Hierarchical Multiagent Reinforcement Learning

Mohammad Ghavamzadeh
Sridhar Mahadevan

CMPSCI Technical Report 04-02

January 25, 2004

Department of Computer Science
140 Governors Drive
University of Massachusetts
Amherst, Massachusetts 01003-4601

Abstract

In this paper, we investigate the use of hierarchical reinforcement learning (HRL) to speed up the acquisition of cooperative multiagent tasks. We introduce a hierarchical multiagent reinforcement learning (RL) framework and propose a hierarchical multiagent RL algorithm called *Cooperative HRL*. In our approach, agents are cooperative and homogeneous (use the same task decomposition). Learning is decentralized, with each agent learning three interrelated skills: how to perform subtasks, which order to do them in, and how to coordinate with other agents. 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*. Since coordination at high levels allows for increased cooperation skills as agents do not get confused by low-level details, we usually define *cooperative subtasks* at the high levels of the hierarchy. This hierarchical approach allows agents to learn coordination faster by sharing information at the level of *cooperative subtasks*, rather than attempting to learn coordination at the level of primitive actions. We use two experimental testbeds to study the empirical performance of the *Cooperative HRL* algorithm. One domain is a simulated two-robot trash collection task. The other domain is a much larger four-agent automated guided vehicle (AGV) scheduling problem. We compare the performance of the *Cooperative HRL* with selfish HRL, as well as single-agent HRL and standard Q-learning algorithms. In the AGV scheduling domain, we also show that the *Cooperative HRL* outperforms widely used industrial heuristics, such as *"first come first serve"*, *"highest queue first"* and *"nearest station first"*.

Later in this paper, we address the issue of rational communication behavior among autonomous agents. The goal is for agents to learn both action and communication policies that together optimize the task given the communication cost. We extend the *Cooperative HRL* algorithm to include communication decisions and propose a cooperative multiagent HRL algorithm called *COM-Cooperative HRL*. In this algorithm, 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 problem.[1]

Keywords: hierarchical reinforcement learning, cooperative multiagent systems, coordination, communication.

1. Introduction

The analysis of multiagent systems is a topic of interest in both economic theory and artificial intelligence (AI). While they have already been widely studied in game theory, only recently they have started to attract interest in AI, where their integration with existing methods constitutes a promising area of research. An optimal policy in a multiagent system may depend on the behavior of other agents, which is often not predictable. It makes learning and adaptation a necessary component of the agent. Multiagent learning studies algorithms for selecting actions for multiple agents coexisting in the same environment. This is a complicated problem, because the behaviors of the other agents can be changing as they also adapt to achieve their own goals. It usually makes the environment non-stationary and often non-Markovian as well [24]. Robosoccer, disaster rescue and e-commerce are examples of challenging multiagent domains that need robust learning algorithms for coordination among multiple agents or effectively responding to other agents [41].

In addition to the existing methods in distributed AI and machine learning, game theory also provides a framework for research in multiagent learning. The game theoretic concepts of stochastic games and Nash equilibria [10, 26] are the foundation for much of the recent research in multiagent learning. Learning algorithms use stochastic games as a natural extension of Markov decision processes (MDPs) to multiple agents. These algorithms can be summarized by broadly grouping them into two categories: *equilibria learners* and *best-response learners*. *Equilibria learners* such as Nash-Q [14], Minimax-Q [20], Friend-or-Foe-Q [21] and gradient ascent learner in [33] seek to learn an equilibrium of the game by iteratively computing intermediate equilibria. They guarantee convergence to their part of an equilibrium solution regardless of the behavior of the other agents. On the other hand, *best-response learners* seek to learn the best response to the other agents. Although not an explicitly multiagent algorithm, Q-learning [40] was one of the first algorithms applied to multiagent problems [8, 38]. WoLF-PHC [6] and joint-state/joint-action learners [5] are another examples of a best-response learner. If an algorithm in which best-response learners playing with each other converges, it must be to a Nash equilibrium [6].

Multiagent learning has been recognized to be challenging 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 and inter-agent communication is usually costly.

Prior work in multiagent learning have addressed the *curse of dimensionality* in many different ways. One natural approach is to restrict the amount of information that is available to each agent and hope to maximize the global payoff by solving local optimization problems for each agent. This idea has been addressed using value function based reinforcement learning (RL) [32] as well as policy gradient based RL [28]. Another approach is to exploit the structure in a multiagent problem using factored value functions. Guestrin et al. [12] integrate these ideas in collaborative multiagent domains. They use value function approximation and approximate 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, this approach does not address the communication cost in its message passing strategy.

Graphical models have also been used to address the curse of dimensionality in multiagent systems. These works seek to transfer the representational and computational benefits that graphical models provide to probabilistic inference in multiagent systems and game theory [17, 18]. The previous works established algorithms for computing Nash equilibria in one-stage games, including efficient algorithms for computing approximate [15] and exact [22] Nash equilibria in tree-structured games and convergent heuristics for computing Nash equilibria in general graphs [25, 39].

The curse of dimensionality has also been addressed in multiagent robotics. Multi-robot learning methods usually reduce the complexity of the problem by not modeling joint states or actions explicitly, such as works by Balch [2] and Mataric [24], among others. In such systems, each robot maintains its position in the formation depending on the locations of the other robots, so there is some implicit communication or sensing of states and actions of the other agents. There has also been work on reducing the parameters needed for Q-learning in multiagent domains, by learning action values over a set of derived features [34]. These derived features are domain specific, and have to be encoded by hand, or constructed by a supervised learning algorithm.

Almost all the above methods ignore this important fact that an agent might not have free access to the other agents' information that are required to make its own decision. In general, the world is partially observable for each agent in a distributed multiagent setting. Partially observable Markov decision processes (POMDPs) have been used to model partial observability in probabilistic AI. A POMDP is a generalization of a MDP in which an agent must base its decisions on incomplete information about the state of the environment. The POMDP framework can be extended to allow for multiple distributed agents to base their decisions on their local observations. This model is called decentralized POMDP (DEC-POMDP) and it has been shown that the decision problem for a DEC-POMDP is NEXP-complete [4]. One way to address partial observability in distributed multiagent domains is to use communication to exchange required information. However, since communication can be costly, in addition to its normal actions, each agent needs to decide about communication with other agents [42, 43]. Pynadath and Tambe [30] extended DEC-POMDP by including communication decisions in the model, and proposed a framework called communicative multiagent team decision problem (COM-MTDP). Since DEC-POMDP can be reduced to COM-MTDP with no communication by copying all the other model features, decision problem for a COM-MTDP is also NEXP-complete [30]. The trade-off between the quality of solution, the cost of communication, and the complexity of the model is currently a very active area of research in multiagent learning and planning.

Our approach in this paper differs from all the above in one key respect, namely the use of hierarchy to speed up multiagent RL [23]. Hierarchical methods constitute a general framework for scaling RL to large domains by using the task structure to restrict the space of policies [3]. Several alternative frameworks for hierarchical RL (HRL) have been proposed, including options [36], HAMs [27] and MAXQ [9]. The key idea underlying our approach is that coordination[1] skills are learned much more efficiently if the agents have a hierarchical representation of the task structure (algorithms for learning task-level coordination have been developed in non-MDP approaches, see [35]). 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 assume each agent is given an initial hierarchical decomposition of the overall task. However, learning is distributed since each agent 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 with their teammates at *cooperative subtasks* and are unaware of them at the other subtasks. *Cooperative subtasks* are usually defined at highest level(s) of the hierarchy. Coordination at high-level provides significant advantage over flat methods by preventing agents to get con-

[1] We are primarily interested in cooperative multiagent problems in this paper.

fused by low-level details and reducing the amount of communication needed for proper coordination among agents.

These benefits can potentially accrue with using any type of HRL algorithm. However, it is necessary to generalize the HRL frameworks to make them more applicable to multiagent learning. In this paper, first we assume that communication is free and propose a hierarchical multiagent RL algorithm called *Cooperative HRL*. We apply this algorithm to a simple two-robot trash collection task and a complex four-agent automated guided vehicle (AGV) scheduling problem and compare its performance and speed with other learning approaches as well as several well-known industrial AGV heuristics. 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 propose a multiagent HRL algorithm, called *COM-Cooperative HRL*. We study the empirical performance of this algorithm as well as the relation between the communication cost and the communication policy using a multiagent taxi problem.

The rest of this paper is organized as follows. Section 2 describes a framework for hierarchical multiagent RL which is used to develop the algorithms of this paper. In Section 3, we introduce a HRL algorithm, called *Cooperative HRL* for learning in cooperative multiagent domains. Section 4 presents experimental results of using the *Cooperative HRL* algorithm in a simple two-robot trash collection task and a more complex four-agent AGV scheduling problem. In Section 5, we illustrate how to incorporate communication decisions in the *Cooperative HRL* algorithm. In this section, after a brief introduction of the communication framework in Section 5.1, we illustrate *COM-Cooperative HRL*, a multiagent HRL algorithm with communication decisions in Section 5.2. Section 6 presents experimental results of using the *COM-Cooperative HRL* algorithm in a multiagent taxi domain. Finally, Section 7 summarizes the paper and discusses some directions for future work.

2. A Hierarchical Multiagent Reinforcement Learning Framework

In this section, we introduce a hierarchical multiagent RL framework for simultaneous learning in multiple levels of hierarchy. This is the framework underlying the hierarchical multiagent RL algorithms presented in this paper. Our treatment builds upon the existing approaches, including the MAXQ value function decomposition [9], hierarchies of abstract machines (HAMs) [27], and the options model [36].

2.1. Motivating Example

Consider sending a team of agents to pick up trash from trash cans over an extended area and accumulate it into one centralized trash bin, from where it might be sent for recycling or disposed. This is a task which can be parallelized among agents in the team. An office (rooms and connecting corridors) type environment with two agents ($A1$ and $A2$) is shown in Figure 1. Agents need to learn three skills here. First, how to do each subtask, such as navigate to trash cans $T1$ or $T2$ or *Dump*, and when to perform *Pick* or *Put* action. Second, the order to do the subtasks, for instance go to $T1$ and collect trash before heading to *Dump*. Finally, how to coordinate with each other, i.e., agent $A1$ can pick up trash from $T1$ whereas agent $A2$ can service $T2$.

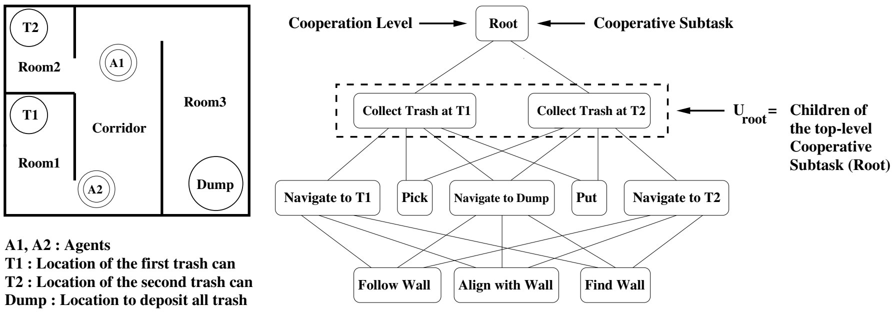


Figure 1. A multiagent trash collection task and its associated task graph.

The strength of the HRL methods (when extended to the multiagent case) is that they can serve as a substrate for efficiently learning all these three types of 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 trash collection task are *root* (the whole trash collection task), *collect trash at* $T1$ and $T2$, *navigate to* $T1$, $T2$ and *Dump*. Each of these subtasks has a set of termination states, and terminates when reaches one of its termination states. Primitive actions are always executable and terminate immediately after execution. After defining subtasks, we must indicate for each subtask, which other primitive or non-primitive subtasks it should employ to reach its goal. For example, *navigate to* $T1$, $T2$ and *Dump* use three primitive actions *find wall*, *align with wall* and *follow wall*. *Collect trash at* $T1$ should use two subtasks *navigate to* $T1$ and *Dump* plus two primitive actions *Put* and *Pick*, and so on. All of this information is summarized by the task graph shown in Figure 1. A key feature of any HRL method is how it supports temporal abstraction, state abstraction and subtask sharing [9].

- **Temporal Abstraction:** *Navigate to* $T1$ is a temporally extended action that can take different numbers of steps to complete depending on the distance to $T1$.
- **State Abstraction:** While an agent is moving toward *Dump*, the status of trash cans $T1$ and $T2$ are completely irrelevant, cannot affect navigation to dump. Therefore, the status of the trash cans $T1$ and $T2$ can be removed from the state space of the *navigate to Dump* subtask.
- **Subtask Sharing:** If the system could learn how to solve the *navigate to Dump* subtask once, then the solution could be shared by both *collect trash at* $T1$ and *collect trash at* $T2$ subtasks.

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. Thus, semi-Markov decision processes (SMDPs) have become the preferred language for modeling temporally extended actions. Semi-Markov decision processes [13, 29] 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 domain 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 (the MMDP model [5]), the multiagent extension of SMDP is not straight forward.

Definition 1: A multiagent SMDP (MSMDP) consists of six components $(\alpha,\mathcal{S},\mathcal{A},P,R,\tau)$ 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 $\vec{a} = \langle a^1, \ldots, a^n \rangle$ 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 $\mathcal{S}$, R and P are as in a SMDP, the set of states of the system being controlled, the reward function mapping $\mathcal{S} \to \Re$, and the state and action dependent multi-step transition probability function $P : \mathcal{S} \times \mathcal{N} \times \mathcal{S} \times \mathcal{A} \to [0,1]$ (where $\mathcal{N}$ is the set of natural numbers). $P(s', N|s, \vec{a})$ denotes the probability that joint-action $\vec{a}$ will cause the system to transition from state s to state s' in N time steps. 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 $\tau_{continue}$ for temporally extended joint-actions were investigated in [31]. 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 completes an action (finishes *collect trash at* $T1$ by putting trash in *Dump*), all other agents interrupt their actions, the next decision epoch occurs and a new joint-action is selected (agent $A1$ chooses to collect trash at $T2$ and agent $A2$ decides to collect trash at $T1$). 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 the other agents finish their current actions. Then, next decision epoch occurs and agents choose next joint-action together. In both these termination strategies, all agents make decision at every decision epoch. $\tau_{continue}$ 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 completes an action, next decision epoch occurs only for that agent and it selects its next action given the actions being performed by the other agents. □

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 $\tau_{continue}$, 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. Since our goal is to design decentralized multiagent RL algorithms, we use the $\tau_{continue}$ termination scheme for joint-action selection in the hierarchical multiagent model and algorithms presented in this paper.

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 next sections.

2.3. Hierarchical Task Decomposition

Mathematically, a task hierarchy such as the one illustrated in Section 2.1 can be modeled by decomposing the overall task MDP M, into a finite set of subtasks $\{M_0, M_1, \ldots, M_n\}$, where M_0 is the *root* task and solving it solves the entire 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. It is described by those state variables that are relevant to subtask i. The range of the state variables describing S_i might be a subset of their range in S, the state space of the overall task MDP M (**state abstraction**).

- 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. The policy for subtask i can only be executed if the current state s belongs to $(S_i - T_i)$.

- 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. □

Each primitive action a is a primitive subtask in this decomposition, such that a is always executable and it terminates immediately after execution.

2.4. Policy Execution

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: $\pi = \{\pi_0, \ldots, \pi_n\}$.

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 the name of 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. If any subtask on the stack terminates, then all subtasks below it are immediately aborted, and control returns to the subtask that had invoked the terminated subtask. Hence, at any time, *root* subtask is located at the bottom and the subtask which is currently being executed is located at the top of the stack. Under a hierarchical policy π, we define a multi-step transition probability function $P_i^\pi : S_i \times \mathcal{N} \times S_i \rightarrow [0,1]$ for each subtask i in the hierarchy, where $P_i^\pi(s', N|s)$ denotes the probability that action $\pi_i(s)$ will cause the system to transition from state s to state s' in N primitive steps.

2.5. Multiagent Setup

In our hierarchical multiagent 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 his/her domain knowledge to recursively decomposes the overall task into a collection of subtasks that he/she believes are important for solving the problem. This information can be summarized by a directed acyclic graph called the task graph. We assume that agents are *homogeneous*, i.e., all agents are given the same task hierarchy.[2] At each level of the hierarchy, the designer of the system defines *cooperative subtasks* to be those subtasks 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. We usually define the *cooperative subtasks* at highest level(s) of the hierarchy. Agents actively coordinate only while making decision at *cooperative subtasks* and are ignorant about the other agents at *non-cooperative subtasks*. Therefore, *cooperative subtasks* are configured to model joint-action values. In the trash collection problem, *root* is defined as a *cooperative subtask*, therefore the top-level of the hierarchy is a *cooperation level.* As a result, *root* is the only member of the *cooperation set* at that level and U_{root} consists of all subtasks located at the second level of the hierarchy, $U_{root} = \{collect\ trash\ at\ T1\ ,\ collect\ trash\ at\ T2\}$ (see Figure 1). As it is clear in the trash collection task, it is more effective that each agent learns high-level coordination knowledge (what is the utility of agent $A2$ collect trash from trash can $T1$ if agent $A1$ is collecting trash from trash can $T2$, and so on), rather than it learns its response to low-level primitive actions of other agents (for instance if agent $A1$ aligns with wall, what should agent $A2$ do).

We define 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 the states 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 about the high-level actions

[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.

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*. For the *cooperative subtask root* in the trash collection problem, the set of agents is $\alpha = \{A1, A2\}$ and its joint-action space, $\mathcal{A}_{root}$, is specified as the cross product of its action set, A_{root}, and U_{root}, $\mathcal{A}_{root} = A_{root} \times U_{root}$. Finally, since we are interested in decentralized control, we use $\tau_{continue}$ termination strategy. Therefore, when an agent terminates a subtask, next decision epoch occurs only for that agent and it selects its next action given the information about the other agents. □

This cooperative multiagent approach has the following pros and cons:

Pros

- Using HRL scales learning to problems with large state spaces by using the task structure to restrict the space of policies.
- Cooperation among agents is faster and more efficient as agents learn joint-action values only at *cooperative subtasks* usually located at the high levels of abstraction and do not get confused by low-level details.
- Since high-level tasks can take a long time to complete, communication is needed only fairly infrequently.
- 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

- The learned policy would not be optimal if agents need to coordinate at the subtasks that are defined as *non-cooperative*. This issue will be addressed in the AGV experiment in Section 4.2, 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 number of parameters to be learned and the complexity of the learning problem.
- 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. If communication is cheap, agents learn to collaborate with each other, and if

communication is expensive, agents prefer to make decision only based on their local view of the overall problem.

- Storing only local state information by agents causes sub-optimality in general. On the other hand, including the state of other agents dramatically increases the complexity of the learning problem and has its own inefficacy. We do not explicitly address this problem in the paper.

2.6. Value Function Decomposition

The value function decomposition used in our framework is similar to the MAXQ value function decomposition [9]. The purpose of value function decomposition is to decompose the value function of the overall task (*root*) under hierarchical policy π, in terms of the value functions of all its descendants in the hierarchy. The value function of subtask i under hierarchical policy π, $V^\pi(i,s)$, is the expected sum of discounted reward until subtask i terminates:

$$V^\pi(i,s) = E\{r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} + \ldots + \gamma^L r_{t+L} | s_t = s, \pi\} \tag{1}$$

Now let us suppose that the policy of subtask 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 $P_i^\pi(s', N|s, \pi_i(s))$. We can rewrite Equation 1 as:

$$V^\pi(i,s) = E\left\{\sum_{k=0}^{N-1} \gamma^k r_{t+k} + \sum_{k=N}^{L} \gamma^k r_{t+k} | s_t = s, \pi\right\} \tag{2}$$

The first summation of Equation 2 is the discounted sum of rewards for executing subtask $\pi_i(s)$ starting in state s until it terminates. In other words, it is $V^\pi(\pi_i(s), s)$, the value function of subtask $\pi_i(s)$ in state s. The second summation is the value of state s' for the current subtask i under hierarchical policy π, $V^\pi(i,s')$, discounted by γ^N:

$$V^\pi(i,s) = V^\pi(\pi_i(s), s) + \sum_{s' \in S_i, N} P_i^\pi(s', N|s, \pi_i(s)) \gamma^N V^\pi(i, s') \tag{3}$$

where s' is the state when subtask $\pi_i(s)$ terminates. Equation 3 is in the form of a Bellman equation. We can restate Equation 3 and derive the following Bellman equation for action-value function:

$$Q^\pi(i,s,a) = V^\pi(a,s) + \sum_{s' \in S_i, N} P_i^\pi(s', N|s,a)\gamma^N Q^\pi(i,s',\pi_i(s')) \quad (4)$$

The summation term in Equation 4 is the expected discounted reward of completing subtask i after executing subtask a in state s. This term is called *completion function* and is defined as follows:

Definition 6: Completion function, $C^\pi(i,s,a)$, is the expected discounted cumulative reward of completing subtask i after execution of subtask a in state s. The reward is discounted back to the point in time where a begins execution.

$$C^\pi(i,s,a) = \sum_{s' \in S_i, N} P_i^\pi(s', N|s,a)\gamma^N Q^\pi(i,s',\pi_i(s')) \quad (5)$$

Now, we can express the action-value function Q as:

$$Q^\pi(i,s,a) = V^\pi(a,s) + C^\pi(i,s,a) \quad (6)$$

and the V function as:

$$V^\pi(i,s) = \begin{cases} Q^\pi(i,s,\pi_i(s)) & \text{if } i \text{ is a non-primitive subtask} \\ \sum_{s' \in S_i} P(s'|s,i)R(s'|s,i) & \text{if } i \text{ is a primitive action} \end{cases} \quad (7)$$

Equations 5, 6 and 7 are the decomposition equations for the hierarchical framework under a fixed hierarchical policy π. These equations recursively decompose the value function for the *root*, $V^\pi(0,s)$, into a set of value and completion functions. The quantities that must be stored to represent the value function decomposition are just the C values for non-primitive subtasks and the V values for primitive actions. Using this decomposition and the stored values, we can recursively calculate all Q values in the hierarchy. For instance, $Q(i,s,a)$ is calculated as follows:

- From Equation 6, $Q(i,s,a) = V(a,s) + C(i,s,a)$. $C(i,s,a)$ is stored by subtask i, so subtask i only needs to calculate $V(a,s)$ by asking subtask a.

- If a is a primitive action, it stores $V(a,s)$ and returns it to subtask i immediately. If a is a non-primitive subtask, then using Equation

7, $V(a,s) = Q(a,s,\pi_a(s))$, and using Equation 6, $Q(a,s,\pi_a(s)) = V(\pi_a(s),s) + C(a,s,\pi_a(s))$. In this case, $C(a,s,\pi_a(s))$ is available at subtask a, and a asks subtask $\pi_a(s)$ for $V(\pi_a(s),s)$.

- This process continues until we reach a primitive action. Since, primitive actions store their V values, all V values are calculated upward in the hierarchy and eventually subtask i receives the value of $V(a,s)$ and calculates $Q(i,s,a)$.

The value function decomposition described above relies on a key principle: the reward function for the parent task is the value function of the child task (see Equations 4 and 6). Now, we show how the single-agent value function decomposition described above can be modified to formulate the joint-value function for *cooperative subtasks*. In our hierarchical multiagent model, *cooperative subtasks* are configured to store the joint completion function values.

Definition 7: The joint completion function for agent j, $C^j(i,s,a^1,\dots,a^{j-1},a^{j+1},\dots,a^n,a^j)$, is the expected discounted cumulative reward of completing *cooperative subtask* i after taking subtask a^j in state s while other agents performing subtasks $a^k, \forall k \in \{1,\dots,n\}, k \neq j$. The reward is discounted back to the point in time where a^j begins execution.

In this definition, i is a *cooperative subtask* at level l of the hierarchy and $\langle a^1,\dots,a^n\rangle$ is a joint-action in the action set of i. 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* i of agent j have the following form:

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

$$\begin{aligned} Q^j(i,s,a^1,\dots,a^{j-1},a^{j+1},\dots,a^n,a^j) = {} & V^j(a^j,s) + \\ & C^j(i,s,a^1,\dots,a^{j-1},a^{j+1},\dots,a^n,a^j) \end{aligned} \tag{8}$$

One important point to note in this equation 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 $V^j(a^j,s,\tilde{a}^1,\dots,\tilde{a}^{j-1},\tilde{a}^{j+1},\dots,\tilde{a}^n)$, where $\tilde{a}^1,\dots,\tilde{a}^{j-1},\tilde{a}^{j+1},\dots,\tilde{a}^n$ belong to U_{l+1}. In this case, in order to calculate $V^j(a^j,s)$ for Equation 8, we marginalize $V^j(a^j,s,\tilde{a}^1,\dots,\tilde{a}^{j-1},\tilde{a}^{j+1},\dots,\tilde{a}^n)$ over $\tilde{a}^1,\dots,\tilde{a}^{j-1},\tilde{a}^{j+1},\dots,\tilde{a}^n$.

We illustrate the above joint-value function decomposition using the trash collection task. The value function decomposition for agent $A1$ at *root* has the following form:

$$Q^1(root, s, collect\ trash\ at\ T2, collect\ trash\ at\ T1) = V^1(collect\ trash\ at\ T1, s) + C^1(root, s, collect\ trash\ at\ T2, collect\ trash\ at\ T1)$$

which represents the value of agent $A1$ performing *collect trash at T1* in the context of the overall task (*root*), when agent $A2$ is executing *collect trash at T2*. Note that this value is decomposed into the value of *collect trash at T1* subtask (the V term), and the completion value of the remainder of the *root* task (the C term).

Given a hierarchical decomposition for any task, we need to find the highest level subtasks at which decomposition Equation 8 provides a sufficiently good approximation of the true value. For the problems used in the experiments of this paper, coordination only at the highest level of the hierarchy is a good compromise between achieving a desirable performance and reducing the number of joint-state-action values that need to be learned. Hence, we define *root* as a *cooperative subtask* and thus the highest level of the hierarchy as a *cooperation level* in these experiments. We extend coordination to the lower levels of the hierarchy by defining *cooperative subtasks* at levels below the *root* in one of the experiments of Section 4.2.

3. A Hierarchical Multiagent Reinforcement Learning Algorithm

In this section, we use the HRL framework described in Section 2 and present a hierarchical multiagent RL algorithm, called *Cooperative HRL*. The pseudo code for this algorithm is shown in Algorithm 1 at the end of the paper. In the *Cooperative HRL*, V and C values can be learned through a standard temporal-difference (TD) learning method based on sample trajectories. One important point to note is that since non-primitive subtasks are temporally extended in time, the update rules for C values used in this algorithm are based on the SMDP model. In this algorithm, an agent starts from the *root* task and chooses a subtask till it reaches a primitive action i. It executes primitive action i in state s, receives reward r and observes resulting state s', the value function V of primitive subtask i is updated using:

$$V_{t+1}(i, s) = (1 - \alpha_t(i))V_t(i, s) + \alpha_t(i)r$$

where $\alpha_t(i)$ is the learning rate for subtask i at time t. This parameter should be gradually decreased to zero in time limit.

Whenever a subtask terminates, the C values are updated for all states visited during the execution of that subtask. Assume an agent is executing non-primitive subtask i and is in state s, then while subtask i does not terminate, it chooses subtask a according to the current exploration policy (softmax or ϵ-greedy with respect to $\pi_i(s)$). If subtask a takes N primitive steps and terminates in state s', the corresponding C value is updated using:

$$C_{t+1}(i,s,a) = (1-\alpha_t(i))C_t(i,s,a) + \alpha_t(i)\gamma^N[C_t(i,s',a^*) + V_t(a^*,s')] \quad (9)$$

where $a^* = argmax_{a' \in A_i}[C_t(i,s',a') + V_t(a',s')]$.

The V values in Equation 9 are calculated using the following equation:

$$V(i,s) = \begin{cases} max_{a \in A_i} Q(i,s,a) & \text{if } i \text{ is a non-primitive subtask} \\ \sum_{s' \in S_i} P(s'|s,i)R(s'|s,i) & \text{if } i \text{ is a primitive action} \end{cases} \quad (10)$$

Similarly, when agent j completes execution of subtask $a^j \in A_i$, the joint completion function C of *cooperative subtask* i located at level l of the hierarchy is updated for all the states visited during the execution of subtask a^j using:

$$\begin{aligned} &C^j_{t+1}(i,s,a^1,\ldots,a^{j-1},a^{j+1},\ldots,a^n,a^j) = \\ &(1-\alpha^j_t(i))C^j_t(i,s,a^1,\ldots,a^{j-1},a^{j+1},\ldots,a^n,a^j) + \\ &\alpha^j_t(i)\gamma^N[C^j_t(i,s',\hat{a}^1,\ldots,\hat{a}^{j-1},\hat{a}^{j+1},\ldots,\hat{a}^n,a^*) + V^j_t(a^*,s')] \end{aligned} \quad (11)$$

where $a^* = argmax_{a' \in A_i}[C^j_t(i,s',\hat{a}^1,\ldots,\hat{a}^{j-1},\hat{a}^{j+1},\ldots,\hat{a}^n,a') + V^j_t(a',s')]$, $a^1,\ldots,a^{j-1},a^{j+1},\ldots,a^n$ and $\hat{a}^1,\ldots,\hat{a}^{j-1},\hat{a}^{j+1},\ldots,\hat{a}^n$ are actions in U_l being performed by the other agents when agent j is in states s and s' respectively. Equation 11 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.

4. Experimental Results for the Cooperative HRL Algorithm

In this section, we demonstrate the performance of the *Cooperative HRL* algorithm proposed in Section 3 using a simple two-robot trash collection domain, and a more complex four-agent AGV scheduling task. In both cases, we first provide a brief overview of the domain, then apply the *Cooperative HRL* algorithm to the problem, and finally compare its performance with other algorithms, such as selfish multi-agent HRL (where each agent acts independently and learns its own optimal policy), single-agent HRL and flat Q-Learning.

4.1. Two-Robot Trash Collection Task

In the single-agent trash collection task, one robot starts in the middle of *Room 1* and learns the task of picking up trash from $T1$ and $T2$ and depositing it into the *Dump*. The goal state is reached when trash from both $T1$ and $T2$ has been deposited in the *Dump*. The state space is the orientation of the robot (N,S,W,E) and another component based on its percept. We assume that a ring of 16 sonars would enable the robot to find out whether it is in a corner, (with two walls perpendicular to each other on two sides of the robot), near a wall (with wall only on one side), near a door (wall on either side of an opening), in a corridor (parallel walls on either side) or in an open area (the middle of the room). Thus, each room is divided into 9 states, the corridor into 4 states, and we have $((9 \times 3) + 4) \times 4 = 124$ locations for a robot. Also, the trash object from trash basket $T1(T2)$ can be at $T1(T2)$, carried with a robot, or at *Dump*. Hence, the total number of states of the environment is $124 \times 3 \times 3 = 1116$ for the single-agent case. Going to the two-agent case would mean that the trash can be at either $T1$ or $T2$ or *Dump*, or carried by one of the two robots. Therefore in the flat case, the size of the state space would grow to $124 \times 124 \times 4 \times 4 \approx 240000$. The environment is fully observable given the above state decomposition. The direction which the robot is facing, in combination with the percept (which includes the room that agent is in) gives a unique value for each situation. The primitive actions considered here are behaviors to find a wall in one of the four directions, align with the wall on left or right side, follow a wall, enter or exit door, align south or north in the corridor, or move in the corridor. In this task, each experiment was repeated ten times and the results averaged.

In the two-robot trash collection task, examination of the learned policy in Figure 2 reveals that the robots have nicely learned all three skills: how to achieve a subtask, what order to do them in, and how to coordinate with each other. In addition, as Figure 3 confirms, the number of steps needed to accomplish the trash collection task is greatly reduced when the two agents coordinate to do the task, compared to when a single agent attempts to carry out the whole task.

Learned Policy for Agent 1

```
root
  navigate to T1
      go to location of T1 in room 1
  pick trash from T1
  navigate to Dump
      exit room 1
      enter room 3
      go to location of Dump in room 3
  put trash collected from T1 in Dump
end
```

Learned Policy for Agent 2

```
root
  navigate to T2
      go to location of T2 in room 2
  pick trash from T2
  navigate to Dump
      exit room 2
      enter room 3
      go to location of Dump in room 3
  put trash collected from T2 in Dump
end
```

Figure 2. This figure shows the policy learned by the *Cooperative HRL* algorithm in the two-robot trash collection task.

4.2. AGV Scheduling Domain

Automated Guided Vehicles (AGVs) are used in flexible manufacturing systems (FMS) for material handling [1]. They are typically used to pick up parts from one location, and drop them off at another location for further processing. Locations correspond to workstations or storage locations. Loads which are released at the drop off point of a

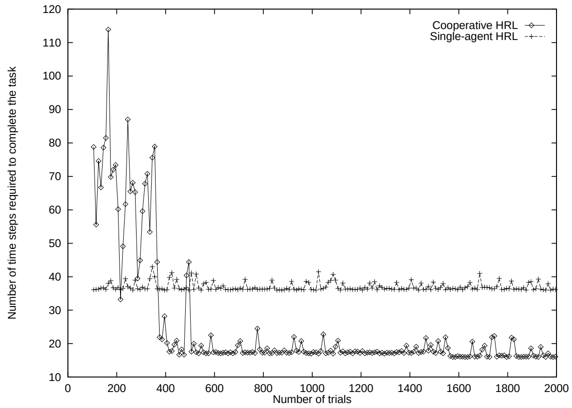


Figure 3. This figure shows that the *Cooperative HRL* algorithm learns the trash collection task with less number of steps than the single-agent HRL algorithm.

workstation wait at its pick up point after the processing is over, so the AGV is able to take it to the warehouse or some other locations. The pick up point is the machine or workstation's output buffer. Any FMS system using AGVs faces the problem of optimally scheduling the paths of the AGVs in the system [19]. For example, a move request occurs when a part finishes at a workstation. If more than one vehicle is empty, the vehicle which would service this request needs to be selected. Also, when a vehicle becomes available and multiple move requests are queued, a decision needs to be made as to which request should be serviced by that vehicle. These schedules obey a set of constraints that reflect the temporal relations between activities and the capacity limitations of a set of shared resources. The uncertain and ever changing nature of the manufacturing environment makes it virtually impossible to plan moves ahead of time. Hence, AGV scheduling requires dynamic dispatching rules, which are dependent on the state of the system like the number of parts in each buffer, the state of the AGV and the process going on at workstations. The system performance is generally measured in terms of the throughput, the on-line inventory, the AGV travel time and the flow time, but the throughput is by far the most important factor. In this case, the throughput is measured in terms of the number of finished assemblies deposited at the unloading deck per unit time. Since this problem is analytically intractable, various heuristics and their combinations are generally used to schedule AGVs [16, 19]. However, the heuristics perform poorly when the constraints on the movement of the AGVs are reduced.

Previously, Tadepalli and Ok [37] studied a single-agent AGV scheduling task using *flat* average-reward RL. However, the multiagent AGV task we study is more complex. Figure 4 shows the layout of the system used for experimental purposes in this paper. $M1$ to $M4$ show workstations in this environment. 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 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 5. 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 the set U_{root}. Coordination skills among agents are learned by using joint-action values at the highest level of the hierarchy as described in Section 3.

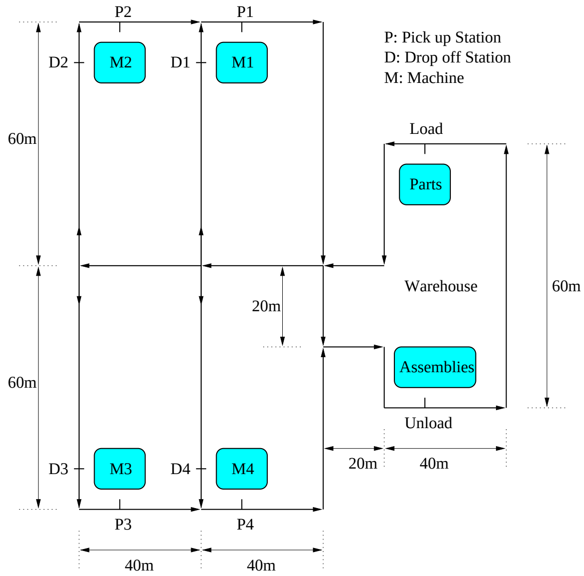


Figure 4. A multiagent AGV scheduling domain. There are four AGVs (not shown) which carry raw materials and finished parts between machines and the warehouse.

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, state space consists of 100 locations, 8 buffers of size 3, 9 possible states of the AGV (carrying part1, ... , carrying assembly1, ... , empty), and 2 values for each part in the warehouse, i.e., $100 \times 4^8 \times 9 \times 2^4 \approx 2^{30}$ states, which is enormous. The state abstraction helps in reducing the state space considerably. Only the relevant state variables are used while storing the completion functions in each node of the task graph. For example, for the

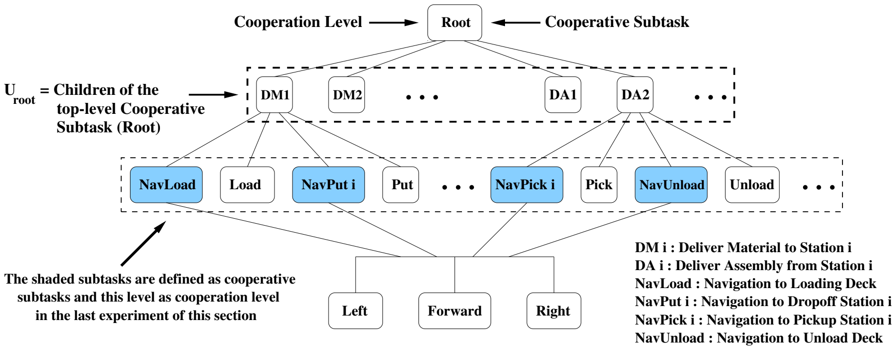


Figure 5. Task graph for the AGV scheduling task.

navigation subtasks, only the *location* state variable is relevant, and this subtask can be learned with 100 values. Hence, for the highest level subtasks *DM1*, ... , *DM4*, the number of relevant states would be $100 \times 9 \times 4 \times 2 \approx 2^{13}$, and for the highest level subtasks *DA1*, ... , *DA4*, the number of relevant states would be $100 \times 9 \times 4 \approx 2^{12}$. This state abstraction gives us a compact way of representing the C and V functions, and speeds up the algorithm.

We now present detailed experimental results on the AGV scheduling task, comparing several learning agents, including single-agent HRL, selfish multiagent HRL, and *Cooperative HRL*, the cooperative multiagent HRL algorithm proposed in Section 3. In the experiments of this section, we assume that there are four agents (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. Table I summarizes the values of the model parameters used in the experiments of this section. In this task, each experiment was conducted five times and the results averaged.

Figure 6 shows the throughput of the system for the three algorithms, single-agent HRL, selfish multiagent HRL and *Cooperative HRL*. As seen in Figure 6, agents learn a little faster initially in the selfish multiagent method, but after some time undulations are seen in the

Table I. Model parameters for the AGV scheduling task.

Parameter	**Distribution**	**Mean (sec)**	**Variance (sec)**
Idle Action	Normal	1	0.1
Primitive Actions	Normal	0.001	0.00005
Assembly Time for Part1	Normal	15	2
Assembly Time for Part2	Normal	24	2
Assembly Time for Part3	Normal	24	2
Assembly Time for Part4	Normal	30	2
Inter-Arrival Time for Parts	Uniform	4	1

graph showing not only that the algorithm does not stabilize, but also that it results in sub-optimal performance. This is due to the fact that two or more agents select the same action, but once the first agent completes the task, the other agents might have to wait for a long time to complete the task, due to the constraints on the number of parts that can be stored at a particular place. The system throughput achieved using the *Cooperative HRL* method is significantly higher than the single-agent HRL and the selfish multiagent HRL algorithms. This difference is even more significant in Figure 7, when the primitive actions have longer execution time, almost $\frac{1}{10^{th}}$ of the average assembly time (the mean execution time of primitive actions is 2 sec).

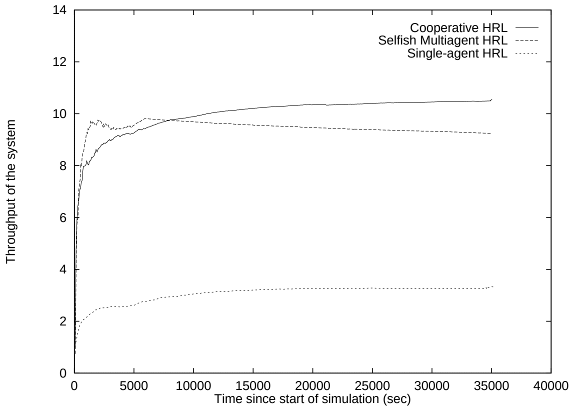


Figure 6. This figure shows that the *Cooperative HRL* algorithm outperforms both the selfish multiagent HRL and the single-agent HRL algorithms when the AGV travel time and load/unload time are very much less compared to the average assembly time.

Figure 8 shows results from an implementation of the single-agent flat Q-Learning with the buffer capacity at each station set at 1. As can

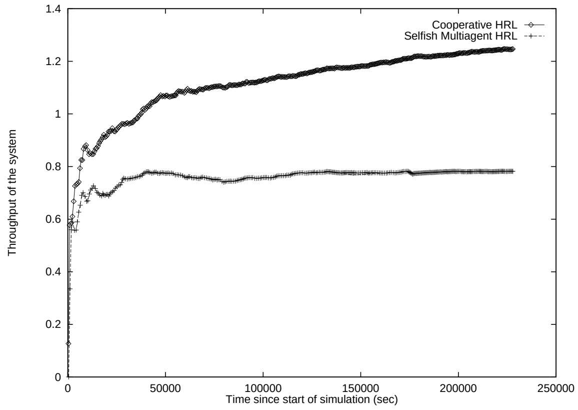


Figure 7. This figure compares the *Cooperative HRL* algorithm with the selfish multiagent HRL, when the AGV travel time and load/unload time are $\frac{1}{10^{th}}$ of the average assembly time.

be seen from the plot, the flat algorithm converges extremely slowly. The throughput at 70,000 sec has gone up to only 0.07, compared with 2.6 for the hierarchical single-agent case. Figure 9 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.

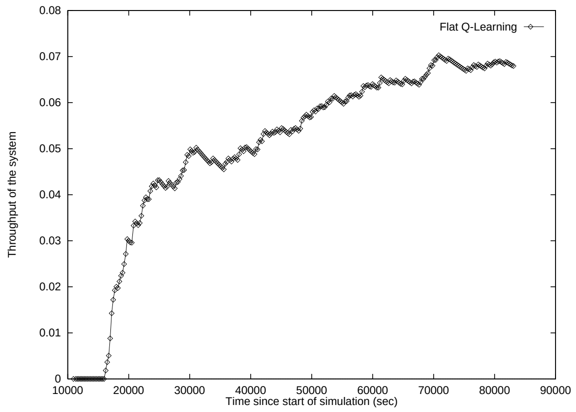


Figure 8. A flat Q-Learner learns the AGV domain extremely slowly showing the need for using a hierarchical task structure.

So far in our experiments in the AGV domain, we only defined *root* as a *cooperative subtask*. Now in our last experiment in this domain, in addition to *root*, we define navigation subtasks at the third level of the hierarchy as *cooperative subtasks*. Therefore, the third level of the hierarchy is also a *cooperation level* and its *cooperation set* contains all navigation subtasks at that level (see Figure 5). We configure the

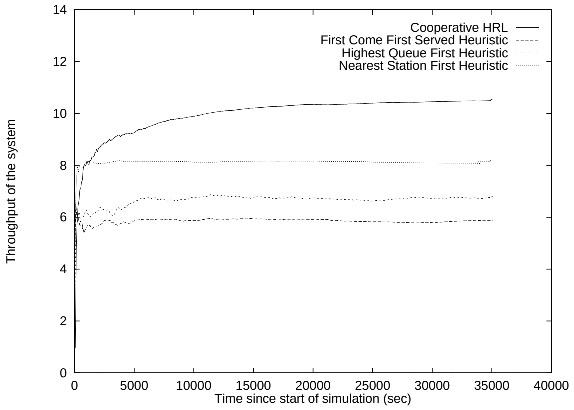


Figure 9. This plot shows that the *Cooperative HRL* algorithm outperforms three well-known widely used industrial heuristics for AGV scheduling.

root and the third level navigation subtasks to represent joint-actions. Figure 10 compares the performance of the system in these two cases. When the navigation subtasks are 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, as shown in Figure 4, thus coordination at the navigation level does not improve the performance of the system. However, there exist problems that adding joint-actions in multiple levels will be worthwhile, even if convergence is slower, due to better overall performance.

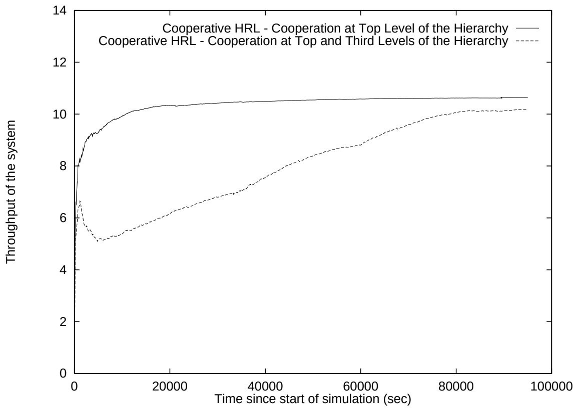


Figure 10. This plot compares the performance of the *Cooperative HRL* algorithm with cooperation at the top level of the hierarchy vs. cooperation at the top and third levels of the hierarchy.

5. Incorporating Communication Decisions in the Model

Communication is used by agents to obtain the local information of their teammates by paying a certain cost. The *Cooperative HRL* algorithm described in Section 3 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 the rest of this paper, we examine the case that communication is not free, but still assume that it is reliable and instantaneous. In this section, we first describe the communication framework and then illustrate how we extend the *Cooperative HRL* algorithm to include communication decisions and propose a new algorithm, called *COM-Cooperative HRL*. The goal of this 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. Finally, in Section 6, 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.

5.1. Communication Framework

Communication usually consists of three steps: *send*, *answer* and *receive*. At the *send* step t_s, agent j decides if communication is necessary, performs a communication action and sends a message to agent i. At the *answer* step $t_a \geq t_s$, agent i receives the message from agent j, updates its local information using the content of the message (if necessary) and sends back the answer (if required). At the *receive* step $t_r \geq t_a$, agent j receives the answer of its message, updates its local information and decides on which non-communicative action to execute. Generally there are two types of messages in a communication framework: *request* and *inform*. For simplicity, we suppose that relative ordering of messages do not change, which means that for two communication actions c_1 and c_2, if $t_s(c_1) < t_s(c_2)$ then $t_a(c_1) \leq t_a(c_2)$ and $t_r(c_1) \leq t_r(c_2)$. The following three types of communication actions are commonly used in a communication model:

- $Tell(j,i)$: agent j sends an *inform* message to agent i.
- $Ask(j,i)$: agent j sends a *request* message to agent i, which is answered by agent i with an *inform* message.
- $Sync(j,i)$: agent j sends an *inform* message to agent i, which is answered by agent i with an *inform* message.

In the *Cooperative HRL* algorithm described in Section 3, we assume free, reliable and instantaneous communication. Hence, the communication protocol of this algorithm is as follows: whenever an agent chooses an action at a *cooperative subtask*, it executes a *Tell* communication action and sends its selected action as an *inform* message to all other agents. As a result, when an agent is going to choose an action at a lth level *cooperative subtask*, it knows actions being performed by all other agents in U_l. *Tell* and *inform* are the only communication action and type of message used in the communication protocol of the *Cooperative HRL* algorithm.

5.2. A Hierarchical Multiagent Reinforcement Learning Algorithm with Communication Decisions

When communication is costly in the *Cooperative HRL* algorithm, it is no longer optimal for the team that each agent broadcasts its action to all its teammates. In this case, each agent must learn to optimally use the communication. To address the communication cost 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 11 for the trash collection task. In this algorithm, when an agent is going to make a decision at a 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 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 (not joint-action-value function) $Q^j(NotCom, s, a)$, where $a \in Children(NotCom)$. When it decides to communicate, it first takes communication action $Ask(j,i), \forall i \in \{1,\dots,j-1,j+1,\dots,n\}$, where n is the number of agents, and sends a *request* message to all other agents. Other agents reply by taking communication action $Tell(i,j)$ and send their action in U_l as an *inform* message to agent j. Then agent j uses its joint-action-value function (not action-value function) $Q^j(Com, s, a^1, \dots, a^{j-1}, a^{j+1}, \dots, a^n, a)$ ($a \in Children(Com)$) to select its next action in U_l. For instance, in the trash collection task, when agent $A1$ dumps trash and is going to move to one of the two trash cans, it should first decide whether to communicate with agent $A2$ in order to inquire its action in $U_{root} = \{collect\ trash\ at\ T1,\ collect\ trash\ at\ T2\}$ or not. To make a communication decision, agent $A1$ 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 action of agent $A2$ in U_{root}, a^{A2}, it selects its action using $Q^1(Com, s, a^{A2}, a)$, where a and a^{A2} both belong to U_{root}.

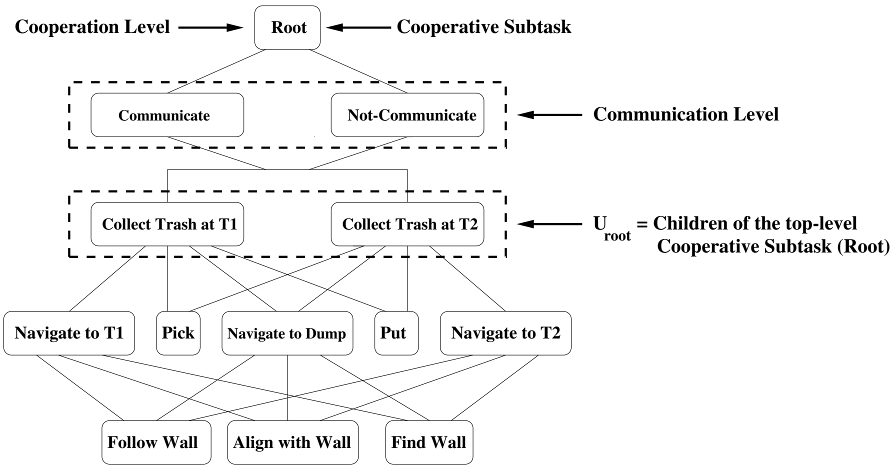


Figure 11. Task graph of the trash collection problem with communication actions.

In the *COM-Cooperative HRL*, we assume that when an agent decides to communicate, it communicates with all 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 $C^1_{n-1} + C^2_{n-1} + \dots + C^{n-1}_{n-1}$, 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 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 reward of completing subtask a^j by agent j in the context of the parent task Com when other agents performing subtasks $a^i, \forall i \in \{1, \ldots, j-1, j+1, \ldots, n\}$. In the trash collection domain, if agent $A1$ communicates with agent $A2$, its value function decomposition would be

$$\begin{aligned}
&Q^1(Com, s, Collect\ Trash\ at\ T2, Collect\ Trash\ at\ T1) = \\
&V^1(Collect\ Trash\ at\ T1, s) + \\
&C^1(Com, s, Collect\ Trash\ at\ T2, Collect\ Trash\ at\ T1)
\end{aligned}$$

which represents the value of agent $A1$ performing subtask *collect trash at T1*, when agent $A2$ is executing subtask *collect trash at T2*. Note that this value is decomposed into the value of subtask *collect trash at T1* and the value of completing subtask $Parent(Com)$ (here $root$ is the parent of subtask Com) after executing subtask *collect trash at T1*. If agent $A1$ does not communicate with agent $A2$, its value function decomposition would be

$$\begin{aligned}
Q^1(NotCom, s, Collect\ Trash\ at\ T1) = &V^1(Collect\ Trash\ at\ T1, s) \\
&+ C^1(NotCom, s, Collect\ Trash\ at\ T1)
\end{aligned}$$

which represents the value of agent $A1$ performing subtask *collect trash at T1*, regardless of the action being executed by agent $A2$.

Again, the V and C values are learned through a standard temporal-difference learning method based on sample trajectories similar to the one presented in Algorithm 1. Completion function values for an action in U_l is updated when we take the action under *Not-Communicate* subtask, and joint completion function values for an action in U_l is updated when it is selected under *Communicate* subtask. In the later case, the actions selected in U_l by the other agents are known as a result of communication and are used to update the joint completion function values.

6. Experimental Results for the COM-Cooperative HRL Algorithm

In this section, we demonstrate the performance of the *COM-Cooperative HRL* algorithm proposed in Section 5.2 using a multiagent taxi problem. We also investigate the relation between the communication policy and the communication cost in this domain.

Consider a 5-by-5 grid world inhabited by two taxis ($T1$ and $T2$) shown in Figure 12. 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[3] 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. This problem can be decomposed into subtasks and the resulting task graph is shown in Figure 12. Taxis need to learn three skills here. First, how to do each subtask, such as *navigate* to B, G, R or Y, and when to perform *Pickup* or *Putdown* action. Second, the order to do the subtasks (for instance go to a station and pickup a passenger before heading to the passenger's destination). Finally, how to communicate and coordinate with each other, i.e., if taxi $T1$ is on its way to pick up a passenger at location *Blue*, taxi $T2$ should serve a passenger at one of the other stations. 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 and 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), and 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 the number of primitive actions 10, (256×10^7). In the hierarchical selfish case (the selfish multiagent HRL algorithm), using state abstraction and the fact that each agent stores only its own state

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

variables, the number of the 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 decision (the *Cooperative HRL* algorithm), the number of values to be learned would be $2 \times 729,815 = 1,459,630$. Finally in the hierarchical cooperative with communication decisions (the *COM-Cooperative HRL* algorithm), this number would be $2 \times 934,615 = 1,869,230$. In the *COM-Cooperative HRL* algorithm, we define *root* as a *cooperative subtask* and the highest level of the hierarchy as a *cooperation level* as shown in Figure 12. 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* is specified as the cross product of the *root* action set and U_{root}. Finally, $\tau_{continue}$ termination scheme is used for joint-action selection in this domain. All the experiments in this section were repeated five times and the results averaged.

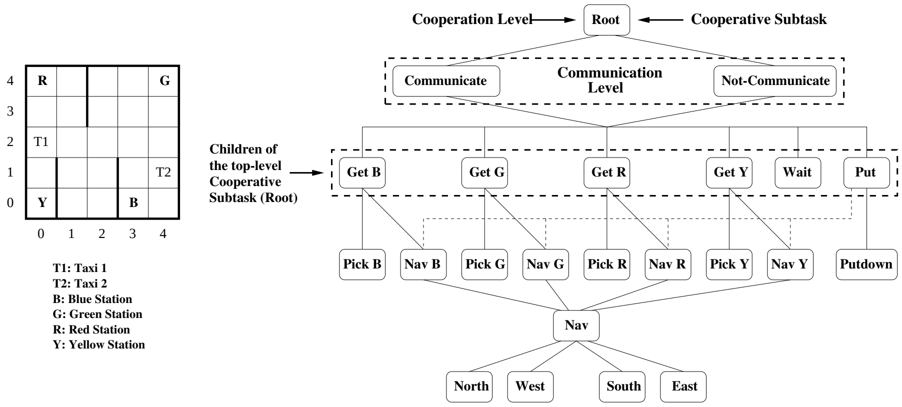


Figure 12. A multiagent taxi domain and its associated task graph.

Figures 13 and 14 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.[4] As seen in Figures 13 and 14, *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*.

[4] The *COM-Cooperative HRL* uses the task graph in Figure 12. The *Cooperative HRL* uses the same task graph without the *communication level*

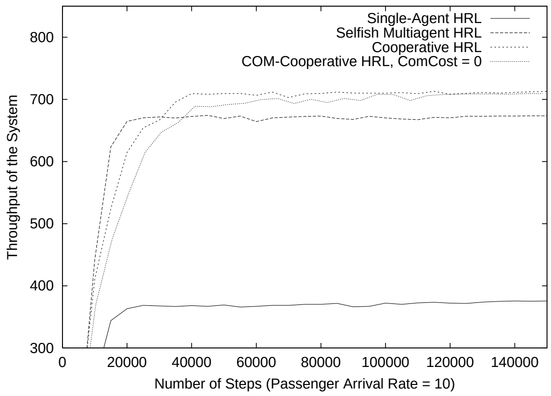


Figure 13. This figure shows that the *Cooperative HRL* and the *COM-Cooperative HRL* with $ComCost = 0$ have better throughput than the selfish multiagent HRL and the single-agent HRL.

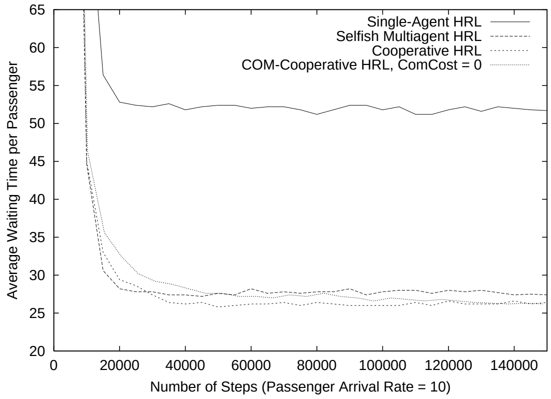


Figure 14. This figure shows that the average waiting time per passenger in the *Cooperative HRL* and the *COM-Cooperative HRL* with $ComCost = 0$ is less than the selfish multiagent HRL and the single-agent HRL.

Figure 15 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 previous 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.

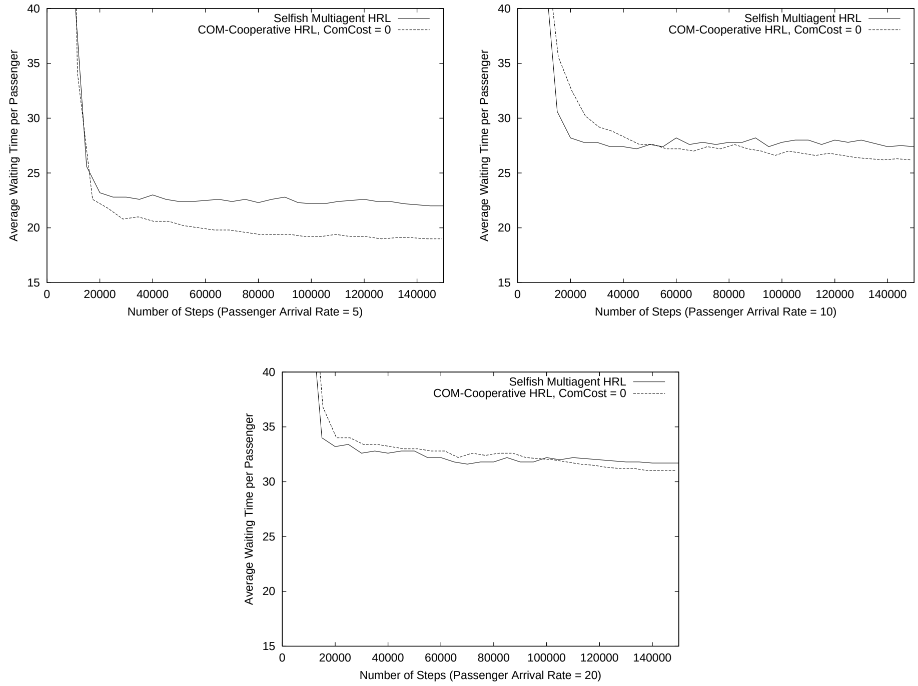
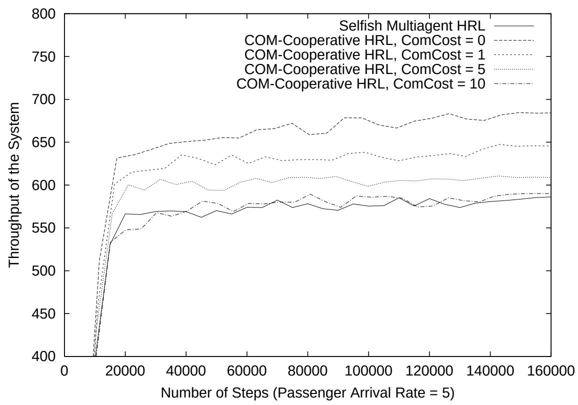


Figure 15. This figure compares the average waiting time per passenger for the selfish multiagent HRL and the *COM-Cooperative HRL* with $ComCost = 0$ for three different passenger arrival rates (5, 10 and 20). It shows that coordination among taxis becomes more crucial as the passenger arrival rate becomes smaller.

Figure 16 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.

7. Conclusion and Future Work

In this paper, we study methods for learning to communicate and act in cooperative multiagent systems using hierarchical reinforcement learning (HRL). The key idea underlying our approach is that coordination



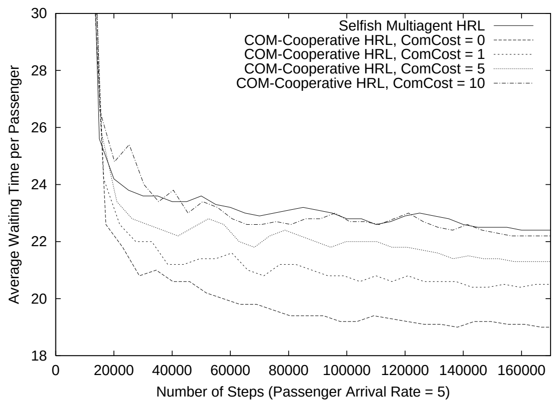


Figure 16. This figure shows that as communication cost increases, the throughput (top) and the average waiting time per passenger (bottom) of the *COM-Cooperative HRL* become closer to the selfish multiagent HRL. It indicates that agents learn to be selfish when communication is expensive.

skills are learned much more efficiently if agents have a hierarchical representation of the task structure. 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 define *cooperative subtasks* to be those subtasks in which coordination among agents has significant effect on 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 other subtasks. Since coordination at high-level allows for increased cooperation skills as agents do not get confused by low-level details, we usually define *cooperative subtasks* at the high levels of the hierarchy. A further advantage of this approach over flat learning methods is that, since high-level subtasks take a long time to complete, communication is needed fairly infrequently.

We exploit the ideas mentioned above and propose two new cooperative multiagent HRL algorithms, *Cooperative HRL* and *COM-Cooperative HRL*. In both algorithms, agents are homogeneous, i.e., use the same task decomposition, learning is decentralized and each agent learns three interrelated skills: how to perform subtasks, which order to do them in, and how to coordinate with other agents. In the *Cooperative HRL*, we assume communication is free and therefore the communication rule is very simple: when an agent takes an action at a *cooperative subtask*, it broadcasts it to all its teammates. We demonstrate the efficacy of this algorithm using two experimental testbeds: a simple simulated two-robot trash collection domain and a much larger four-agent automated guided vehicle (AGV) scheduling problem. We compare the performance of the *Cooperative HRL* algorithm with other algorithms such as the selfish multiagent HRL, the single-agent HRL and flat Q-learning algorithms in these domains. In the AGV scheduling domain, we also show that the *Cooperative HRL* outperforms widely used industrial heuristics, such as *"first come first serve"*, *"highest queue first"* and *"nearest station first"*.

In the *COM-Cooperative HRL* algorithm, we address the issue of rational communicative behavior among autonomous agents. The goal is to learn both action and communication policies that together optimize the task given the communication cost. This algorithm is an extension of the *Cooperative HRL* by including communication decisions in the model. We add a communication level to the hierarchical decomposition of the problem below each *cooperation level*. In this algorithm, the communication rule is more complicated than the *cooperative HRL*: before selecting an action at a *cooperative subtask*, each agent has to decide if it is worthwhile to perform a communication action in order to acquire the actions taken by the other agents at the *cooperative subtasks* at the same level of the hierarchy. 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.

There are a number of directions for future work which can be briefly outlined. An immediate question that arises is the classes of cooperative multiagent problems in which the proposed algorithms converge to a good approximation of optimal policy. The experiments of this paper show that the effectiveness of these algorithms is most apparent in tasks where agents rarely interact at the low levels (for example two trash collection robots may rarely need to exit through the same door at the same time). However, the algorithms can be easily generalized and adapted to constrained environments where agents are constantly running into one another (for example ten robots in a small room all trying to leave the room at the same time) by extending cooperation to lower levels of the hierarchy. This will result in a much larger set of action values that need to be learned, and consequently learning will be much slower. A number of extensions would be useful, from studying the scenario where agents are heterogeneous, to recognizing the high-level subtasks being performed by other agents using a history of observations instead of direct communication [7]. 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.

Another direction for future work is to study different termination schemes for composing temporally extended actions. We used $\tau_{continue}$ termination strategy in the algorithms proposed in this paper. However, it would be beneficial to investigate τ_{any} and τ_{all} termination schemes in our model. Many other manufacturing and robotics problems can benefit from these algorithms. Combining the proposed algorithms with function approximation and factored action models, which makes them more appropriate for continuous state problems, is also an important area of research. In this direction, we presented a family of HRL algorithms suitable for problems with continuous state space, using a mixture of policy gradient-based RL and value function-based RL methods [11]. We believe these algorithms can be easily extended to cooperative multiagent domains. The success of the proposed algorithms 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.

Acknowledgements

The computational experiments were carried out in the Autonomous Agents Laboratory in the Department of Computer Science and Engineering at Michigan State University under the Defense Advanced Research Projects Agency, DARPA contract No. DAANO2-98-C-4025, and the Autonomous Learning Laboratory in the Department of Computer Science at University of Massachusetts Amherst under NASA contract No. NAg-1445 #1. The first author would like to thank Balaraman Ravindran for his useful comments.

Notes

[1] This article significantly extends our previous conference paper [23].

References

1. Askin, R. and C. Standridge: 1993, *Modeling and Analysis of Manufacturing Systems*. John Wiley and Sons.
2. Balch, T. and R. Arkin: 1998, 'Behavior-based Formation Control for Multi-robot Teams'. *IEEE Transactions on Robotics and Automation* **14**, 1–15.
3. Barto, A. and S. Mahadevan: 2003, 'Recent Advances in Hierarchical Reinforcement Learning'. *Discrete Event Systems (Special Issue on Reinforcement Learning)* **13**, 41–77.
4. Bernstein, D. S., S. Zilberstein, and N. Immerman: 2000, 'The Complexity of Decentralized Control of Markov Decision Processes'. In: *Proceedings of the Sixteenth International Conference on Uncertainty in Artificial Intelligence (UAI)*. pp. 32–37.
5. Boutilier, C.: 1999, 'Sequential Optimality and Coordination in Multi-Agent Systems'. In: *Proceedings of the Sixteenth International Joint Conference on Artificial Intelligence (IJCAI)*. pp. 478–485.
6. Bowling, M. and M. Veloso: 2002, 'Multiagent Learning Using a Variable Learning Rate'. *Artificial Intelligence* **136**, 215–250.
7. Bui, H., S. Venkatesh, and G. West: 2002, 'Policy Recognition in the Abstract Hidden Markov Model'. *Journal of Artificial Intelligence Research* **17**, 451–499.
8. Crites, R. and A. Barto: 1998, 'Elevator Group Control using Multiple Reinforcement Learning Agents'. *Machine Learning* **33**, 235–262.
9. Dietterich, T.: 2000, 'Hierarchical reinforcement learning with the MAXQ value function decomposition'. *Journal of Artificial Intelligence Research* **13**, 227–303.
10. Filar, J. and K. Vrieze: 1997, *Competitive Markov Decision Processes*. Springer Verlag.
11. Ghavamzadeh, M. and S. Mahadevan: 2003, 'Hierarchical Policy Gradient Algorithms'. In: *Proceedings of the Twentieth International Conference on Machine Learning*. pp. 226–233.
12. Guestrin, C., M. Lagoudakis, and R. Parr: 2002, 'Coordinated Reinforcement Learning'. In: *Proceedings of the Nineteenth International Conference on Machine Learning*. pp. 227–234.
13. Howard, R.: 1971, *Dynamic Probabilistic Systems: Semi-Markov and Decision Processes*. John Wiley and Sons.
14. Hu, J. and M. Wellman: 1998, 'Multiagent Reinforcement Learning: Theoretical Framework and an Algorithm'. In: *Proceedings of the Fifteenth International Conference on Machine Learning*. pp. 242–250.
15. Kearns, M., M. Littman, and S. Singh: 2001, 'Graphical Models for Game Theory'. In: *Proceedings of the Seventeenth International Conference on Uncertainty in Artificial Intelligence(UAI)*.

16. Klein, C. and J. Kim: 1996, 'AGV Dispatching'. *International Journal of Production Research* **34**, 95–110.
17. Koller, D. and B. Milch: 2001, 'Multiagent Influence Diagrams for Representing and Solving Games'. In: *Proceedings of the Seventeenth International Joint Conference on Artificial Intelligence (IJCAI)*. pp. 1027–1034.
18. La Mura, P.: 2000, 'Game Networks'. In: *Proceedings of the Sixteenth International Conference on Uncertainty in Artificial Intelligence(UAI)*.
19. Lee, J.: 1996, 'Composite Dispatching Rules for Multiple-Vehicle AGV Systems'. *SIMULATION* **66**, 121–130.
20. Littman, M.: 1994, 'Markov Games as a Framework for Multi-Agent Reinforcement Learning'. In: *Proceedings of the Eleventh International Conference on Machine Learning*. pp. 157–163.
21. Littman, M.: 2001, 'Friend-or-Foe Q-Learning in General-Sum Games'. In: *Proceedings of the Eighteenth International Conference on Machine Learning*.
22. Littman, M., M. Kearns, and S. Singh: 2001, 'An Efficient Exact Algorithm for Singly Connected Graphical Games'. In: *Neural Information Processing Systems (NIPS)*.
23. Makar, R., S. Mahadevan, and M. Ghavamzadeh: 2001, 'Hierarchical multi-agent reinforcement learning'. In: *Proceedings of the Fifth International Conference on Autonomous Agents*. pp. 246–253.
24. Mataric, M.: 1997, 'Reinforcement Learning in the Multi-Robot Domain'. *Autonomous Robots* **4**, 73–83.
25. Ortiz, L. and M. Kearns: 2002, 'Nash Propagation for Loopy Graphical Games'. In: *Neural Information Processing Systems (NIPS)*.
26. Owen, G.: 1995, *Game Theory*. Academic Press.
27. Parr, R.: 1998, *Hierarchical Control and Learning for Markov Decision Processes*. PhD thesis, University of California at Berkeley.
28. Peshkin, L., K. Kim, N. Meuleau, and L. Kaelbling: 2000, 'Learning to Cooperate via Policy Search'. In: *Proceedings of the Sixteenth International Conference on Uncertainty in Artificial Intelligence (UAI)*. pp. 489–496.
29. Puterman, M.: 1994, *Markov Decision Processes*. Wiley Interscience.
30. Pynadath, D. and M. Tambe: 2002, 'The communicative multiagent team decision problem: Analyzing teamwork theories and models'. *Journal of Artificial Intelligence Research (JAIR)* **16**, 389–426.
31. Rohanimanesh, K. and S. Mahadevan: 2002, 'Learning to Take Concurrent Actions'. In: *Proceedings of the Sixteenth Annual Conference on Neural Information Processing Systems (NIPS)*.
32. Schneider, J., W. Wong, A. Moore, and M. Riedmiller: 1999, 'Distributed Value Functions'. In: *Proceedings of the Sixteenth International Conference on Machine Learning (ICML)*. pp. 371–378.
33. Singh, S., M. Kearns, and Y. Mansour: 2000, 'Nash Convergence of Gradient Dynamics in General-Sum Games'. In: *Proceedings of the Sixteenth International Conference on Uncertainty in Artificial Intelligence (UAI)*. pp. 541–548.
34. Stone, P. and M. Veloso: 1999, 'Team-Partitioned, Opaque-Transition Reinforcement Learning'. In: *Proceedings of the Third International Conference on Autonomous Agents*.
35. Sugawara, T. and V. Lesser: 1998, 'Learning to Improve Coordinated Actions in Cooperative Distributed Problem-Solving Environments'. *Machine Learning* **33**, 129–154.

36. Sutton, R., D. Precup, and S. Singh: 1999, 'Between MDPs and Semi-MDPs: A Framework for Temporal Abstraction in Reinforcement Learning'. *Artificial Intelligence* **112**, 181–211.
37. Tadepalli, P. and D. Ok: 1996, 'Scaling up Average Reward Reinforcement Learning by Approximating the Domain Models and the Value Function'. In: *Proceedings of the Thirteenth International Conference on Machine Learning.* pp. 471–479.
38. Tan, M.: 1993, 'Multi-Agent Reinforcement Learning: Independent vs. Cooperative Agents'. In: *Proceedings of the Tenth International Conference on Machine Learning.* pp. 330–337.
39. Vickrey, D. and D. Koller: 2002, 'Multiagent Algorithms for Solving Graphical Games'. In: *Proceedings of the National Conference on Artificial Intelligence (AAAI).*
40. Watkins, C.: 1989, *Learning from Delayed Rewards.* PhD thesis, Kings College, Cambridge, England.
41. Weiss, G.: 1999, *Multi-agent Systems: A Modern Approach to Distributed Artificial Intelligence.* MIT Press.
42. Xuan, P. and V. Lesser: 2002, 'Multiagent Policies: from Centralized ones to Decentralized ones'. In: *Proceedings of the First International Joint Conference on Autonomous Agents and Multiagent Systems.*
43. Xuan, P., V. Lesser, and S. Zilberstein: 2001, 'Communication Decisions in Multi-Agent Cooperation: Model and Experiments'. In: *Proceedings of the Fifth International Conference on Autonomous Agents.* pp. 616–623.

Algorithm 1 The Cooperative HRL algorithm.

Function Cooperative-HRL(Agent j, Task i at the lth level of the hierarchy, State s)
let *Seq* $= \{\}$ be the sequence of (*state-visited, actions in* $\bigcup_{k=1}^{L} U_k$ *being performed by the other agents*) while executing i /* L is the number of levels in the hierarchy */
if i is a primitive action **then**
  execute action 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 /* i is a non-primitive subtask */
  while i has not terminated **do**
    if i is a *cooperative subtask* **then**
      choose action a^j according to the current exploration policy $\pi_i^j(s, a^1, \ldots, a^{j-1}, a^{j+1}, \ldots, a^n)$
      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 action a^j
      observe result state s' and $\hat{a}^1, \ldots, \hat{a}^{j-1}, \hat{a}^{j+1}, \ldots, \hat{a}^n$ actions in U_l being performed by the other agents
      let $a^* = argmax_{a' \in A_i} [C_t^j(i, s', \hat{a}^1, \ldots, \hat{a}^{j-1}, \hat{a}^{j+1}, \ldots, \hat{a}^n, a') + V_t^j(a', s')]$
      let $N = 0$
      for each (s, $a^1, \ldots, a^{j-1}, a^{j+1}, \ldots, a^n$) in *ChildSeq* from the beginning **do**
        $N = N + 1$
        $C_{t+1}^j(i, s, a^1, \ldots, a^{j-1}, a^{j+1}, \ldots, a^n, a^j) \longleftarrow (1-\alpha_t^j(i)) C_t^j(i, s, a^1, \ldots, a^{j-1}, a^{j+1}, \ldots, a^n, a^j) + \alpha_t^j(i) \gamma^N [C_t^j(i, s', \hat{a}^1, \ldots, \hat{a}^{j-1}, \hat{a}^{j+1}, \ldots, \hat{a}^n, a^*) + V_t^j(a^*, s')]$
      end for
    else /* i is not a *cooperative subtask* */
      choose action 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 action a^j
      observe result state s'
      let $a^* = argmax_{a' \in A_i} [C_t^j(i, s', a') + V_t^j(a', s')]$
      let $N = 0$
      for each state 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