Hierarchical Average Reward Reinforcement Learning

Mohammad Ghavamzadeh
Sridhar Mahadevan

CMPSCI Technical Report 03-19

June 25, 2003

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

Abstract

Hierarchical reinforcement learning (HRL) is the study of mechanisms for exploiting the structure of tasks in order to learn more quickly. By decomposing tasks into subtasks, fully or partially specified subtask solutions can be reused in solving tasks at higher levels of abstraction. The theory of semi-Markov decision processes provides a theoretical basis for HRL. Several variant representational schemes based on SMDP models have been studied in previous work, all of which are based on the *discrete-time discounted SMDP* model. In this approach, policies are learned that maximize the long-term discounted sum of rewards.

In this paper we investigate two formulations of HRL based on the *average-reward* SMDP model, both for discrete time and continuous time. In the average-reward model, policies are sought that maximize the expected reward per step. The two formulations correspond to two different notions of optimality that have been explored in previous work on HRL: *hierarchical optimality*, which corresponds to the set of optimal policies in the space defined by a task hierarchy, and a weaker local model called *recursive optimality*. What distinguishes the two models in the average reward framework is the optimization of subtasks. In the recursively optimal framework, subtasks are treated as continuing, and solved by finding gain optimal policies given the policies of their children. In the hierarchical optimality framework, the aim is to find a globally gain optimal policy within the space of policies defined by the hierarchical decomposition. We present algorithms that learn to find recursively and hierarchically optimal policies under discrete-time and continuous-time average reward SMDP models.

We use four experimental testbeds to study the empirical performance of our proposed algorithms. The first two domains are relatively simple, and include a small autonomous guided vehicle (AGV) scheduling problem and a modified version of the well-known Taxi problem. The other two domains are larger real-world single-agent and multiagent AGV scheduling problems. We model these AGV scheduling tasks using both discrete-time and continuous-time models and compare the performance of our proposed algorithms with each other, as well as with other HRL methods and to standard Q-learning. In the large AGV domain, we also show that our proposed algorithms outperform widely used industrial heuristics, such as *"first come first serve"*, *"highest queue first"* and *"nearest station first"*.

Keywords: Hierarchical Reinforcement Learning, Semi-Markov Decision Processes, Average Reward, Hierarchical and Recursive Optimality.

1. Introduction

Reinforcement learning (RL) (Bertsekas and Tsitsiklis, 1996, Sutton and Barto, 1998) is the study of algorithms that enable agents embedded in stochastic environments to 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 sample-based approaches use dynamic programming (DP) (Bertsekas, 1995, Puterman, 1994), 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.

Broadly speaking, the two main approaches to RL are to search the policy space directly using the gradient of the parametric representation of the policy with respect to some performance metric – the so-called *policy gradient* formulation (Marbach, 1998, Baxter and Bartlett, 2001) – or, instead, to 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). In this paper, we focus on the value function-based approach, although we have recently begun to investigate hierarchical policy gradient RL algorithms as well (Ghavamzadeh and Mahadevan, 2003).

The asymptotic convergence of value function-based RL algorithms, such as Q-learning (Watkins, 1989) or TD(λ) (Sutton, 1988), is only assured in restricted cases, typically when the values are represented explicitly for each state. Often, real-world problems require using *function approximators* for which convergence is not guaranteed in general. In such cases, convergence is guaranteed only if the value function is approximated using a *linear* superposition of basis feature values, and samples are generated using an *on policy* distribution. However, even if asymptotic convergence was theoretically guaranteed by adhering to these restrictions, in practice these algorithms can take hundreds of thousands of epochs to converge.

The central focus of this paper is to present new algorithms for reinforcement learning, applicable to discrete-time and continuous-time continuing tasks, using which convergence occurs much more rapidly than with traditional Q-learning (Watkins, 1989). 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 subtasks (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).

Fortunately, a well-known statistical model is available to treat variable length actions: the *semi-Markov decision process* (SMDP) model (Howard, 1971, Puterman, 1994). 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. Early work in RL on the SMDP model studied extensions of algorithms such as Q-learning (Bradtke and Duff, 1995, Mahadevan et al., 1997b). This early work on SMDP models was then expanded to include hierarchical task models over fully or partially specified lower level subtasks. The options model (Sutton et al., 1999) in its simplest form studied how to learn policies given fully specified policies for executing subtasks. The hierarchical abstract machines (HAMs) formulation (Parr, 1998) showed how hierarchical learning could be achieved even when the policies for lower-level subtasks were only partially specified. Lastly, the MAXQ framework (Dietterich, 2000) provided a fully comprehensive framework for hierarchical learning where instead of specifying policies for subtasks, the learner is given *pseudo*-reward functions. While a full comparison of these variant approaches is beyond the scope of this paper, what these treatments have in common is that they are all based on the discrete-time discounted reward SMDP framework.

The average-reward formulation has been shown to be more appropriate for a wide class of continuing tasks. A primary goal of continuing tasks, including manufacturing, scheduling, queuing and inventory control, is to find a *gain optimal policy* that maximizes (minimizes) the long-run average reward (cost) over time. Although average reward RL has been extensively studied, using both the discrete-time MDP model (Schwartz, 1993, Mahadevan, 1996, Tadepalli and Ok, 1996a, Marbach, 1998, Van-Roy, 1998) as well as the continuous-time SMDP model (Mahadevan et al., 1997b, Wang and Mahadevan, 1999), prior work has been limited to *flat* policy representations.

In this paper, we extend previous work on hierarchical reinforcement learning to the average reward SMDP framework and present discrete-time and continuous-time hierarchical average reward RL algorithms corresponding to two notions of optimality in HRL: *hierarchical optimality* and *recursive optimality*. A secondary contribution of this paper is to illustrate how HRL can be applied to more interesting (and practical) domains than has been illustrated previously. In particular, we focus on autonomous guided vehicle (AGV) scheduling, although our approach easily generalizes to other problems, such as transfer line production control (Gershwin, 1994, Wang and Mahadevan, 1999). We use four experimental testbeds to study the empirical performance of our proposed algorithms. The first two domains are simple, a small autonomous guided vehicle (AGV) scheduling problem and a modified version of the Taxi problem (Dietterich, 2000). The other domains are much larger real-world single-agent and multiagent AGV scheduling problems. We model these AGV scheduling tasks using both discrete-time and continuous-time models and compare the performance of our proposed algorithms with each other, as well as with the MAXQ method (Dietterich, 2000) and to standard Q-learning. In the multiagent AGV domain, we also 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) describes a framework for hierarchical reinforcement learning which is used to develop the algorithms of this paper. In Section (3), we present two discrete-time and two continuous-time hierarchical average reward RL algorithms. Section (3.1) reviews average reward discrete-time SMDPs. In Section

(3.3), we present discrete-time and continuous-time hierarchically optimal average reward RL algorithms. In Section (3.4), we investigate different methods to formulate subtasks in a recursively optimal hierarchical average reward RL framework and present discrete-time and continuous-time recursively optimal hierarchical average reward RL algorithms. Section (4) provides a brief overview of the Automated Guided Vehicle (AGV) scheduling problem, which is used in the experimental study presented in this paper. Section (5) presents experimental results of using the proposed algorithms in a simple AGV scheduling problem, a modified version of the Taxi problem and large real-world single-agent and multiagent AGV scheduling problems. Section (6) summarizes the paper and discusses some directions for future work. Finally, we list the notation used in this paper in Appendix A.

2. A Framework for Hierarchical Reinforcement Learning

In this section, we introduce a general hierarchical reinforcement learning framework for simultaneous learning at multiple levels of the hierarchy. Our treatment builds upon the existing approaches, including the MAXQ value function decomposition (Dietterich, 2000), hierarchies of abstract machines (HAMs) (Parr, 1998), and the options model (Sutton et al., 1999). We describe the common principles underlying these variant formulations below, ignoring some of the subtle differences between the frameworks. In the next section, we will extend this framework to average reward model and present our hierarchical average reward reinforcement learning algorithms.

2.1 Motivating Example

Hierarchical reinforcement learning methods provide a general framework for scaling reinforcement learning to problems with large state spaces by using the task structure to restrict the space of policies. In these methods, the designer of the system uses his/her domain knowledge to recursively decompose the overall task into a collection of subtasks that he/she believes are important for solving the problem.

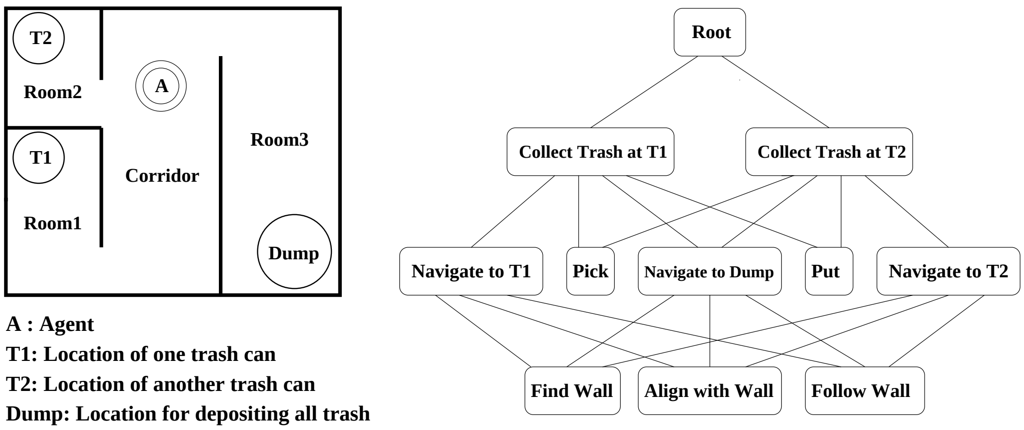

Figure 1: A (simulated) robot trash collection task and its associated task graph.

Let us illustrate the idea using a simple search task shown in Figure (1). Consider the case where in an office (rooms and connecting corridors) type environment, a robot is assigned the task of picking up trash from trash cans ($T1$ and $T2$) over an extended area and accumulating it into one centralized trash bin ($Dump$), from where it might be sent for recycling or disposed. The main subtasks in this problem are *root* (the whole trash collection task), *collect trash at* $T1$ and $T2$, *navigate to* $T1$, $T2$ and $Dump$. Each of these subtasks is defined by a set of termination states, and terminates when reaches one of its termination states. After defining subtasks, we must indicate for each subtask, which other subtasks or primitive actions it should employ to reach its goal. For example, *navigate to* $T1$, $T2$ and $Dump$ use three primitive actions *find wall*, *align with wall* and *follow wall*. *Collect trash at* $T1$ uses two subtasks *navigate to* $T1$ and $Dump$, plus two primitive actions Put and $Pick$, and so on.

All of this information can be summarized by a directed acyclic graph called the *task graph*. The task graph for the trash collection problem is shown in Figure (1). A key challenge for any HRL method is how to support temporal abstraction, state abstraction and subtask sharing.

- **Temporal Abstraction:** The process of navigating to $T1$ is a temporally extended action that can take different lengths of time to complete depending on the distance to $T1$.
- **State Abstraction:** While the agent is moving toward the $Dump$, the status of trash cans $T1$ and $T2$ are 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 *navigate to Dump* subtask.
- **Subtask Sharing:** If the system could learn how to solve the *navigate to Dump* subtask once, then the solution could be shared by both *collect trash at* $T1$ and $T2$ subtasks.

2.2 Temporal Abstraction using SMDPs

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 (Mahadevan et al., 1997a). We briefly explain the basic SMDP model here, leaving details of the average-reward formulation to later sections. Semi-Markov decision processes 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 actions. Thus, the time between transitions may be several time units and can depend on the transition that is made.

An SMDP is defined as a four tuple (S,A,P,R), where S is a finite set of states, A is the set of actions, $P : S \times \mathcal{N} \times S \times A \rightarrow [0,1]$ is a set of state and action dependent multi-step transition probabilities, and R is the reward function. $P(s', N|s, a)$ denotes the probability that action a will cause the system to transition from state s to state s' in N

time steps. This transition is at decision epochs only. Basically, the SMDP model represents snapshots of the system at decision points, whereas the so-called *natural process* describes the evolution of the system over all times.

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

Mathematically, a task hierarchy such as the one illustrated above can be modeled by decomposing the overall task MDP M, into a finite set of subtasks $\{M_0, M_1, \ldots, M_n\}$, where M_0 is the *root* task and solving it solves the entire MDP M.

Definition 1: Each *non-primitive* subtask i (i is not a primitive action) consists of five components $(S_i, I_i, T_i, A_i, R_i)$:

- S_i is the *state space* for subtask i. It is described by those state variables that are relevant to subtask i. The range of the state variables describing S_i might be a subset of their range in S (state abstraction).

- I_i is the *initiation set* for subtask i. Subtask i can be initiated only in states belong to I_i.

- T_i is the set of *terminal states* for subtask i. Subtask i terminates when it reaches a state in T_i. The policy for subtask i can only be executed if the current state s belongs to $(S_i - T_i)$.

- A_i is the *set of actions* that can be performed to achieve subtask i. These actions can either be primitive actions from A (the set of primitive actions for MDP M), or they can be other subtasks.

- R_i is the *reward structure* inside subtask i and could be different from the reward function of MDP M. Besides the reward of the overall task (MDP M), each subtask i can use additional rewards to guide its local learning (Ng et al., 1999). Additional rewards are only used inside each subtask and do not propagate to upper levels in the hierarchy. If the reward structure inside a subtask is different than the reward function of the overall task, we need to define two types of value functions for the subtask, *internal value functions* and *external value functions.* Internal value functions are defined based on both the local reward structure of the subtask and the reward of the overall task, and only used in learning the subtask. On the other hand, external value functions are defined only based on the reward function of the overall task and propagated to higher levels in the hierarchy to be used in learning the global policy.

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.

2.3 Policy Execution

If we have a policy for each subtask in this model, it gives us a policy for the overall task. This collection of policies is called a *hierarchical policy.*

Definition 2: A hierarchical policy μ is a set with a policy for each of the subtasks in the hierarchy: $\mu = \{\mu_0, \dots, \mu_n\}$.

The hierarchical policy is executed using a stack discipline, similar to ordinary programming languages. Each subtask policy takes a state and returns the name of a primitive action to execute or the name of a subtask to invoke. When a subtask is invoked, its name is pushed onto the *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* subtask 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 μ, we define a multi-step transition probability $P_i^\mu : S_i \times \mathcal{N} \times S_i \rightarrow [0,1]$ for each subtask i in the hierarchy, where $P_i^\mu(s', N|s)$ denotes the probability that action $\mu_i(s)$ will cause the system to transition from state s to state s' in N *primitive steps.* We also define a single-step transition probability function for each subtask i under hierarchical policy μ by marginalizing the multi-step transition probability function P_i^μ, as $F_i^\mu(s'|s) = \sum_{N=1}^{\infty} P_i^\mu(s', N|s)$. $F_i^\mu(s', n|s)$ denotes the n-step (or abstract) transition probability from state s to state s' under hierarchical policy μ, where n is the number of actions taken by subtask i, not the number of primitive actions taken in this transition. In this paper, we use the abstract transition probability F to model state transition at the subtask level and transition probability P to model state transition at the level of primitive actions.

Definition 3: Under a hierarchical policy μ, each subtask i can be modeled by an SMDP consists of components (S_i, A_i, P_i^μ, R_i).

2.4 Local versus Global Optimality

In the HRL framework, the designer imposes a hierarchy on the problem to incorporate prior knowledge and thereby reduces the size of the space that must be searched to find a good policy. However, this hierarchy constrains the space of possible policies so that it may not be possible to represent the optimal policy or its value function and hence make it impossible to learn the optimal policy. If we cannot learn the optimal policy, the next best target would be to learn the best policy that is consistent with the given hierarchy. Two notions of optimality have been explored in previous work on hierarchical reinforcement learning:

Definition 4: *Hierarchical optimality* is a global optimum consistent with the given hierarchy. In this form of optimality, the policy for each individual subtask is not necessarily optimal, but the policy for the entire hierarchy is optimal. The HAMQ HRL algorithm

(Parr, 1998) and the SMDP Q-learning algorithm for a fixed set of options (Sutton et al., 1999) both converge to a hierarchically optimal policy. More formally, a hierarchical optimal policy for MDP M is a hierarchical policy which has the best performance among all policies consistent with the given hierarchy.

Definition 5: *Recursive optimality* is a weaker but more flexible form of optimality which only guarantees that the policy of each subtask is optimal given the policies of its children. It is an important and flexible form of optimality because it permits each subtask to learn a locally optimal policy while ignoring the behavior of its ancestors in the hierarchy. This increases the opportunities for subtask sharing and state abstraction. The MAXQ-Q HRL algorithm (Dietterich, 2000) converges to a recursively optimal policy. More formally, a recursive optimal policy for MDP M with hierarchical decomposition $\{M_0, M_1, \dots, M_n\}$ is a hierarchical policy $\mu = \{\mu_0, \dots, \mu_n\}$ such that for each subtask M_i, the corresponding policy μ_i is optimal for the SMDP defined by the tuple $(S_i, A_i, P_i^{\mu}, R_i)$.

2.5 Value Function Definitions

For recursive optimality, the goal is to find a hierarchical policy $\mu = \{\mu_0, \dots, \mu_n\}$ such that for each subtask M_i in the hierarchy, the expected cumulative reward of executing policy μ_i and the policies of all descendants of M_i is maximized. In this case, the value function to be learned for subtask i under hierarchical policy μ must contain only the reward received during the execution of subtask i. We call this the *projected value function* and define it as follows:

Definition 6: The projected value function of hierarchical policy μ on subtask M_i, denoted $\hat{V}^{\mu}(i, s)$, is the expected cumulative reward of executing policy μ_i and the policies of all descendants of M_i starting in state $s \in S_i$ until M_i terminates.

The expected cumulative reward outside a subtask is not a part of its projected value function. It makes the projected value function of a subtask dependent only on itself and its descendants.

On the other hand, for hierarchical optimality, the goal is to find a hierarchical policy that maximizes the expected cumulative reward. In this case, the value function to be learned for subtask i under hierarchical policy μ must contain the reward received during the execution of subtask i and the reward after subtask i terminates. We call this the *hierarchical value function*. The hierarchical value function of a subtask includes the expected reward outside the subtask and therefore depends on the subtask and all its ancestors up to the root of the hierarchy. In the case of hierarchical optimality, we need to consider the contents of the *Task Stack* as an additional part of the state space of the problem, since a subtask might be shared by multiple parents.

Definition 7: Ω is the space of possible values of the *Task Stack* for hierarchy $\mathcal{H}$.

Let us define a joint state space X as the cross product of *Task Stack* values Ω and the states S in hierarchy $\mathcal{H}$. We define the hierarchical value function using state space X as:

Definition 8: A hierarchical value function for subtask M_i in state $x = (\omega, s)$ and under hierarchical policy μ, denoted $V^\mu(i, x)$, is the expected cumulative reward of following the hierarchical policy μ starting in state $s \in S_i$ and *Task Stack* ω.

The current subtask i is a part of the *Task Stack* ω and as a result is a part of state x. So we can exclude it from the hierarchical value function notation and write $V^\mu(i, x)$ as $V^\mu(x)$. However, we keep the current subtask i as a part of the hierarchical value function notation to simplify the notation in the following section.

3. Hierarchical Average Reward Reinforcement Learning

Given the above fundamental principles of HRL, we can now proceed to describe our hierarchical average reward formulation. We begin with a review of average reward discrete-time SMDPs.

3.1 Discrete-time Average Reward SMDPs

The theory of infinite-horizon SMDPs with the average reward criterion is more complex than that for discounted models (Howard, 1971, Puterman, 1994). To simplify exposition we assume that for every stationary policy, the embedded Markov chain has a unichain transition probability matrix.[1] Under this assumption, the expected average reward of every stationary policy does not vary with the initial state.

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

$$V_N^\mu(s) = E_s^\mu \left\{ \sum_{k=0}^{N-1} r(s_k, a_k) \right\}$$

The average expected reward or gain $g^\mu(s)$ for a policy μ in state s can be defined by taking the ratio of the expected total reward and the number of decision epochs. The gain $g^\mu(s)$ of a policy μ can be expressed as

$$g^\mu(s) = \liminf_{N \to \infty} \frac{E_s^\mu \{ \sum_{k=0}^{N-1} r(s_k, a_k) \}}{N}$$

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

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

1. The underlying Markov chain for every stationary policy has a single recurrent class, and a (possibly empty) set of transient states.

The expected average adjusted sum of rewards H^μ for policy μ is defined as

$$H^\mu(s) = E_s^\mu\{\sum_{k=0}^{\infty}[r(s_k, a_k) - g^\mu(s_k)]\} = E_s^\mu\{\sum_{k=0}^{\infty}[r(s_k, a_k) - g^\mu]\}$$

The Bellman equation for the average adjusted value function H^μ can be written as

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

The average adjusted action value function $L^\mu(s, a)$ represents the average adjusted value of doing action a in state s once, and then following policy μ subsequently is defined as

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

3.2 Assumptions

In this paper, we consider *continuing* HRL problems for which the following assumptions hold.

Assumption 1 (Continuing Root Task) The *root* of the hierarchy is a *continuing* task, i.e., the root task goes on continually without terminating.

Assumption 2 (Root Task Recurrence) There exists a state $s_0^* \in S_0$ such that, for every hierarchical policy μ and for every state $s \in S_0$, we have[2]

$$\sum_{n=1}^{|S_0|} F_0^\mu(s_0^*, n|s) > 0$$

where n is the number of steps at the level of *root* task, not the number of primitive actions as defined in Section (2.3).

Assumption (2) is equivalent to assuming that the underlying Markov chain for every policy of the *root* task has a single recurrent class and the state s_0^* is a recurrent state.[3] Under this assumption, the balance equations for policy μ

$$\begin{aligned} \sum_{s=1}^{|S_0|} F_0^\mu(s'|s)\pi_0^\mu(s) &= \pi_0^\mu(s'), \qquad\qquad s' = 1, \dots, |S_0| - 1 \\ \sum_{s=1}^{|S_0|} \pi_0^\mu(s) &= 1 \end{aligned} \tag{1}$$

have a unique solution $\pi_0^\mu = (\pi_0^\mu(1), \dots, \pi_0^\mu(|S_0|))$. We refer to π_0^μ as the steady state probability vector of the Markov chain with transition probability $F_0^\mu(s'|s)$ and to $\pi_0^\mu(s)$ as the steady state probability of being in state s.

2. Notice that the *root* task is represented as subtask M_0 in the HRL framework described in Section (2).
3. This assumption can be relaxed by assuming that the MDP corresponding to the *root* task is *unichain*.

If assumption (2) holds, the gain g^μ is well defined for every hierarchical policy μ and does not depend on the initial state. We have the following relation:

$$g^\mu = \sum_{s \in S_0} \pi_0^\mu(s) r(s, \mu_0(s))$$

When assumption (2) holds, we are interested in finding a hierarchical control policy μ^* which maximizes the gain, i.e.,

$$g^{\mu^*} \geq g^\mu, \qquad \text{for all } \mu \tag{2}$$

We refer to a hierarchical policy μ^* which satisfies condition (2) as a *gain optimal policy*, and to g^{μ^*} as the *optimal average reward* or the *optimal gain.*

However, since the policy learned for the *root* task involves the policies of its children, the type of optimality achieved at *root* depends on how we formulate subtasks in the hierarchy. We already addressed two notions of optimality: *hierarchical optimality* and *recursive optimality.* In Section (3.3), we introduce an algorithm to find a *hierarchically gain optimal policy* (a hierarchical policy that has the maximum gain among all hierarchical policies) (Ghavamzadeh and Mahadevan, 2002). In Section (3.4), we investigate different approaches for finding a *recursively gain optimal policy* (a hierarchical policy in which the policy at each node has the maximum gain given the policies of its children) and introduce a recursively gain optimal average reward HRL algorithm.

3.3 Hierarchically Gain Optimal Average Reward RL Algorithm

In this section, we consider problems for which assumptions (1) and (2) (*Continuing Root Task*) and (*Root Task Recurrence*) hold, i.e., the average reward for *root* (overall problem) is well defined for every hierarchical policy and does not vary with initial state. We use the hierarchical RL framework described in Section (2). Since we are interested in finding the hierarchical optimal policy, we include the contents of the *Task Stack* as a part of the state space of the problem. We also replace value function and action-value function with average adjusted value function and average adjusted action-value function in the hierarchical model of Section (2).

The hierarchical average adjusted value function H for hierarchical policy μ and subtask i, denoted $H^\mu(i,x)$, is the average adjusted sum of rewards earned of following policy μ starting in state $x = (\omega, s)$ until i terminates plus the expected average adjusted reward outside subtask i:

$$H^\mu(i,x) = \lim_{N \to \infty} E_x^\mu \{ \sum_{k=0}^{N-1} (r(x_k, a_k) - g^\mu) \} \tag{3}$$

where g^μ is the gain of the *root* task and we call it *global gain* of the hierarchical policy μ.

Now let us suppose that the first action chosen by μ is invoked and executed for a number of primitive steps N_1 and terminates in state $x_1 = (\omega, s_1)$ according to $P_i^\mu(x_1, N_1 | x, \mu_i(x))$ and after that subtask i itself executes for n_2 steps at the level of subtask i (n_2 is the number of actions taken by subtask i, not the number of primitive actions) and terminates in state $x_2 = (\omega, s_2)$ according to abstract transition probability $F_i^\mu(x_2, n_2|x_1)$. We can write Equation (3) in the form of a Bellman equation:

$$\begin{aligned} H^\mu(i,x) &= r(x,\mu_i(x)) - g^\mu y_i(x,\mu_i(x)) + \sum_{N_1, s_1 \in S_i} P_i^\mu(x_1, N_1|x,\mu_i(x))\hat{H}^\mu(i,x_1) \\ &+ \sum_{s_1 \in S_i} F_i^\mu(x_1|x,\mu_i(x)) \sum_{n_2, s_2 \in S_i} F_i^\mu(x_2,n_2|x_1) H^\mu(Parent(i),(\omega \nearrow i, s_2)) \end{aligned} \tag{4}$$

where $\hat{H}^\mu(i,.)$ is the projected average adjusted value function of hierarchical policy μ on subtask i, and $\omega \nearrow i$ is the content of the *Task Stack* after popping subtask i off. Notice that $\hat{H}$ does not contain the reward outside the current subtask and should be distinguished with the hierarchical average adjusted value function H, which includes the sum of rewards outside the current subtask.

Since $r(x,\mu_i(x))$ is the expected total reward between two decision epochs of subtask i, given that the system occupies state x at the first decision epoch and decision maker chooses action $\mu_i(x)$, we have

$$r(x,\mu_i(x)) = \hat{V}^\mu_{y_i(x,\mu_i(x))}(\mu_i(x),(\mu_i(x) \searrow \omega, s)) = \hat{H}^\mu(\mu_i(x),(\mu_i(x) \searrow \omega, s)) + g^\mu y_i(x,\mu_i(x))$$

where $\mu_i(x) \searrow \omega$ is the content of the *Task Stack* after pushing subtask $\mu_i(x)$ onto it. By replacing $r(x,\mu_i(x))$ from the above expression, Equation (4) can be written as

$$\begin{aligned} H^\mu(i,x) &= \hat{H}^\mu(\mu_i(x),(\mu_i(x) \searrow \omega, s)) + \sum_{N_1, s_1 \in S_i} P_i^\mu(x_1, N_1|x,\mu_i(x))\hat{H}^\mu(i,x_1) \\ &+ \sum_{s_1 \in S_i} F_i^\mu(x_1|x,\mu_i(x)) \sum_{n_2, s_2 \in S_i} F_i^\mu(x_2,n_2|x_1) H^\mu(Parent(i),(\omega \nearrow i, s_2)) \end{aligned} \tag{5}$$

We can re-state Equation (5) for hierarchical average adjusted action-value function as

$$\begin{aligned} L^\mu(i,x,a) &= \hat{H}^\mu(a,(a \searrow \omega, s)) + \sum_{N_1, s_1 \in S_i} P_i^\mu(x_1, N_1|x,a)\hat{H}^\mu(i,x_1) \\ &+ \sum_{s_1 \in S_i} F_i^\mu(x_1|x,a) \sum_{n_2, s_2 \in S_i} F_i^\mu(x_2,n_2|x_1) L^\mu(Parent(i),(\omega \nearrow i, s_2),\mu_{parent(i)}(\omega \nearrow i, s_2)) \end{aligned}$$

and we can re-express the definition for $\hat{H}$ as

$$\hat{H}^\mu(i,s) = \begin{cases} \hat{L}^\mu(i,s,\mu_i(s)) & \text{if } i \text{ is a composite action} \\ \sum_{s'} P(s'|s,i)[r(s'|s,i) - g^\mu] & \text{if } i \text{ is a primitive action} \end{cases} \tag{6}$$

where $\hat{L}$ is the projected average adjusted action-value function.

The above formulas can be used to obtain update equations for $\hat{H}$, $\hat{L}$ and L in this framework. Pseudo-code for the resulting algorithm is shown in Algorithm (1). After running for appropriate time, this algorithm should generate a gain-optimal policy that maximizes the average reward for the overall task. In this algorithm, primitive subtasks update only their projected average adjusted value functions $\hat{H}$ (line 5), while non-primitive subtasks update both their projected average adjusted action-value functions $\hat{L}$ and hierarchical average adjusted action-value functions L (lines 17 and 18). We store only one global gain g and update it after each non-random primitive action (line 7). In update formulas at lines 17 and 18, the projected average adjusted value function $\hat{H}(a,(a \searrow \omega, s))$ is the reward of executing action a in state (ω, s) under subtask i and is recursively calculated by subtask a and its descendants using Equation (6).

Algorithm 1 The discrete-time hierarchically gain optimal average reward RL algorithm.

1: **Function HO-AR(Task i, State $x = (\omega, s)$)**
2: let $Seq = \{\}$ be the sequence of *states visited* while executing i
3: **if** i is a primitive action **then**
4: execute action i in state x, observe state $x' = (\omega, s')$ and reward $r(s'|s,i)$
5: $\hat{H}_{t+1}(i,x) \longleftarrow (1-\alpha_t)\hat{H}_t(i,x) + \alpha_t[r(s'|s,i) - g_t]$
6: **if** i and all its ancestors are non-random actions **then**
7: update the global average reward $g_{t+1} = \frac{r_{t+1}}{n_{t+1}} = \frac{r_t + r(s'|s,i)}{n_t + 1}$
8: **end if**
9: push *state* $x_1 = (\omega \nearrow i, s)$ into the beginning of *Seq*
10: **else**
11: **while** i has not terminated **do**
12: choose action a according to the current exploration policy $\mu_i(x)$
13: let *ChildSeq*=HO-AR(a, $(a \searrow \omega, s)$), where *ChildSeq* is the sequence of states visited while executing action a
14: observe result state $x' = (\omega, s')$
15: let $a^* = argmax_{a' \in A_i(s')} L_t(i, x', a')$
16: **for** each $x = (\omega, s)$ in *ChildSeq* from the beginning **do**
17: $\hat{L}_{t+1}(i,x,a) \leftarrow (1-\alpha_t)\hat{L}_t(i,x,a) + \alpha_t[\hat{H}_t(a,(a \searrow \omega, s)) + \hat{L}_t(i,x',a^*)]$
18: $L_{t+1}(i,x,a) \leftarrow (1-\alpha_t)L_t(i,x,a) + \alpha_t[\hat{H}_t(a,(a \searrow \omega, s)) + L_t(i,x',a^*)]$
19: replace state $x = (\omega, s)$ with $x_1 = (\omega \nearrow i, s)$ in the *ChildSeq*
20: **end for**
21: append *ChildSeq* onto the front of *Seq*
22: $x = x'$
23: **end while**
24: **end if**
25: **return** *Seq*
26: **end HO-AR**

This algorithm can be easily extended to continuous-time by changing the update formulas for $\hat{H}$ and g in lines 5 and 7 as

$$\hat{H}_{t+1}(i,x) \longleftarrow (1-\alpha_t)\hat{H}_t(i,x) + \alpha_t[k(s,i) + r(s'|s,i)\tau(s'|s,i) - g_t\tau(s'|s,i)]$$

$$g_{t+1} = \frac{r_{t+1}}{t_{t+1}} = \frac{r_t + k(s,i) + r(s'|s,i)\tau(s'|s,i)}{t_t + \tau(s'|s,i)}$$

where $\tau(s'|s,i)$ is the time elapsing between states s and s', $k(s,i)$ is the fixed reward of taking action i in state s and $r(s'|s,i)$ is the reward rate for the time that the natural process remains in state s' between decision epochs.

3.4 Recursively Gain Optimal Average Reward RL

In the previous section, we introduced hierarchically gain optimal average reward RL algorithms for both discrete and continuous time problems. In the proposed average reward algorithms, we define only a global gain for the entire hierarchy to guarantee global optimality for the overall task. The hierarchical policy has the highest gain among all policies consistent with the given hierarchy. However, there might exist a subtask where its policy must be locally suboptimal so that the overall policy becomes optimal.

Recursive optimality is a kind of local optimality in which the policy at each node is optimal given the policies of its children. The reason to seek recursive optimality rather than hierarchical optimality is that recursive optimality makes it possible to solve each subtask without reference to the context in which it is executed. This leaves open the question of what local optimality criterion should be used for each subtask except *root* in the recursive optimal average reward HRL setting.[4] One possibility is to simply optimize the total reward of every subtask in the hierarchy except *root*. Another possibility, investigated in (Ghavamzadeh and Mahadevan, 2001), is to treat subtasks as average reward problems that maximize their gain given the policies of their children. We will describe this approach in detail later in this section. Finally the third approach, pursued in (Seri and Tadepalli, 2002), is to optimize subtasks using their expected total relativized reward with respect to the gain of the overall task (gain of the *root* task). Seri and Tadepalli (Seri and Tadepalli, 2002) introduce a model-based algorithm called *Hierarchical H-Learning* (*HH-Learning*). For every subtask, this algorithm learns the action model and maximizes the expected total average adjusted reward with respect to the gain of the overall task at each state. In their approach, the projected average adjusted value functions with respect to the gain of the overall task satisfy the following Bellman equations:

$$\hat{H}^\mu(i,s) = \begin{cases} r(s'|s,i) - g^\mu \tau(s'|s,i) & \text{if } i \text{ is a primitive action} \\ 0 & \text{if } s \text{ is a goal state for subtask } i \\ max_{a \in A_i(s)}[\hat{H}^\mu(a,s) + \sum_{N,s' \in S_i} P_i^\mu(s',N|s,a)\hat{H}^\mu(i,s')] & \text{otherwise} \end{cases} \quad (7)$$

The first term of the last part of Equation (7), $\hat{H}^\mu(a,s)$, denotes the expected total average adjusted reward during the execution of subtask a, and the second term denotes the expected total average adjusted reward from then on until the completion of subtask i. Since the expected average adjusted reward after subtask i execution is not a component of the average adjusted value function, this approach does not necessarily allow for hierarchical gain optimality (as will be shown in experiments of Section (5)). Moreover, the policy learned for each subtask using this approach is not context free, because each node maximizes its relativized reward with respect to the gain of the overall policy. However, this method finds the hierarchically gain optimal policy when the *result distribution invariance* condition holds (Seri and Tadepalli, 2002).

On the other hand, the approach in which subtasks are treated as average reward problems (Ghavamzadeh and Mahadevan, 2001) might fail to find the hierarchical gain optimal

4. Like the previous section, we consider those problems for which assumptions (1) and (2) (*Continuing Root Task*) and (*Root Task Recurrence*) hold. Thus, for *root*, the goal is to maximize its gain, given the policies for its descendants.

policy as shown in (Seri and Tadepalli, 2002). However the policy learned at each node using this method maximizes the gain of the node given the policies of its children. Therefore, it is independent of the context in which it is executed and could be reused in other hierarchies. Now we first describe this approach in detail and then introduce an algorithm for finding recursively optimal average reward policies.

In HRL methods, we typically assume that every time a subtask except *root* is called, it starts at one of its initial states and terminates at one of its terminal states after a finite number of steps. Therefore, we make the following assumption for every subtask i in the hierarchy except *root*. Under this assumption, each instantiation of a subtask can be considered as an episode and each subtask as an episodic problem.

Assumption 3 (Subtask Termination) There exists a distinguished state $s_i^* \in S_i$ such that, for all hierarchical stationary policies μ and every terminal state s_i^T, we have

$$F_i^{\mu}(s_i^*|s_i^T, \mu_i(s_i^T)) = 1 \quad \text{and} \quad r_i(s_i^*|s_i^T, \mu_i(s_i^T)) = 0$$

and, for all non-terminal states $s \in S_i$ of the subtask, we have

$$F_i^{\mu}(s_i^*|s, \mu_i(s)) = 0$$

and finally, for all states $s \in S_i$, we have

$$F_i^{\mu}(s_i^*, n|s) > 0$$

where $n = |S_i|$ is the number of states in the state space of the subtask.

Although subtasks are episodic problems, when the overall task is continuing, they are executed an infinite number of times and therefore can be modeled as continuing problems using the model described in Figure (2). In this model, each subtask i terminates at one of its terminal states $s_i^T \in T_i$. All terminal states transit with probability one and reward zero to a distinguished state s_i^*. Finally, the distinguished state transits with reward zero to one of the initial states ($\in I_i$) of the subtask. It is important for the validity of this model to fix the value of the distinguished state equal to zero.

Under this model, for every hierarchical policy μ, each subtask i in the hierarchy (except *root*) can be modeled as a Markov chain with transition probabilities

$$F_{i,\bar{\pi}}^{\mu}(s'|s, \mu_i(s)) = \begin{cases} F_i^{\mu}(s'|s, \mu_i(s)) & s \neq s_i^* \\ \bar{\pi}_i(s') & s = s_i^* \end{cases} \tag{8}$$

and rewards

$$r_{i,\bar{\pi}}(s'|s, \mu_i(s)) = r_i(s'|s, \mu_i(s))$$

where $\bar{\pi}_i$ is a probability distribution on initial states of subtask i.

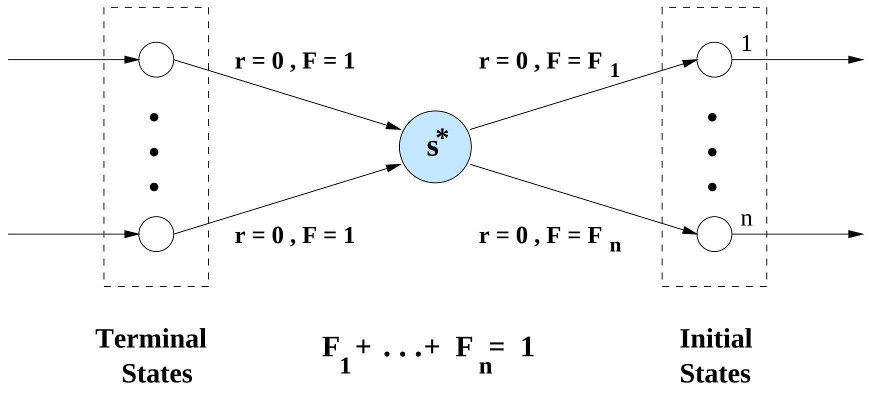

Figure 2: This figure shows how each subtask in the hierarchy except *root* can be modeled as a continuing task. In this figure, F and r are transition probability and reward.

Let $F^{\mu}_{i,\bar{\pi}}$ be the transition matrix with entries $F^{\mu}_{i,\bar{\pi}}(s'|s,\mu_i(s))$ and let $\mathcal{F}_{i,\bar{\pi}}$ be the set of all such transition matrices. We have the following result for all subtasks in the hierarchy except *root*.

Theorem 1 Let assumption (3) (*Subtask Termination*) hold. Then, for every $F^{\mu}_{i,\bar{\pi}} \in \mathcal{F}^{\mu}_{i,\bar{\pi}}$ and every state $s \in S_i$, we have[5]

$$\sum_{n=1}^{|S_i|} F^{\mu}_{i,\bar{\pi}}(s_i^*, n|s) > 0$$

Theorem (1) is equivalent to assuming that the underlying Markov chain for every hierarchical policy μ of any subtask i in the hierarchy has a single recurrent class and state s_i^* is its recurrent state. Under this assumption, for every subtask i in the hierarchy, the balance equations for every hierarchical policy μ have a unique solution $\pi^{\mu}_{i,\bar{\pi}}$ and the average reward $g^{\mu}_{i,\bar{\pi}}$ is well defined and does not depend on the initial state. Using this model, we define the average reward of subtask i under the hierarchical policy μ as:

$$g^{\mu}_{i,\bar{\pi}} = \sum_{s \in S_i} \pi^{\mu}_{i,\bar{\pi}}(s) r_{i,\bar{\pi}}(s'|s,\mu_i(s))$$

where $\pi^{\mu}_{i,\bar{\pi}}(s)$ is the steady state probability of being in state s under hierarchical policy μ.

In the next section, we illustrate the recursively optimal average reward algorithm using the above formulation. We consider problems for which assumptions (1), (2) and (3) (*Continuing Root Task*), (*Root Task Recurrence*) and (*Subtask Termination*) hold and every subtask in the hierarchy except *root* is modeled as an average reward problem using the model in Figure (2) and Equation (8), i.e., the average reward for every subtask in the hierarchy including *root* is well defined for every policy and does not vary with initial state.

5. This theorem is a restatement of the lemma 5 in page 34 of Peter Marbach's thesis (Marbach, 1998), which is applicable to the model described in Figure (2).

3.4.1 RECURSIVELY GAIN OPTIMAL AVERAGE REWARD RL ALGORITHM

In this section, we describe a discrete-time recursively optimal average reward HRL algorithm.[6] Since we are interested in finding a recursive optimal policy, we can exclude the contents of the *Task Stack* from the state space of the problem. We also use the hierarchical model of Section (2) with projected average adjusted value function and projected average adjusted action-value function.

We show how the overall projected average adjusted value function (the projected average adjusted value function of the *root* task) of a hierarchical policy is decomposed into a collection of projected average adjusted value functions of individual subtasks in this algorithm. The projected average adjusted value function of hierarchical policy μ on subtask i, denoted $\hat{H}^\mu(i,s)$, is the average adjusted (with respect to local gain g_i^μ) sum of rewards earned of following policy μ_i (and the policies of all descendants of subtask i) starting in state s until subtask i terminates. Now let us suppose that the first action chosen by μ is invoked and executed for a number of primitive steps N and terminates in state s' according to $P_i^\mu(s',N|s)$. We can write the projected average adjusted value function in the form of a Bellman equation as

$$\hat{H}^\mu(i,s) = r(s,\mu_i(s)) - g_i^\mu y_i(s,\mu_i(s)) + \sum_{N,s'\in S_i} P_i^\mu(s',N|s,\mu_i(s))\hat{H}^\mu(i,s') \tag{9}$$

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

$$r(s,\mu_i(s)) = \hat{V}^\mu_{y_i(s,\mu_i(s))}(\mu_i(s),s) = \hat{H}^\mu(\mu_i(s),s) + g^\mu_{\mu_i(s)} y_i(s,\mu_i(s))$$

By replacing $r(s,\mu_i(s))$ from the above expression, Equation (9) can be written as

$$\hat{H}^\mu(i,s) = \hat{H}^\mu(\mu_i(s),s) - (g_i^\mu - g^\mu_{\mu_i(s)}) y_i(s,\mu_i(s)) + \sum_{N,s'\in S_i} P_i^\mu(s',N|s,\mu_i(s))\hat{H}^\mu(i,s') \tag{10}$$

We can re-state Equation (10) for projected action-value function as follows:

$$\hat{L}^\mu(i,s,a) = \hat{H}^\mu(a,s) - (g_i^\mu - g_a^\mu) y_i(s,a) + \sum_{N,s'\in S_i} P_i^\mu(s',N|s,a)\hat{L}^\mu(i,s',\mu_i(s'))$$

In the above equation, the term

$$-(g_i^\mu - g_a^\mu) y_i(s,a) + \sum_{N,s'\in S_i} P_i^\mu(s',N|s,a)\hat{L}^\mu(i,s',\mu_i(s'))$$

denotes the average adjusted reward of completing subtask i after executing action a in state s. We call this term *completion function* and denote it by $C^\mu(i,s,a)$. With this definition, we can express the average adjusted action-value function $\hat{L}^\mu$ recursively as

$$\hat{L}^\mu(i,s,a) = \hat{H}^\mu(a,s) + C^\mu(i,s,a)$$

6. The continuous-time recursively optimal average reward HRL algorithm is similar and was introduced in (Ghavamzadeh and Mahadevan, 2001).

and we can re-express the definition for $\hat{H}$ as

$$\hat{H}^{\mu}(i,s) = \begin{cases} \hat{L}^{\mu}(i,s,\mu_i(s)) & \text{if } i \text{ is a composite action} \\ \sum_{s'} P(s'|s,i)[r(s'|s,i) - g_i^{\mu}] & \text{if } i \text{ is a primitive action} \end{cases} \tag{11}$$

The above formulas can be used to obtain update equations for $\hat{H}$ and C functions in discrete-time recursively optimal average reward model. Pseudo-code for this algorithm is shown in Algorithm (2). After running for appropriate time, this algorithm should generate a recursively gain-optimal policy that maximizes the gain of each subtask given the policies of its children. In this algorithm, a gain is defined for every subtask in the hierarchy (even primitive subtasks) and this gain is updated every time the subtask is non-randomly chosen. Primitive subtasks update their projected average adjusted value functions $\hat{H}$ (line 5) and gain (line 7), whereas non-primitive subtasks update their completion functions C (line 19) and gain (line 21). The projected average adjusted value function $\hat{H}$ for non-primitive subtasks used at lines 15 and 19 is recursively calculated using Equation (11).

4. The AGV Scheduling Task

In this section, we provide a brief overview of the AGV scheduling problem used in the experiments of this paper. 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 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 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 (Klein and Kim, 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 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. 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). 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. In the next section, we

Algorithm 2 The discrete-time recursively optimal average reward HRL algorithm.

Function RO-AR(Task i, State s)
let $Seq = \{\}$ be the sequence of (*state-visited*, *reward*) while executing i
if i is a primitive action **then**
  execute action i in state s, receive reward $r(s'|s,i)$ and observe state s'
  $\hat{H}_{t+1}(i,s) \longleftarrow (1-\alpha_t)\hat{H}_t(i,s) + \alpha_t(r(s'|s,i) - g(i))$
  if i and all its ancestors are non-random actions **then**
    update gain of subtask i $g_{t+1}(i) = \frac{r_{t+1}(i)}{n_{t+1}(i)} = \frac{r_t(i)+r(s'|s,i)}{n_t(i)+1}$
  end if
  push (*state* s, *reward* $r(s'|s,i)$) onto the front of Seq
else
  while i has not terminated **do**
    choose action a according to the current exploration policy $\mu_i(s)$
    let *ChildSeq*=RO-AR(a, s), where *ChildSeq* is the sequence of (*state-visited*, *reward*) while executing action a
    observe result state s'
    let $a^* = argmax_{a' \in A_i(s')}[C_t(i,s',a') + \hat{H}_t(a',s')]$
    let $N = 0$; $\rho = 0$;
    for each (s, r) in *ChildSeq* from the beginning **do**
      $N = N + 1$; $\rho = \rho + r$;
      $C_{t+1}(i,s,a) \longleftarrow (1-\alpha_t)C_t(i,s,a) + \alpha_t[C_t(i,s',a^*) + \hat{H}_t(a^*,s') - (g_t(i) - g_t(a))N]$
      if a and all its ancestors are non-random actions **then**
        update gain of subtask i $g_{t+1}(i) = \frac{r_{t+1}(i)}{n_{t+1}(i)} = \frac{r_t(i)+\rho}{n_t(i)+N}$
      end if
    end for
    append *ChildSeq* onto the front of Seq
    $s = s'$
  end while
end if
return Seq
end RO-AR

study both single-agent and more complex multiagent AGV scheduling tasks and apply the HRL algorithms described in previous section to these tasks.

5. Experimental Results

The goal of this section is to demonstrate the efficacy of the algorithms proposed in this paper. We show the type of the optimality that they converge to as well as their performance and speed comparing to other algorithms. We conduct four sets of experiments in this section. In Section (5.1), we apply five hierarchical RL algorithms to a simple discrete-time AGV scheduling problem. The advantage of using this simple domain is that it clearly demonstrates the difference between hierarchical and recursive optimal policies and differences between the optimality criteria achieved by these algorithms. In Section

(5.2), we use a modified version of the well-known Taxi problem (Dietterich, 2000). Since hierarchical and recursive optimal policies are not different in this domain, we just test two hierarchically optimal algorithms on this problem. Then we will turn to more complex multiagent and single-agent AGV tasks in Sections (5.3), and (5.4) to demonstrate the performance and speed of the proposed algorithms in complex domains. In Section (5.3), we use a complex continuous-time multiagent AGV scheduling problem and compare the performance and speed of our continuous-time recursively gain optimal average reward algorithm with continuous-time recursively optimal discounted reward HRL algorithm introduced in (Ghavamzadeh and Mahadevan, 2001) as well as three widely used industrial AGV scheduling heuristics. Finally in Section (5.4), we model a single-agent AGV scheduling task as discrete and continuous time problems and apply three hierarchical RL algorithms as well as a flat RL algorithm to both models.

5.1 Simple AGV Scheduling Problem

In this section, we apply the *discrete-time hierarchically gain optimal algorithm* (HO-AR) described in Section (3.3), the *discrete-time recursively gain optimal algorithm* (RO-AR) illustrated in Section (3.4.1), and *HH-Learning*, the algorithm proposed by Seri and Tadepalli (Seri and Tadepalli, 2002) to a simple AGV scheduling task. We also test MAXQ (*a recursively optimal discounted reward HRL algorithm*) (Dietterich, 2000) and a *hierarchically optimal discounted reward RL algorithm* (HO-DR) on this task. We derived HO-DR algorithm by combining the three parts value function decomposition introduced by Andre and Russell (Andre and Russell, 2002) and MAXQ hierarchical task decomposition (Dietterich, 2000). These experimental results clearly demonstrate the difference between hierarchical and recursive optimal policies and between the optimality criteria achieved by the above algorithms.

A small AGV domain is depicted in Figure (3). In this domain there are two machines ($M1$ and $M2$) that produce parts to be delivered to corresponding destination stations ($G1$ and $G2$). Since machines and destination stations are in two different rooms, the AGV has to pass one of the two doors ($D1$ and $D2$) every time it goes from one room to another. Part 1 is more important than part 2, therefore the AGV gets a reward of 20 when part 1 delivered to destination $G1$ and a reward of 1 when part 2 delivered to destination $G2$. The AGV receives a reward of -1 for all other actions. This task is deterministic and the state variables are AGV *location* and AGV *status* (empty, carry part 1 or carry part 2), which is total of $26 \times 3 = 78$ states. In all experiments, we use the task graph shown in Figure (3) and set the discount factor to 0.99 for discounted reward algorithms. We tried several discounting factors and 0.99 yielded the best performance. Using this task graph, hierarchical and recursive optimal policies are different. Since delivering part 1 has more reward than part 2, the hierarchically optimal policy is one in which the AGV always serves machine $M1$. In the recursively optimal policy, the AGV switches from serving machine $M1$ to serving machine $M2$ and vice versa. In this policy, the AGV goes to machine $M1$, picks up a part of type 1, goes to goal $G1$ via door $D1$, drops the part there, then passes through door $D2$, goes to machine $M2$, picks up a part of type 2, goes to goal $G2$ via door $D2$ and then switches again to machine $M1$ and so on so forth.

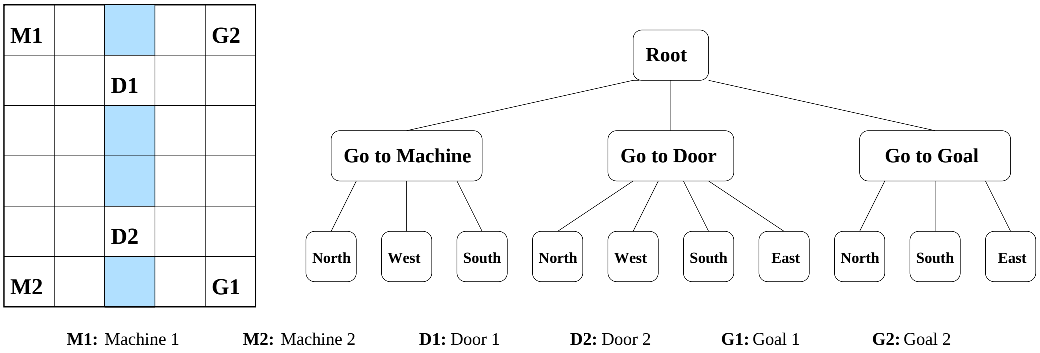

Figure 3: A simple AGV domain (left) and its associated task graph (right).

Among the algorithms we applied to this task, the hierarchically gain optimal average reward RL (HO-AR) and the hierarchically optimal discounted reward RL (HO-DR) algorithms find the hierarchically optimal policy, where the other algorithms only learn the recursively optimal policy. Figure (4) demonstrates the throughput of the system for the above algorithms. In this figure, the throughput of the system is the number of parts deposited at the destination stations weighted by their reward $((part1 \times 20) + (part2 \times 1))$ in 10000 time steps. Each experiment was conducted ten times and the results averaged.

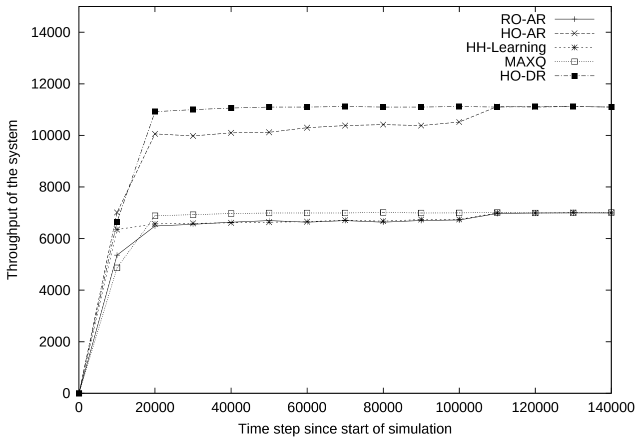

Figure 4: This plot shows that HO-DR and HO-AR algorithms learn the hierarchically optimal policy while MAXQ, RO-AR and HH-Learning only find the recursively optimal policy for the simple AGV task.

5.2 Modified Taxi Problem

In this section, we apply the *discrete-time hierarchically gain optimal average reward RL algorithm* (HO-AR) described in Section (3.3) and the *discounted reward hierarchically optimal RL algorithm* (HO-DR) to a modified version of the well-known Taxi problem (Dietterich, 2000).

Unlike the original Taxi problem (Dietterich, 2000), the version used in this paper is a continuing task. A 5-by-5 grid world inhabited by a taxi agent is shown in Figure (5). There are four stations, marked as B(lue), G(reen), R(ed) and Y(ellow). The taxi starts in a randomly chosen location and passengers randomly appear at these four stations. The passenger at each station wishes to be transported to one of the other three stations (also chosen randomly). The taxi must go to one of the passenger's locations, pick up the passenger, go to its destination location and drop off the passenger there. Then, passengers once again randomly appear in four stations and the task continues. Each navigation action with probability 0.7 causes the taxi to move one cell in the corresponding direction, and with probability 0.3 moves the agent in one of the other three directions, each with probability 0.1. The system performance is measured in terms of the number of passengers dropped off at their destinations per a fixed number of time steps. The state variables in this task are *taxi location*, *taxi status*, *status of each station* (whether there is a passenger waiting at that station or not), and *destination of passenger at each station*, which equals 512,000 states.

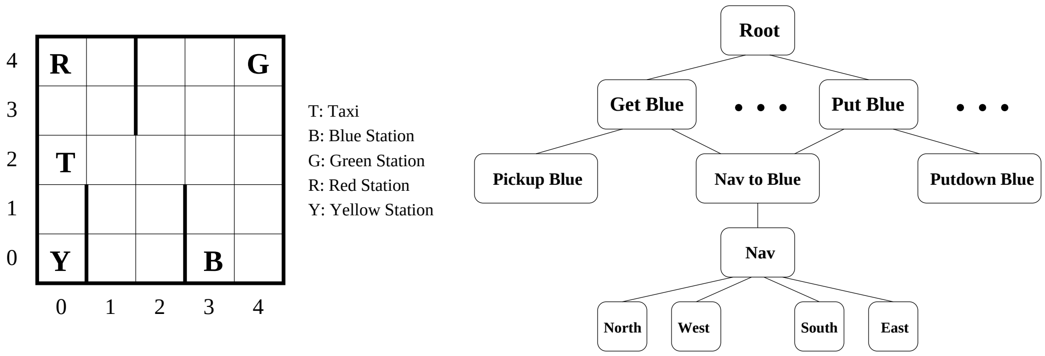

Figure 5: The Taxi Domain (left) and its associated task graph (right).

Figure (6) compares the proposed discrete-time hierarchically gain optimal algorithm (HO-AR) with the discrete-time hierarchically optimal discounted reward algorithm (HO-DR) showing the better performance of the average reward algorithm. Each experiment was conducted ten times and the results averaged. With the task graph depicted in Figure (5), the hierarchical and recursive optimal policies are not different for this problem. Hence, we did not test the recursively optimal algorithms on this domain.

5.3 Multiagent AGV Scheduling Problem (Continuous-Time Model)

In this section, we apply *continuous-time recursively optimal discounted reward HRL algorithm* introduced in (Ghavamzadeh and Mahadevan, 2001) and *continuous-time recursively gain optimal average reward algorithm* (RO-AR) illustrated in Section (3.4.1) to a complex multiagent AGV scheduling problem and compare their performance and speed with each other, as well as several well-known AGV scheduling heuristics.

We use a modified version of the above two continuous-time algorithms. This modification makes them well suited to multiagent problems. The most salient feature of this extension is that the top level (the level immediately below the root) of the hierarchy is

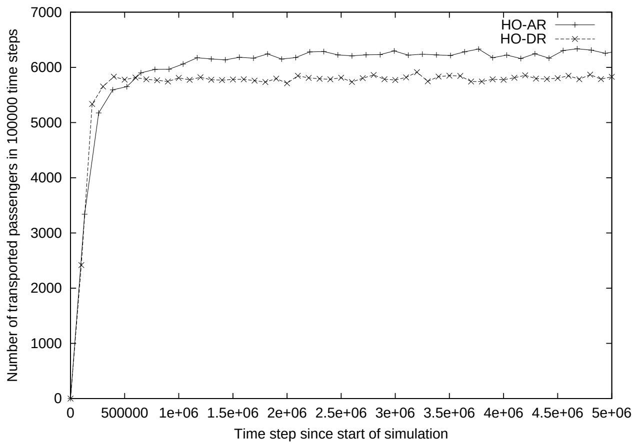

Figure 6: This plot shows that HO-AR algorithm works better than the discounted reward HO-DR (with discount factor 0.9) on the modified taxi problem. We tried several discounting factors and 0.9 yielded the best performance.

configured to store the completion function values C for joint abstract actions of all agents. The completion function for agent j, $C^j(i, s, a^1, ..., a^j, ..., a^n)$, is defined as the expected (discounted or undiscounted) reward of completion of subtask a^j by agent j in the context of the other agents performing subtasks $a^i, \forall i \neq j \in \{1, ..., n\}$, where s is the local state of agent j not the joint state. This method reduces the number of joint state-action values that need to be learned in a complex multiagent task, and provides a sufficiently good approximation of the true value functions (see (Makar et al., 2001) for details).

Figure (7) shows the layout of the AGV scheduling problem used in this experiment. $M1$ 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_i's), to the warehouse. The AGV travel is unidirectional (as the arrows show).

Each agent uses a copy of the task graph in Figure (8). 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 of the hierarchy as described above.

The state of the environment consists of the number of parts in the pickup and drop-off stations of each machine, and whether the warehouse contains parts of each of the four types. In addition, each agent keeps track of its own location and status as a part of its state space. Thus, in the flat case, state space consists of 100 locations, 8 buffers of size 3, 9 possible states of the AGV (carrying Part1, ... , carrying Assembly1, ... , empty), and 2 values for each part in the warehouse, i.e. $100 \times 4^8 \times 9 \times 2^4 \approx 2^{30}$ states, which is enormous. *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* subtask, only the location state variable is relevant, and this subtask can be learned with 100 values. Hence, for the highest level actions *DM1*, ...

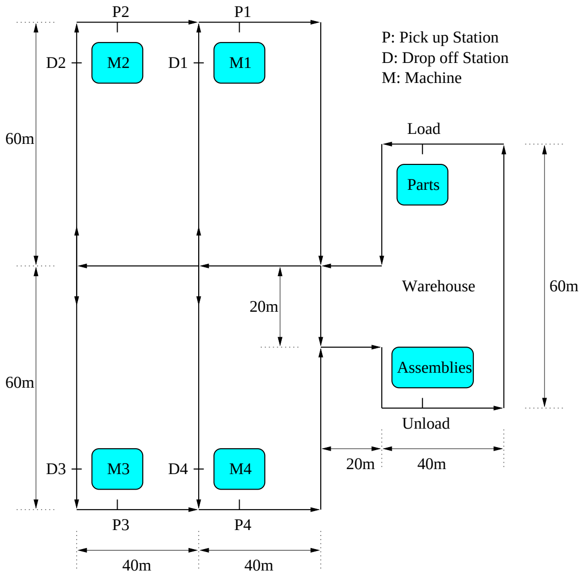

Figure 7: A multiagent AGV scheduling task. There are four AGV agents (not shown) which carry raw materials and finished parts between machines and warehouse.

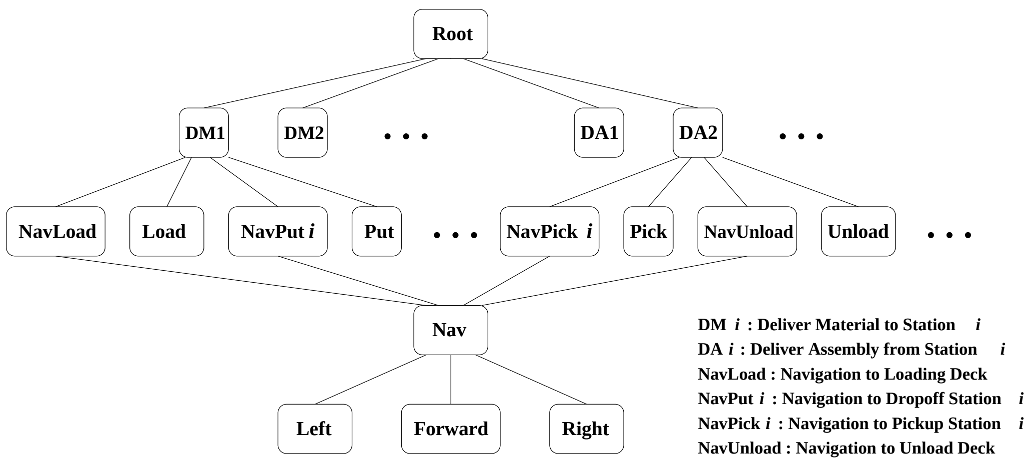

Figure 8: Task graph for the AGV scheduling task.

, *DM4*, the number of relevant states would be $100 \times 9 \times 4 \times 2 \approx 2^{13}$, and for the highest level actions *DA1*, $\ldots$, *DA4*, the number of relevant states would be $100 \times 9 \times 4 \approx 2^{12}$. This state abstraction gives us a compact way of representing the C functions, and speeds up the algorithm (Dietterich, 2000).

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 (*navigation* actions, *load* and *unload*) is normally distributed with mean 1000 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.

Parameter	**Type of 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

Table 1: Model Parameters

Figure (9) shows the throughput of the system for continuous-time recursively optimal discounted reward HRL algorithm and continuous-time recursively gain optimal average reward (RO-AR) algorithm. 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. This figure also compares these two 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.

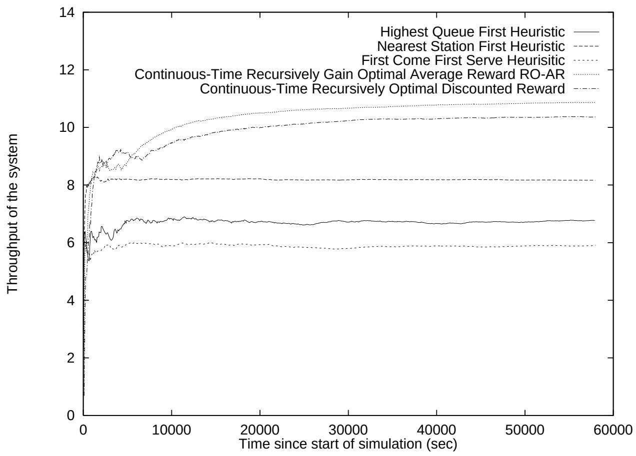

Figure 9: This plot shows continuous-time recursively gain optimal average reward (RO-AR) algorithm outperforms continuous-time recursively optimal discounted reward algorithm. It also demonstrates both these algorithms outperform three well-known widely used (industrial) heuristics for AGV scheduling.

5.4 AGV Scheduling Problem (Discrete and Continuous Time Models)

In this section, we describe two sets of experiments on a modified version of the AGV scheduling task described in Section (5.3). In the experiments of this section, we assume a single-agent problem with only three machines in the environment (Figure (7) without machine $M3$). It reduces the number of states to 1,347,192 states. We model this AGV scheduling task using both discrete-time and continuous-time models, and compare the performance and speed of three HRL algorithms: *hierarchically gain optimal RL* (HO-AR), *hierarchically optimal discounted reward RL* (HO-DR) and *recursively gain optimal RL* (RO-AR) as well as a *non-hierarchical average reward* algorithm. In both sets of experiments, we use the task graph for the AGV scheduling task shown in Figure (8) and discount factors 0.9 and 0.95 for discounted reward algorithms. In both experiments, using a discount factor of 0.95 yielded better performance.

The discrete-time 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 12 time steps and variance of 2 time steps. The percentage of $Part1$, $Part2$ and $Part3$ in the part arrival process are 40, 35 and 25 respectively. The time required for assembling the various parts are Poisson random variables with means 6, 10 and 12 time steps for $Part1$, $Part2$ and $Part3$ respectively, and variance 2 time steps. Table (2) shows the parameters of the discrete-time model.

Parameter	**Distribution**	**Mean (steps)**	**Variance (steps)**
Assembly Time for Part1	Poisson	6	2
Assembly Time for Part2	Poisson	10	2
Assembly Time for Part3	Poisson	12	2
Inter-Arrival Time for Parts	Uniform	12	2

Table 2: Parameters of the Discrete-Time Model

The continuous-time experimental results were generated with the following model parameters. The time required for execution of each primitive action is a normal random variable with mean 10 sec and variance 2 sec. The inter-arrival time for parts at the warehouse is uniformly distributed with a mean of 100 seconds and variance of 20 seconds. The percentage of $Part1$, $Part2$ and $Part3$ in the part arrival process are 40, 35 and 25 respectively. The time required for assembling the various parts are normal random variables with means 100, 120 and 180 sec for $Part1$, $Part2$ and $Part3$ respectively, and variance 20 sec. Table (3) contains the parameters of the continuous-time model. In both cases, each experiment was conducted five times and the results averaged.

Parameter	**Type of Distribution**	**Mean (sec)**	**Variance (sec)**
Execution Time for Primitive Actions	Normal	10	2
Assembly Time for Part1	Normal	100	20
Assembly Time for Part2	Normal	120	20
Assembly Time for Part3	Normal	180	20
Inter-Arrival Time for Parts	Uniform	100	20

Table 3: Parameters of the Continuous-Time Model

Figure (10) compares the proposed discrete-time hierarchically gain optimal algorithm (HO-AR) described in Section (3.3) with the discrete-time discounted reward hierarchically optimal algorithm (HO-DR) and the discrete-time recursively gain optimal algorithm (RO-AR) illustrated in Section (3.4.1). The graph shows the improved performance of the proposed discrete-time average reward algorithm (HO-AR). This figure also shows that the HO-AR algorithm converges faster to the same throughput as the non-hierarchical average reward algorithm. The non-hierarchical average reward algorithm used in this experiment is relative value iteration (RVI) Q-learning (Abounadi et al., 2001). The difference in convergence speed between flat and hierarchical algorithms becomes more significant as we increase the number of states.

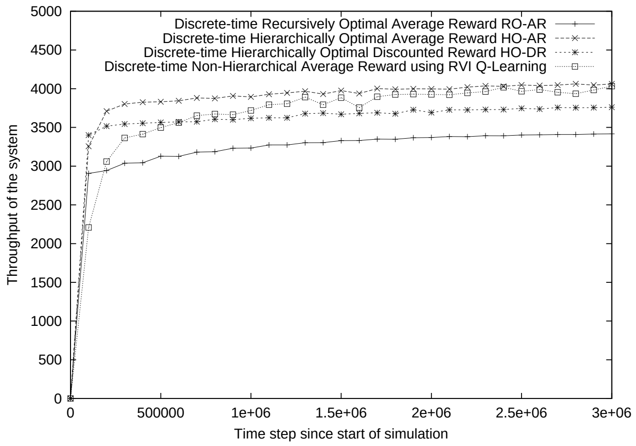

Figure 10: This plot shows that the discrete-time HO-AR algorithm performs better than both the discounted reward HO-DR and RO-AR algorithms on the AGV scheduling task. It also demonstrates the faster convergence of the HO-AR algorithm comparing to the non-hierarchical average reward algorithm (RVI Q-learning).

Figure (11) compares the continuous-time hierarchically gain optimal algorithm (HO-AR) proposed in Section (3.3) with the continuous-time hierarchically optimal discounted reward HO-DR algorithm and the continuous-time recursively gain optimal algorithm (RO-AR) described in Section (3.4.1). The graph shows that the HO-AR converges to the same performance as the discounted reward HO-DR algorithm. Both clearly have better performance than the average reward recursively optimal algorithm (RO-AR). This figure also shows that the HO-AR algorithm converges faster to the same throughput as the non-hierarchical average reward algorithm. The non-hierarchical average reward algorithm used in this experiment is a continuous-time version of the relative value iteration (RVI) Q-learning (Abounadi et al., 2001). The difference in convergence speed between flat and hierarchical algorithms becomes more significant as we increase the number of states.

These results in the last two sections are consistent with the hypothesis that the undiscounted optimality paradigm is superior to the discounted framework for learning a gain-optimal policy, since undiscounted methods do not need careful tuning of the discount factor to find gain-optimal policies.

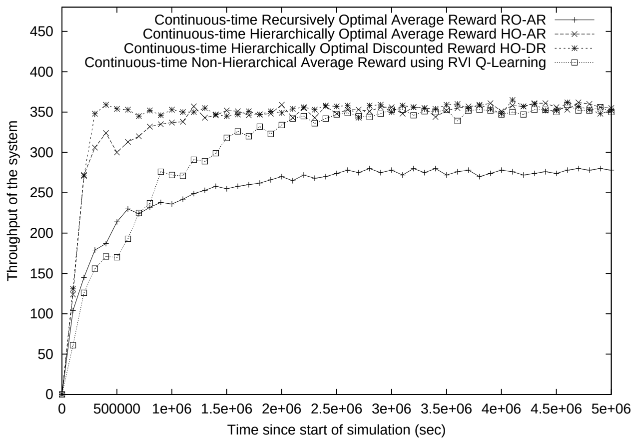

Figure 11: This plot shows that the continuous-time HO-AR converges to the same performance as the discounted reward HO-DR, and both outperform the recursively optimal average reward algorithm on the AGV scheduling task. It also demonstrates the faster convergence of the HO-AR compared to the flat average reward algorithm (RVI Q-learning).

6. Conclusion and Future Work

This paper presents new discrete-time and continuous-time hierarchical reinforcement learning algorithms applicable to continuing tasks, including manufacturing, scheduling, queuing and inventory control. These algorithms are based on the average-reward SMDP model, which has been shown to be more appropriate for a wide class of continuing tasks. These hierarchical average-reward reinforcement learning algorithms can be categorized into two groups corresponding to two notions of optimality that have been studied in previous work on HRL: *hierarchical optimality* and *recursive optimality.* Hierarchically gain optimal average reward RL algorithms aim to find a globally gain optimal policy within the space of policies defined by the hierarchical decomposition. In the recursive optimal average reward HRL setting, the formulation of learning algorithms directly depends on the local optimality criterion used for each subtask in the hierarchy. In this paper, we investigate several methods to formulate subtasks, however, recursively gain optimal average reward RL algorithms proposed in this paper treat subtasks as continuing problems and solve them by finding gain optimal policies given the policies of their children. A secondary contribution of this paper is to illustrate how hierarchical reinforcement learning can be applied to more interesting and practical domains than has been shown previously. In particular, we focus on AGV scheduling, although our approach easily generalizes to other industrial optimization problems such as transfer line production control.

The effectiveness of the proposed algorithms were tested using four experimental testbeds: a small AGV scheduling domain, a modified version of the Taxi problem, and much larger real-world single-agent and multiagent AGV domains. The proposed algorithms performed well in all domains, and in particular, in the multiagent AGV domain, we showed that our proposed algorithms 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. An immediate question that arises is the convergence of the algorithms to recursive and hierarchical optimal policies. These results should provide some theoretical validity to the proposed methods, in addition to their empirical effectiveness demonstrated in this paper. It is obvious that many other manufacturing and robotics problems can benefit from these algorithms. Combining hierarchical reinforcement learning with function approximation and factored action models is an important area for research. In this direction, we are currently working to develop a hierarchical reinforcement learning framework suitable for problems with continuous state spaces, using a mixture of policy gradient-based RL and value function-based RL methods (Ghavamzadeh and Mahadevan, 2003). We used the flexibility provided by recursive optimality and developed HRL algorithms for learning in this hierarchical hybrid framework. In this hybrid framework, when the overall task is *continuing*, we formulate the *root* task as a *continuing* problem using the average reward criterion. Since the policy learned at *root* involves policies of its children, the type of optimality achieved at *root* depends on how we formulate other subtasks in the hierarchy. We are currently investigating different formulations for average reward recursive optimality described in this paper, in our hierarchical hybrid framework. Finally, deriving abstractions automatically is another fundamental problem that needs to be addressed.

Acknowledgments

We would like to thank Prasad Tadepalli, David Andre and Balaraman Ravindran for their useful comments. The computational experiments were carried out in 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 Autonomous Learning Laboratory in the Department of Computer Science at University of Massachusetts Amherst, under NASA contract No. NAg-1445 #1.

References

J. Abounadi, D. P. Bertsekas, and V. S. Borkar. Learning algorithms for Markov decision processes with average cost. *SIAM Journal on Control and Optimization*, 40, 2001.

D. Andre and S. J. Russell. State abstraction for programmable reinforcement learning agents. In *Proceedings of the Eighteenth AAAI*, pages 119–125, 2002.

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

J. Baxter and P. L. Bartlett. Infinite-horizon policy gradient estimation. *Journal of Artificial Intelligence Research*, 15:319–350, 2001.

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

D. P. Bertsekas and J. N. Tsitsiklis. *Neuro-Dynamic Programming*. Athena Scientific, Belmont, Massachusetts, 1996.

S. J. Bradtke and M. O. Duff. Reinforcement learning methods for continuous-time Markov decision problems. In *Proceedings of the Seventh International Conference on Advances in Neural Information Processing Systems*. MIT Press, 1995.

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

S. Gershwin. *Manufacturing Systems Engineering*. Prentice Hall, 1994.

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

M. Ghavamzadeh and S. Mahadevan. Hierarchically optimal average reward reinforcement learning. In *Proceedings of the Nineteenth International Conference on Machine Learning*, pages 195–202, 2002.

M. Ghavamzadeh and S. Mahadevan. Hierarchical policy gradient algorithms. *Proceedings of the Twentieth International Conference on Machine Learning*, 2003.

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

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

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

S. Mahadevan, N. Khaleeli, and N. Marchalleck. Designing agent controllers using discrete-event Markov models. In *AAAI Fall Symposium on Model-Directed Autonomous Systems*, 1997a.

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

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.

P. Marbach. *Simulated-Based Methods for Markov Decision Processes*. Ph.D. Thesis, Massachusetts Institute of Technology, 1998.

A. Ng, D. Harada, and S. Russell. Policy invariance under reward transformations: Theory and application to reward shaping. In *Proceedings of the Sixteenth International Conference on Machine Learning*, 1999.

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

M. L. Puterman. *Markov Decision Processes.* Wiley Inter-science, 1994.

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

S. Seri and P. Tadepalli. Model-based hierarchical average-reward reinforcement learning. In *Proceedings of the Nineteenth International Conference on Machine Learning*, pages 562–569, 2002.

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

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

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 the Thirteenth International Conference on Machine Learning*, 1996b.

B. Van-Roy. *Learning and Value Function Approximation in Complex Decision Processes.* Ph.D. Thesis, Massachusetts Institute of Technology, 1998.

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.* Ph.D. Thesis, Kings College, Cambridge, England, 1989.

Appendix A. List of Notation

Notation	Definition
$\mathcal{H}$	hierarchy
$\mathcal{N}$	set of natural numbers
S	set of states
Ω	set of possible values for *Task Stack* in hierarchy
$X = \Omega \times S$	joint state space of *Task Stack* values and states
$x = (\omega, s)$	joint state value x formed by *Task Stack* value ω and state value s
$\omega \nearrow i$	popping subtask i off the *Task Stack* with content ω
$i \searrow \omega$	pushing subtask i onto the *Task Stack* with content ω
$\lvert S \rvert$	cardinality of the set S
A	set of actions
P	transition probability function
R	reward function
S_i	set of states for subtask i
A_i	set of actions for subtask i
R_i	reward function for subtask i
I_i	initiation set for subtask i
T_i	termination set for subtask i
s_i^T	a terminal state of subtask i, $s_i^T \in T_i$
μ_i	policy for subtask i
$\mu = \{\mu_0, \dots, \mu_n\}$	hierarchical policy
μ^*	hierarchical optimal policy
$P_i^\mu(s', N \vert s)$	probability that action $\mu_i(s)$ causes transition from state s to state s' in N primitive steps under the hierarchical policy μ
$F_i^\mu(s', n \vert s)$	probability of transition from state s to state s' in n actions taken by subtask i under the hierarchical policy μ
g^μ	gain of policy μ
$y(s, a)$	expected number of transition steps until the next decision epoch
$\pi^\mu(s)$	steady state probability of being in state s for Markov chain defined by policy μ
π^μ	steady state probability vector of the Markov chain defined by policy μ
$\bar{\pi}_i$	probability distribution on initial states of subtask i
V^μ	hierarchical value function of hierarchical policy μ
$\hat{V}^\mu$	projected value function of hierarchical policy μ
H^μ	hierarchical average adjusted value function of hierarchical policy μ
$\hat{H}^\mu$	projected average adjusted value function of hierarchical policy μ
L^μ	hierarchical average adjusted action-value function of hierarchical policy μ
$\hat{L}^\mu$	projected average adjusted action-value function of hierarchical policy μ
C^μ	completion function of hierarchical policy μ

Table 4: List of notation used in the paper