication cost increases, the performance of the *COM-Cooperative HRL* becomes closer to the selfish multiagent HRL. It indicates that when communication is expensive, agents learn not to communicate and to be selfish.





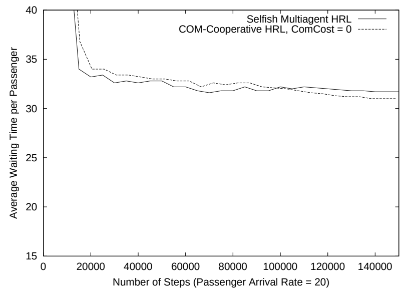
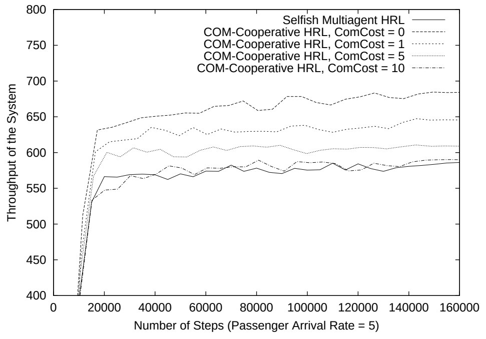


Figure 6. This figure compares the average waiting time per passenger for the selfish multiagent HRL and the *COM-Cooperative HRL* with $ComCost = 0$, for three different passenger arrival rates (5, 10 and 20). It shows that coordination among taxis becomes more important as the passenger arrival rate becomes smaller.

5. Conclusion and Future Work

In this paper, we investigate methods for learning to communicate and act in cooperative multiagent systems using hierarchical reinforcement learning (HRL). The use of hierarchy speeds up learning in multiagent domains by making it possible to learn coordination skills at the level of subtasks instead of primitive actions. We introduce a new cooperative multiagent HRL algorithm, called *COM-Cooperative HRL*, by extending our previously reported algorithm [9] to include communication decisions. In the *COM-Cooperative HRL*, we define *cooperative subtasks* to be those subtasks in which coordination among agents significantly improves the performance of the system. Those levels of the hierarchy to which *cooperative subtasks* belong are called *cooperation levels*. Each agent learns joint-action-values at *cooperative subtasks* by communicating with its teammates, and is unaware of them at the other subtasks. We add a communication level to the task hierarchy, below each *cooperation level*. Before selecting an action at a *cooperation level*, agents decide if it is worthwhile to perform a communication action to acquire the actions chosen by the other agents at the same level. It allows agents to learn a policy to optimize the communication needed for proper coordination, given the communication cost. We study the empirical performance of the *COM-Cooperative HRL* algorithm as well as the relation between the communication cost and the communication policy using a multiagent taxi problem.

A number of extensions would be useful, from studying the scenario where agents are heterogeneous, to rec-



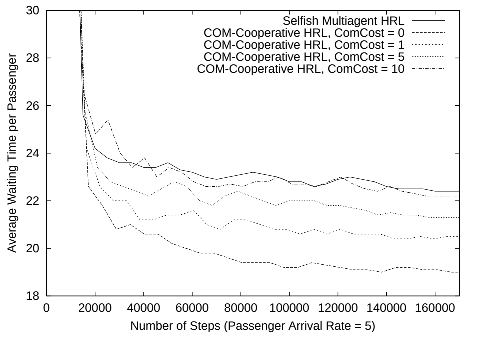


Figure 7. This figure shows that as communication cost increases, the throughput (top) and the average waiting time per passenger (bottom) of the *COM-Cooperative HRL* become closer to the selfish multiagent HRL. It indicates that agents learn to be selfish when communication is expensive.

ognizing the high-level subtasks being performed by the other agents using a history of observations 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 [3]. 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. Many other manufacturing and robotics problems can benefit from this algorithm. We are currently applying the *COM-Cooperative HRL* to a complex four-agent AGV scheduling problem used in our previous paper [9]. Combining our algorithm with function approximation and factored action models, which makes it more appropriate for continuous state problems, is also an important area of research. The success of the proposed algorithm depends on providing agents with a good initial hierarchical task decomposition. Therefore, deriving abstractions automatically is an essential problem to study. Finally, studying those communication features that have not been considered in our model, such as message delay and probability of loss, is another fundamental problem that needs to be addressed.

References

[1] C. Boutilier. Sequential optimality and coordination in multiagent systems. In *Proceedings of the Sixteenth International Joint Conference on Artifi cial Intelligence (IJCAI)*, 1999.

[2] M. Bowling and M. Veloso. Multiagent learning using a variable learning rate. *Artifi cial Intelligence*, 136:215–250, 2002.

[3] H. Bui, S. Venkatesh, and G. West. Policy recognition in the Abstract Hidden Markov Model. *Journal of Artifi cial Intelligence Research*, 17:451–499, 2002.

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

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

[6] M. Ghavamzadeh and S. Mahadevan. Hierarchical multiagent reinforcement learning. *UMASS Computer Science Technical Report*, 2004.

[7] C. Guestrin, M. Lagoudakis, and R. Parr. Coordinated reinforcement learning. In *Proceedings of the Nineteenth International Conference on Machine Learning*, 2002.

[8] R. Howard. *Dynamic Probabilistic Systems: Semi-Markov and Decision Processes*. John Wiley and Sons., 1971.

[9] R. Makar, S. Mahadevan, and M. Ghavamzadeh. Hierarchical multi-agent reinforcement learning. In *Proceedings of the Fifth International Conference on Autonomous Agents*, 2001.

[10] L. Peshkin, K. Kim, N. Meuleau, and L. Kaelbling. Learning to cooperate via policy search. In *Proceedings of the Sixteenth International Conference on Uncertainty in Artifi cial Intelligence (UAI)*, 2000.

[11] D. Pynadath and M. Tambe. The communicative multiagent team decision problem: Analyzing teamwork theories and models. *Journal of Artifi cial Intelligence Research (JAIR)*, 16:389–426, 2002.

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

[13] J. Schneider, W. Wong, A. Moore, and M. Riedmiller. Distributed value functions. In *Proceedings of the Sixteenth International Conference on Machine Laerning (ICML)*, 1999.

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

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

[16] P. Xuan, V. Lesser, and S. Zilberstein. Communication decisions in multi-agent cooperation: Model and experiments. In *Proceedings of the Fifth International Conference on Autonomous Agents*, 2001.