Auton Agent Multi-Agent Sys (2006) 13: 197–229
DOI 10.1007/s10458-006-7035-4

Hierarchical multi-agent reinforcement learning

Mohammad Ghavamzadeh · Sridhar Mahadevan · Rajbala Makar

Published online: 4 April 2006
Springer Science+Business Media, LLC 2006

Abstract In this paper, we investigate the use of hierarchical reinforcement learning (HRL) to speed up the acquisition of cooperative multi-agent tasks. We introduce a hierarchical multi-agent reinforcement learning (RL) framework, and propose a hierarchical multi-agent RL algorithm called *Cooperative HRL*. In this framework, agents are cooperative and homogeneous (use the same task decomposition). Learning is decentralized, with each agent learning three interrelated skills: how to perform each individual subtask, the order in which to carry them out, 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*. A fundamental property of the proposed approach is that it 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 study the empirical performance of the *Cooperative HRL* algorithm using two testbeds: a simulated two-robot trash collection task, and a larger four-agent automated guided vehicle (AGV) scheduling problem. We compare the performance and speed of *Cooperative HRL* with other learning algorithms, as well as several well-known industrial AGV heuristics. We also address the issue of rational communication behavior among autonomous agents in this paper. The goal is for agents to learn both action and communication policies that together optimize the task given a communication cost. We extend the multi-agent HRL framework to include communication decisions and propose a cooperative multi-agent 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 an agent makes a decision at a *cooperative*

M. Ghavamzadeh (✉)
Department of Computing Science, University of Alberta, AB T6G 2E8, Canada
e-mail: mgh@cs.ualberta.ca

S. Mahadevan
Department of Computer Science, University of Massachusetts Amherst, MA 01003, USA
e-mail: mahadeva@cs.umass.edu

R. Makar
Agilent Technologies, Santa Rosa, CA 95403, USA
e-mail: makar@agilent.com

subtask, it decides if it is worthwhile to perform a communication action. A communication action has a certain cost and provides the agent with the actions selected by the other agents at a *cooperation level*. We demonstrate the efficiency of the *COM-Cooperative HRL* algorithm as well as the relation between the communication cost and the learned communication policy using a multi-agent taxi problem.

Keywords Hierarchical reinforcement learning · Cooperative multi-agent systems · Coordination · Communication

1. Introduction

A multi-agent system is a system in which several interacting, intelligent agents pursue some set of goals or perform some set of tasks [43]. In these systems, decisions of an agent usually depend on the behavior of the other agents, which is often not predictable. It makes learning and adaptation a necessary component of an agent. Multi-Agent learning studies algorithms for selecting actions for multiple agents coexisting in the same environment. This is a complicated problem, because the behavior 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 [25]. Robosoccer, disaster rescue, and e-commerce are examples of challenging multi-agent domains that need robust learning algorithms for coordination among multiple agents or for effectively responding to other agents.

Multi-Agent 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 a decision are not fully observable and inter-agent communication is usually costly. Prior work in multi-agent learning has addressed these issues in many different ways, as we will discuss in detail in Section 2.

In this paper, we investigate the use of hierarchical reinforcement learning (HRL) to address the *curse of dimensionality* and *partial observability* in order to accelerate learning in cooperative[1] multi-agent systems. Our approach differs from the previous work in one key respect, namely the use of task hierarchies to scale multi-agent reinforcement learning (RL). We originally proposed this approach in [24], and subsequently extended it in [12]. 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 [38], HAMs [28], and MAXQ [9]. The key idea underlying our approach is that coordination skills are learned much more efficiently if the agents have a hierarchical representation of the task structure. Algorithms for learning task-level coordination have already been developed in non-MDP approaches [37], however to the best of our knowledge, our work has been the first attempt to use task-level coordination in an MDP setting. The use of hierarchy speeds up learning in multi-agent 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

[1] We are primarily interested in cooperative multi-agent problems in this paper.

them at the other subtasks. *Cooperative subtasks* are usually defined at the highest level(s) of a hierarchy. Coordination at high-level provides significant advantage over flat methods by preventing agents from getting confused by low-level details and reducing the amount of communication needed for proper coordination among agents.

These benefits can be potentially achieved using any type of HRL algorithm. However, it is necessary to generalize the HRL frameworks to make them more applicable to multi-agent learning. In this paper, initially we assume that communication is free and propose a hierarchical multi-agent RL algorithm called *Cooperative HRL*. We apply the *Cooperative HRL* algorithm to a simple two-robot trash collection task and a complex four-agent automated guided vehicle (AGV) scheduling problem. We compare its performance and speed with selfish multi-agent HRL, as well as single-agent HRL and standard Q-learning algorithms. In the AGV scheduling problem, we also demonstrate that the *Cooperative HRL* algorithm outperforms widely used industrial heuristics, such as *"first come first serve"*, *"highest queue first"*, and *"nearest station first"*. 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 multi-agent 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 learned communication policy using a multi-agent taxi problem.

The rest of this paper is organized as follows. Section 2 provides a brief overview of the related work in multi-agent learning. Section 3 describes a framework for hierarchical multi-agent RL which is used to develop the algorithms of this paper. In Section 4, we introduce a HRL algorithm, called *Cooperative HRL* for learning in cooperative multi-agent domains. Section 5 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 6, 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 6.1, we illustrate *COM-Cooperative HRL*, a multi-agent HRL algorithm with communication decisions in Section 6.2. Section 7 presents experimental results of using the *COM-Cooperative HRL* algorithm in a multi-agent taxi problem. Finally, Section 8 summarizes the paper and discusses some directions for future work.

2. Related work

The analysis of multi-agent systems has been a topic of interest in both economic theory and artificial intelligence (AI). While multi-agent systems have been widely studied in game theory, only in the last one or two decades they have started to attract interest in AI, where their integration with existing methods constitutes a promising area of research. The game theoretic concepts of stochastic games and Nash equilibria [10, 27] are the foundation for much of the recent research in multi-agent 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 [15], Minimax-Q [21], and Friend-or-Foe-Q [22] seek to learn an equilibrium of the game by iteratively computing intermediate equilibria. Under certain conditions, 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 multi-agent algorithm, Q-learning [42] was one of the first algorithms applied to multi-agent problems [8, 40]. WoLF-PHC [6], joint-state/joint-action learners [5], and the gradient ascent learner in [35] are other 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].

The RL framework has been well-studied in multi-agent domains. Prior work in multi-agent RL has 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 RL [34] as well as policy gradient based RL [29]. Another approach is to exploit the structure in a multi-agent problem using factored value functions. Guestrin et al. [13] integrate these ideas in collaborative multi-agent 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 multi-agent systems. The goal is to transfer the representational and computational benefits that graphical models provide to probabilistic inference in multi-agent systems and game theory [18, 19]. The previous work established algorithms for computing Nash equilibria in one-stage games, including efficient algorithms for computing approximate [16] and exact [23] Nash equilibria in tree-structured games, and convergent heuristics for computing Nash equilibria in general graphs [26, 41].

The curse of dimensionality has also been addressed in multi-agent robotics. Multi-robot learning methods usually reduce the complexity of the problem by not modeling joint states or actions explicitly, such as work by Balch and Arkin [2] and Mataric [25], among others. In such systems, each robot maintains its position in a 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 multi-agent domains, by learning action values over a set of derived features [36]. These derived features are domain specific, and have to be encoded by hand, or constructed by a supervised learning algorithm.

In a cooperative multi-agent setting, it is usually necessary for each agent to have information about the other agents in order to make its own decision. Almost all the above methods ignore the important fact that an agent may not have free access to these information. In general, the world is partially observable for each agent in a distributed multi-agent setting. Partially observable Markov decision processes (POMDPs) have been used to model partial observability in probabilistic AI. A POMDP is a generalization of an 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 multi-agent 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 [44, 45]. Pynadath and Tambe [31] extended DEC-POMDP by including communication decisions in the model, and proposed a framework called communicative multi-agent 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 [31]. 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 multi-agent learning and planning.

3. Hierarchical multi-agent reinforcement learning

In this section, we introduce a hierarchical multi-agent RL framework in which agents are capable of learning simultaneously at multiple levels of hierarchy. This is the framework underlying the hierarchical multi-agent RL algorithms presented in this paper. The main contribution of this framework is that it enables agents to exploit the hierarchical structure of the task in order to learn coordination strategies more efficiently. Our hierarchical multi-agent RL framework can be viewed as extending the existing single-agent HRL methods, including hierarchies of abstract machines (HAMs) [28], options [38], and MAXQ [9], especially the MAXQ value function decomposition [9], to the cooperative multi-agent setting.

3.1. Motivating example

We use a simple example to illustrate the overall approach. 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 Fig. 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 carry out the subtasks, for example 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 $T2$.

The strength of the HRL methods (when extended to the multi-agent domains) 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 it 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

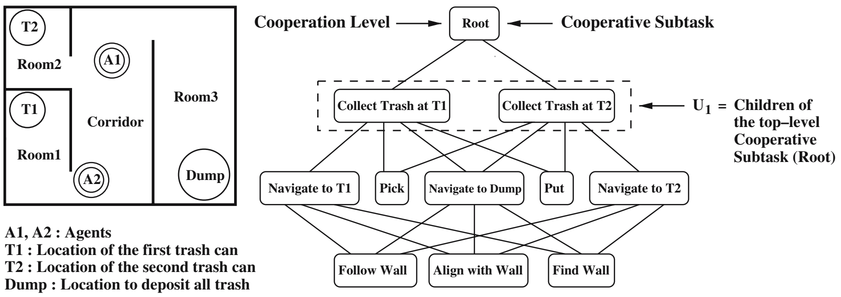


Fig. 1 A multi-agent trash collection task and its associated task graph

goal. For example, *navigate to* $T1$, $T2$, and *Dump* use three primitive actions *find wall*, *align with wall*, and *follow wall*. *Collect trash at* $T1$ uses two subtasks *navigate to* $T1$ and *Dump*, plus two primitive actions *Put* and *Pick*, and so on. All of this information is summarized by a directed acyclic graph called the *task graph*. The task graph for the trash collection problem is shown in Fig. 1. This hierarchical model is able to support **state abstraction** (while the agent is moving toward the *Dump*, the status of trash cans $T1$ and $T2$ is irrelevant and cannot affect this navigation process. Therefore, the variables defining the status of trash cans $T1$ and $T2$ can be removed from the state space of the *navigate to Dump* subtask), and **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 $T2$ subtasks).

3.2. Multi-agent semi-Markov decision processes

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 [14, 30] extend the MDP model in several aspects. Decisions are only made at discrete points in time. The 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 multi-agent domains when a team of agents controls the process, and introduce the **multi-agent 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 multi-agent extension of MDP, the MMDP model [5], the multi-agent extension of SMDP requires extending the notion of a decision making event.

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

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

Three termination strategies τ_{any}, τ_{all}, and τ_{continue} for temporally extended joint actions were introduced and analyzed in [32]. 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 (e.g., agent $A1$ 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 (e.g., 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, the next decision epoch occurs and all the agents choose the next joint action together. In both these termination strategies, all the agents choose actions at every decision epoch. The τ_{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 whose activities have not terminated are not interrupted and only those agents whose actions have terminated select new actions. In this termination strategy, only a subset of the agents choose action at each decision epoch. When an agent completes an action, the next decision epoch occurs only for that agent and it selects its next action given the actions being performed by the other agents.

The three termination strategies described above are the most common, but not the only termination schemes in cooperative multi-agent systems. A wide range of termination strategies can be defined based on them. Of course, not all these strategies are appropriate for any given multi-agent task. We categorize termination strategies as *synchronous* and *asynchronous*. In **synchronous** schemes, such as τ_{any} and τ_{all}, all the agents select action at every decision epoch and therefore we need a centralized mechanism to synchronize the agents at decision epochs. In **asynchronous** strategies, such as τ_{continue}, only a subset of the agents choose action at each decision epoch. In this case, there is no need for a centralized mechanism to synchronize the agents and decision making can take place in a decentralized fashion. Since our goal is to design decentralized multi-agent RL algorithms, we use the τ_{continue} termination scheme for joint action selection in the hierarchical multi-agent model and algorithms presented in this paper.

While SMDP theory provides the theoretical underpinnings of temporal abstraction by modeling 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 decompose value function, etc. We examine these issues in the next sections.

3.3. Hierarchical task decomposition

A task hierarchy such as the one illustrated in Section 3.1 can be modeled by decomposing the overall task MDP M, into a finite set of subtasks $\{M_0, \ldots, M_{m-1}\}$, where M_0 is the *root* task and solving it solves the entire MDP M.

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

– S_i is the **state space** for subtask M_i. It is described by those state variables that are relevant to subtask M_i.[2] The range of a state variable describing S_i might be a subset of its range in $\mathcal{S}$ (the state space of MDP M).

[2] State variables relevant to subtask M_i are those state variables that are important in solving subtask M_i.

- $I_i \subset S_i$ is the **initiation set** for subtask M_i. Subtask M_i can be initiated only in states belonging to I_i.
- $T_i \subset S_i$ is the **set of terminal states** for subtask M_i. Subtask M_i terminates when it reaches a state in T_i. A policy for subtask M_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 M_i. These actions can be either primitive actions from $\mathcal{A}$ (the set of primitive actions for MDP M), or they can be other subtasks. Technically, A_i is a function of state, since it may differ from one state to another. However, we will suppress this dependence in our notation.
- R_i is the **reward function** of subtask M_i.

Each primitive action a is a primitive subtask in this decomposition, such that a is always executable and it terminates immediately after execution. From now on in this paper, we use subtask to refer to non-primitive subtasks.

3.4. Policy execution

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

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

A hierarchical policy is executed 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 Task-Stack and its policy is executed until it enters one of its terminal states. When a subtask terminates, its name is popped off the Task-Stack. If any subtask on the Task-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, the *Root* task is located at the bottom and the subtask which is currently being executed is located at the top of the Task-Stack. Under a hierarchical policy π, we define a multi-step transition probability function $P_i^\pi : S_i \times \mathbb{N} \times S_i \to [0, 1]$ for each subtask M_i in the hierarchy, where $P_i^\pi(s', N|s)$ denotes the probability that the hierarchical policy π will cause the system to transition from state s to state s' in N primitive steps at subtask M_i.

3.5. Multi-agent setup

In our hierarchical multi-agent framework, we assume that there are n agents in the environment, cooperating with each other to accomplish a task. The designer of the system uses her/his domain knowledge to recursively decompose the overall task into a collection of subtasks that she/he believes are important for solving the problem.[3] 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.[4] At each level of the hierarchy, the designer of the system defines *cooperative subtasks* to be those in which coordination among agents significantly increases the performance of the overall task. The set

[3] Providing hierarchical decompositions manually is not a difficult process in many problems. However, we are eventually interested in deriving hierarchical decompositions automatically, and it is currently an active line of research in RL (see [3] for more details on automatic abstraction).

[4] 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.

of all *cooperative subtasks* at a certain level of the hierarchy is called the *cooperation set* of that level. Each level of the hierarchy with non-empty *cooperation set* is called a *cooperation level*. The union of the children of the lth level *cooperative subtasks* is represented by U_l. Since high-level coordination allows for increased cooperation skills as agents do not get confused by low-level details, we usually define *cooperative subtasks* at the highest level(s) of the hierarchy. Agents actively coordinate while making decision at *cooperative subtasks* and are ignorant about other agents at *non-cooperative subtasks*. Thus, we configure *cooperative subtasks* to model joint action values.

In the trash collection problem, we define *Root* as a *cooperative subtask*. As a result, the top-level of the hierarchy is a *cooperation level*, *Root* is the only member of the *cooperation set* at the top-level, and U_1 consists of all subtasks located at the second level of the hierarchy, $U_1 = \{collect\ trash\ at\ T1,\ collect\ trash\ at\ T2\}$ (see Fig. 1). As it is clear in this problem, it is more effective that an agent learns high-level coordination knowledge (what is the utility of agent $A2$ collecting trash from trash can $T1$ if agent $A1$ is collecting trash from trash can $T2$), rather than learning its response to low-level primitive actions of the other agents (what agent $A2$ should do if agent $A1$ aligns with wall). Therefore we define single-agent policies for *non-cooperative subtasks* and joint policies for *cooperative subtasks*.

Definition 4 Under a hierarchical policy π, each *non-cooperative subtask* M_i can be modeled by an SMDP consisting of components $(S_i, A_i, P_i^{\pi}, R_i)$.

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

Υ is the set of n agents in the team. We assume that agents have only local state information and ignore the states of the other agents. 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 multi-agent RL problem. This approximation is based on the fact that an agent can get a rough idea of what state the other agents might be in just by knowing the high-level actions being performed by them. The action space is joint and is defined as $\mathcal{A}_i = A_i \times (U_l)^{n-1}$, where U_l is the union of the action sets of all the lth level *cooperative subtasks*. For the *cooperative subtask Root* in the trash collection problem, the set of agents is $\Upsilon = \{A1, A2\}$, and its joint action space, $\mathcal{A}_{\text{root}}$, is specified as the cross product of its action set, A_{root}, and U_1, $\mathcal{A}_{\text{root}} = A_{\text{root}} \times U_1$. Finally, since we are interested in decentralized control, we use the τ_{continue} termination strategy. Therefore, when an agent completes a subtask, the next decision epoch occurs only for that agent and it selects its next action given the information about the other agents. □

This cooperative multi-agent 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 level(s) 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 have not been defined as *cooperative*. This issue will be addressed in one of the AGV experiments in Section 5.2, 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 as a result increases 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 6 by including communication decisions in the model. If communication is cheap, agents learn to cooperate 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 the other agents increases the complexity of the learning problem and has its own inefficiency. We do not explicitly address this problem in the paper.

3.6. Value function decomposition

A value function decomposition splits the value of a state or a state-action pair into multiple additive components. Modularity in the hierarchical structure of a task allows us to carry out this decomposition along subtask boundaries. The value function decomposition used in our hierarchical framework is similar to the MAXQ value function decomposition [9]. The purpose of a value function decomposition is to decompose the value function of the overall task (*Root*) under a hierarchical policy π, $V^{\pi}(0, s)$, in terms of the value functions of all the subtasks in the hierarchy. The value function of subtask M_i under a hierarchical policy π, $V^{\pi}(i, s)$, is the expected sum of discounted reward until subtask M_i terminates and can be written as:

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

Now let us suppose that the policy of subtask M_i, π_i, chooses subtask $\pi_i(s)$ in state s, this subtask executes for a number of steps N and terminates in state s' according to $P_i^{\pi}(s', N|s, \pi_i(s))$. 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 on the right-hand side 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 on the right-hand side of the equation is the value of state s' for the current subtask M_i under hierarchical policy π, $V^{\pi}(i, s')$, discounted by γ^N, where s' is the current state when subtask $\pi_i(s)$ terminates and N is the number of transition steps from state s to state s'. We can therefore write the Equation 2 in the form of a Bellman equation:

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

Equation 3 can be re-stated for the action-value function as follows:

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

The right-most term in this equation is the expected discounted cumulative reward of completing subtask M_i after executing subtask M_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 M_i after execution of subtask M_a in state s. The reward is discounted back to the point in time where M_a begins execution.

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

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

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

and the value function V as:

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

Equations 5, 6, and 7 are referred to as decomposition equations for a hierarchy 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 functions for the individual subtasks, $M_1, \ldots, M_{m-1}$, and the individual completion functions $C^{\pi}(i,s,a)$ for $i = 1, \ldots, m-1$. The fundamental quantities that must be stored to represent the value function decomposition are the C values for non-primitive subtasks and the V values for primitive actions. This decomposition is summarized graphically in Fig. 2.

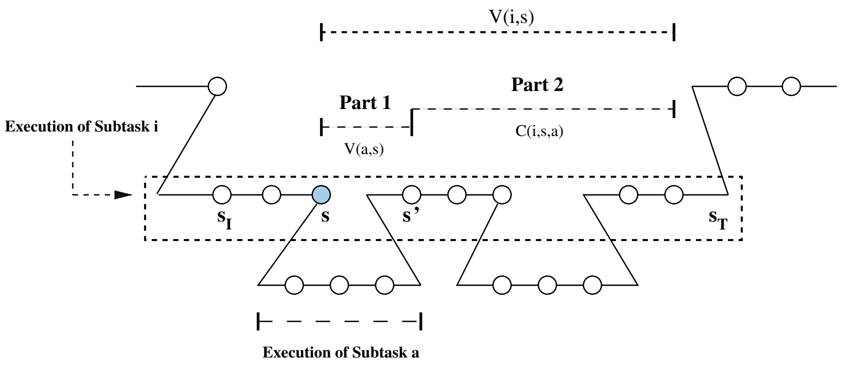


Fig. 2 This figure shows the decomposition for $V(i,s)$, the value function of subtask M_i for the shaded state s. Each circle is a state of the SMDP visited by the agent. Subtask M_i is initiated at state s_I and terminates at state s_T. The value function $V(i,s)$ is broken into two parts: **Part 1)** the value function of subtask M_a for state s, and **Part 2)** the completion function, the expected discounted cumulative reward of completing subtask M_i after executing subtask M_a in state s

Using this decomposition and the stored values, we can recursively calculate all the Q values in a hierarchy. For example, $Q(i, s, a)$ 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 M_i, so subtask M_i only needs to calculate $V(a, s)$ by asking subtask M_a.
– If M_a is a primitive action, it stores $V(a, s)$ and returns it to subtask M_i immediately. If M_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 M_a, and M_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 M_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 multi-agent model, we configure *cooperative subtasks* to store the joint completion function values.

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

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

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

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

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 $V^j(a^j, s, \tilde{a}^1, \ldots, \tilde{a}^{j-1}, \tilde{a}^{j+1}, \ldots, \tilde{a}^n)$, where $\tilde{a}^1, \ldots, \tilde{a}^{j-1}, \tilde{a}^{j+1}, \ldots, \tilde{a}^n$ belong to U_{l+1}. In this case, $V^j(a^j, s)$ is replaced by $V^j(a^j, s, \tilde{a}^1, \ldots, \tilde{a}^{j-1}, \tilde{a}^{j+1}, \ldots, \tilde{a}^n)$ in Equation 8.

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:

$$\begin{aligned} &Q^1(root, s, collect\ trash\ at\ T2, collect\ trash\ at\ T1) \\ &\quad = V^1(collect\ trash\ at\ T1, s) + C^1(root, s, collect\ trash\ at\ T2, collect\ trash\ at\ T1) \end{aligned}$$

which represents the value of agent $A1$ performing *collect trash at T*1 in the context of the overall task (*Root*), when agent $A2$ is executing *collect trash at T*2. Note that this value is decomposed into the value of subtask *collect trash at T*1 (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 the levels below *Root* in one of the experiments of Section 5.2.

4. A Hierarchical multi-agent reinforcement learning algorithm

In this section, we use the hierarchical multi-agent RL framework described in Section 3 and present a hierarchical multi-agent RL algorithm, called *Cooperative HRL*. The pseudo code for this algorithm is shown in Table 1. In the *Cooperative HRL* algorithm, the V and C values can be learned through a standard temporal-difference (TD) learning method based on sample trajectories. One important point to note is that since non-primitive subtasks are temporally extended in time, the update rules for the 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 until it reaches a primitive action M_i. It executes primitive action M_i in state s, receives reward r, observes resulting state s', and updates the value function V of primitive subtask M_i 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 primitive action M_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 a non-primitive subtask M_i and is in state s, then while subtask M_i does not terminate, it chooses subtask M_a according to the current exploration policy (softmax or ϵ-greedy with respect to $\pi_i(s)$). If subtask M_a takes N primitive steps and terminates in state s', the corresponding C value is updated using:

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

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

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

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

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

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

Table 1 The Cooperative HRL algorithm

```
1:  Function Cooperative-HRL(Agent $j$, Task $M_i$ at the $l$th level of the hierarchy, State $s$)
2:  let Seq={ } be the sequence of (state-visited, actions in $\bigcup_{k=1}^{L} U_k$ being performed by the other agents)
    while executing $M_i$     /* $L$ is the number of levels in the hierarchy */
3:  if $M_i$ is a primitive action then
4:      execute action $M_i$ in state $s$, receive reward $r(s'|s,i)$ and observe state $s'$
5:      $V_{t+1}^{j}(i,s) \longleftarrow [1-\alpha_t^j(i)]V_t^j(i,s) + \alpha_t^j(i) r(s'|s,i)$
6:      push (state $s$, actions in $\{U_l | l \text{ is a cooperation level}\}$ being performed by the
        other agents) onto the front of Seq
7:  else     /* $M_i$ is a non-primitive subtask */
8:      while $M_i$ has not terminated do
9:          if $M_i$ is a cooperative subtask then
10:             choose subtask $a^j$ according to the current exploration policy $\pi_i^j(s, a^1, \ldots,$
                $a^{j-1}, a^{j+1}, \ldots, a^n)$
11:             let ChildSeq = Cooperative-HRL($j$, $a^j$, $s$), where ChildSeq is the sequence
                of (state-visited, actions in $\bigcup_{k=1}^{L} U_k$ being performed by the other agents)
                while executing subtask $a^j$
12:             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
13:             let $a^* = \arg\max_{a' \in A_i} [C_t^j(i, s', \hat{a}^1, \ldots, \hat{a}^{j-1}, \hat{a}^{j+1}, \ldots, \hat{a}^n, a') + V_t^j(a', s')]$
14:             let $N = 0$
15:             for each $(s, a^1, \ldots, a^{j-1}, a^{j+1}, \ldots, a^n)$ in ChildSeq from the beginning do
16:                 $N = N + 1$
17:                 $C_{t+1}^j(i, s, a^1, \ldots, a^{j-1}, a^{j+1}, \ldots, a^n, a^j) \longleftarrow$
                    $[1-\alpha_t^j(i)]C_t^j(i, s, a^1, \ldots, a^{j-1}, a^{j+1}, \ldots, a^n, a^j) +$
                    $\alpha_t^j(i)\gamma^N [C_t^j(i, s', \hat{a}^1, \ldots, \hat{a}^{j-1}, \hat{a}^{j+1}, \ldots, \hat{a}^n, a^*) + V_t^j(a^*, s')]$
18:             end for
19:         else     /* $M_i$ is not a cooperative subtask */
20:             choose subtask $a^j$ according to the current exploration policy $\pi_i^j(s)$
21:             let ChildSeq = Cooperative-HRL($j$, $a^j$, $s$), where ChildSeq is the sequence
                of (state-visited, actions in $\bigcup_{k=1}^{L} U_k$ being performed by the other agents)
                while executing subtask $a^j$
22:             observe result state $s'$
23:             let $a^* = \arg\max_{a' \in A_i} [C_t^j(i, s', a') + V_t^j(a', s')]$
24:             let $N = 0$
25:             for each state $s$ in ChildSeq from the beginning do
26:                 $N = N + 1$
27:                 $C_{t+1}^j(i, s, a^j) \leftarrow [1-\alpha_t^j(i)]C_t^j(i, s, a^j) + \alpha_t^j(i)\gamma^N [C_t^j(i, s', a^*) + V_t^j(a^*, s')]$
28:             end for
29:         end if
30:         append ChildSeq onto the front of Seq
31:         $s = s'$
32:     end while
33: end if
34: return Seq
35: end Cooperative-HRL
```

where $a^* = \arg\max_{a' \in A_i} [C_t^j(i, s', \hat{a}^1, \ldots, \hat{a}^{j-1}, \hat{a}^{j+1}, \ldots, \hat{a}^n, a') + V_t^j(a', s')]$, $a^1, \ldots, a^{j-1}, a^{j+1}, \ldots, a^n$ 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 Table 1.

5. Experimental results for the cooperative HRL algorithm

In this section, we demonstrate the performance of the *Cooperative HRL* algorithm proposed in Section 4 using a two-robot trash collection problem, and a more complex four-agent AGV scheduling task. In these experiments, 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.

5.1. 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 nine states, the corridor into four 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, the experiment was repeated ten times and the results averaged.

In the two-robot trash collection task, examination of the learned policy in Fig. 3 reveals that the robots have nicely learned all three skills: how to achieve a subtask, what order to carry them out, and how to coordinate with each other. The coordination strategy learned by the robots here is robot $A1$ collects trash only from trash can $T1$ and robot $A2$ collects trash only from trash can $T2$. In addition, as Figure 4 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.

5.2. 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 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 using AGVs faces the problem of optimally scheduling the paths of the AGVs in the system [20]. For example, a move request occurs when a part finishes at a workstation. If

Fig. 3 This figure shows the policy learned by the *Cooperative HRL* algorithm in the two-robot trash collection 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
```

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 manufacturing environments makes it difficult 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 [17, 20]. However, the heuristics perform poorly when the constraints on the movement of the AGVs are reduced.

Previously, Tadepalli and Ok [39] studied a single-agent AGV scheduling task using *flat* average-reward RL. However, the multi-agent AGV task we study is more complex. Figure 5 shows the layout of the AGV scheduling domain used in the experiments of this paper. $M1$ to $M4$ show workstations in this environment. Parts of type i have to be carried to the

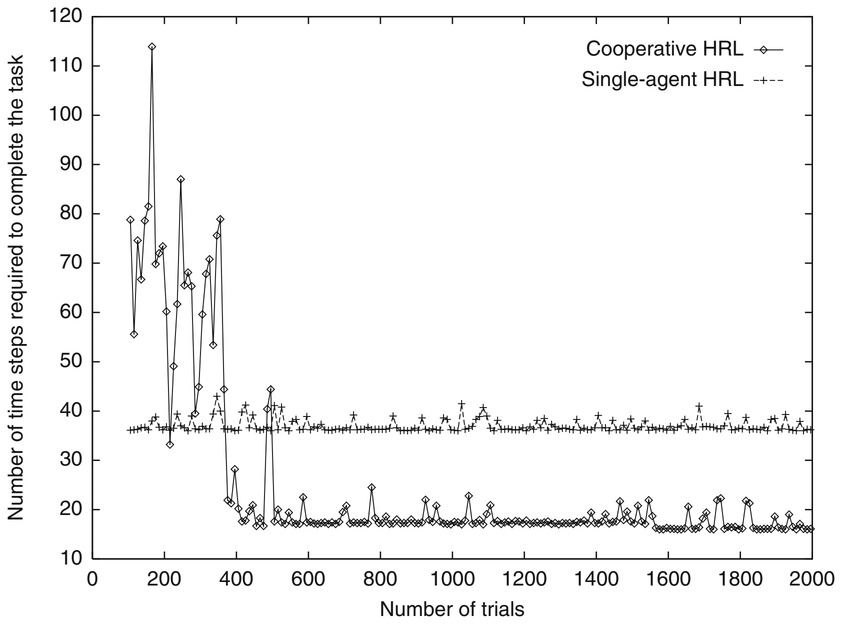


Fig. 4 This figure shows that the *Cooperative HRL* algorithm learns the trash collection task with fewer number of steps than the single-agent HRL algorithm

drop-off station at workstation i, D_i, and the assembled parts brought back from the pick-up stations of workstations, P_i's, to the warehouse. The AGV travel is unidirectional (as the arrows show). This task is decomposed using the task graph in Fig. 6. 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 ($DM1, \ldots, DM4, DA1, \ldots, DA4$) belong to set U_1. Coordination skills among agents are learned by using joint action-values at the highest level of the hierarchy as described in Section 4.

The state of the environment consists of the number of parts in the pick-up and drop-off stations of each machine, and whether the warehouse contains parts of each of the four types. In addition, each agent keeps track of its own location and status as a part of its state space. Thus, in the flat case, the state space consists of 100 locations, eight buffers of size 3, 9 possible states of AGV (carrying part1,. . ., carrying assembly1, . . ., empty), and 2 values for each part in the warehouse, i.e., $100 \times 4^8 \times 9 \times 2^4 \approx 10^9$ states. 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 navigation subtasks, only the *location* state variable is relevant, and this subtask can be learned with 100 values. Hence, for the highest level subtasks $DM1, \ldots, DM4$, the number of relevant states would be $100 \times 9 \times 4 \times 2 = 7,200$, and for the highest level subtasks $DA1, \ldots, DA4$, the number of relevant states would be $100 \times 9 \times 4 = 3,600$. 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 multi-agent HRL, and *Cooperative HRL*, the cooperative multi-agent HRL algorithm proposed in Section 4. In the experiments of this section, we assume that there are four agents (AGVs) in the environment. The

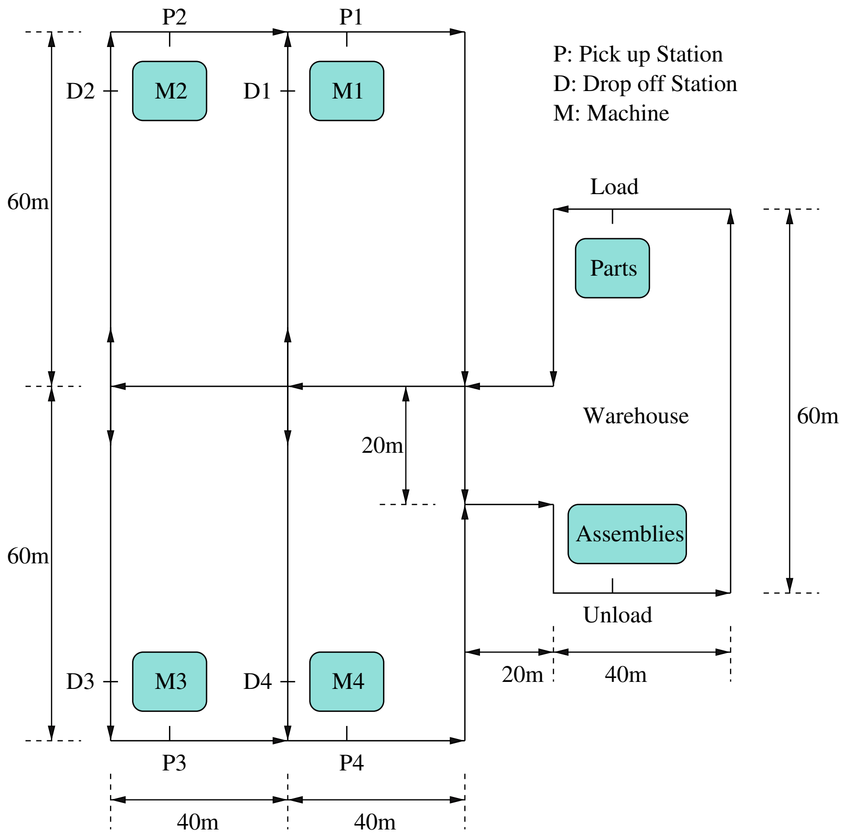


Fig. 5 A multi-agent AGV scheduling domain. There are four AGVs (not shown) which carry raw materials and finished parts between machines and the warehouse

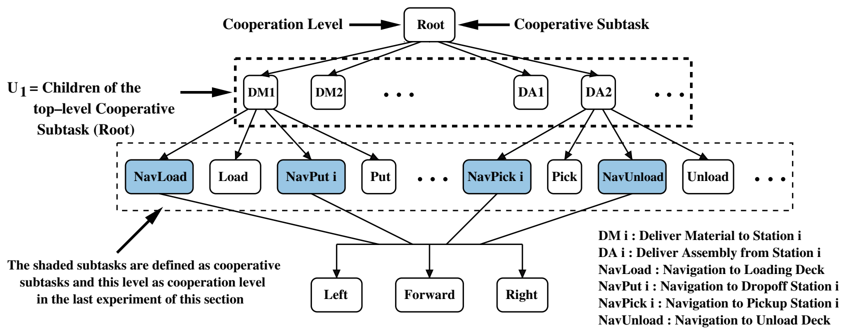

Cooperation Level
Root
Cooperative Subtask
U1 = Children of the top–level Cooperative Subtask (Root)
DM1
DM2
DA1
DA2
NavLoad
Load
NavPut i
Put
NavPick i
Pick
NavUnload
Unload
The shaded subtasks are defined as cooperative subtasks and this level as cooperation level in the last experiment of this section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

Fig. 6 Task graph for the AGV scheduling task

Table 2 Model parameters for the multi-agent 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

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 2 summarizes the values of the model parameters used in the experiments of this section. In these algorithms, learning rate α is set to 0.2, and exploration starts with 0.3, remains unchanged until the performance reaches to a certain level, and then is decreased by a factor of 1.01 every 60 sec. We use discount factors 0.9, 0.95, and 0.99 in these algorithms. Using discount factor 0.99 yielded better performance in all the algorithms. In this task, each experiment was conducted five times and the results were averaged.

Figure 7 shows the throughput of the system for the three algorithms, single-agent HRL, selfish multi-agent HRL, and *Cooperative HRL*. As seen in Fig. 7, agents learn a little faster initially in the selfish multi-agent method, but after some time the algorithm results in sub-optimal performance. This is due to the fact that two or more 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 higher than the single-agent HRL and the selfish multi-agent HRL algorithms. This difference is even more significant in Fig. 8, when the primitive actions have longer execution time, almost $\frac{1}{10\text{th}}$ of the average assembly time (the mean execution time of primitive actions is 2 sec).

Figure 9 shows the results from an implementation of the single-agent flat Q-Learning with the buffer capacity at each station set at 1. As can 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 10 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.

So far in our experiments in the AGV problem, we only defined *Root* as a *cooperative subtask*. Now in our last experiment in this problem, 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 the navigation subtasks at that level (see Fig. 6). We configure the *Root* and the third level navigation subtasks to represent joint actions. Figure 11 compares the performance of the system in these two cases. When the navigation subtasks are configured to represent joint actions, learning

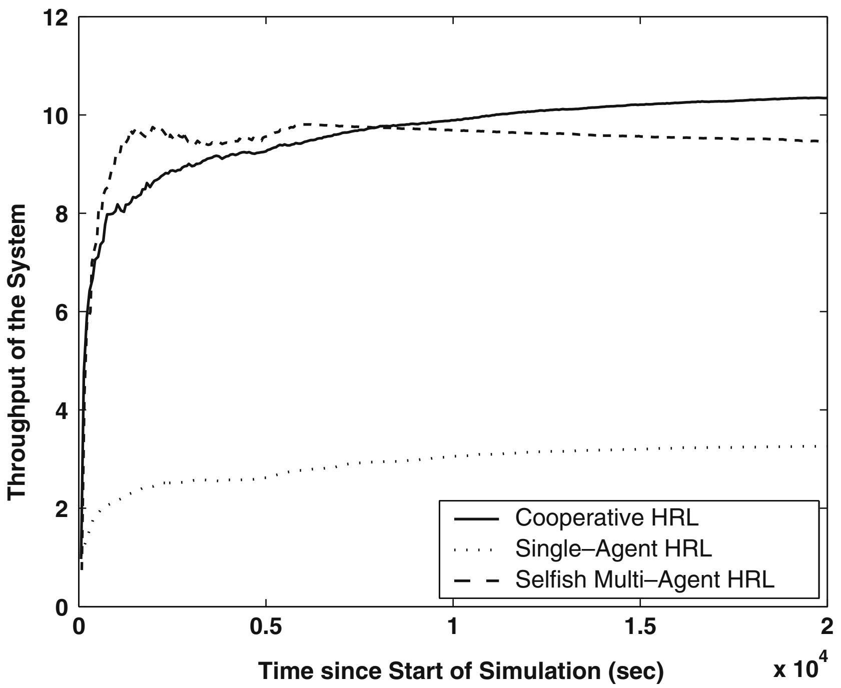


Fig. 7 This figure shows that the *Cooperative HRL* algorithm outperforms both the selfish multi-agent HRL and the single-agent HRL algorithms when the AGV travel time and load/unload time are very much less compared to the average assembly time

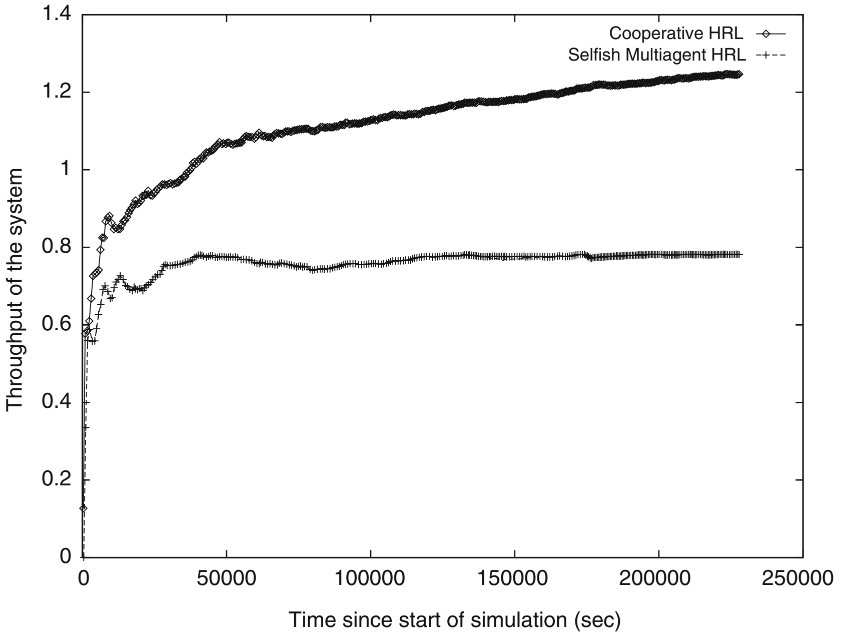


Fig. 8 This figure compares the *Cooperative HRL* algorithm with the selfish multi-agent HRL algorithm, when the AGV travel time and load/unload time are $\frac{1}{10\text{th}}$ of the average assembly time

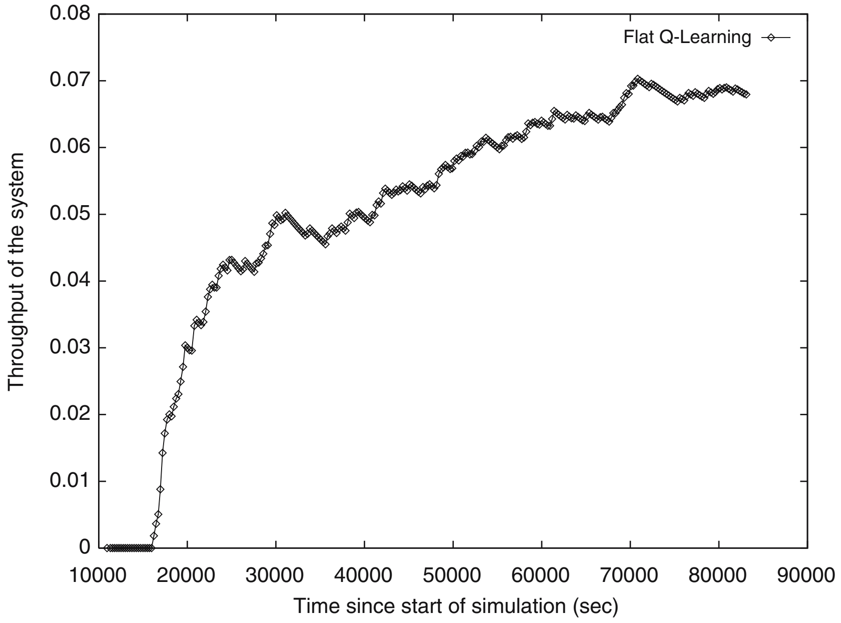


Fig. 9 A flat Q-Learner learns the AGV task extremely slowly showing the need for using a hierarchical task structure

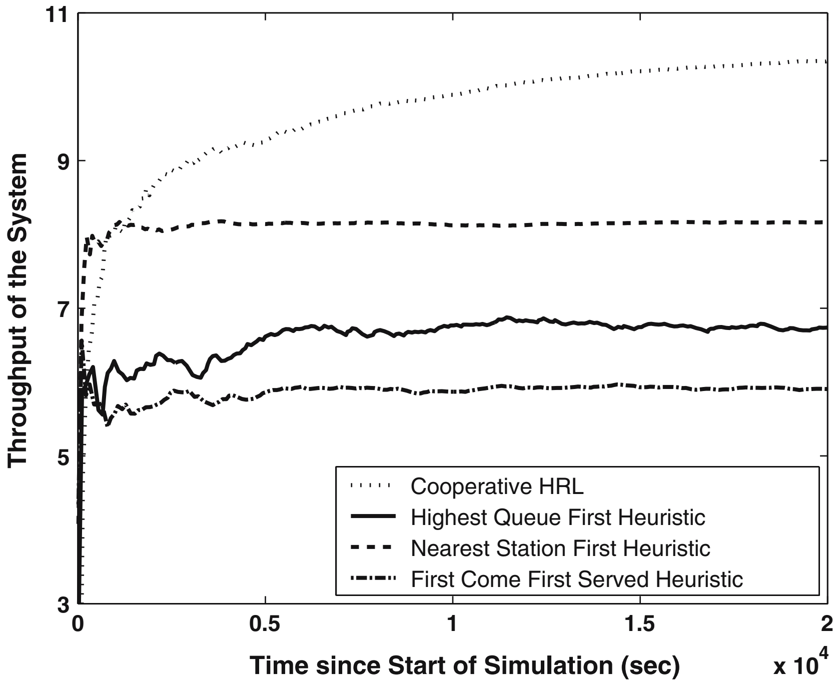


Fig. 10 This plot shows that the *Cooperative HRL* algorithm outperforms three well-known widely used industrial heuristics for AGV scheduling

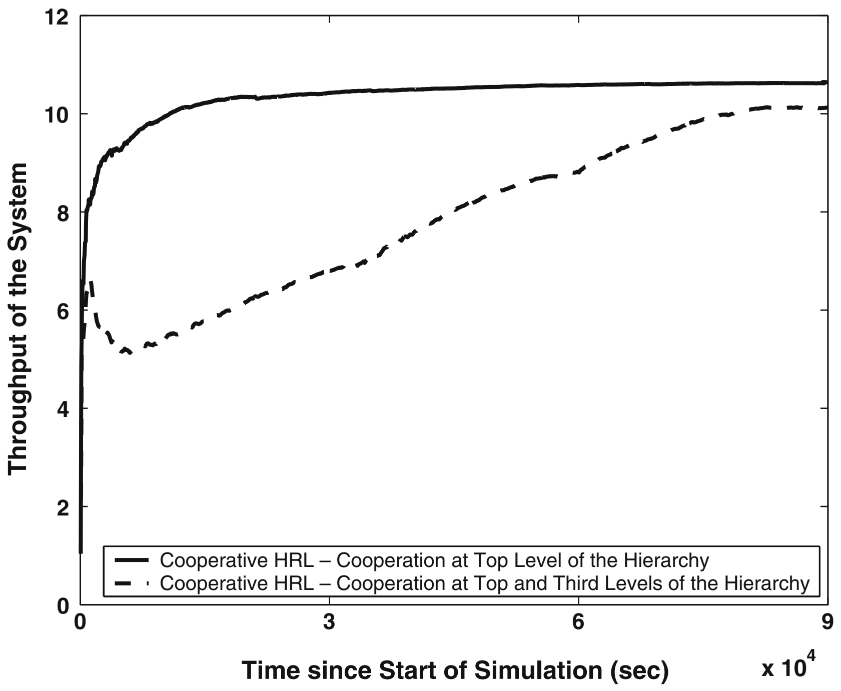


Fig. 11 This plot compares the performance of the *Cooperative HRL* algorithm with cooperation at the top level of the hierarchy vs. cooperation at the top and third levels of the hierarchy

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 Fig. 5, thus coordination at the navigation level does not improve the performance of the system. However, there exist problems that having joint actions at multiple levels will be worthwhile, even if convergence is slower, due to better overall performance.

6. Hierarchical multi-agent RL with communication decisions

Communication can be viewed as a decision taken by agents to obtain local information of their teammates, which may incur a certain cost. The *Cooperative HRL* algorithm described in Section 4 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 an 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 use communication optimally by taking into account its long term return and its immediate cost.

In the remainder of this paper, we examine the case where 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, one policy for each of the subtasks in the hierarchy including the communication subtasks) to maximize the team utility given the communication cost. Finally, in Section 7, 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 multi-agent taxi problem.

6.1. Communication among agents

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 4, 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 an 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.

6.2. A hierarchical multi-agent RL algorithm with communication decisions

When communication is costly, it is no longer optimal for the team that each agent broadcasts all its actions to its teammates. In this case, each agent must learn to use communication optimally. 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 Fig. 12 for the trash collection task. In this algorithm, when an agent is going to make a decision at an lth level *cooperative subtask*, it first decides whether to communicate (takes *Communicate* action) with the other agents to acquire their actions in U_l, or do not communicate (takes *Not-Communicate* action) and selects its action without inquiring new information about its teammates. Agents decide about communication by comparing the expected value of

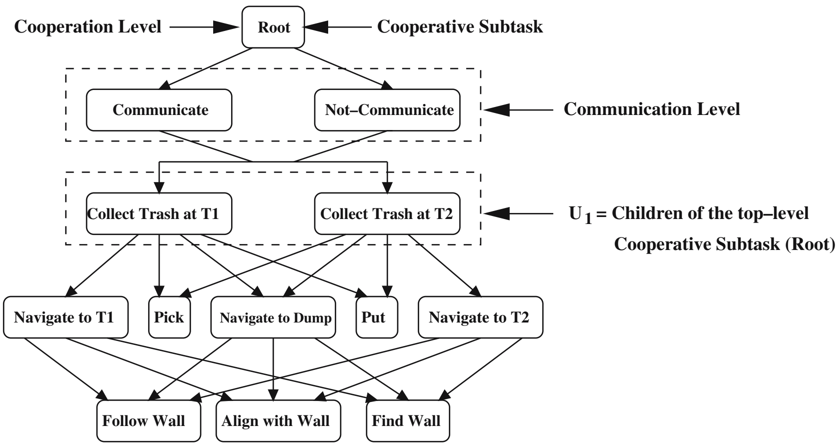


Fig. 12 Task graph of the trash collection problem with communication actions

communication $Q(Parent(Com), s, Com)$ with the expected value of not communicating with the other agents $Q(Parent(NotCom), s, NotCom)$. If agent j decides not to communicate, it chooses an action like a selfish agent by using its action-value (not joint action-value) function $Q^j(NotCom, s, a)$, where $a \in Children(NotCom)$. Upon the completion of the selected action, the expected value of not communicate $Q^j(Parent(NotCom), s, NotCom)$ is updated using the sum of all rewards received during the execution of this action. When agent j decides to communicate, it first takes the communication action $Ask(j, i), \forall i \in \{1, \ldots, n\}, i \neq j$, where n is the number of agents, and sends a *request* message to all the other agents. Other agents reply by taking the communication action *Tell*(i, j) and send their actions in U_l as an *inform* message to agent j. Then agent j uses its joint action-value (not action-value) function $Q^j(Com, s, a^1, \ldots, a^{j-1}, a^{j+1}, \ldots, a^n, a), a \in Children(Com)$ to select its next action in U_l. Upon the termination of the selected action, the expected value of communication $Q^j(Parent(Com), s, Com)$ is updated using the sum of all rewards received during the execution of this action plus the communication cost.

For example, 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 about its action in $U_1 = \{collect\ trash\ at\ T1,\ collect\ trash\ at\ T2\}$ or not. To make a communication decision, agent $A1$ compares $Q^1(Root, s, NotCom)$ with $Q^1(Root, s, Com)$. If it chooses not to communicate, it selects its action using $Q^1(NotCom, s, a)$, where $a \in U_1$. If it decides to communicate, after acquiring the action of agent $A2$ in U_1, a^{A2}, it selects its own action using $Q^1(Com, s, a^{A2}, a)$, where a and a^{A2} both belong to U_1.

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

In the *COM-Cooperative HRL* algorithm, *Communicate* subtasks are configured to store joint completion function values and *Not-Communicate* subtasks are configured to store completion function values. The joint completion function for agent j, $C^j(Com, s, a^1, \ldots, a^{j-1}, a^{j+1}, \ldots, a^n, a^j)$ is defined as the expected discounted cumulative reward of completing the *cooperative subtask* $Parent(Com)$ after executing subtask a^j in state s, while other agents performing subtasks $a^i, \forall i \in \{1, \ldots, n\}, i \neq j$. In the trash collection domain, if agent $A1$ communicates with agent $A2$, its value function decomposition would be

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

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 $A2$.

In the *COM-Cooperative HRL* algorithm, the V and C values are learned through a standard temporal-difference learning method based on sample trajectories similar to the one presented in the *Cooperative HRL* algorithm shown in Table 1. Completion function values for an action in U_l are updated when we take an action under a *Not-Communicate* subtask, and joint completion function values for an action in U_l are updated when it is selected under a *Communicate* subtask. 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. If an action in U_l is selected under a *Not-Communicate* subtask, upon its termination, $Q(Parent(NotCom), s, NotCom)$ is updated using the sum of all rewards received during the execution of the action. If an action in U_l is selected under a *Communicate* subtask, upon its termination, $Q(Parent(Com), s, Com)$ is updated using the sum of all rewards received during the execution of the action plus the communication cost.

7. Experimental results for the COM-cooperative HRL algorithm

In this section, we demonstrate the performance of the *COM-Cooperative HRL* algorithm proposed in Section 6.2 using a multi-agent 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 Fig. 13. There are four stations 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[5] at these four stations and wish to be transported to one of the other stations chosen randomly. Taxis must

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

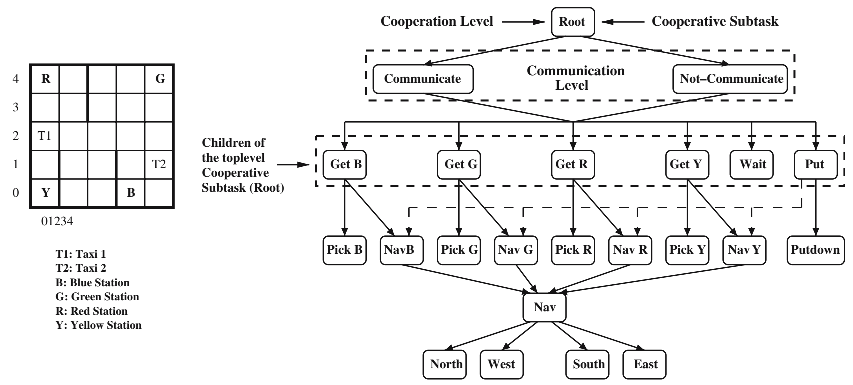


Fig. 13 A multi-agent taxi domain and its associated task graph

go to the location of a passenger, pick up the passenger, go to her/his destination station, 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 5,000 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 Fig. 13. 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 carry out the subtasks, i.e., for example 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 B, taxi $T2$ should serve a passenger at one of the other stations.

The state variables in this task are locations of the taxis (25 values each), status of the taxis (five values each, taxi is empty or transporting a passenger to one of the four stations), and status of the stations B, G, R, and Y (four values each, station is empty or has a passenger whose destination is one of the other three stations). Thus, in the multi-agent flat case, the size of the state space would grow to 4×10^6. The size of the Q table is this number multiplied by the number of primitive actions 10, which is 4×10^7. In the selfish multi-agent HRL algorithm, using state abstraction and the fact that each agent stores only its own state 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 *Cooperative HRL* algorithm, the number of the values to be learned would be $2 \times 729{,}815 = 1,459{,}630$. Finally in the *COM-Cooperative HRL* algorithm, this number would be $2 \times 934, 615 = 1, 869, 230$. In the *Cooperative HRL* and *COM-Cooperative HRL* algorithms, we define *Root* as a *cooperative subtask* and the highest level of the hierarchy as a *cooperation level* as shown in Fig. 13. Thus, *Root* is the only member of the *cooperation set* at that level, and $U_1 = A_{Root} = \{GetB, GetG, GetR, GetY, Wait, Put\}$. The joint action space for *Root* is specified as the cross product of the *Root* action set and U_1. Finally, the τ_{continue} termination scheme is used for joint action selection in this problem. In these algorithms, learning rate α is set to 0.1, and exploration starts with $\epsilon = 0.1$ and then is decreased by a factor of $\frac{1}{1+\epsilon}$ every 5,000 steps. We use discount factors 0.9, 0.95, and 0.99 in these algorithms. Using discount factor 0.99 yielded better performance in all the algorithms. All the experiments in this section were repeated five times and the results were averaged.

Figures 14 and 15 show the throughput of the system and the average waiting time per passenger for four algorithms, single-agent HRL, selfish multi-agent HRL, *Cooperative HRL*,

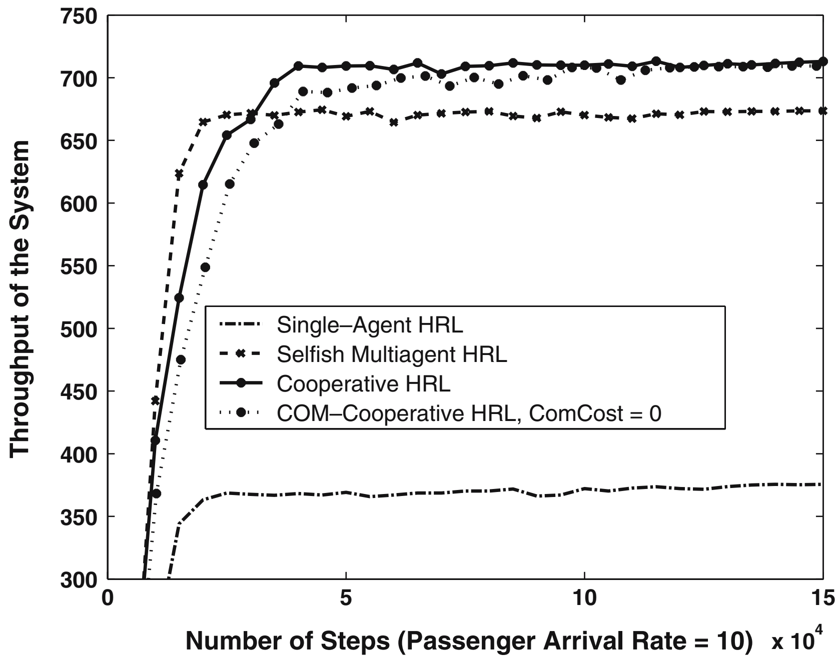


Fig. 14 This figure shows that *Cooperative HRL* and *COM-Cooperative HRL* with $ComCost = 0$ have better throughput than selfish multi-agent HRL and single-agent HRL

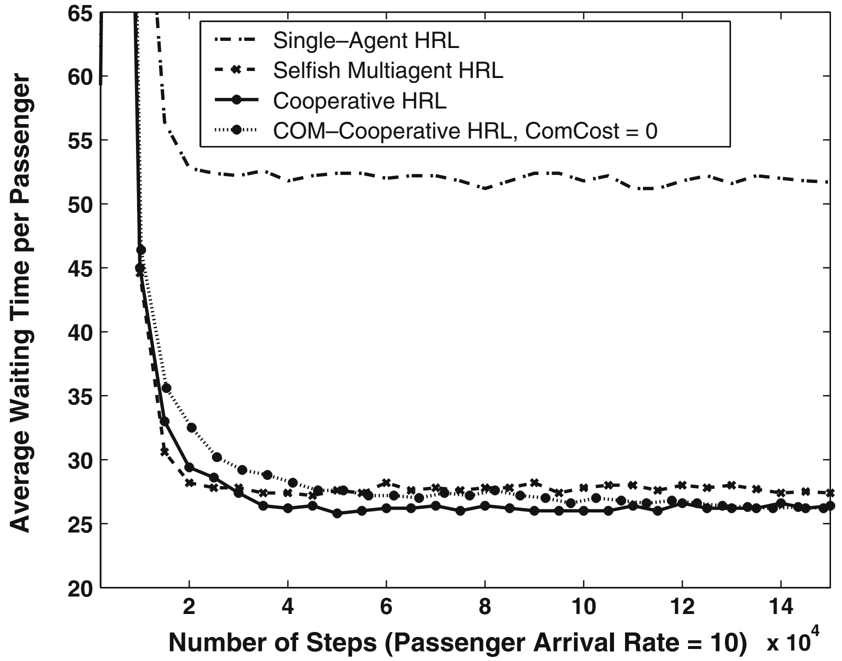


Fig. 15 This figure shows that the average waiting time per passenger in *Cooperative HRL* and *COM-Cooperative HRL* with $ComCost = 0$ is less than selfish multi-agent HRL and single-agent HRL

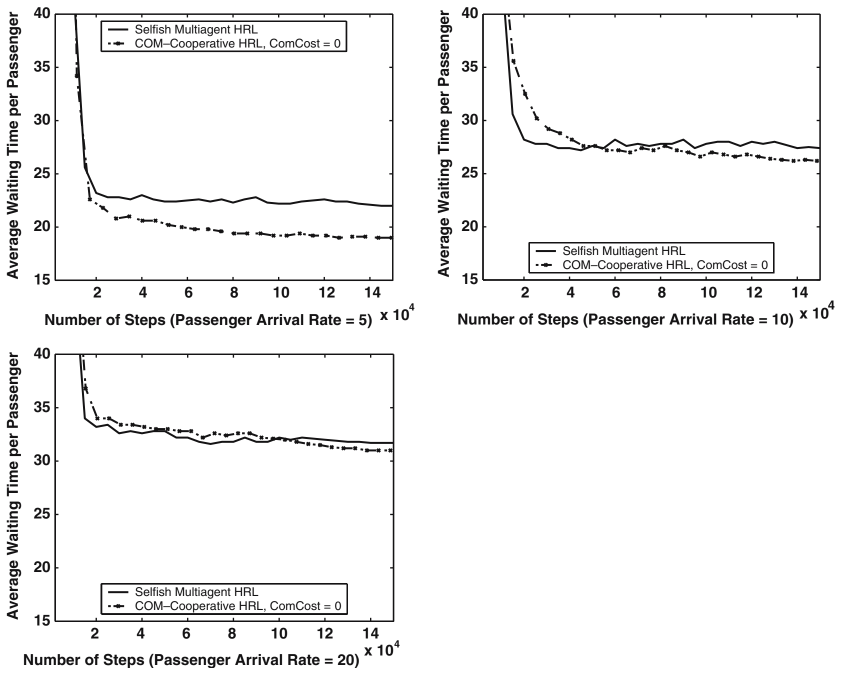


Fig. 16 This figure compares the average waiting time per passenger for selfish multi-agent HRL and *COM-Cooperative HRL* with $ComCost = 0$ for three different passenger arrival rates 5, 10, and 20. It shows that coordination among taxis becomes more crucial as the passenger arrival rate becomes smaller

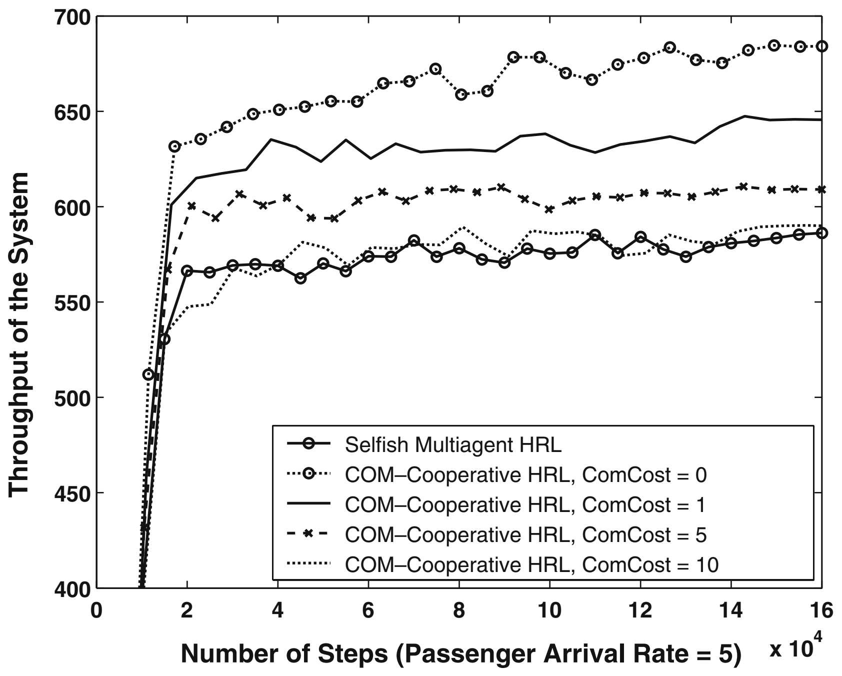
and *COM-Cooperative HRL* when communication cost is zero.[6] As seen in Figures 14 and 15, *Cooperative HRL* and *COM-Cooperative HRL* with $ComCost = 0$ have better throughput and average waiting time per passenger than selfish multi-agent HRL and single-agent HRL.[7] The *COM-Cooperative HRL* algorithm learns slower than *Cooperative HRL*, due to more parameters to be learned in this model. However, it eventually converges to the same performance as the *Cooperative HRL* algorithm does.

Figure 16 compares the average waiting time per passenger for multi-agent selfish HRL and *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, it is possible 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 picks 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 17 demonstrates the relation between the learned communication policy and the communication cost. These two figures show the throughput and the average waiting time

[6] The *COM-Cooperative HRL* algorithm uses the task graph in Fig. 13. The *Cooperative HRL* algorithm uses the same task graph without the *communication level*.

[7] The setting of the $ComCost$ value to zero is for comparison purposes. Different values of the $ComCost$ will be explored later in this section.



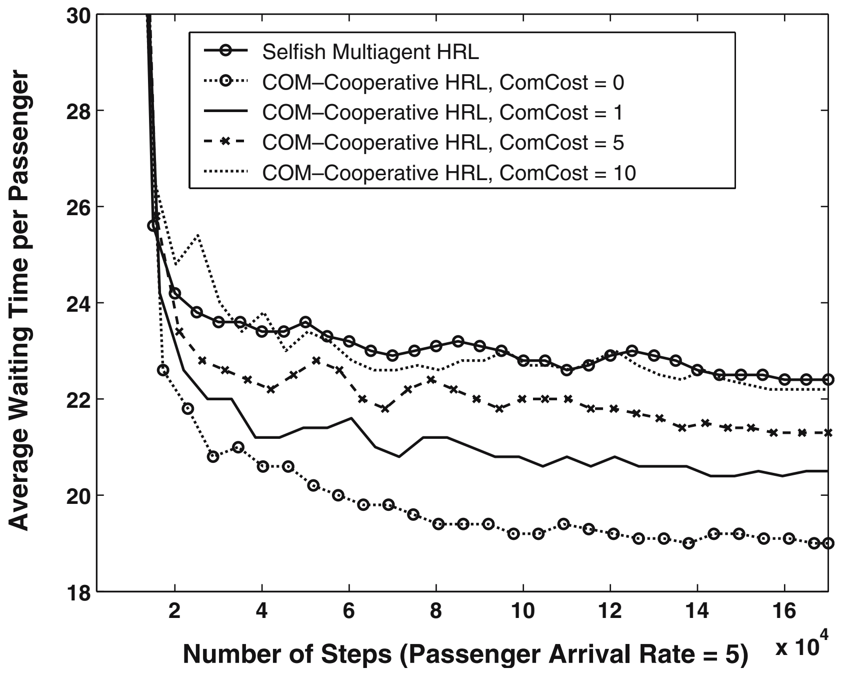


Fig. 17 This figure shows that as communication cost increases, the throughput (top) and the average waiting time per passenger (bottom) of the *COM-Cooperative HRL* algorithm become closer to those for the selfish multi-agent HRL algorithm. It indicates that agents learn to be selfish when communication is expensive

per passenger for selfish multi-agent HRL and *COM-Cooperative HRL* when communication cost equals 0, 1, 5, and 10. In both figures, as the communication cost increases, the performance of the *COM-Cooperative HRL* algorithm becomes closer to the performance of the selfish multi-agent HRL algorithm. It indicates that when communication is expensive, agents learn not to communicate and to be selfish.

8. Conclusions and future work

Multi-agent learning has been recognized to be challenging for two main reasons: the *curse of dimensionality* and *partial observability*. In a multi-agent system, each agent usually needs to know the states and actions of the other agents in order to make its own decision. As a result, the number of state-action values to be learned increases dramatically with the number of agents (the *curse of dimensionality*). This problem can be divided into two problems known as the joint action space problem and the joint state space problem. Moreover, the states and actions of the other agents are not often fully observable and inter-agent communication is usually costly (*partial observability*). In this paper, we studied the use of hierarchical reinforcement learning (HRL) to address these problems, and to accelerate learning to communicate and act in cooperative multi-agent systems. The key idea underlying our approach is that coordination skills are learned much more efficiently if agents have a hierarchical representation of the task structure. The use of hierarchy speeds up learning in cooperative multi-agent domains by making it possible to learn coordination skills at the level of subtasks instead of primitive actions. Algorithms for learning task-level coordination have already been developed in non-MDP approaches [37], however to the best of our knowledge, this work is the first attempt to use task-level coordination in an MDP setting. We proposed two new cooperative multi-agent HRL algorithms, *Cooperative HRL* and *COM-Cooperative HRL* using the above idea. 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, order in which to carry them out, and how to coordinate with other agents.

In the *Cooperative HRL* algorithm, we assume communication is free and therefore agents do not need to decide if communication with their teammates is necessary. We demonstrated the efficiency of this algorithm using two experimental testbeds: a simulated two-robot trash collection task, and a much larger four-agent automated guided vehicle (AGV) scheduling problem. We compared the performance of the *Cooperative HRL* algorithm with other algorithms such as selfish multi-agent HRL, single-agent HRL, and flat Q-learning in these problems. In the AGV scheduling task, we also showed that the *Cooperative HRL* algorithm 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 addressed 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 *Cooperative HRL* by including communication decisions in the model. We studied the empirical performance of the *COM-Cooperative HRL* algorithm as well as the relation between communication cost and the learned communication policy using a multi-agent taxi problem.

The hierarchical multi-agent models and algorithms presented in this paper address the joint action space problem by allowing the agents to learn joint action values only at *cooperative subtasks* usually located at the high level(s) of hierarchy. They also address the communication problem by allowing coordination at the level of subtasks instead of primitive actions. Since high-level subtasks can take a long time to complete, communication is needed only fairly infrequently. Additionally, the *COM-Cooperative HRL* algorithm proposed in this paper presents a method to optimize communication by including communication decisions in the hierarchical model. We did not directly address the joint state space problem in this paper. Although the proposed models do not prevent us from using joint state space for the subtasks in a hierarchy, we avoided dealing with its complexity by storing only local state information by each agent. However, using hierarchy can alleviate the joint state space problem in cooperative multi-agent systems. First, only *cooperative subtasks* can be defined as joint state space problems. It is an approximation, but it would be a good approximation if agents rarely need to cooperate at *non-cooperative subtasks*. Second, state abstraction in a hierarchy (which is an important and inseparable feature of any hierarchical abstraction) can help to reduce the size of the joint state space at a *cooperative subtask*. The joint state space at a *cooperative subtask* contains only those state variables that are relevant to the *cooperative subtask* and ignores the rest of the state variables. Third, since each agent can get a rough idea of the state of the other agents just by knowing about their high-level subtasks, it would sometimes be even possible to achieve a reasonably good performance by storing only local state information at *cooperative subtasks*, as shown in the experiments of this paper.

There is a number of directions for future work which can be briefly outlined. An immediate question that arises is to define the classes of cooperative multi-agent problems in which the proposed algorithms converge to a good approximation of the optimal policy. The experiments of this paper show that the effectiveness of these algorithms is most apparent in tasks where agents rarely interact at low levels (for example in the trash collection task, two robots may rarely need to exit through the same door at the same time). However, the algorithms can be 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 the 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 slower as shown in the AGV experiment depicted in Fig. 11. Moreover in this case, there are usually many subtasks in which it would be crucial for the agents to know the states of their teammates in order to make their own decisions. Therefore, we may have to define these subtasks as joint state space problems, which will result in even much larger set of state-action values that need to be learned, and consequently much slower learning.

A number of extensions would be useful, from studying the scenario where agents are heterogeneous, to recognizing the high-level subtasks being performed by the other agents using a history of observations (plan recognition and activity modeling) instead of direct communication. In the later case, we assume that each agent can observe its teammates and uses its observations to extract their high-level subtasks. Good examples for this approach are games such as soccer, football, or basketball, in which players often extract the strategy being performed by their teammates using recent observations instead of direct communication. Saria and Mahadevan presented a theoretical framework for online probabilistic plan recognition in cooperative multi-agent systems [33]. Their model extends the abstract hidden Markov model (AHMM) [7] to cooperative multi-agent domains. We believe that the model presented by Saria and Mahadevan can be combined with the learning algorithms proposed in this paper to reduce communication by learning to recognize the high-level subtasks being performed by the other agents. In the cooperative multi-agent models proposed in this paper, agents cooperate only at *cooperative subtasks*, which are predefined by the designer of the system. Another useful extension is where agents are given the ability to discover *cooperative subtasks*, or more general, the ability to decide (or learn to decide) autonomously, when to cooperate and with whom to cooperate.

Another direction for future work is to study different termination schemes for composing temporally extended actions. We used τ_{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 and/or action spaces, using a mixture of policy gradient-based RL and value function-based RL methods [11]. We believe that the algorithms proposed in this paper can be combined with the algorithms presented in [11] to be used in multi-agent domains with continuous state and/or action. 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 first author would like to thank Balaraman Ravindran for his useful comments. 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. Part of this article was written while the first author was supported by iCORE at the Department of Computing Science at University of Alberta.

References

1. Askin, R., & Standridge, C. (1993). *Modeling and analysis of manufacturing systems*. John Wiley and Sons.
2. Balch, T., & Arkin, R. (1998). Behavior-based formation control for multi-robot Teams. *IEEE Transactions on Robotics and Automation, 14*, 1–15.
3. Barto, A., & Mahadevan, S. (2003) Recent advances in hierarchical reinforcement learning, *Discrete Event Systems Special Issue on Reinforcement Learning, 13*, 41–77.
4. Bernstein, D., Zilberstein, S., & Immerman, N. (2000). The complexity of decentralized control of markov decision processes. In *Proceedings of the sixteenth international conference on uncertainty in artificial intelligence* (pp. 32–37).
5. Boutilier, C. (1999). Sequential optimality & coordination in multi-agent systems. In *Proceedings of the sixteenth international joint conference on artificial intelligence* (pp. 478–485).
6. Bowling, M., & Veloso, M. (2002). Multiagent learning using a variable learning rate. *Artificial Intelligence, 136*, 215–250.
7. Bui, H., Venkatesh, S., & West, G. (2002). Policy recognition in the abstract hidden markov model. *Journal of Artificial Intelligence Research, 17*, 451–499.
8. Crites, R., & Barto, A. (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., & Vrieze, K. (1997). *Competitive Markov decision processes.* Springer Verlag.
11. Ghavamzadeh, M., & Mahadevan, S. (2003). Hierarchical policy gradient algorithms. In *Proceedings of the twentieth international conference on machine learning* (pp. 226–233).
12. Ghavamzadeh, M., & Mahadevan, S. (2004). Learning to communicate and act using hierarchical reinforcement learning. In *Proceedings of the third international joint conference on autonomous agents and multiagent systems* (pp. 1114–1121).
13. Guestrin, C., Lagoudakis, M., & Parr, R. (2002). Coordinated reinforcement learning. In *Proceedings of the nineteenth international conference on machine learning* (pp. 227–234).

14. Howard, R. (1971). *Dynamic probabilistic systems: Semi-Markov and decision processes*. John Wiley and Sons.
15. Hu, J., & Wellman, M. (1998). Multiagent reinforcement learning: Theoretical framework and an algorithm. In *Proceedings of the fifteenth international conference on machine learning* (pp. 242–250).
16. Kearns, M., Littman, M., & Singh, S. (2001). Graphical models for game theory. In *Proceedings of the seventeenth international conference on uncertainty in artificial intelligence* (pp. 253–260).
17. Klein, C., & Kim, J. (1996). AGV dispatching. *International Journal of Production Research, 34*, 95–110.
18. Koller, D., & Milch, B. Multiagent influence diagrams for representing and solving games. In *Proceedings of the seventeenth international joint conference on artificial intelligence* (pp. 1027–1034).
19. La Mura, P. (2000). Game Networks. In *Proceedings of the sixteenth international conference on uncertainty in artificial intelligence*.
20. Lee, J. (1996). Composite dispatching rules for multiple-vehicle agv systems. *Simulation, 66*, 121–130.
21. 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).
22. Littman, M. (2001). Friend-or-Foe Q-learning in general-sum games. In *Proceedings of the eighteenth international conference on machine learning* (pp. 322–328).
23. Littman, M., Kearns, M., & Singh, S. (2001). An efficient exact algorithm for singly connected graphical games. In *Proceedings of neural information processing systems* (pp. 817–824).
24. Makar, R., Mahadevan, S., & Ghavamzadeh, M. (2001). Hierarchical multi-agent reinforcement learning. In *Proceedings of the fifth international conference on autonomous agents* (pp. 246–253).
25. Mataric, M. (1997). Reinforcement learning in the multi-robot domain (1997). *Autonomous Robots, 4*, 73–83.
26. Ortiz, L., & Kearns, M. (2002). Nash propagation for loopy graphical games. In *Proceedings of neural information processing systems*.
27. Owen, G. (1995). *Game theory*. Academic Press.
28. Parr, R. (1998). *Hierarchical control and learning for Markov decision processes*, PhD thesis, University of California, Berkeley.
29. Peshkin, L., Kim, K., Meuleau, N., & Kaelbling, L. (2000). Learning to cooperate via policy search. In *Proceedings of the sixteenth international conference on uncertainty in artificial intelligence* (pp. 489–496).
30. Puterman, M. (1994). *Markov decision processes*. Wiley Interscience.
31. Pynadath, D., & Tambe, M. (2002). The communicative multiagent team decision problem: Analyzing teamwork theories and models. *Journal of Artificial Intelligence Research, 16*, 389–426.
32. Rohanimanesh, K., & Mahadevan, S. Learning to take concurrent actions. In *Proceedings of the sixteenth annual conference on neural information processing systems*.
33. Saria, S., & Mahadevan, M. (2004). Probabilistic plan recognition in multiagent systems. In *Proceedings of the fourteenth international conference on automated planning and scheduling* (pp. 12–22).
34. Schneider, J., Wong, W., Moore, A., & Riedmiller, M. Distributed value functions. In *Proceedings of the sixteenth international conference on machine learning* (pp. 371–378).
35. Singh, S., Kearns, M., & Mansour, Y. (2000). Nash convergence of gradient dynamics in general-sum games. In *Proceedings of the sixteenth international conference on uncertainty in artificial intelligence* (pp. 541–548).
36. Stone, P., & Veloso, M. (1999). Team-partitioned, opaque-transition reinforcement learning. In *Proceedings of the third international conference on autonomous agents* (pp. 206–212).
37. Sugawara, T., & Lesser, V. Learning to improve coordinated actions in cooperative distributed problem-solving environments. *Machine Learning, 33*, 129–154.
38. Sutton, R., Precup, D., & Singh, S. (1999). Between MDPs and semi-MDPs: A framework for temporal abstraction in reinforcement learning. *Artificial Intelligence, 112*, 181–211.
39. Tadepalli, P., & Ok, D. 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).
40. Tan, M. (1993). Multi-agent reinforcement learning: Independent vs. cooperative agents. In *Proceedings of the tenth international conference on machine learning* (pp. 330–337).
41. Vickrey, D., & Koller, D. (2002). Multiagent algorithms for solving graphical games. In *Proceedings of the national conference on artificial intelligence* (pp. 345–351).
42. Watkins, C. (1989). *Learning from delayed rewards*, PhD thesis, Kings College, Cambridge, England.
43. Weiss, G. (1999). *Multi-agent systems: A modern approach to distributed artificial intelligence.* MIT Press.
44. Xuan, P., Lesser, V., & Zilberstein, S. (2001). Communication decisions in multi-agent cooperation: Model and experiments. In *Proceedings of the fifth international conference on autonomous agents* (pp. 616–623).
45. Xuan, P., & Lesser, V. (2002). Multiagent policies: From centralized ones to decentralized ones. In *Proceedings of the first international joint conference on autonomous agents and multiagent systems* (pp. 1098–1105).