Extending Hierarchical Reinforcement Learning to Continuous-Time, Average-Reward, and Multi-Agent Models

Mohammad Ghavamzadeh
Sridhar Mahadevan
Rajbala Makar

CMPSCI Technical Report 03-23

July 9, 2003

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

Abstract

Hierarchical reinforcement learning (HRL) is a general framework that studies how to exploit the structure of actions and tasks to accelerate policy learning in large domains. Prior work on HRL has been limited to the discrete-time discounted reward semi-Markov decision process (SMDP) model. In this paper we generalize the setting of HRL to *average-reward*, *continuous-time* and *multi-agent* SMDP models. We also describe experimental results from a large-scale real-world domain, attesting to the benefits of HRL generally, and to our extensions more specifically.

Although in principle any HRL framework could suffice, we focus in this paper on the MAXQ framework. We describe three new hierarchical reinforcement learning algorithms: *continuous-time discounted reward MAXQ*, *discrete-time average reward MAXQ*, and *continuous-time average reward MAXQ*. We also investigate the use of hierarchical reinforcement learning to speed up the acquisition of cooperative multiagent tasks. We extend the MAXQ framework to the multiagent case which we term *cooperative MAXQ*, where each agent uses the same task hierarchy. Learning is decentralized, with each agent learning three interrelated skills: *how to perform subtasks*, *which order to do them in*, and *how to coordinate with other agents*. Coordination skills among agents are learned by using joint actions at the highest level(s) of the hierarchy.

We use two experimental testbeds to study the empirical performance of our proposed extensions. One domain is a simulated robot trash collection task. The other domain is a much larger real-world multi-agent autonomous guided vehicle (MAGV) problem. We compare the performance of our proposed algorithms with each other, as well as with the original MAXQ method and to standard Q-learning. In the MAGV domain, we show that our proposed extensions outperform widely used industrial heuristics, such as "*first come first serve*", "*highest queue first*" and "*nearest station first*".

Keywords: Hierarchical Reinforcement Learning, Multiagent Systems, Semi-Markov Decision Processes, Average-Reward.

1. Introduction

Reinforcement learning (RL) (Sutton and Barto, 1998) is a class of problems whereby agents must learn what actions to take in different situations or *states* in order to maximize a scalar feedback function (or *reward*) over time. The mapping from states to actions is referred to as a *policy* or a closed-loop plan. Learning occurs based on the idea that the tendency to produce an action should be reinforced if it produces favorable long-term rewards, and weakened otherwise. From the perspective of control theory, RL algorithms can be shown to be approximations to classical approaches to solving optimal control problems. The classical approaches use dynamic programming (DP) (Bertsekas, 1995), which requires perfect knowledge of the system dynamics and payoff function. Reinforcement learning has the advantage of potentially being able to find optimal solutions (or close-to-optimal) solutions in domains where models are not known or unavailable.

Instead of learning the policy directly, most approaches in RL learn an indirect target function, referred to as the *value function* as it represents the long-term payoff associated with states or state-action pairs. The policy can be recovered from a value function by choosing "greedy" actions that maximize the value of states nearby (or immediate state action pairs). Although it is possible to show theoretically that RL algorithms, such as Q-learning (Watkins, 1989) or TD(λ) (Sutton, 1988), converge asymptotically to produce the *optimal* value function (one that dominates all other value functions over the state space), convergence is only assured in restricted cases. Often, real-world problems require using nonlinear *function approximators* for which convergence is not guaranteed. Moreover, even if asymptotic convergence was theoretically guaranteed by using a restricted class of parametric approximators, in practice one is often interested in convergence within a reasonable time bound. Real-time convergence is usually extremely slow with algorithms such as Q-learning, when they are combined with nonlinear function approximators, such as neural nets. For example, in the well-known multiagent elevator task (Crites and Barto, 1998), convergence takes on the order of millions of steps of simulated time.

The central focus of this paper is to present new algorithms for reinforcement learning, applicable to continuous-time multiagent tasks such as the elevator problem, using which convergence occurs much more rapidly than with traditional Q-learning. The new algorithms are based on extending *hierarchical reinforcement learning* (HRL), a general framework for scaling reinforcement learning to problems with large state spaces by using the task (or action) structure to restrict the space of policies. The key principle underlying HRL is to develop learning algorithms that do not need to learn policies from scratch, but instead reuse existing policies for simpler sub-tasks (or macro actions). The difficulty with using the traditional framework for reusing learned policies is that decision making no longer occurs in synchronous unit-time steps, as is traditionally assumed in RL. Instead, decision-making occurs in epochs of variable length, such as when a distinguishing state is reached (e.g., an intersection in a robot navigation task), or a subtask is completed (e.g., the elevator arrives on the first floor). Furthermore, these variable length intervals cause the system dynamics to be non-Markov with respect to immediate state transitions, since the state resulting from a macro action might depend on all states that occurred since that macro was initiated. For example, a macro action undertaken by a robot for cleaning a room by sweeping the floor twice and then exiting will require remembering when the robot started (you cannot predict its next state merely by observing that the robot was near the door).

Fortunately, a well-known statistical model is available to treat variable length state dynamics: the semi-Markov decision process (SMDP) model. Here, state transition dynamics is specified not only by the state where an action was taken, but also parameters specifying the length of time since the action was taken. Work in HRL to date, including hierarchical abstract machines (HAMs) (Parr, 1998), options (Sutton et al., 1999) and MAXQ (Dietterich, 2000), have used a discrete-time discounted SMDP framework. However a wide class of tasks have continuous-time nature. Therefore, these frameworks, which are limited to the discrete-time SMDP model, should be generalized to continuous-time SMDPs and also to the average-reward framework. A primary goal of most of these tasks, such as AGV scheduling, queuing and inventory control, is to find a gain optimal policy that maximizes (minimizes) the long-run average reward (cost). Although average reward RL has been extensively studied, using both the discrete-time MDP model (Schwartz, 1993, Mahadevan, 1996, Tadepalli and Ok, 1996a) as well as the continuous-time SMDP model (Mahadevan et al., 1997, Wang and Mahadevan, 1999), prior work has been limited to "flat" algorithms. Therefore, it is necessary to extend hierarchical reinforcement learning methods such as MAXQ, which have been limited to discounted reward SMDP model, to the average reward SMDP framework.

One of the principal contributions of this paper is to extend HRL to more interesting (and practical) settings, including continuous-time SMDP models, average-reward optimality models, and also to multi-agent domains. These extensions make HRL more widely applicable to many interesting practical problems, including the elevator domain, Robosoccer (Stone and Veloso, 1999), transfer line production control (Wang and Mahadevan, 1999), multi-agent autonomous guided vehicle (MAGV) scheduling, and so on.

We focus our extension of HRL to the MAXQ framework, although in principle, our approach can be applied also to HAMS and options. We describe three new hierarchical reinforcement learning algorithms: *continuous-time discounted reward MAXQ*, *discrete-time average reward MAXQ*, and *continuous-time average reward MAXQ*. We also extend the MAXQ framework to the multiagent case which we term *cooperative MAXQ*, where each agent uses the same task hierarchy to cooperatively learn complex tasks. Learning is decentralized, with each agent learning three interrelated skills: *how to perform subtasks*, *which order to do them in*, and *how to coordinate with other agents*. Coordination skills among agents are learned by using joint actions at the highest level(s) of the hierarchy.

We use two experimental testbeds to study the empirical performance of our proposed extensions. One domain is a simulated robot trash collection domain. The other domain is a much larger real-world multi-agent autonomous guided vehicle (MAGV) domain. We compare the performance of our proposed algorithms with each other, as well as with the original MAXQ method and to standard Q-learning. In the MAGV domain, we show that our proposed extensions outperform widely used industrial heuristics, such as "*first come first serve*", "*highest queue first*" and "*nearest station first*".

The rest of this paper is organized as follows. Section 2 provides an overview of multi-agent reinforcement learning. Section 3 introduces the continuous-time SMDP framework under both discounted and average reward paradigms. Section 4 describes the original discrete-time discounted MAXQ framework, and illustrates it using a multiagent robot trash collection task. Section 5, Section 6, and Section 7 describe the continuous-time discounted reward MAXQ, discrete-time average reward MAXQ, and continuous-time average reward MAXQ algorithms, respectively. Section 8 describes the multiagent cooperative MAXQ algorithm. Section 9 describes the real-world multiagent testbed of *AGV scheduling*. Section 10 presents experimental results of using proposed algorithms in the trash collection task and the multiagent AGV scheduling problem. Finally, Section 11 summarizes the paper and discusses some directions for future work.

2. Hierarchical Multiagent Reinforcement Learning

Consider sending a team of robots to carry out reconnaissance of an indoor environment to check for intruders. This problem is naturally viewed as a multiagent task (Weiss, 1999). The most effective strategy will require coordination among the individual robots. A natural decomposition of this task would be to assign different parts of the environments, for example rooms, to different robots. In this paper, we are interested in learning algorithms for such *cooperative* multiagent tasks, where the agents learn the coordination skills by trial and error. The key idea underlying our approach is simply that coordination skills are learned much more efficiently if the robots have a hierarchical representation of the task structure (algorithms for learning task-level coordination have been developed in non-MDP approaches, see (Sugawara and Lesser, 1998). In particular, rather than each robot learning its response to low-level primitive actions of the other robots (for instance if robot-1 goes forward, what should robot-2 do), it learns high-level coordination knowledge (what is the utility of robot-2 searching room-2 if robot-1 is searching room-1, and so on).

Multiagent reinforcement learning has been recognized to be very challenging, since the number of parameters to be learned increases dramatically with the number of agents. In addition, when agents carry out actions in parallel, the environment is usually non-stationary and often non-Markovian as well (Mataric, 1997). We do not address the non-stationary aspect of multiagent learning in this paper. One approach that has been successful in the past is to have agents learn policies that are parameterized by the modes of interaction (Wang and Mahadevan, 1999). Prior work in multiagent reinforcement learning can be decomposed into work on competitive models vs. cooperative models. Littman (Littman, 1994), and Hu and Wellman (Hu and Wellman, 1998), among others, have studied the framework of Markov games for competitive multiagent learning.

Here, we are primarily interested in the cooperative case. The work on cooperative learning can be further separated based on the extent to which agents need to communicate with each other. For example, in completely accessible models such as multiagent Markov decision processes (MMDP) (Boutilier, 1999), agents have complete access to both the global state and the global joint action. Tan (Tan, 1993) and Xuan (Xuan et al., 2001) also exemplify the approach of requiring communication of global states and/or actions at every step. Tan extends flat Q-learning to multiagent learning by using joint state-action values. This approach requires communication of states and actions at every step. Xuan assumes agents have to communicate to obtain other agent's local state information. In the elevator domain (Crites and Barto, 1998), agents share a common state description and a global reinforcement signal, but do not model joint actions.

There are also studies of multiagent learning which do not model joint states or actions explicitly, such as by Balch (Balch and Arkin, 1998) and Mataric (Mataric, 1997), among others. In such systems, each robot maintains its position in the formation depending on the locations of the other robots, so there is some implicit communication or sensing of states and actions of other agents. There has also been work on reducing the parameters needed for Q-learning in multiagent domains, by learning action values over a set of derived features(Stone and Veloso, 1999). These derived features are domain-specific, and have to be encoded by hand, or constructed by a supervised learning algorithm.

Our approach differs from all the above in one key respect, namely the use of HRL to speed up cooperative multiagent reinforcement learning. We assume each agent is given an initial hierarchical decomposition of the overall task (as described below, we adopt the MAXQ framework). However, the learning is distributed since each agent has only a local view of the overall state space. Furthermore, each agent learns joint abstract action-values by communicating with each other only the high-level subtasks that they are doing. Since high-level tasks can take a long time to complete, communication is needed only fairly infrequently (this is another significant advantage over flat methods).

A further advantage of the use of hierarchy in multiagent learning is that it makes it possible to learn coordination skills at the level of abstract actions. The agents learn joint action values only at the highest level(s) of abstraction in the proposed framework. This allows for increased cooperation skills as agents do not get confused by low level details. In addition, each agent has only local state information, and is ignorant about the other agent's location. This is based on the idea that in many cases, an agent can get a rough idea of what state the other agent might be in just by knowing about the high level action being performed by the other agent. Also, keeping track of just this information greatly simplifies the underlying reinforcement learning problem.

These benefits can potentially accrue with using any type of hierarchical learning algorithm, though in this paper we only describe results using the MAXQ framework. The reason that we decided to adopt the MAXQ framework as a basis for our multiagent algorithm is the fact that the MAXQ method stores the value function in a distributed way in all nodes in the subtask graph. The value function is propagated upwards from the lower level nodes whenever a high level node needs to be evaluated. This propagation enables the agent to simultaneously learn subtasks and high level tasks. Thus, by using this method, agents learn the coordination skills and the individual low level tasks and subtasks all at once.

However, it is necessary to generalize the MAXQ framework to make it more applicable to multiagent learning. A broad class of multiagent optimization tasks, such as AGV scheduling, can be viewed as discrete-event dynamic systems. For such tasks, the termination predicate used in MAXQ has to be redefined to take care of the fact that the completion of certain subtasks might depend on the occurrence of an event rather than just a state of the environment.

3. Semi-Markov Decision Processes

We begin with a review of continuous-time semi-Markov decision processes. Semi-Markov decision processes (SMDPs) are useful in modeling temporally extended actions. They extend the discrete-time MDP model in several aspects. Time is modeled as a continuous entity and decisions are only made at discrete points in time (or events). The state of the system may change continually between decisions, unlike MDPs where state changes are only due to actions.

An SMDP is defined as a five tuple (S,A,P,R,F), where S is a finite set of states, A is the set of actions, P is a set of state and action dependent transition probabilities, R is the reward function, and F is a function giving probability of transition times for each state-action pair. $P(s'|s,a)$ denotes the probability that action a will cause the system to transition from state s to state s'. This transition is 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. $F(t|s,a)$ is the probability that the next decision epoch occurs within t time units after the agent chooses action a in state s at a decision epoch. From F and P, we can compute Φ by

$$\Phi(t,s'|s,a) = P(s'|s,a)F(t|s,a)$$

where Φ denotes the probability that the system will be in state s' for the next decision epoch, at or before t time units after choosing action a in state s, at the last decision epoch. In discrete-time semi-Markov decision processes, $P(s',N|s,a)$ denotes the probability that the system will be in state s' for the next decision epoch, at or before N time steps after choosing action a in state s, at the last decision epoch. The reward function for continuous-time SMDPs is more complex than in the MDP model. In addition to the fixed reward of taking action a in state s, $k(s,a)$, an additional reward may be accumulated at rate $c(s',s,a)$ for the time the natural process remains in state s' between decision epochs. Formally, the expected reward between two decision epochs, given that the system is in state s and chooses action a in the first decision epoch, is expressed as

$$r(s,a) = k(s,a) + E_s^a\{\int_0^\tau c(W_t,s,a)dt\}$$

where τ is the transition time to the second decision epoch and W_t denotes the state of the natural process during this transition.

3.1 Discounted Models

We now give a short overview of infinite-horizon discounted semi-Markov decision processes (Puterman, 1994, Bradtke and Duff, 1995). We assume continuous-time discounting at rate $\beta > 0$, which means that the present value of one reward unit received t time units in the future equals $e^{-\beta t}$. In this model, for policy π, $v^\pi(s)$ denotes the expected infinite-horizon discounted reward, given that the process occupies state s at the first decision epoch and is defined by

$$v^\pi(s) = E_s^\pi\{\sum_{n=0}^{\infty} e^{-\beta\sigma_n}[k(s_n,a_n) + \int_{\sigma_n}^{\sigma_{n+1}} e^{-\beta(t-\sigma_n)}c(W_t,s_n,a_n)dt]\} \tag{1}$$

In the above expression, $\sigma_0, \sigma_1, \ldots$ represent the times of successive decision epochs and $e^{-\beta\sigma_n}$ transforms the reward to values at the first decision epoch. In this model, the expected discounted reward between two decision epochs is defined as

$$r(s,a) = k(s,a) + \int_0^\infty \sum_{s' \in S} [\int_0^u e^{-\beta t} c(s',s,a) P(s'|s,a) dt] F(du|s,a) \tag{2}$$

Using Equation (2), we can re-express the value function in Equation (1) as

$$v^\pi(s) = r(s,\pi(s)) + \sum_{s' \in S} \int_0^\infty e^{-\beta t} v^\pi(s') \Phi(dt, s'|s,\pi(s))$$

The action value function $Q^\pi(s,a)$ represents the discounted cumulative reward of doing an action a in state s once, and then following policy π subsequently.

$$Q^\pi(s,a) = r(s,a) + \sum_{s' \in S} P(s'|s,a) \int_0^\infty e^{-\beta t} Q^\pi(s', \pi(s')) F(dt|s,a)$$

3.2 Average Reward Models

The theory of infinite-horizon semi-Markov decision processes with the average reward criterion is more complex than that for discounted models (Puterman, 1994, Mahadevan, 1996). To simplify exposition we assume that for every stationary policy, the embedded Markov chain has a unichain transition probability matrix. Under this assumption, the expected average reward of every stationary policy does not vary with the initial state. In this section we illustrate both discrete-time and continuous-time average reward semi-Markov decision processes.

3.2.1 Discrete-Time Average Reward Models

For policy π, state $s \in S$ and number of time steps $N \geq 0$, $v_N^\pi(s)$ denotes the expected total reward generated by the policy π up to time step N, given that the system occupies state s at time 0 and is defined as

$$v_N^\pi(s) = E_s^\pi\{\sum_{u=0}^{N-1} r(s_u, a_u)\}$$

The average expected reward or gain $g^\pi(s)$ for a policy π at state s can be defined by taking the limit inferior of the ratio of the expected total reward up until the Nth decision epoch to the number of decision epochs. So, the gain of a policy $g^\pi(s)$ can be expressed as the ratio

$$g^\pi(s) = \lim_{N \to \infty} \frac{E_s^\pi\{\sum_{u=0}^{N-1} r(s_u, a_u)\}}{N}$$

For unichain MDPs, the gain of any policy is state independent and we can write $g^\pi(s) = g^\pi$. For each transition, the expected number of transition steps is defined as:

$$y(s,a) = E_s^a\{N\} = \sum_{N=0}^{\infty} N \sum_{s' \in S} P(s', N|s,a)$$

In discrete-time unichain average reward SMDPs, the expected average adjusted sum of rewards h^π for stationary policy π is defined as

$$h^\pi(s) = E_s^\pi\{\sum_{u=0}^{\infty}[r(s_u, a_u) - g^\pi(s_u)]\} = E_s^\pi\{\sum_{u=0}^{\infty}[r(s_u, a_u) - g^\pi]\} \tag{3}$$

The Bellman equation for discrete-time unichain average reward SMDPs is defined based on the h function in Equation (3) and can be written as

$$h^\pi(s) = r(s, \pi(s)) - g^\pi y(s, \pi(s)) + \sum_{s' \in S, N} P(s', N|s, \pi(s))h^\pi(s')$$

The action value function $R^\pi(s, a)$ represents the average adjusted value of doing an action a in state s once, and then following policy π subsequently.

$$R^\pi(s, a) = r(s, a) - g^\pi y(s, a) + \sum_{s' \in S, N} P(s', N|s, a)R^\pi(s', \pi(s'))$$

3.2.2 Continuous-Time Average Reward Models

For policy π, state $s \in S$ and time $t \geq 0$, $v_t^\pi(s)$ denotes the expected total reward generated by the process up to time t, given that the system occupies state s at time 0 and is defined as

$$v_t^\pi(s) = E_s^\pi\{\sum_{n=0}^{v_t - 1} k(s_n, a_n) + \int_0^t c(W_u, s_{v_u}, a_{v_u})du\}$$

where v_u is the number of decisions made up to time t. In this model, the expected total reward between two decision epochs is defined as

$$r(s, a) = k(s, a) + \int_0^\infty \sum_{s' \in S}[\int_0^u c(s', s, a)P(s'|s, a)dt]F(du|s, a)$$

The average expected reward or gain $g^\pi(s)$ for a policy π at state s can be defined by taking the limit inferior of the ratio of the expected total reward up until the nth decision epoch to the expected total time until the nth decision epoch. So, the gain of a policy $g^\pi(s)$ can be expressed as the ratio

$$g^\pi(s) = \lim_{n \to \infty} \frac{E_s^\pi\{\sum_{i=0}^n [k(s_i, a_i) + \int_{\sigma_i}^{\sigma_{i+1}} c(W_t, s_i, a_i)dt]\}}{E_s^\pi\{\sum_{i=0}^n \tau_i\}}$$

For unichain MDPs, the gain of any policy is state independent and we can write $g^\pi(s) = g^\pi$. For each transition, the expected transition time is defined as:

$$y(s, a) = E_s^a\{\tau\} = \int_0^\infty t \sum_{s' \in S} \Phi(dt, s'|s, a)$$

In continuous-time unichain average reward SMDPs, the expected average adjusted sum of rewards h^π for stationary policy π is defined as

$$h^\pi(s) = V_t^\pi(s) - g^\pi t \tag{4}$$

where t is the time at which the decision epoch occurs. The Bellman equation for unichain average reward SMDPs is defined based on the h function in Equation (4) and can be written as

$$h^\pi(s) = r(s, \pi(s)) - g^\pi y(s, \pi(s)) + \sum_{s' \in S} P(s'|s, \pi(s)) \int_0^\infty h^\pi(s') F(dt|s, \pi(s))$$

The action value function $R^\pi(s, a)$ represents the average adjusted value of doing an action a in state s once, and then following policy π subsequently.

$$R^\pi(s, a) = r(s, a) - g^\pi y(s, a) + \sum_{s' \in S} P(s'|s, a) \int_0^\infty R^\pi(s', \pi(s')) F(dt|s, a)$$

4. The MAXQ Framework

The reinforcement learning algorithms introduced in this paper extend the MAXQ value function decomposition developed originally in the context of the discrete-time SMDP model (for single agents) (Dietterich, 2000). This approach involves the use of a graph to store a distributed value function. The overall task is first decomposed into subtasks up to the desired level of detail, and the task graph is constructed. We illustrate the idea using a simple two-robot search task shown in Figure 1. Consider the case where a robot is assigned the task of picking up trash from trash cans over an extended area and accumulating it into one centralized trash bin, from where it might be sent for recycling or disposed. This is a task which can be parallelized, if we have more than one agent working on it. An office (rooms and connecting corridors) type environment is shown in figure. A1 and A2 represent the two agents in the figure. Note the agents need to learn three skills here. First, how to do each subtask, such as navigating to $T1$ or $T2$ or $Dump$, and when to perform $Pickup$ or $Putdown$ action. Second, the agents also need to learn the order to do subtasks (for instance, go to $T1$ and collect trash before heading to the $Dump$). Finally, the agents also need to learn how to coordinate with other agents (i.e. $Agent1$ can pick up trash from $T1$ whereas $Agent2$ can service $T2$). The strength of the MAXQ framework (when extended to the multiagent case) is that it can serve as a substrate for learning all these three types of skills.

This trash collection task can be decomposed into subtasks and the resulting task graph is shown in figure 2. The task graph is then converted to the MAXQ graph, which is shown in figure 3. The MAXQ graph has two types of nodes: MAX nodes (triangles) and Q nodes (rectangles), which represent the different actions that can be done under their parents. Note that MAXQ allows learning of shared subtasks. For example, the navigation task Nav is common to several parent tasks.

The multiagent learning scenario under investigation in this paper can now be illustrated. Imagine the two robots start to learn this task with the same MAXQ graph structure. We can distinguish between two learning approaches, *selfish* and *cooperative*. In the *selfish* case, the two robots learn with the given MAXQ structure, but make no attempt to communicate with each other. In the *cooperative* case, the MAXQ structure is modified such that the Q nodes at the level(s) immediately under the root task include the joint action done by both robots. For instance, each robot learns the joint Q-value of navigating to trash $T1$ when

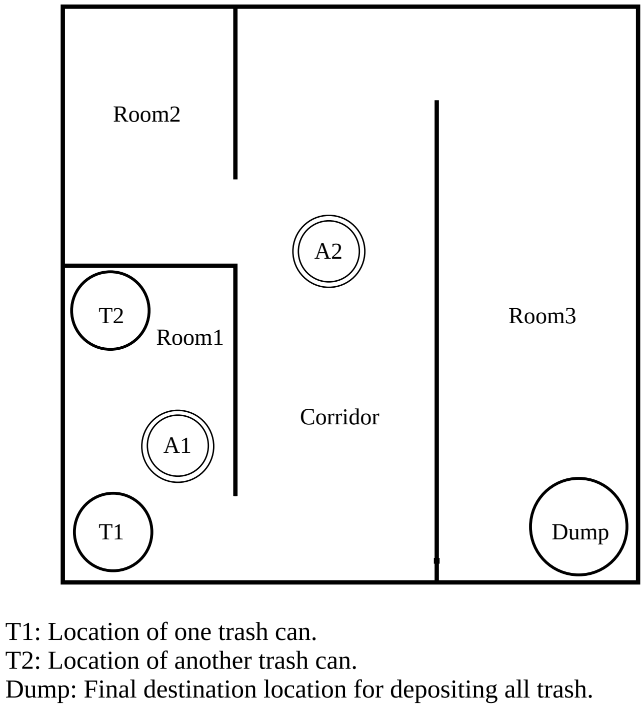


Figure 1: A (simulated) multiagent robot trash collection task.

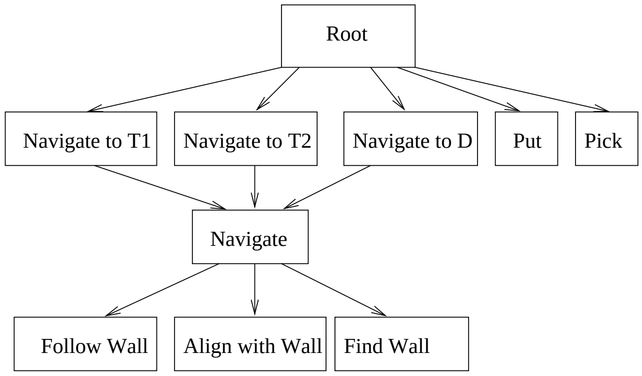


Figure 2: The task graph for the trash collection task.

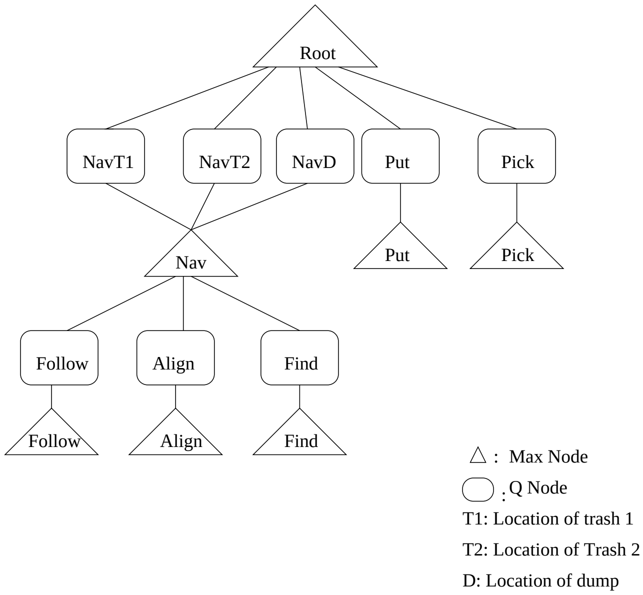


Figure 3: The MAXQ graph for the trash collection task.

the other robot is either navigating to $T1$ or $T2$ or $Dump$ or doing a Put or $Pick$ action. As we will show in a more complex domain below, cooperation among the agents results in superior learned performance than in the selfish case, or indeed the flat case when the agents do not use a task hierarchy at all.

More formally, the MAXQ method decomposes an MDP M into a set of subtasks $M_0, M_1, ..., M_n$. Each subtask is a three tuple $(T_i, A_i, \tilde{R}_i)$ defined as:

- $T_i(s_i)$ is a termination predicate which partitions the state space S into a set of active states S_i, and a set of terminal states T_i. The policy for subtask M_i can only be executed if the current state $s \in S_i$.

- A_i is a set of actions that can be performed to achieve subtask M_i. These actions can either be primitive actions from A, the set of primitive actions for the MDP, or they can be other subtasks.

- $\tilde{R}_i(s^{'}|s, a)$ is the pseudo reward function, which specifies a *pseudo-reward* for each transition from a state $s \in S_i$ to a terminal state $s^{'} \in T_i$. This *pseudo-reward* tells how desirable each of the terminal states is for this particular subtask.

Each primitive action a is a primitive subtask in the MAXQ decomposition, such that a is always executable, it terminates immediately after execution, and it's *pseudo-reward* function is uniformly zero. The projected value function V^{π} is the value of executing hierarchical policy π starting in state s, and at the root of the hierarchy. The outside completion function $(C(i, s, a))$ is the expected cumulative discounted reward of completing subtask M_i after invoking the subroutine for subtask M_a in state s. It is computed without any reference to $\tilde{R}_i$. This completion function will be used by parent tasks to compute $V(i, s)$, the expected reward for performing action i starting in state s. The inside completion function

($\tilde{C}(i,s,a)$) is a completion function, which is used only inside node i in order to discover the locally optimal policy for task M_i. This function incorporates rewards both from the real reward function, $R(s'|s,a)$, and from the pseudo-reward function, $\tilde{R}_i(s')$.

The value function $V(i,s)$ for doing task i in state s is calculated by decomposing it into two parts: the value of the subtask which is independent of the parent task, and the value of the completion of the task, which of course depends on the parent task.

$$V(i,s) = \begin{cases} \max_a Q(i,s,a) & \text{if i is composite} \\ \sum_{s'} P(s' \mid s,i) R(s' \mid s,i) & \text{if i is primitive} \end{cases}$$

$$Q(i,s,a) = V(a,s) + C(i,s,a) \tag{5}$$

where $Q(i,s,a)$ is the action value of doing subtask a in state s in the context of parent task i.

The Q values and the C values can be learned through a standard temporal-difference learning method, based on sample trajectories (see (Dietterich, 2000) for details). One important point to note here is that since subtasks are temporally extended in time, the update rules used here are based on the SMDP model.

Let us assume that an agent is in state s while doing task i, and chooses subtask j to execute. Let this subtask terminate after N steps and result in state s'. Then, the SMDP Q-learning rule used to update the outside and inside completion functions are given by

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

$$\tilde{C}_{t+1}(i,s,j) \leftarrow (1-\alpha_t) \tilde{C}_t(i,s,j) + \alpha_t \gamma^N [\tilde{R}_i(s') + \tilde{C}_t(i,s',a^*) + V_t(a^*,s')]$$

where $a^* = argmax_{a'} [\tilde{C}_t(i,s',a') + V_t(a',s')]$.

A *hierarchical* policy π is a set containing a policy for each of the subtasks in the problem: $\pi = \{\pi_0 \dots \pi_n\}$. The projected value function in the hierarchical case, denoted by $V^\pi(s)$, is the value of executing hierarchical policy π starting in state s and starting at the root of the task hierarchy. A *recursively optimal* policy for MDP M with MAXQ decomposition $\{M_0 \dots M_n\}$ is a hierarchical policy $\pi = \{\pi_0 \dots \pi_n\}$ such that for each subtask M_i the corresponding policy π_i is optimal for the SMDP defined by the set of states S_i, the set of actions A_i, the state transition probability function $P^\pi(s^{'}, N|s,a)$, and the reward function given by the sum of the original reward function $R(s^{'}|s,a)$ and the pseudo-reward function $\overline{R}_i(s^{'})$. The MAXQ learning algorithm has been proven to converge to π_r^*, the unique recursively optimal policy for MDP M and MAXQ graph H, where $M = (S,A,P,R,P_0)$ is a discounted infinite horizon MDP with discount factor γ, and H is a MAXQ graph defined over subtasks $\{M_0 \dots M_n\}$.

5. Continuous-Time Discounted Reward MAXQ Algorithm

At the center of the MAXQ method for hierarchical reinforcement learning is the MAXQ value function decomposition. We show how the overall value function for a policy is decomposed into a collection of value functions for individual subtasks for the continuous-time discounted reward model. The projected value function of hierarchical policy π on subtask M_i, denoted $V^\pi(i,s)$, is the expected cumulative discounted reward of executing π_i (and the policies of all descendants of M_i) starting in state s until M_i terminates. The value $V^\pi(i,s)$ has the following form in the continuous-time discounted reward framework:

$$V^\pi(i,s) = E_s^\pi\{\sum_{n=0}^{\infty} e^{-\beta\sigma_n} r(s_n,a_n)\} \tag{6}$$

where $r(s_n,a_n)$ is defined using Equation 2. Now let us suppose that the first action chosen by π_i is invoked and executes for a number of steps N and terminates in state s' according to $P_i^\pi(s'|s,a)$. We can rewrite Equation 6 as

$$V^\pi(i,s) = E_s^\pi\{\sum_{n=0}^{N-1} e^{-\beta\sigma_n} r(s_n,a_n) + e^{-\beta\sigma_N}\sum_{n=0}^{\infty} e^{-\beta\sigma_n} r(s_{N+n},a_{N+n})\} \tag{7}$$

The first summation on the right-hand side of Equation 7 is the discounted sum of rewards for executing subroutine $\pi_i(s)$ starting in state s until it terminates, in other words, it is $V^\pi(\pi_i(s),s)$, the projected value function for the child task $M_{\pi_i(s)}$. The second term on the right-hand side of the equation is the value of s' for the current task i, $V^\pi(i,s')$, discounted by $e^{-\beta t}$, where s' is the current state when subroutine $\pi_i(s)$ terminates and t is the sample transition time from state s to state s'. We can write Equation 7 in the form of a Belmopan equation:

$$V^\pi(i,s) = V^\pi(\pi_i(s),s) + \sum_{s'\in S_i} P_i(s'|s,\pi_i(s)) \int_0^\infty e^{-\beta t} V^\pi(i,s') F_i(dt|s,\pi_i(s)) \tag{8}$$

Equation 8 can be re-stated for action-value function decomposition as follows:

$$Q^\pi(i,s,a) = V^\pi(a,s) + \sum_{s'\in S_i} P_i(s'|s,a) \int_0^\infty e^{-\beta t} Q^\pi(i,s',\pi_i(s')) F_i(dt|s,a)$$

The right-most term in this equation is the expected discounted cumulative reward of completing task M_i after executing action a in state s. This term is called the *completion function* and is denoted by $C^\pi(i,s,a)$. With this definition, we can express the Q function recursively as

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

and we can re-express the definition for V as

$$V^\pi(i,s) = \begin{cases} Q^\pi(i,s,\pi_i(s)) & \text{if i is composite} \\ k_i(s,i) + \int_0^\infty \sum_{s'\in S_i} P_i(s'|s,i)[\int_0^u e^{-\beta t} c_i(s',s,i)dt] F_i(du|s,i) & \\ & \text{if i is primitive} \end{cases}$$

We can use the above formulas to obtain update equations for value function V, outside completion function C and inside completion function $\tilde{C}$ in the continuous-time discounted reward model. Pseudo-code for the resulting algorithm is shown in Algorithm 1 [1] (Ghavamzadeh and Mahadevan, 2001).

1. We use the notation $u \xleftarrow{\alpha} v$ in Algorithm 1, Algorithm 2 and Algorithm 3 as an abbreviation for the stochastic approximation update rule $u \leftarrow (1-\alpha)u + \alpha v$.

Algorithm 1 The continuous-time discounted reward MAXQ algorithm.

function MAXQ(MaxNode i, State s)
let Seq={} be the sequence of (states visited, transition times) while executing i
if i is a primitive MaxNode **then**
    execute action i in state s, observe state s' in τ time units, receive lump portion of reward $k(s,i)$ and continuous portion of reward with rate $r(s',s,i)$
    $V_{t+1}(i,s) \xleftarrow{\alpha} [k(s,i) + \frac{1-e^{-\beta\tau}}{\beta} r(s',s,i)]$
    push (state s, transition time τ) into the beginning of Seq
else
    while i has not terminated **do**
        choose action a according to the current exploration policy $\pi_i(s)$
        let ChildSeq=MAXQ(a,s), where ChildSeq is the sequence of (states visited, transition times) while executing action a
        observe result state s'
        let $a^* = argmax_{a' \in A_i(s')}[\tilde{C}_t(i,s',a') + V_t(a',s')]$
        $T = 0$;
        for (s,τ) in ChildSeq from the beginning do
            $T = T + \tau$
            $\tilde{C}_{t+1}(i,s,a) \xleftarrow{\alpha} e^{-\beta T}[\tilde{R}_i(s') + \tilde{C}_t(i,s',a^*) + V_t(a^*,s')]$
            $C_{t+1}(i,s,a) \xleftarrow{\alpha} e^{-\beta T}[C_t(i,s',a^*) + V_t(a^*,s')]$
        end for
        append ChildSeq onto the front of Seq
        $s = s'$
    end while
end if
return Seq
end MAXQ

6. Discrete-Time Average Reward MAXQ Algorithm

We now describe a new discrete-time average reward hierarchical reinforcement learning algorithm based on the MAXQ framework. To simplify exposition, we assume that for every possible stationary policy of each subtask in the hierarchy, the embedded Markov chain has a unichain transition probability matrix. Under this assumption every subtask in the hierarchy is a unichain SMDP. This means the expected average reward of every stationary policy for each subtask in the hierarchy does not vary with initial state. As we mentioned earlier, value function decomposition is the heart of the MAXQ method. We show how the overall h function for a policy is decomposed into a collection of h functions for individual subtasks in the discrete-time average reward MAXQ method. The projected h function of hierarchical policy π on subtask M_i, denoted $h^\pi(i,s)$, is the average adjusted sum of rewards earned of following policy π_i (and the policies of all descendants of M_i) starting in state s until M_i terminates:

$$h^\pi(i,s) = \lim_{N\to\infty} E_s^\pi\{\sum_{u=0}^{N-1}(r(s_u,a_u) - g^i)\} \tag{9}$$

where g^i is the gain of subtask M_i. Now let us suppose that the first action chosen by π is invoked and executes for a number of steps N and terminates in state s' according to $P_i^\pi(s',N|s,a)$. We can write Equation 9 in the form of a Bellman equation:

$$h^\pi(i,s) = r(s,\pi_i(s)) - g^i y_i(s,\pi_i(s)) + \sum_{s'\in S_i,N} P_i(s',N|s,\pi_i(s))h^\pi(i,s') \tag{10}$$

Since $r(s,\pi_i(s))$ is the expected total reward between two decision epochs of subtask i, given that the system occupies state s at the first decision epoch and decision maker chooses action $\pi_i(s)$ and the number of time steps until next decision epoch is N, we have

$$r(s,\pi_i(s)) = V^\pi_{y_i(s,\pi_i(s))}(\pi_i(s),s) = h^\pi(\pi_i(s),s) + g^{\pi_i(s)}y_i(s,\pi_i(s))$$

By replacing $r(s,\pi_i(s))$ from the above expression, Equation 10 can be written as

$$h^\pi(i,s) = h^\pi(\pi_i(s),s) - (g^i - g^{\pi_i(s)})y_i(s,\pi_i(s)) + \sum_{s'\in S_i,N} P_i(s',N|s,\pi_i(s))h^\pi(i,s') \tag{11}$$

We can re-state Equation 11 for action-value function decomposition as follows:

$$R^\pi(i,s,a) = h^\pi(a,s) - (g^i - g^a)y_i(s,a) + \sum_{s'\in S_i,N} P_i(s',N|s,a)R^\pi(i,s',\pi_i(s'))$$

In the above equation, the term

$$-(g^i - g^a)y_i(s,a) + \sum_{s'\in S_i,N} P_i(s',N|s,a)R^\pi(i,s',\pi_i(s'))$$

denotes the average adjusted reward of completing task M_i after executing action a in state s. This term is called the completion function and is denoted by $C^\pi(i,s,a)$. With this definition, we can express the R function recursively as

$$R^\pi(i,s,a) = h^\pi(a,s) + C^\pi(i,s,a)$$

and we can re-express the definition for h as

$$h^\pi(i,s) = \begin{cases} R^\pi(i,s,\pi_i(s)) & \text{if i is composite} \\ \sum_{s'} P(s'|s,i)[r(s'|s,i) - g^i] & \text{if i is primitive} \end{cases}$$

The above formulas can be used to obtain update equations for h function, outside completion function C and inside completion function $\tilde{C}$ in the discrete-time average reward model. Pseudo-code for the resulting algorithm is shown in Algorithm 2. As mentioned above, all subtasks in the hierarchy, even primitive actions, are modeled by a unichain SMDP.

7. Continuous-Time Average Reward MAXQ Algorithm

We now describe a new average reward hierarchical reinforcement learning algorithm based on the MAXQ framework. To simplify exposition, we assume that for every possible stationary policy of each subtask in the hierarchy, the embedded Markov chain has a unichain transition probability matrix. Under this assumption every subtask in the hierarchy is a unichain SMDP. This means the expected average reward of every stationary policy for each subtask in the hierarchy does not vary with initial state. As we mentioned earlier, value function decomposition is the heart of the MAXQ method. We show how the overall h function for a policy is decomposed into a collection of h functions for individual subtasks in the continuous-time average reward MAXQ method. The projected h function of hierarchical policy π on subtask M_i, denoted $h^\pi(i,s)$, is the average adjusted sum of rewards earned of following policy π_i (and the policies of all descendants of M_i) starting in state s until M_i terminates:

$$h^\pi(i,s) = \lim_{N\to\infty} E_s^\pi\{\sum_{t=0}^{N-1}(r(s_t,a_t) - g^i\tau_t)\} \tag{12}$$

where τ's and g^i are the length of decision epochs and gain of subtask M_i respectively. Now let us suppose that the first action chosen by π is invoked and executes for a number of steps and terminates in state s' according to $P_i^\pi(s'|s,a)$. We can write Equation 12 in the form of a Bellman equation:

$$h^\pi(i,s) = r(s,\pi_i(s)) - g^i y_i(s,\pi_i(s)) + \sum_{s'\in S_i} P_i(s'|s,\pi_i(s)) \int_0^\infty h^\pi(i,s') F_i(dt|s,\pi_i(s)) \tag{13}$$

Since $r(s,\pi_i(s))$ is the expected total reward between two decision epochs of subtask i, given that the system occupies state s at the first decision epoch and decision maker chooses action $\pi_i(s)$ and the expected length of time until next decision epoch is $y_i(s,\pi_i(s))$, we have

$$r(s,\pi_i(s)) = V^\pi_{y_i(s,\pi_i(s))}(\pi_i(s),s) = h^\pi(\pi_i(s),s) + g^{\pi_i(s)} y_i(s,\pi_i(s))$$

By replacing $r(s,\pi_i(s))$ from the above expression, Equation 13 can be written as

$$h^\pi(i,s) = h^\pi(\pi_i(s),s) - (g^i - g^{\pi_i(s)}) y_i(s,\pi_i(s)) + \sum_{s'\in S_i} P_i(s'|s,\pi_i(s)) \int_0^\infty h^\pi(i,s') F_i(dt|s,\pi_i(s)) \tag{14}$$

Algorithm 2 The discrete-time average reward MAXQ algorithm.

function MAXQ(MaxNode i, State s)
let Seq={} be the sequence of (states visited, transition times, reward) while executing i
if i is a primitive MaxNode **then**
  execute action i in state s, receive reward r, and observe state s'
  $h_{t+1}(i,s) \longleftarrow (1-\alpha)h_t(i,s) + \alpha(r(s',s,i) - g^i)$
  if i is a non-random action **then**
    update average reward or gain of subtask i
    $g^i_{t+1} = \frac{r_{t+1}(i)}{n_{t+1}(i)} = \frac{r_t(i)+r_t}{n_t(i)+1}$
  end if
  push (state s, reward r_t) into the beginning of Seq
else
  while i has not terminated **do**
    choose action a according to the current exploration policy $\pi_i(s)$
    let ChildSeq=MAXQ(a,s), where ChildSeq is the sequence of (states visited, transition times) while executing action a
    observe result state s'
    let $a^* = argmax_{a' \in A_i(s')}[\tilde{C}_t(i,s',a') + h_t(a',s')]$
    let $N = 1$; $R = 0$;
    for each (s,r) in ChildSeq from the beginning do
      $R = R + r$;
      $\tilde{C}_{t+1}(i,s,a) \xleftarrow{\alpha} [\tilde{R}_i(s') - (g^i_t - g^a_t)N + \tilde{C}_t(i,s',a^*) + h_t(a^*,s')]$
      $C_{t+1}(i,s,a) \xleftarrow{\alpha} [C_t(i,s',a^*) + h_t(a^*,s') - (g^i_t - g^a_t)N]$
      $N = N + 1$
    if a is a non-random action **then**
      update average reward or gain of subtask i
      $g^i_{t+1} = \frac{r_{t+1}(i)}{n_{t+1}(i)} = \frac{r_t(i)+R}{n_t(i)+N}$
    end if
    end for
    append ChildSeq onto the front of Seq
    $s = s'$
  end while
end if
return Seq
end MAXQ

We can re-state Equation 14 for action-value function decomposition as follows:

$$R^\pi(i,s,a) = h^\pi(a,s) - (g^i - g^a)y_i(s,a) + \sum_{s' \in S_i} P_i(s'|s,a) \int_0^\infty R^\pi(i,s',\pi_i(s'))F_i(dt|s,a)$$

In the above equation, the term

$$-(g^i - g^a)y_i(s,a) + \sum_{s' \in S_i} P_i(s'|s,a) \int_0^\infty R^\pi(i,s',\pi_i(s'))F_i(dt|s,a)$$

denotes the average adjusted reward of completing task M_i after executing action a in state s. This term is called the completion function and is denoted by $C^\pi(i,s,a)$. With this definition, we can express the R function recursively as

$$R^\pi(i,s,a) = h^\pi(a,s) + C^\pi(i,s,a)$$

and we can re-express the definition for h as

$$h^\pi(i,s) = \begin{cases} R^\pi(i,s,\pi_i(s)) & \text{if i is composite} \\ \quad k(s,i) + \int_0^\infty \sum_{s' \in S_i} P_i(s'|s,i)[\int_0^u c(s',s,i)dt]F_i(du|s,i) & \\ \qquad - g^i \sum_{s' \in S_i} P_i(s'|s,i) \int_0^\infty tF_i(dt|s,i) & \text{if i is primitive} \end{cases}$$

The above formulas can be used to obtain update equations for h function, outside completion function C and inside completion function $\tilde{C}$ in the continuous-time average reward model. Pseudo-code for the resulting algorithm is shown in Algorithm 3 (Ghavamzadeh and Mahadevan, 2001) As mentioned above, all subtasks in the hierarchy, even primitive actions, are modeled by a unichain SMDP.

8. Multiagent MAXQ Algorithm (Cooperative MAXQ)

The MAXQ decomposition of the Q-function relies on a key principle: the reward function for the parent task is the value function of the child task (see Equation 5). We show how this idea can be extended to joint-action values. The most salient feature of the extended MAXQ algorithm, which is proposed in this section, is that the top level(s) (the level immediately below the root, and perhaps lower levels) of the hierarchy is (are) configured to store the completion function (C) values for joint (abstract) actions of all agents. The completion function $C^j(i,s,a^1,a^2...a^j...a^n)$ is defined as the expected discounted reward of completion of subtask a^j by agent j in the context of the other agents performing subtasks $a^i, \forall i \in \{1,...,n\}, i \neq j$ (Makar et al., 2001).

More precisely, the decomposition equations used for calculating the projected value function V have the following form (for agent j)

$$V^j(i,s,a^1,\dots,a^{j-1},a^{j+1},\dots,a^n) = \begin{cases} \max_{a^j} Q^j(i,s,a^1 \dots a^j \dots a^n) & \text{if i is composite} \\ \sum_{s'} P(s' \mid s,i)R(s' \mid s,i) & \text{if i is primitive} \end{cases}$$

Algorithm 3 The continuous-time average reward MAXQ algorithm.

function MAXQ(MaxNode i, State s)
let Seq={} be the sequence of (states visited, transition times, reward) while executing i
if i is a primitive MaxNode **then**
  execute action i in state s, observe state s' in τ time units, receive lump portion of reward $k(s,i)$ and continuous portion of reward with rate $r(s',s,i)$
  $h_{t+1}(i,s) \xleftarrow{\alpha} [k(s,i) + r(s',s,i)\tau - g_t^i\tau]$
  if i is a non-random action **then**
    update average reward or gain of subtask i
    $g_{t+1}^i = \frac{r_{t+1}(i)}{t_{t+1}(i)} = \frac{r_t(i)+k(s,i)+r(s',s,i)\tau}{t_t(i)+\tau}$
  end if
  push (state s, transition time τ, reward $\rho = k(s,i) + r(s',s,i)\tau$) into the beginning of Seq
else
  while i has not terminated **do**
    choose action a according to the current exploration policy $\pi_i(s)$
    let ChildSeq=MAXQ(a,s), where ChildSeq is the sequence of (states visited, transition times) while executing action a
    observe result state s'
    let $a^* = argmax_{a' \in A_i(s')}[\tilde{C}_t(i,s',a') + h_t(a',s')]$
    $T = 0; \quad R = 0;$
    for (s,τ,ρ) in ChildSeq from the beginning do
      $T = T + \tau; \quad R = R + \rho;$
      $\tilde{C}_{t+1}(i,s,a) \xleftarrow{\alpha} [\tilde{R}_i(s') - (g_t^i - g_t^a)T + \tilde{C}_t(i,s',a^*) + h_t(a^*,s')]$
      $C_{t+1}(i,s,a) \xleftarrow{\alpha} [C_t(i,s',a^*) + h_t(a^*,s') - (g_t^i - g_t^a)T]$
    if a is a non-random action **then**
      update average reward or gain of subtask i
      $g_{t+1}^i = \frac{r_{t+1}(i)}{t_{t+1}(i)} = \frac{r_t(i)+R}{t_t(i)+T}$
    end if
    end for
    append ChildSeq onto the front of Seq
    $s = s'$
  end while
end if
return Seq
end MAXQ

$$Q^j(i, s, a^1 \ldots a^j \ldots a^n) = V^j(a^j, s) + C^j(i, s, a^1 \ldots a^j \ldots a^n) \tag{15}$$

at the highest (or lower than the highest as needed) level(s) of the hierarchy, where joint action values are being modeled, and a^j is the action being performed by agent j. Compare the decomposition in Equation 5 with Equation 15. Given a MAXQ hierarchy M for any given task, we need to find the highest level at which this equation provides a sufficiently good approximation of the true value. For both the AGV and the trash collection domain, the subtasks immediately below the root seem to be a good compromise between good performance and reducing the number of joint state action values that need to be learned.

To illustrate the multiagent MAXQ algorithm, for the two-robot trash collection task, if we set up the joint action-values at only the highest level of the MAXQ graph, we get the following value function decomposition for $Agent1$:

$$Q_t^1(Root, s, NavT1, NavT2) = V_t^1(NavT1, s) + C_t^1(Root, s, NavT1, NavT2)$$

which represents the value of $Agent1$ doing task $NavT1$ in the context of the overall $Root$ task, when $Agent2$ is doing task $NavT2$. Note that this value is decomposed into the value of the $NavT1$ subtask itself and the completion cost of the remainder of the overall task. In this example, the multiagent MAXQ decomposition embodies the heuristic that the value of $Agent1$ doing the subtask $NavT1$ is independent of whatever $Agent2$ is doing.

A recursive algorithm is used for learning the C values. Thus, an agent starts from the root task and chooses a subtask till it gets to a primitive action. The primitive action is executed, the reward observed, and the leaf V values updated. Whenever any subtask terminates, the $C(i, s, a)$ values are updated for all states visited during the execution of that subtask. Similarly, when one of the tasks at the level just below the root task terminates, the $C(i, s, a^1, \ldots, a^n)$ values are updated according to the MAXQ learning algorithm.

9. The AGV Scheduling Task

Automated Guided Vehicles (AGVs) are used in flexible manufacturing systems (FMS) for material handling (Askin and Standridge, 1993). 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 dropoff 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 pickup point is the machine or workstation's output buffer. Any FMS system using AGVs faces the problem of optimally scheduling the paths of AGVs in the system(Lee, 1996). For example, a move request occurs when a part finishes at a workstation. If more than one vehicle is empty, the vehicle which would service this request needs to be selected. Also, when a vehicle becomes available, and multiple move requests are queued, a decision needs to be made as to which request should be serviced by that vehicle. These schedules obey a set of constraints that reflect the temporal relationships between activities and the capacity limitations of a set of shared resources.

The uncertain and ever changing nature of the manufacturing environment makes it virtually impossible to plan moves ahead of time. Hence, AGV scheduling requires dynamic dispatching rules, which are dependent on the state of the system like the number of parts

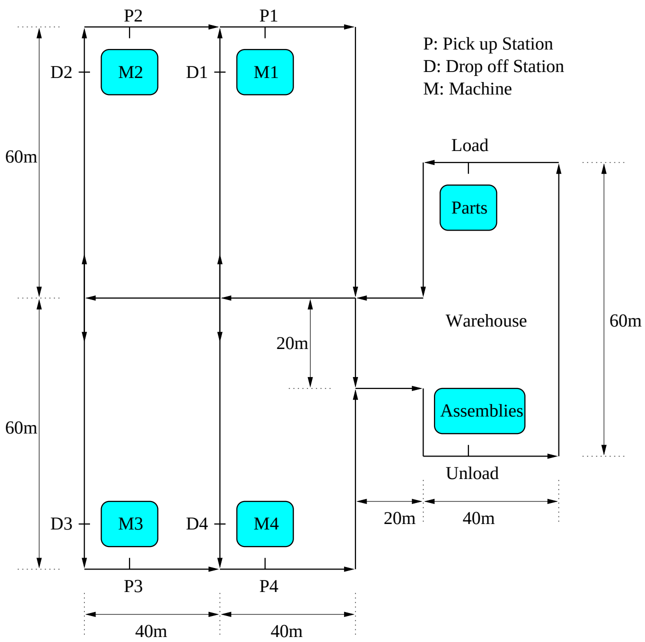
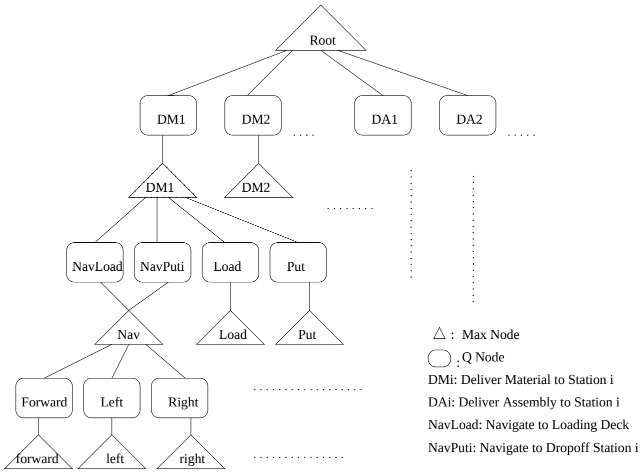


Figure 4: A multiple automatic guided vehicle (AGV) optimization task. There are four AGV agents (not shown) which carry raw materials and finished parts between the machines and the warehouse.

in each buffer, the state of the AGV and the processing going on at the workstations. The system performance is generally measured in terms of the throughput, the online 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 (Klein and Kim, 1996, Lee, 1996). However, the heuristics perform poorly when the constraints on the movement of the AGVs are reduced.

Previously, Tadepalli and Ok (Tadepalli and Ok, 1996b) studied a single agent AGV scheduling task using "flat" average-reward reinforcement learning. However, the multiagent AGV task we study is more complex. Figure 4 shows the layout of the system used for experimental purposes in this paper. $M1$ to $M4$ show workstations in this environment. Parts of type i have to be carried to drop off station at workstation i, D_i, and the assembled parts brought back from pick up stations of workstations, P_is, to the warehouse. The AGV travel is unidirectional (as the arrows show).

The termination predicate has been redefined to take care of the fact that the completion of certain tasks might depend on the occurrence of an event rather than just a state of the environment. For example, if we consider the $DM1$ subtask in the AGV problem (see Figure 5), the state of the system at the beginning of the subtask might be the same as that at the end, as the system is very dynamic. New parts continuously arrive at the warehouse, and the machines start and end work on parts at random intervals. Also, the actions of



(b)

Figure 5: MAXQ graph for the AGV scheduling task.

a number of agents affects the environment. This kind of discrete event model makes it necessary to have termination of subtasks to be defined in terms of events. Hence, a subtask terminates when the event associated with that subtask is triggered by the robot performing the subtask, for example DM1 subtask terminates when the *"unload of material 1 at drop off station of machine 1"* event occurs.

9.1 State Abstraction

The state of the environment consists of the number of parts in the pickup station and in the dropoff station 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 state as a part of the state space. Thus, in the flat case, the size of the state space is ≈ 100 locations, 3 parts in each buffer, 9 possible states of the AGV (carrying Part1, ..., carrying Assembly1, ..., Empty), and 2 values for each part in the warehouse, i.e. $100 \times 4^8 \times 9 \times 2^4 \approx 2^{30}$, which is enormous. The MAXQ 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 *Navigate* subtask, only the location state variable is relevant, and this subtask can be learned with 100 values. Hence, for the highest level actions DM1, $\ldots$, DM4, the number of relevant states would be $100 \times 9 \times 4 \times 2 \approx 2^{13}$, and for highest level actions DA1, $\ldots$, DA4, the number of states would be $100 \times 9 \times 4 \approx 2^{12}$. For the lower level state space, the action with the largest state space is Navigate with 100 values. This state abstraction gives us a compact way of representing the C functions, and speeds up the algorithm (Dietterich, 2000).

10. Experimental Results

We first describe experiments in the simple two-robot trash collection problem, and then we will turn to the more complex multiagent AGV task and apply the cooperative discrete and continuous time MAXQ algorithms to this problem.

10.1 Trash Collection Task (Discrete Time Model)

We first provide more details of how we implemented the trash collection task. In the single agent scenario, 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 Dump. The state space here is the orientation of the robot (N,S,W,E), and another component based on its percept. We assume that a ring of 16 sonars would enable the robot to find out whether it is in a corner, (with two walls perpendicular to each other on two sides of the robot), near a wall (with wall only on one side), near a door (wall on either side of an opening), in a corridor (parallel walls on either side) or in an open area (the middle of the room). Thus, each room is divided into 9 states, and the corridor into 4 states. Thus, we have $((9 \times 3) + 4) \times 4$, or 124 locations for a robot. Also, the trash object from trash basket $T1$ can be at $T1$, carried with robot, or at *Dump*, and the trash object from trash basket $T2$ can be at $T2$, carried by robot, or at *Dump*. Thus the total number of environment states is $124 \times 3 \times 3$, or 1116 for the single agent case. Going to the two-agent case would mean that the trash can be at either $T1$ or $T2$, *Dump*, or carried by one of the two robots. Thus, in the flat case, the size of the state space would grow to $124 \times 124 \times 4 \times 4$, or $\approx 24 \times 10^4$.

The environment is fully observable given this state decomposition, as the direction which the robot is facing, in combination with the percept (which includes the room the agent is in) gives a unique value for each location. 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 wall, enter or exit door, align south or north in the corridor, or move in the corridor.

In the two-robot trash collection task, examination of the learned policy in Figure 6 reveals that the robots have nicely learned all three skills: how to achieve a subtask, what order to do them in, and how to coordinate with other agents. In addition, as Figure 7 confirms, the number of steps needed to do 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.

10.2 AGV Domain (Discrete-Time Model)

We now present detailed experimental results on the AGV scheduling task, comparing several learning agents, including a single agent using MAXQ, selfish multiple agents using MAXQ (where each agent acts independently and learns its own optimal policy), and the new cooperative multiagent MAXQ approach. In this domain, there are four agents (each AGV is an agent).

The experimental results were generated with the following model parameters. The inter-arrival time for parts at the warehouse is uniformly distributed with a mean of 4 sec

Learned Policy for Agent 1

root
navigate to trash 1
go to location of trash 1 in room 1
pick trash 1
navigate to bin
exit room 1
enter room 3
go to location of dump in room 3
put trash 1 in dump
end

Learned Policy for Agent 2

root
navigate to trash 2
go to location of trash 2 in room 1
pick trash 2
navigate to bin
exit room 1
enter room 3
go to location of dump in room 3
put trash 2 in dump
end

Figure 6: This figure shows the policy learned by the cooperative multiagent MAXQ algorithm in the trash collection task.

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 uniformly 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 (AGV navigation actions, load and unload) is 1000 micro sec. The execution time for idle action is 1 sec. Each experiment was conducted five times and the results averaged.

Figure 8 shows the throughput of the system for the three types of approaches. As seen in Figure 8, the agents learn a little faster initially in the selfish multiagent method, but after some time, undulations are seen in the graph showing not only that the algorithm does not stabilize, but also that it results in sub-optimal performance. This is due to the fact that two or more agents select the same action, but once the first agent completes the task, the other agents might have to wait for a long time to complete the task, due to the constraints on the number of parts that can be stored at a particular place. The system throughput

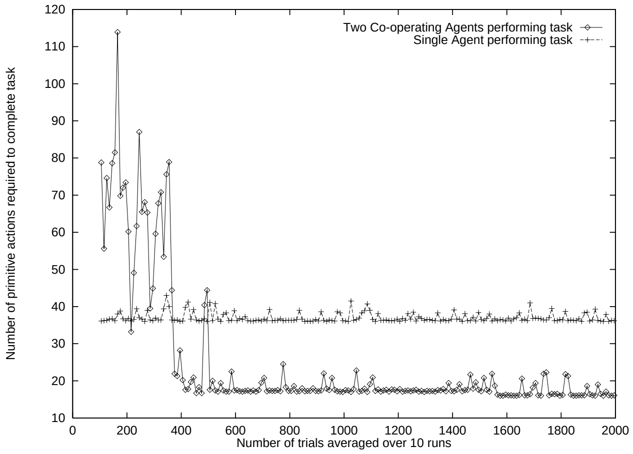


Figure 7: Number of actions needed to complete the trash collection task.

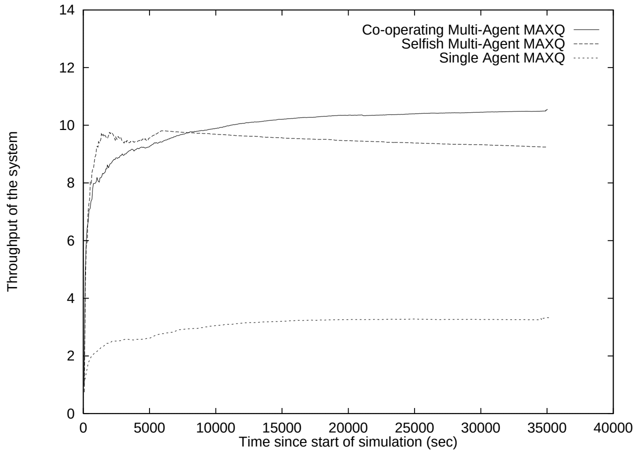


Figure 8: This figure shows that the cooperative multiagent MAXQ approach outperforms both the selfish (non-cooperative) and single agent MAXQ approaches when the AGV travel time is very much less compared to the assembly time. Learning curves are averaged over five runs.

achieved using the new cooperative multiagent MAXQ method is significantly higher than the single agent or selfish multiagent case. This difference is even more significant in figure 9, as when the primitive actions have longer execution time, almost $\frac{1}{10^{th}}$ the average assembly time (the execution time of primitive actions is 2 sec).

Figure 10 shows results from an implementation of a single flat Q-Learning agent 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 11 compares the cooperative multiagent MAXQ algorithm with several well-known AGV scheduling rules, showing clearly the improved performance of the reinforcement learning method. Finally, Figure 12

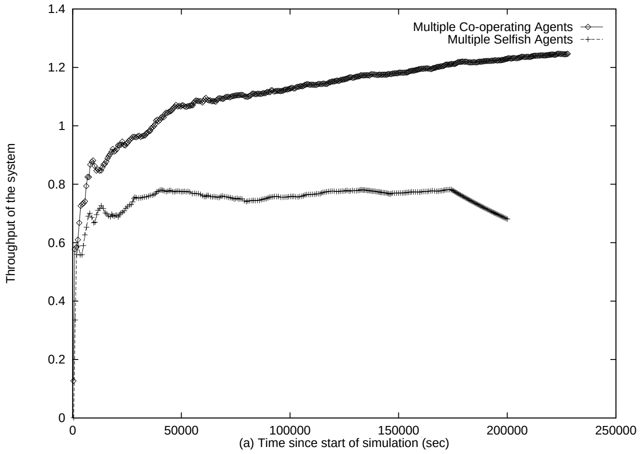


Figure 9: This figure compares the cooperative multiagent MAXQ approach with the selfish (non-cooperative) MAXQ approach, when the AGV travel time and load/unload time is $\frac{1}{10^{th}}$ the average assembly time. Learning curves are averaged over five runs.

shows that when the Q-nodes at the top two levels of the hierarchy are configured to represent joint action-values, learning is considerably slower (since the number of parameters is increased significantly), and the overall performance is not better. The lack of improvement is due in part to the fact that the second layer of the MAXQ hierarchy is concerned with navigation. Adding joint actions does not help improve navigation because coordination is not necessary in this environment. However, it might turn out that adding joint actions in multiple layers will be worthwhile, even if convergence is slower, due to better overall task performance.

10.3 AGV Domain (Continuous-Time Model)

We now apply the two proposed continuous-time algorithms described in section 4 to the AGV scheduling task and compare their performance and speed with each other, as well as several well-known AGV scheduling heuristics.

The experimental results were generated with the following model parameters. The inter-arrival time for parts at the warehouse is uniformly distributed with a mean of 4 sec and variance of 1 sec. The percentage of Part1, Part2, Part3 and Part4 in the part arrival process are 20, 28, 22 and 30 respectively. The time required for assembling the various parts is normally distributed with means 15, 24, 24 and 30 sec for Part1, Part2, Part3 and Part4 respectively, and the variance 2 sec. The execution time of primitive actions (AGV navigation actions, load and unload) is normally distributed with mean 1000 micro sec and variance 50 micro sec. The execution time of idle action is normally distributed with mean 1 sec and variance 0.1 sec. Table 1 shows the value of all the parameters of the continuous-time model used in the experimental results of this section. Each experiment was conducted five times and the results averaged.

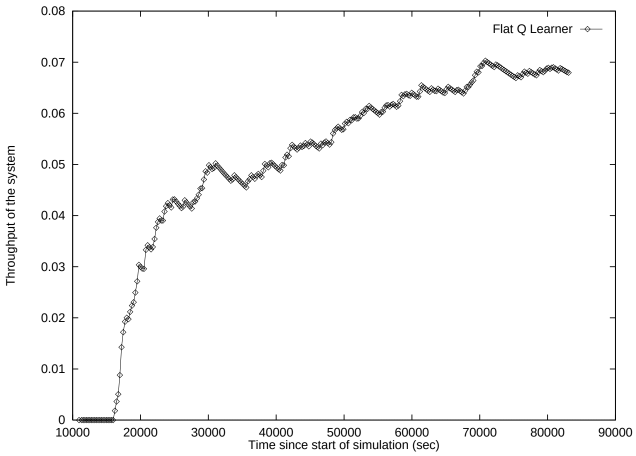


Figure 10: A flat Q-learner learns the AGV domain extremely slowly, showing the need for using a hierarchical task structure. Note the y axis of this plot is greatly expanded upward comparing to Figure 8.

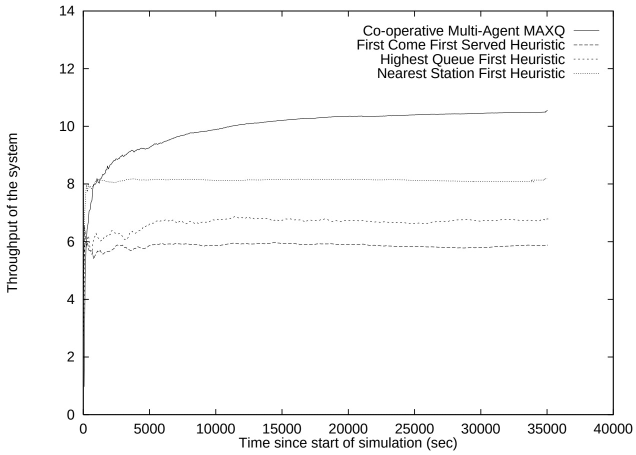


Figure 11: This plot shows the multiagent MAXQ outperforms three well-known widely used (industrial) heuristics for AGV scheduling. Learning curves are averaged over five runs.

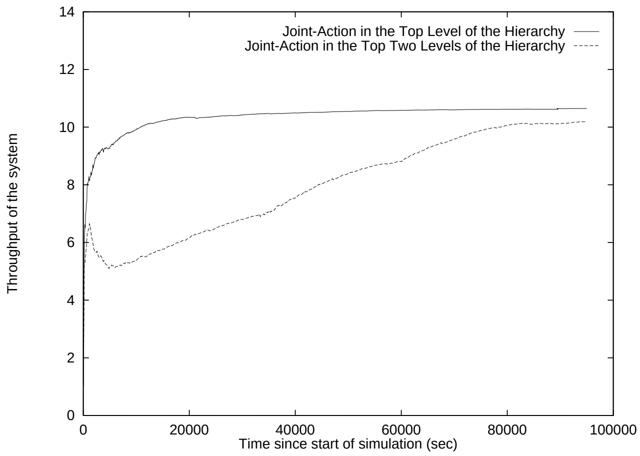


Figure 12: This plot compares the performance of the multiagent MAXQ algorithm with joint actions at the top level vs. joint actions at the top two levels. Learning curves are averaged over five runs.

Table 1: Model Parameters

Parameter	**Type of Distribution**	**Mean**	**Variance**
Idle Action	Normal	1 (sec)	0.1 (sec)
Primitive Actions	Normal	1000 (micro sec)	50 (micro sec)
Assembly Time for Part1	Normal	15 (sec)	2 (sec)
Assembly Time for Part2	Normal	24 (sec)	2 (sec)
Assembly Time for Part3	Normal	24 (sec)	2 (sec)
Assembly Time for Part4	Normal	30 (sec)	2 (sec)
Inter-Arrival Time for Parts	Uniform	4 (sec)	1 (sec)

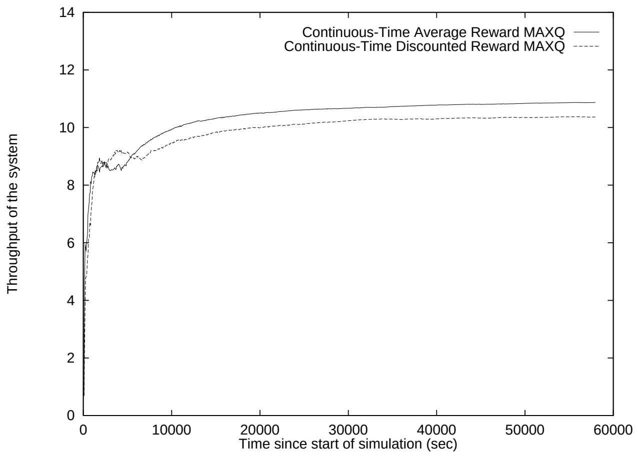


Figure 13: This plot shows continuous-time average reward multiagent MAXQ algorithm outperforms continuous-time discounted reward multiagent MAXQ algorithm. Learning curves averaged over five runs.

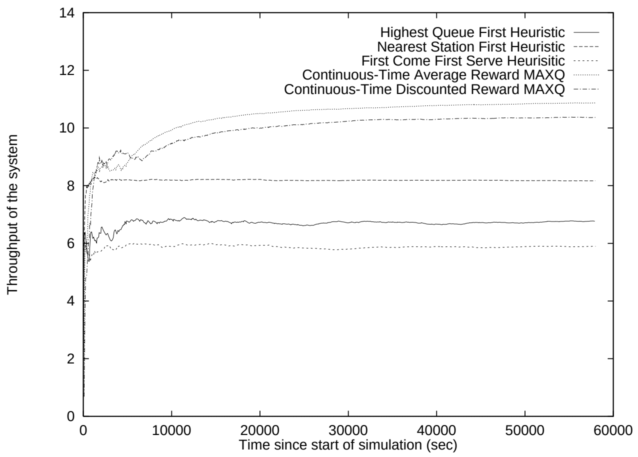


Figure 14: This plot shows both continuous-time average reward and discounted reward multiagent MAXQ algorithms outperform three well-known widely used (industrial) heuristics for AGV scheduling. Learning curves averaged over five runs.

Figure 13 shows the throughput of the system for continuous-time discounted and average reward MAXQ algorithms proposed in this paper. As seen in this figure, the agents learn a little faster initially in the discounted reward method, but the final system throughput achieved using the average reward algorithm is higher than the discounted reward case.

Figure 14 compares the proposed continuous-time MAXQ algorithms 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 reinforcement learning methods.

11. Conclusion and Future Work

This paper extended the framework of hierarchical reinforcement learning to continuous-time, average-reward, and multiagent domains. Using the MAXQ framework as an example, we described three new hierarchical reinforcement learning algorithms: *continuous-time discounted reward MAXQ*, *discrete-time average reward MAXQ*, and *continuous-time average reward MAXQ*. We also extended the MAXQ framework to the multiagent case (*cooperative MAXQ*), where each agent uses the same task hierarchy. Learning is decentralized, with each agent learning three interrelated skills: *how to perform subtasks*, *which order to do them in*, and *how to coordinate with other agents*. Coordination skills among agents are learned by using joint actions at the highest level(s) of the hierarchy.

The effectiveness of the proposed algorithms were tested using two experimental testbeds: a simulated robot trash collection domain, and a much larger real-world multi-agent autonomous guided vehicle (MAGV) domain. The proposed algorithms performed well in both domains, and in particular, in the MAGV domain, we showed that our proposed extensions outperform widely used industrial heuristics, such as "*first come first serve*", "*highest queue first*" and "*nearest station first*".

There are a number of directions for future work which can be briefly outlined. In the continuous-time and average-reward extensions of MAXQ, an immediate question that arises is the convergence of the algorithms to *recursively optimal* policies. These results should provide some theoretical validity to the proposed methods, in addition to their empirical effectiveness demonstrated in this paper. The multiagent extension to MAXQ is more difficult to analyze theoretically, due to the inherent complexity of the multiagent setting. However, a number of empirical extensions would be useful, from modeling the cost of communication among agents, to studying the scenario where agents begin with dissimilar task hierarchies. Finally, although our work primarily focused on the MAXQ framework, the key ideas underlying our proposed methods could be equally well applied to other HRL frameworks, such as options and HAMs.

Acknowledgments

This work was supported in part by the Defense Advanced Research Projects Agency, DARPA contract No. DAANO2-98-C-4025, and in part by a Knowledge and Distributed Intelligence (KDI) grant ECS-9873531 from the National Science Foundation. The computational experiments were carried out in the Autonomous Agents laboratory in the Department of Computer Science and Engineering at Michigan State University.

References

R.G. Askin and C.R. Standridge. *Modeling and Analysis of Manufacturing Systems.* John Wiley and Sons, 1993.

Tucker Balch and Ronald Arkin. Behavior-based formation control for multi-robot teams. *IEEE Transactions on Robotics and Automation*, 14(6):1–15, 1998.

D.P. Bertsekas. *Dynamic Programming and Optimal Control.* Athena Scientific, Belmont, Massachusetts, 1995.

C. Boutilier. Sequential optimality and coordination in multi-agent systems. In *Proceedings of the Sixteenth International Joint Conference on Artificial Intelligence (IJCAI)*, 1999.

S. J. Bradtke and M. O. Duff. Reinforcement learning methods for continuous-time markov decision problems. In *Advances in Neural Information Processing Systems.* MIT Press, 1995.

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

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

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

J. Hu and M. Wellman. Multiagent reinforcement learning: Theoretical framework and an algorithm. In *Fifteenth International Conference on Machine Learning*, pages 242–250, 1998.

C.M. Klein and J. Kim. Agv dispatching. *International Journal of Production Research*, 34(1):95–110, 1996.

J. Lee. Composite dispatching rules for multiple-vehicle agv systems. *SIMULATION*, 66 (2):121–130, 1996.

Michael Littman. Markov games as a framework for multi-agent reinforcement learning. In *Proceedings of the Eleventh International Conference on Machine Learning*, pages 157–163, 1994.

S. Mahadevan. Average reward reinforcement learning: foundations, algorithms, and empirical results. *Machine Learning*, 22:159–196, 1996.

S. Mahadevan, N. Marchalleck, T. Das, and A. Gosavi. Self-improving factory simulation using continuous-time average reward reinforcement learning. In *Proceedings of the Fourtheenth International Conference on Machine Learning*, 1997.

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

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

R.E. Parr. *Hierarchical Control and Learning for Markov Decision Processes.* PhD Thesis, University of California, Berkeley, 1998.

M. L. Puterman. *Markov Decision Processes.* Wiley Interscience, 1994.

A. Schwartz. A reinforcement learning method for maximizing undiscounted rewards. In *Proceedings of the Tenth International Conference on Machine Learning*, 1993.

P. Stone and M. Veloso. Team-partitioned, opaque-transition reinforcement learning. *Third International Conference on Autonomous Agents*, pages 86–91, 1999.

T. Sugawara and V. Lesser. Learning to improve coordinated actions in cooperative distributed problem-solving environments. *Machine Learning*, 33:129–154, 1998.

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

R. C. Sutton and A. G. Barto. *An Introduction to Reinforcement Learning.* MIT Press, 1998.

R.S. Sutton. Learning to predict by the methods of temporal differences. *Machine Learning*, 3:9–44, 1988.

P. Tadepalli and D. Ok. Auto-exploratory average reward reinforcement learning. In *Proceedings of the Thirteenth AAAI*, pages 881–887, 1996a.

P. Tadepalli and D. Ok. Scaling up average reward reinforcement learning by approximating the domain models and the value function. In *Proceedings of International Machine Learning Conference*, 1996b.

Ming Tan. Multi-agent reinforcement learning: Independent vs. cooperative agents. In *Proceedings of the Tenth International Conference on Machine Learning*, pages 330–337, 1993.

G. Wang and S. Mahadevan. Hierarchical optimization of policy-coupled semi-markov decision processes. In *Proceedings of the Sixteenth International Conference on Machine Learning*, 1999.

C.J. Watkins. *Learning from Delayed Rewards.* PhD thesis, Kings College, Cambridge, England, May 1989.

G. Weiss. *Multiagent Systems: A Modern Approach to Distributed Artificial Intelligence.* MIT Press, 1999.

P. Xuan, V. Lesser, and S. Zilberstein. Communication decisions in multi-agent cooperation: Model and experiments. In *Proceedings of the Fifth International Conference on Autonomous Agents*, pages 616–623, 2001.