Journal of Machine Learning Research 8 (2007) 2629-2669 Submitted 6/03; Revised 8/07; Published 11/07

Hierarchical Average Reward Reinforcement Learning

Mohammad Ghavamzadeh MGH@CS.UALBERTA.CA
Department of Computing Science
University of Alberta
Edmonton, AB T6G 2E8, CANADA

Sridhar Mahadevan MAHADEVA@CS.UMASS.EDU
Department of Computer Science
University of Massachusetts
Amherst, MA 01003-4601, USA

Editor: Michael Littman

Abstract

Hierarchical reinforcement learning (HRL) is a general framework for scaling reinforcement learning (RL) to problems with large state and action spaces by using the task (or action) structure to restrict the space of policies. Prior work in HRL including HAMs, options, MAXQ, and PHAMs has been limited to the *discrete-time discounted reward semi-Markov decision process* (SMDP) model. The average reward optimality criterion has been recognized to be more appropriate for a wide class of continuing tasks than the discounted framework. Although average reward RL has been studied for decades, prior work has been largely limited to flat policy representations.

In this paper, we develop a framework for HRL based on the *average reward* optimality criterion. We investigate two formulations of HRL based on the average reward SMDP model, both for discrete-time and continuous-time. These formulations correspond to two notions of optimality that have been previously explored in HRL: *hierarchical optimality* and *recursive optimality*. We present algorithms that learn to find hierarchically and recursively optimal average reward policies under discrete-time and continuous-time average reward SMDP models.

We use two automated guided vehicle (AGV) scheduling tasks as experimental testbeds to study the empirical performance of the proposed algorithms. The first problem is a relatively simple AGV scheduling task, in which the hierarchically and recursively optimal policies are different. We compare the proposed algorithms with three other HRL methods, including a hierarchically optimal discounted reward algorithm and a recursively optimal discounted reward algorithm on this problem. The second problem is a larger AGV scheduling task. We model this problem using both discrete-time and continuous-time models. We use a hierarchical task decomposition in which the hierarchically and recursively optimal policies are the same for this problem. We compare the performance of the proposed algorithms with a hierarchically optimal discounted reward algorithm and a recursively optimal discounted reward algorithm, as well as a non-hierarchical average reward algorithm. The results show that the proposed hierarchical average reward algorithms converge to the same performance as their discounted reward counterparts.

Keywords: semi-Markov decision processes, hierarchical reinforcement learning, average reward reinforcement learning, hierarchical and recursive optimality

1. Introduction

Sequential decision making under uncertainty is a fundamental problem in artificial intelligence (AI). Many sequential decision making problems can be modeled using the Markov decision process (MDP) formalism. A MDP (Howard, 1960; Puterman, 1994) models a system that we are interested in controlling as being in some state at each time step. As a result of actions, the system moves through some sequence of states and receives a sequence of rewards. The goal is to select actions to maximize (minimize) some measure of long-term reward (cost), such as the expected discounted sum of rewards (costs), or the expected average reward (cost).

Reinforcement learning (RL) is a machine learning framework for solving sequential decision-making problems. Despite its successes in a number of different domains, including backgammon (Tesauro, 1994), job-shop scheduling (Zhang and Dietterich, 1995), dynamic channel allocation (Singh and Bertsekas, 1996), elevator scheduling (Crites and Barto, 1998), and helicopter flight control (Ng et al., 2004), current RL methods do not scale well to high dimensional domains—they can be slow to converge and require many training samples to be practical for many real-world problems. This issue is known as the *curse of dimensionality*: the exponential growth of the number of parameters to be learned with the size of any compact encoding of system state (Bellman, 1957). Recent attempts to combat the curse of dimensionality have turned to principled ways of exploiting abstraction in RL. This leads naturally to hierarchical control architectures and associated learning algorithms.

Hierarchical reinforcement learning (HRL) is a general framework for scaling RL to problems with large state spaces by using the task (or action) structure to restrict the space of policies. Hierarchical decision making represents policies over fully or partially specified temporally extended actions. Policies over temporally extended actions cannot be simply treated as single-step actions over a coarser time scale, and therefore cannot be represented in the MDP framework since actions take variable durations of time. Semi-Markov decision process (SMDP) (Howard, 1971; Puterman, 1994) is a well-known statistical framework for modeling temporally extended actions. Action duration in a SMDP can depend on the transition that is made. The state of the system may change continually between actions, unlike MDPs where state changes are only due to actions. Therefore, SMDPs have become the main mathematical model underlying HRL methods.

Prior work in HRL including hierarchies of abstract machines (HAMs) (Parr, 1998), options (Sutton et al., 1999; Precup, 2000), MAXQ (Dietterich, 2000), and programmable HAMs (PHAMs) (Andre and Russell, 2001; Andre, 2003) has been limited to the discrete-time discounted reward SMDP model. In these methods, policies are learned that maximize the long-term discounted sum of rewards. On the other hand, the average reward optimality criterion has been shown to be more appropriate for a wide class of *continuing* tasks than the well-studied discounted framework. 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 studied using both the discrete-time MDP model (Schwartz, 1993; Mahadevan, 1996; Tadepalli and Ok, 1996a,b, 1998; 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 addition to being an appropriate optimality criterion for continuing tasks, average reward optimality allows for more efficient state abstraction in HRL than discounted reward optimality, as will be discussed in Section 5.

In this paper, we extend previous work on HRL to the average reward setting, and investigate two formulations of HRL based on average reward SMDPs. These two formulations correspond to two notions of optimality in HRL: *hierarchical optimality* and *recursive optimality* (Dietterich, 2000). We extend the MAXQ hierarchical RL method (Dietterich, 2000) and introduce a HRL framework for simultaneous learning of policies at multiple levels of a task hierarchy. We then use this HRL framework to derive discrete-time and continuous-time algorithms that learn to find hierarchically and recursively optimal average reward policies. In these algorithms, we assume that the overall task (the *root* of the hierarchy) is continuing. **Hierarchically optimal average reward RL** (HAR) algorithms find a hierarchical policy within the space of policies defined by the hierarchical decomposition that maximizes the *global gain*. **Recursively optimal average reward RL** (RAR) algorithms treat non-primitive subtasks as continuing average reward problems, where the goal at each subtask is to maximize its gain given the policies of its children. We investigate the conditions under which the policy learned by RAR algorithm at each subtask is independent of the context in which it is executed and therefore can be reused by other hierarchies. We use two automated guided vehicle (AGV) scheduling tasks as experimental testbeds to study the empirical performance of the proposed algorithms. The first problem is a relatively simple AGV scheduling task, in which the hierarchically and recursively optimal policies are different. We compare the proposed algorithms with three other HRL methods, including a hierarchically optimal discounted reward algorithm and a recursively optimal discounted reward algorithm on this problem. The second problem is a relatively larger AGV scheduling task. We model this problem using both discrete-time and continuous-time models. We use a hierarchical task decomposition where the hierarchically and recursively optimal policies are the same. We compare the performance of the proposed algorithms with a hierarchically optimal discounted reward algorithm and a recursively optimal discounted reward algorithm, as well as a non-hierarchical average reward algorithm. The results show that the proposed hierarchical average reward algorithms converge to the same performance as their discounted reward counterparts.

The rest of this paper is organized as follows. Section 2 provides a brief overview of HRL. In Section 3, we concisely describe discrete-time SMDPs, and discuss average reward optimality in this model. Section 4 describes the HRL framework, which is used to develop the algorithms of this paper. In Section 5, we extend the previous work on HRL to the average reward setting, and study two formulations of HRL based on the average reward SMDP model. In Section 5.1, we present discrete-time and continuous-time hierarchically optimal average reward RL (HAR) algorithms. In Section 5.2, we investigate different methods to formulate subtasks in a recursively optimal hierarchical average reward RL setting, and present discrete-time and continuous-time recursively optimal average reward RL (RAR) algorithms. We demonstrate the type of optimality achieved by HAR and RAR algorithms as well as their empirical performance and convergence speed compared to other algorithms using two AGV scheduling problems in Section 6. Section 7 summarizes the paper and discusses some directions for future work. For convenience, a table of the symbols used in this paper is given in Appendix A.

2. An Overview of Hierarchical Reinforcement Learning

Hierarchical reinforcement learning (HRL) is a class of learning algorithms that share a common approach to scaling up reinforcement learning (RL). 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. Subtasks form the basis of hierarchical specifications of action sequences because they can include other subtasks in their definitions. It is similar to the familiar idea of subroutines from programming languages. A subroutine can call other subroutines as well as execute primitive commands. A subtask as an open-loop control policy is inappropriate for most interesting control purposes, especially the control of stochastic systems. HRL methods generalize the subtask idea to closed-loop policies or, more precisely, closed-loop partial policies because they are generally defined for a subset of the state space. The partial policies must also have well-defined termination conditions. The partial policies with well-defined termination conditions are sometimes called temporally extended actions. Work in HRL has followed three trends: focusing on subsets of the state space in a divide and conquer approach (state space decomposition), grouping sequences or sets of actions together (temporal abstraction), and ignoring differences between states based on the context (state abstraction). Much of the work in HRL falls into several of these categories. Barto and Mahadevan (2003) provide a more detailed introduction to HRL.

Mahadevan and Connell (1992) were among the first to systematically investigate the use of task structure to accelerate RL. In their work, the robot was given a pre-specified task decomposition, and learned a set of local policies instead of an entire global policy. Singh (1992) investigated reinforcement learning using abstract actions of different temporal granularity using a hierarchy of models with variable temporal resolution. Singh applied the mixture of experts framework as a special-purpose task selection architecture to switch between abstract actions. Kaelbling (1993a,b) investigated using subgoals to learn sub-policies. Dayan and Hinton (1993) describe Feudal RL, a hierarchical technique which uses both temporal abstraction and state abstraction to recursively partition the state space and the time scale from one level to the next.

One key limitation of all the above methods is that decisions in HRL are no longer made at synchronous time steps, as is traditionally assumed in RL. Instead, agents make decisions intermittently, where each epoch can be 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). In a SMDP, state-transition dynamics is specified not only by the state where an action was taken, but also by parameters specifying the length of time since the action was taken. Early work on the SMDP model extended algorithms such as Q-learning to continuous-time (Bradtke and Duff, 1995; Mahadevan et al., 1997b). The early work on SMDP was then expanded to include hierarchical task models over fully or partially specified lower level subtasks, which led to developing powerful HRL models such as hierarchies of abstract machines (HAMs) (Parr, 1998), options (Sutton et al., 1999; Precup, 2000), MAXQ (Dietterich, 2000), and programmable HAMs (PHAMs) (Andre and Russell, 2001; Andre, 2003).

In the options framework policies are defined over not just primitive actions, but over fully specified lower-level policies. In the HAMs formulation, hierarchical learning could be achieved even when the policies for lower-level subtasks were only partially specified. The MAXQ model is one of the first methods to combine temporal abstraction with state abstraction. It provides a more comprehensive framework for hierarchical learning where instead of policies for subtasks, the learner is given pseudo-reward functions. Unlike options and HAMs, MAXQ does not rely directly on reducing the entire problem to a single SMDP. Instead, a hierarchy of SMDPs is created whose solutions can be learned simultaneously. The key feature of MAXQ is the decomposed representation of the value function. The MAXQ method views each subtask as a separate SMDP, and thus represents the value of a state within that SMDP as composed of the reward for taking an action at that state (which might be composed of many rewards along a trajectory through a subtask) and the expected reward for completing the subtask. To isolate the subtask from the calling context, MAXQ uses the notion of a pseudo-reward. At the terminal states of a subtask, the agent is rewarded according to the pseudo-reward, which is set a priori by the designer, and does not depend on what happens after leaving the current subtask. Each subtask can then be treated in isolation from the rest of the problem with the caveat that the solutions learned are only recursively optimal. Each action in the recursively optimal policy is optimal with respect to the subtask containing the action, all descendant subtasks, and the pseudo-reward chosen by the designer of the system. Another important contribution of MAXQ is the idea that state abstraction can be done separately on the different components of the value function, which allows one to perform *dynamic abstraction*. We describe the MAXQ framework and related concepts such as recursive optimality and value function decomposition in Section 4. In the PHAM model, Andre and Russell extended HAMs and presented an agent-design language for RL. Andre and Russell (2002) also addressed the issue of safe state abstraction in their model. Their method yields state abstraction while maintaining hierarchical optimality.

3. Discrete-time Semi-Markov Decision Processes

Semi-Markov decision processes (SMDPs) (Howard, 1971; Puterman, 1994) extend the Markov decision process (MDP) (Howard, 1971; Puterman, 1994) model by allowing actions that can take multiple time steps to complete. Note that SMDPs do not theoretically provide additional expressive power but they do provide a convenient formalism for temporal abstraction. The duration of an action can depend on the transition that is made.[1] The state of the system may change continually between actions unlike MDPs where state changes are only due to actions. Thus, SMDPs have become the preferred language for modeling temporally extended actions (Mahadevan et al., 1997a) and, as a result, the main mathematical model underlying hierarchical reinforcement learning (HRL).

A SMDP is defined as a five tuple $\langle \mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R}, I \rangle$. All components are defined as in a MDP except the transition probability function $\mathcal{P}$ and the reward function $\mathcal{R}$. $\mathcal{S}$ is the set of states of the world, $\mathcal{A}$ is the set of possible actions from which the agent may choose on at each decision epoch, and $I : \mathcal{S} \rightarrow [0,1]$ is the initial state distribution. The transition probability function $\mathcal{P}$ now takes the duration of the actions into account. The transition probability function $\mathcal{P} : \mathcal{S} \times \mathbb{N} \times \mathcal{S} \times \mathcal{A} \rightarrow [0,1]$ is a multi-step transition probability function ($\mathbb{N}$ is the set of natural numbers), where $P(s', N|s, a)$ 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. If we marginalize $P(s', N|s, a)$ over N, we will obtain $F(s'|s, a)$ the transition probability for the embedded MDP. The term $F(s'|s, a)$ denotes the probability that the system occupies state s' at the next decision epoch, given that the decision maker chooses action a in state s at the current decision epoch. The key difference in the reward function for SMDPs is that the rewards can accumulate over the entire duration of an action. As a result, the reward in a SMDP for taking an action in a state depends on the evolution of the system during the execution of the action. Formally, the reward in a SMDP is modeled as a function $\mathcal{R} : \mathcal{S} \times \mathcal{A} \rightarrow \mathbb{R}$ ($\mathbb{R}$

1. Continuous-time SMDPs typically allow arbitrary continuous action durations.

is the set of real numbers), with $r(s,a)$ representing the expected total reward between two decision epochs, given that the system occupies state s at the first decision epoch and the agent chooses action a. This expected reward contains all necessary information about the reward to analyze the SMDP model. For each transition in a SMDP, the expected number of time steps until the next decision epoch is defined as

$$y(s,a) = \mathbf{E}[N|s,a] = \sum_{N\in\mathbb{N}} N \sum_{s'\in\mathcal{S}} P(s',N|s,a).$$

The notion of policy and the various forms of optimality are the same for SMDPs as for MDPs. In infinite-horizon SMDPs, the goal is to find a policy that maximizes either the expected discounted reward or the average expected reward. We discuss the average reward optimality criterion for the SMDP model in the next section.

3.1 Average Reward Semi-Markov Decision Processes

The theory of infinite-horizon SMDPs with the average reward optimality criterion is more complex than that for discounted models (Howard, 1971; Puterman, 1994). The aim of average reward SMDP algorithms is to compute policies that yield the highest average reward or gain. The average reward or gain of a policy μ at state s, $g^\mu(s)$, can be defined as the ratio of the expected total reward and the expected total number of time steps of following policy μ starting at state s

$$g^\mu(s) = \liminf_{n\to\infty} \frac{\mathbf{E}\left[\sum_{t=0}^{n-1} r(s_t,\mu(s_t))|s_0=s,\mu\right]}{\mathbf{E}\left[\sum_{t=0}^{n-1} N_t|s_0=s,\mu\right]}.$$

In this equation, N_t is the total number of time steps until the next decision epoch, when agent takes action $\mu(s_t)$ in state s_t.

A key observation that greatly simplifies the design of average reward algorithms is that for *unichain* SMDPs,[2] the gain of any policy is state independent, that is

$$g^\mu(s) = g^\mu = \liminf_{n\to\infty} \frac{\mathbf{E}\left[\sum_{t=0}^{n-1} r(s_t,\mu(s_t))|\mu\right]}{\mathbf{E}\left[\sum_{t=0}^{n-1} N_t|\mu\right]}, \qquad \forall s\in\mathcal{S}. \tag{1}$$

To simplify exposition, we consider only *unichain* SMDPs in this paper. When the state space of a SMDP, $\mathcal{S}$, is finite or countable, Equation 1 can be written as

$$g^\mu = \frac{\bar{F}^\mu r^\mu}{\bar{F}^\mu y^\mu}, \tag{2}$$

where F^μ and $\bar{F}^\mu = \lim_{n\to\infty}\frac{1}{n}\sum_{t=0}^{n-1}(F^\mu)^t$ are the transition probability matrix and the *limiting matrix* of the embedded Markov chain for policy μ, respectively,[3] and r^μ and y^μ are vectors with elements $r(s,\mu(s))$ and $y(s,\mu(s))$, for all $s\in\mathcal{S}$. Under the *unichain* assumption, $\bar{F}$ has equal rows, and therefore the right hand side of Equation 2 is a vector with elements all equal to g^μ.

2. In unichain SMDPs, the underlying Markov chain for every stationary policy has a single recurrent class, and a (possibly empty) set of transient states.
3. The *limiting matrix* $\bar{F}$ satisfies the equality $\bar{F}F = \bar{F}$.

In the average reward SMDP model, a policy μ is measured using a different value function, namely the average-adjusted sum of rewards earned following that policy[4]

$$H^{\mu}(s) = \lim_{n\to\infty} \mathbf{E}\left\{\sum_{t=0}^{n-1}\left[r(s_t,\mu(s_t)) - g^{\mu}y(s_t,\mu(s_t))\right]|s_0 = s,\mu\right\}.$$

The term H^{μ} is usually referred to as the **average-adjusted value function**. Furthermore, the average-adjusted value function satisfies the Bellman equation

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

Similarly, the **average-adjusted action-value function** for a policy μ, L^{μ}, is defined, and it satisfies the Bellman equation

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

4. A Framework for Hierarchical Reinforcement Learning

In this section, we describe a general hierarchical reinforcement learning (HRL) framework for simultaneous learning of policies at multiple levels of a hierarchy. Our treatment builds upon existing methods, including HAMs (Parr, 1998), options (Sutton et al., 1999; Precup, 2000), MAXQ (Dietterich, 2000), and PHAMs (Andre and Russell, 2002; Andre, 2003), and, in particular, uses the MAXQ value function decomposition. We extend the MAXQ framework by including the three-part value function decomposition (Andre and Russell, 2002) to guarantee hierarchical optimality, as well as reward shaping (Ng et al., 1999) to reduce the burden of exploration. Rather than redundantly explain MAXQ and then our hierarchical framework, we will present our model and note throughout this section where the key pieces were inspired by or are directly related to MAXQ. In the next section, we will extend this framework to the average reward model and present our hierarchical average reward reinforcement learning algorithms.

4.1 Motivating Example

In the HRL framework, the designer of the system imposes a hierarchy on the problem to incorporate domain knowledge and thereby reduces the size of the space that must be searched to find a good policy. The designer recursively decomposes the overall task into a collection of subtasks that are important for solving the problem.

Let us illustrate the main ideas using a simple search task shown in Figure 1. Consider the domain of an office-type environment (with rooms and connecting corridors), where 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*). For simplicity, we assume that the robot can observe its true location in the environment. 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. After defining subtasks, we must indicate, for each subtask, which

4. This limit assumes that all policies are aperiodic. For periodic policies, it changes to the Cesaro limit $H^{\mu}(s) = \lim_{n\to\infty}\frac{1}{n}\sum_{k=0}^{n-1}\mathbf{E}\left\{\sum_{t=0}^{k}\left[r(s_t,\mu(s_t)) - g^{\mu}y(s_t,\mu(s_t))\right]|s_0 = s,\mu\right\}$ (Puterman, 1994).

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. Similar to MAXQ, all of this information can be summarized by a directed acyclic graph called **task graph**. The task graph for the trash-collection problem is shown in Figure 1. This hierarchical model is able to support **state abstraction** (while the agent is moving toward the *Dump*, the status of trash cans $T1$ and $T2$ is irrelevant and cannot affect this navigation process. Therefore, the variables defining the status of trash cans $T1$ and $T2$ can be removed from the state space of the *navigate to Dump* subtask), and **subtask sharing** (if the system could learn how to solve the *navigate to Dump* subtask once, then the solution could be shared by both *collect trash at* $T1$ and $T2$ subtasks.)

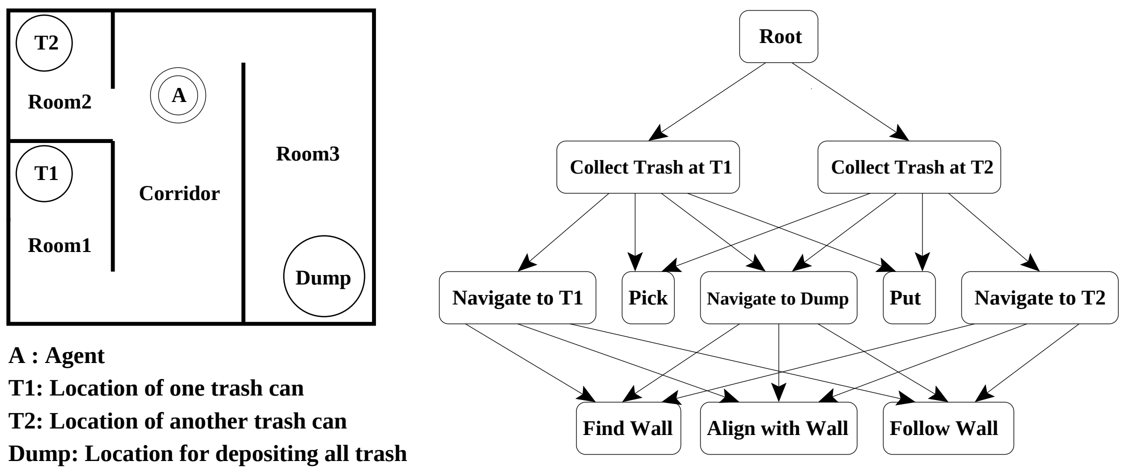

Figure 1: A robot trash-collection task and its associated task graph.

Like HAMs (Parr, 1998), options (Sutton et al., 1999; Precup, 2000), MAXQ (Dietterich, 2000), and PHAMs (Andre and Russell, 2001; Andre, 2003), this framework also relies on the theory of SMDPs. While SMDP theory provides the theoretical underpinnings of temporal abstraction by modeling actions that take varying amounts of time, the SMDP model provides little in the way of concrete representational guidance, which is critical from a computational point of view. In particular, the SMDP model does not specify how tasks can be broken up into subtasks, how to decompose value functions etc. We examine these issues next.

As in MAXQ, a task hierarchy such as the one illustrated above can be modeled by decomposing the overall task MDP $\mathcal{M}$, into a finite set of subtasks $\{M_0, M_1, \ldots, M_{m-1}\}$,[5] where M_0 is the *root* task. Solving M_0 solves the entire MDP $\mathcal{M}$.

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

5. m is the total number of subtasks in the hierarchy.

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

- $I_i \subseteq S_i$ is the **initiation set** for subtask M_i. Subtask M_i can be initiated only in states belonging to I_i.

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

- A_i is the **set of actions** that can be performed to achieve subtask M_i. These actions can be either primitive actions from $\mathcal{A}$ (the set of primitive actions for MDP $\mathcal{M}$), or they can be other subtasks. Technically, A_i is a function of states, since it may differ from one state to another. However, we will suppress this dependence in our notation.

- R_i is the **reward structure** inside subtask M_i and could be different from the reward function of MDP $\mathcal{M}$. Here, we use the idea of reward shaping (Ng et al., 1999) and define a more general reward structure than MAXQ's. Reward shaping is a method for guiding an agent toward a solution without constraining the search space. Besides the reward of the overall task MDP $\mathcal{M}$, each subtask M_i can use additional rewards to guide its local learning. 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 from the reward function of the overall task, we need to define two types of value functions for each subtask, internal value function and external value function. **Internal value function** is defined based on both the local reward structure of the subtask and the reward of the overall task, and only is used in learning the subtask. On the other hand, **external value function** is defined only based on the reward function of the overall task and is propagated to the higher levels in the hierarchy to be used in learning the global policy. This reward structure for each subtask in our framework is more general than the one in MAXQ, and of course, includes MAXQ's pseudo-reward.[6] $\square$

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.

4.2 Policy Execution

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

Definition 2: A hierarchical policy μ is a set of policies, one policy for each subtask in the hierarchy: $\mu = \{\mu_0, \dots, \mu_{m-1}\}$. $\square$

6. The MAXQ pseudo-reward function is defined only for transitions to terminal states, and is zero for non-terminal states.

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* task is located at the bottom and the subtask which is currently being executed is located at the top of the Task-Stack.

Under a hierarchical policy μ, we define a multi-step transition probability $P_i^{\mu} : S_i \times \mathbb{N} \times S_i \rightarrow [0,1]$ for each subtask M_i in the hierarchy, where $P_i^{\mu}(s', N|s)$ denotes the probability that hierarchical policy μ will cause the system to transition from state s to state s' in N primitive steps at subtask M_i. We also define a multi-step abstract transition probability $F_i^{\mu} : S_i \times \mathbb{N} \times S_i \rightarrow [0,1]$ for each subtask M_i under the hierarchical policy μ. The term $F_i^{\mu}(s', N|s)$ denotes the N-step abstract transition probability from state s to state s' under hierarchical policy μ at subtask M_i, where N is the number of actions taken by subtask M_i, not the number of primitive actions taken in this transition. In this paper, we use the multi-step abstract transition probability F_i^{μ} to model state transition at the subtask level, and the multi-step transition probability P_i^{μ} to model state transition at the level of primitive actions. For $N = 1$, $F_i^{\mu}(s', 1|s)$ is the transition probability for the embedded MDP at subtask M_i. We can write $F_i^{\mu}(s', 1|s)$ as $F_i^{\mu}(s'|s)$, and it can also be obtained by marginalizing $P_i^{\mu}(s', N|s)$ over N as described in Section 3.

4.3 Local versus Global Optimality

Using a hierarchy reduces the size of the space that must be searched to find a good policy. However, a 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 the previous work on hierarchical reinforcement learning, **hierarchical optimality** and **recursive optimality** (Dietterich, 2000).

Definition 3: A hierarchically optimal policy for MDP $\mathcal{M}$ is a hierarchical policy which has the best performance among all policies consistent with the given hierarchy. In other words, 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 locally 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; Precup, 2000) both converge to a hierarchically optimal policy. □

Definition 4: 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 opportunity for subtask sharing and state abstraction. The MAXQ-Q HRL algorithm (Dietterich, 2000) converges to a recursively optimal policy. □

4.4 Value Function Definitions

For recursive optimality, the goal is to find a hierarchical policy $\mu = \{\mu_0, \ldots, \mu_{m-1}\}$ 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 M_i under hierarchical policy μ must contain only the reward received during the execution of subtask M_i. We call this the **projected value function** after Dietterich (2000), and define it as follows:

Definition 5: The projected value function of a 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 the subtask 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 M_i under hierarchical policy μ must contain the reward received during the execution of subtask M_i, and the reward after subtask M_i terminates. We call this the **hierarchical value function**, following Dietterich (2000). 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 6: Ω is the space of possible values of the Task-Stack for hierarchy $\mathcal{H}$. □

Let us define joint state space $\mathcal{X} = \Omega \times \mathcal{S}$ for the hierarchy $\mathcal{H}$ as the cross product of the set of the Task-Stack values Ω and the state space $\mathcal{S}$. We also define a transition probability function of the Markov chain that results from flattening the hierarchy using the hierarchical policy μ, $m^{\mu} : \mathcal{X} \times \mathcal{X} \to [0,1]$, where $m^{\mu}(x'|x)$ denotes the probability that the hierarchical policy μ will cause the system to transition from state $x = (\omega, s)$ to state $x' = (\omega', s')$ at the level of primitive actions. We will use this transition probability function in Section 5.1 to define *global gain* for a hierarchical policy. Finally, we define the hierarchical value function using the joint state space $\mathcal{X}$ as follows:

Definition 7: A hierarchical value function for subtask M_i in state $x = (\omega, s)$ 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 M_i is a part of the Task-Stack ω and as a result is a part of the state x. So we can exclude it from the hierarchical value function notation and write $V^{\mu}(i,x)$ as $V^{\mu}(x)$. However for clarity, we use $V^{\mu}(i,x)$ in the rest of this paper.

Theorem 1: Under a hierarchical policy μ, each subtask M_i can be modeled by a SMDP consisting of components $(S_i, A_i, P_i^{\mu}, \overline{R}_i)$, where $\overline{R}_i(s,a) = \hat{V}^{\mu}(a,s)$ for all $a \in \mathcal{A}_i$. $\square$

This theorem is similar to Theorem 1 in Dietterich (2000). Using this theorem, we can define a recursive optimal policy for MDP $\mathcal{M}$ with hierarchical decomposition $\{M_0, M_1, \ldots, M_{m-1}\}$ as a hierarchical policy $\mu = \{\mu_0, \ldots, \mu_{m-1}\}$ such that for each subtask M_i, the corresponding policy μ_i is optimal for the SMDP defined by the tuple $(S_i, A_i, P_i^{\mu}, \overline{R}_i)$.

4.5 Value Function Decomposition

A value function decomposition splits the value of a state or a state-action pair into multiple additive components. Modularity in the hierarchical structure of a task allows us to carry out this decomposition along subtask boundaries. In this section, we first describe the two-part or MAXQ decomposition proposed by Dietterich (2000), and then the three-part decomposition proposed by Andre and Russell (2002). We use both decompositions in our hierarchical average reward framework depending on the type of optimality (hierarchical or recursive) that we are interested in.

The two-part value function decomposition is at the center of the MAXQ method. The purpose of this decomposition is to decompose the projected value function of the *root* task, $\hat{V}^{\mu}(0,s)$, in terms of the projected value functions of all the subtasks in the hierarchy. The projected value of subtask M_i at state s under hierarchical policy μ can be written as

$$\hat{V}^{\mu}(i,s) = \mathbf{E}\left[\sum_{k=0}^{\infty} \gamma^k r(s_k, a_k) | s_0 = s, \mu\right]. \tag{3}$$

Now, let us suppose that the first action chosen by μ_i 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 rewrite Equation 3 as

$$\hat{V}^{\mu}(i,s) = \mathbf{E}\left[\sum_{k=0}^{N-1} \gamma^k r(s_k, a_k) + \sum_{k=N}^{\infty} \gamma^k r(s_k, a_k) | s_0 = s, \mu\right]. \tag{4}$$

The first summation on the right-hand side of Equation 4 is the discounted sum of rewards for executing subtask $\mu_i(s)$ starting in state s until it terminates. In other words, it is $\hat{V}^{\mu}(\mu_i(s), s)$, the projected value function of the child task $\mu_i(s)$. The second term on the right-hand side of the equation is the projected value of state s' for the current task M_i, $\hat{V}^{\mu}(i, s')$, discounted by γ^N, where s' is the current state when subroutine $\mu_i(s)$ terminates and N is the number of transition steps from state s to state s'. We can therefore write Equation 4 in the form of a Bellman equation:

$$\hat{V}^{\mu}(i,s) = \hat{V}^{\mu}(\mu_i(s), s) + \sum_{N, s' \in S_i} P_i^{\mu}(s', N|s) \gamma^N \hat{V}^{\mu}(i, s'). \tag{5}$$

Equation 5 can be restated for the projected action-value function as follows:

$$\hat{Q}^{\mu}(i,s,a) = \hat{V}^{\mu}(a,s) + \sum_{N, s' \in S_i} P_i^{\mu}(s', N|s,a) \gamma^N \hat{Q}^{\mu}(i, s', \mu_i(s')).$$

The right-most term in this equation is the expected discounted cumulative reward of completing subtask M_i after executing action a in state s. Dietterich called this term **completion function** and denoted it by

$$C^{\mu}(i,s,a) = \sum_{N, s' \in S_i} P_i^{\mu}(s', N|s,a) \gamma^N \hat{Q}^{\mu}(i, s', \mu_i(s')). \tag{6}$$

With this definition, we can express the projected action-value function recursively as

$$\hat{Q}^{\mu}(i,s,a) = \hat{V}^{\mu}(a,s) + C^{\mu}(i,s,a), \tag{7}$$

and we can rewrite the definition for projected value function as

$$\hat{V}^{\mu}(i,s) = \begin{cases} \hat{Q}^{\mu}(i,s,\mu_i(s)) & \text{if } M_i \text{ is a non-primitive subtask,} \\ r(s,i) & \text{if } M_i \text{ is a primitive action.} \end{cases} \tag{8}$$

Equations 6 to 8 are referred to as two-part value function decomposition equations for a hierarchy under a hierarchical policy μ. These equations recursively decompose the projected value function for the *root* into the projected value functions for the individual subtasks, $M_1, \dots, M_{m-1}$, and the individual completion functions $C^{\mu}(j,s,a)$, $j = 1, \dots, m-1$. The fundamental quantities that must be stored to represent this value function decomposition are the C values for all non-primitive subtasks and the V values for all primitive actions.[7] The two-part value function decomposition is summarized graphically in Figure 2. As mentioned in Section 4.4, since the expected reward after execution of subtask M_i is not a component of the projected action-value function, the two-part value function decomposition allows only for recursive optimality.

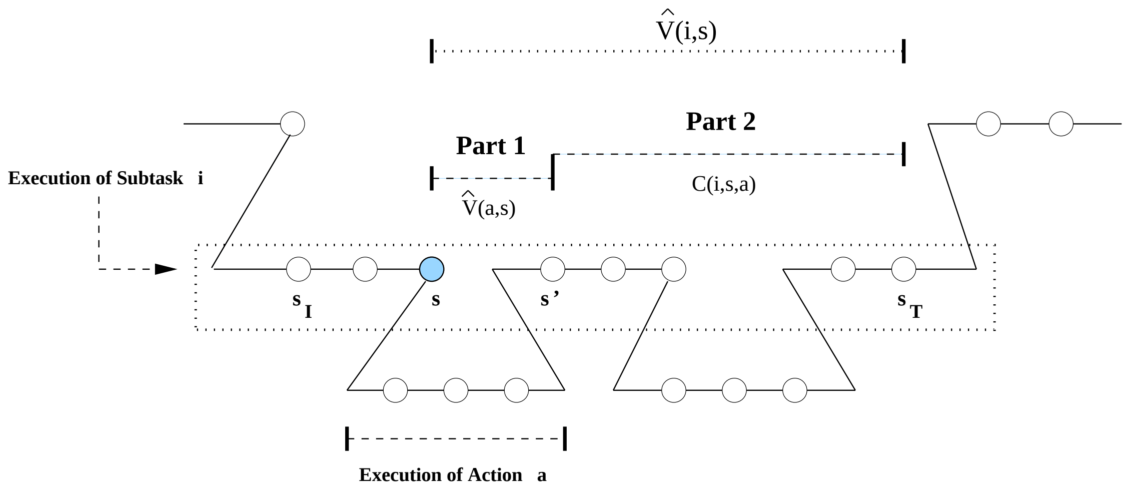

Figure 2: This figure shows the two-part decomposition for $\hat{V}(i,s)$, the projected value function of subtask M_i for the shaded state s. Each circle is a state of the SMDP visited by the agent. Subtask M_i is initiated at state s_I and terminates at state s_T. The projected value function $\hat{V}(i,s)$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

Andre and Russell (2002) proposed a three-part value function decomposition to achieve hierarchical optimality. They added a third component for the expected sum of rewards outside the current subtask to the two-part value function decomposition. This decomposition decomposes the hierarchical value function of each subtask into three parts. As shown in Figure 3, these three parts

7. The projected value function and value function are the same for a primitive action.

correspond to executing the current action (which might itself be a subtask), completing the rest of the current subtask (so far is similar to the MAXQ decomposition), and all actions outside the current subtask.

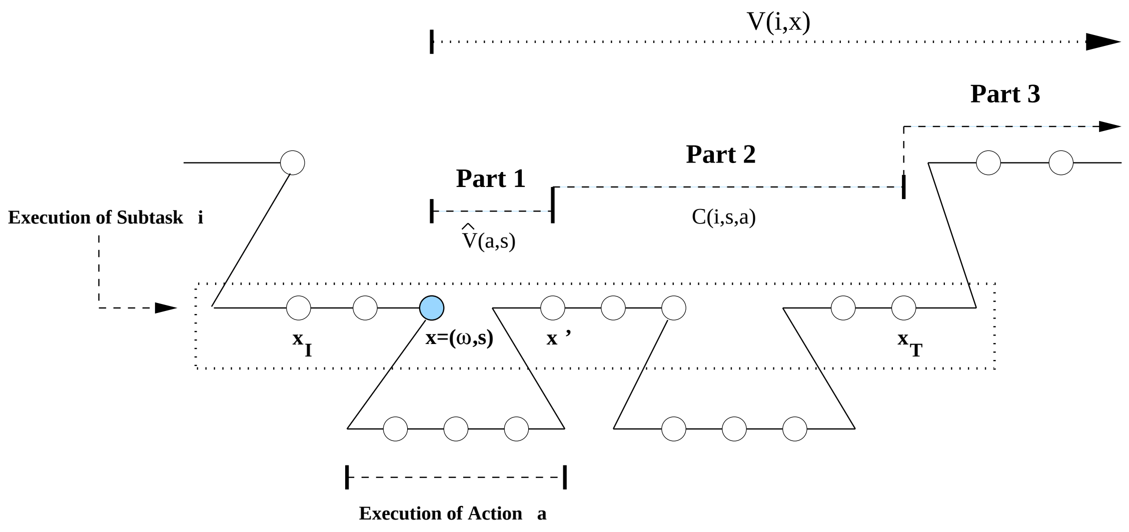

Figure 3: This figure shows the three-part decomposition for $V(i,x)$, the hierarchical value function of subtask M_i for the shaded state $x = (\omega, s)$. Each circle is a state of the SMDP visited by the agent. Subtask M_i is initiated at state x_I and terminates at state x_T. The hierarchical value function $V(i,x)$ is broken into three parts: **Part 1)** the projected value function of subtask M_a for state s, **Part 2)** the completion function, the expected discounted cumulative reward of completing subtask M_i after executing action a in state s, and **Part 3)** the sum of all rewards after termination of subtask M_i.

5. Hierarchical Average Reward Reinforcement Learning

As described in Section 1, the average reward formulation is more appropriate for a wide class of *continuing* tasks including manufacturing, scheduling, queuing, and inventory control than the more well-studied discounted framework. Moreover, average reward optimality allows for more efficient state abstraction in HRL than the discounted reward formulation. Consider the case that a set of state variables $\mathcal{Y}_a$ is irrelevant for the result distribution of action (subtask) M_a, when M_a is executed under subtask M_i. It means that for all policies executed by M_a and its descendants, and for all pairs of states s_1 and s_2 in S_i (the state space of subtask M_i) that differ only in their values for the state variables in $\mathcal{Y}_a$, we have

$$P_i^\mu(s', N|s_1, a) = P_i^\mu(s', N|s_2, a) \qquad , \qquad \forall s' \in S_i \quad , \quad \forall N \in \mathbb{N}.$$

Dietterich (2000) first defined this condition and called it *result distribution irrelevance*. If this condition is satisfied for subtask M_a, then the completion function values of its parent task M_i can be represented compactly, that is, all states $s \in S_i$ that differ only in their values for the state variables in $\mathcal{Y}_a$ have the same completion function, and therefore their completion function values can be represented only by one quantity $C^\mu(i,s,a)$, defined by Equation 6.

The definition of result distribution irrelevance can be weakened to eliminate N, the number of steps. All that is needed is that for all pairs of states s_1 and s_2 in S_i that differ only in the irrelevant state variables, $F_i^\mu(s'|s_1,a) = F_i^\mu(s'|s_2,a)$ for all $s' \in S_i$. Although the result distribution irrelevance condition would rarely be satisfied, we often find cases where the weakened result distribution irrelevance condition is true.

Under this revised definition, the compact representation of a completion function still holds in the undiscounted case, but not in the discounted formulation. Consider, for example, the *collect trash at T1* subtask in the robot trash-collection problem described in Section 4.1. No matter what location the robot has in state s, it will be at the *Dump* location when the *collect trash at T1* subtask terminates. Hence, the starting location is irrelevant to the resulting location of the robot, and $F_{Root}^\mu(s'|s_1, collect\ trash\ at\ T1) = F_{Root}^\mu(s'|s_2, collect\ trash\ at\ T1)$ for all states s_1 and s_2 in S_{Root} that differ only in the robot's location. However, if we were using discounted reward optimality, the robot's location would not be irrelevant, because the probability that the *collect trash at T1* subtask will terminate in N steps would depend on the location of the robot, which could differ in states s_1 and s_2. Different values of N will produce different amounts of discounting in Equation 6, and hence we cannot ignore the robot location when representing the completion function for the *collect trash at T1* subtask. When we use undiscounted optimality, such as average reward, we can use the weakened result distribution irrelevance and still represent the completion function for the *collect trash at T1* subtask with only one quantity.

In this section, we extend previous work on hierarchical reinforcement learning (HRL) to the average reward framework, and investigate two formulations of HRL based on the average reward SMDP model. These two formulations correspond to two notions of optimality in HRL: *hierarchical optimality* and *recursive optimality* described in Section 4.3. We present discrete-time and continuous-time algorithms to find hierarchically and recursively optimal average reward policies. In these algorithms, we assume that the overall task (the *root* of the hierarchy) is continuing. In the **hierarchically optimal average reward RL** (HAR) algorithms, the aim is to find a hierarchical policy within the space of policies defined by the hierarchical decomposition that maximizes the *global gain* (Ghavamzadeh and Mahadevan, 2002). In the **recursively optimal average reward RL** (RAR) algorithms, we treat subtasks as continuing average reward problems, where the goal at each subtask is to maximize its gain given the policies of its children (Ghavamzadeh and Mahadevan, 2001). We investigate the conditions under which the policy learned by the RAR algorithm at each subtask is independent of the context in which it is executed and therefore can be reused by other hierarchies. In Section 6, we use two automated guided vehicle (AGV) scheduling tasks as experimental testbeds to study the empirical performance of the proposed algorithms. We model the second AGV task using both discrete-time and continuous-time models. We compare the performance of our proposed algorithms with other HRL methods and a non-hierarchical average reward RL algorithm in this problem.

5.1 Hierarchically Optimal Average Reward RL Algorithm

Given the basic concepts of the average reward SMDP model described in Section 3.1, the fundamental principles of HRL, and the HRL framework in Section 4, we now describe a hierarchically optimal average reward RL formulation. Since we are interested in hierarchical optimality, we include the contents of the Task-Stack as a part of the state space of the problem. In this section, we consider HRL problems for which the following assumptions hold.

Assumption 1 (Continuing Root Task): The *root* of the hierarchy is a *continuing* task, that is, the root task continues without termination. □

Assumption 2: For every hierarchical policy μ, the Markov chain that results from flattening the hierarchy using the hierarchical policy μ, represented by the transition probability matrix m^μ (defined in Section 4.4), has a single recurrent class and a (possibly empty) set of transient states. □

If Assumptions 1 and 2 hold, the gain[8]

$$g^\mu = \left(\lim_{n\to\infty} \frac{1}{n} \sum_{t=0}^{n-1} (m^\mu)^t \right) r^\mu = \overline{m}^\mu r^\mu \tag{9}$$

is well defined for every hierarchical policy μ and does not depend on the initial state. In Equation 9, $\overline{m}^\mu$ is the *limiting matrix* of the Markov chain that results from flattening the hierarchy using the hierarchical policy μ, and satisfies the equality $\overline{m}^\mu m^\mu = \overline{m}^\mu$, and r^μ is a vector with elements $r(x,\mu(x))$, for all $x \in \mathcal{X}$. We call g^μ the **global gain** under the hierarchical policy μ. The *global gain*, g^μ, is the gain of the Markov chain that results from flattening the hierarchy using the hierarchical policy μ.

Here, we are interested in finding a hierarchical policy μ^* that maximizes the *global gain*

$$g^{\mu^*} \geq g^\mu, \qquad\qquad \text{for all } \mu. \tag{10}$$

We refer to a hierarchical policy μ^* which satisfies Equation 10 as a *hierarchically optimal average reward* policy, and to g^{μ^*} as the *hierarchically optimal average reward* or the *hierarchically optimal gain*.

We replace the value and the action-value functions in the HRL framework of Section 4 with the average-adjusted value and the average-adjusted action-value functions described in Section 3.1. The hierarchical average-adjusted value function for hierarchical policy μ and subtask M_i, denoted $H^\mu(i,x)$, is the average-adjusted sum of rewards earned by following hierarchical policy μ starting in state $x = (\omega, s)$ until M_i terminates, plus the expected average-adjusted reward outside subtask M_i

$$H^\mu(i,x) = \lim_{N\to\infty} \mathbf{E} \left\{ \sum_{k=0}^{N-1} \left[r^\mu(x_k,a_k) - g^\mu y^\mu(x_k,a_k) \right] | x_0 = x, \mu \right\}. \tag{11}$$

Here, the rewards are adjusted with g^μ, the *global gain* under the hierarchical policy μ.

Now, let us suppose that the first action chosen by μ_i is executed for a number of primitive steps N_1 and terminates in state $x_1 = (\omega, s_1)$ according to multi-step transition probability $P_i^\mu(x_1, N_1 | x, \mu_i(x))$, and then subtask M_i itself executes for N_2 steps at the level of subtask M_i (N_2 is the number of actions taken by subtask M_i, not the number of primitive actions) and terminates in state $x_2 = (\omega, s_2)$ according to multi-step abstract transition probability $F_i^\mu(x_2, N_2 | x_1)$. We can rewrite Equation 11 in the form of a Bellman equation as

8. Under the *unichain* assumption, $\overline{m}^\mu$ has equal rows. Therefore, the right hand side of Equation 9 is a vector with elements all equal to g^μ.

$$H^\mu(i,x) = r_i^\mu(x,\mu_i(x)) - g^\mu y_i^\mu(x,\mu_i(x)) + \sum_{N_1,s_1\in S_i} P_i^\mu(x_1,N_1|x,\mu_i(x)) \left[\hat{H}^\mu(i,x_1) + \sum_{N_2,s_2\in S_i} F_i^\mu(x_2,N_2|x_1) H^\mu(Parent(i),(\omega \nearrow i,s_2))\right], \tag{12}$$

where $\hat{H}^\mu(i,.)$ is the projected average-adjusted value function of the hierarchical policy μ and subtask M_i, $y_i^\mu(x,\mu_i(x))$ is the expected number of time steps until the next decision epoch of subtask M_i after taking action $\mu_i(x)$ in state x and following the hierarchical policy μ afterward, and $\omega \nearrow i$ is the content of the Task-Stack after popping subtask M_i off. Notice that $\hat{H}$ does not contain the average-adjusted rewards outside the current subtask and should be distinguished from the hierarchical average-adjusted value function H, which includes the sum of average-adjusted rewards outside the current subtask.

Since $r_i^\mu(x,\mu_i(x))$ is the expected reward between two decision epochs of subtask M_i, given that the system occupies state x at the first decision epoch, and the agent chooses action $\mu_i(x)$, we have

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

where $\mu_i(x) \searrow \omega$ is the content of the Task-Stack after pushing subtask $\mu_i(x)$ onto it. By replacing $r_i^\mu(x,\mu_i(x))$ from the above expression, Equation 12 can be written as

$$H^\mu(i,x) = \hat{H}^\mu(\mu_i(x),(\mu_i(x) \searrow \omega,s)) + \sum_{N_1,s_1\in S_i} P_i^\mu(x_1,N_1|x,\mu_i(x)) \left[\hat{H}^\mu(i,x_1) + \sum_{N_2,s_2\in S_i} F_i^\mu(x_2,N_2|x_1) H^\mu(Parent(i),(\omega \nearrow i,s_2))\right]. \tag{13}$$

We can restate Equation 13 for hierarchical average-adjusted action-value function as

$$L^\mu(i,x,a) = \hat{H}^\mu(a,(a \searrow \omega,s)) + \sum_{N_1,s_1\in S_i} P_i^\mu(x_1,N_1|x,a) \left[\hat{H}^\mu(i,x_1) + \sum_{N_2,s_2\in S_i} F_i^\mu(x_2,N_2|x_1) L^\mu(Parent(i),(\omega \nearrow i,s_2),\mu_{parent(i)}(\omega \nearrow i,s_2))\right]. \tag{14}$$

From Equation 14, we can rewrite the hierarchical average-adjusted action-value function L recursively as

$$L^\mu(i,x,a) = \hat{H}^\mu(a,(a \searrow \omega,s)) + C^\mu(i,x,a) + CE^\mu(i,x,a), \tag{15}$$

where

$$C^\mu(i,x,a) = \sum_{N_1,s_1\in S_i} P_i^\mu(x_1,N_1|x,a) \hat{H}^\mu(i,x_1), \tag{16}$$

and

$$CE^{\mu}(i,x,a) = \sum_{N_1,s_1\in S_i} P_i^{\mu}(x_1,N_1|x,a) \left[\sum_{N_2,s_2\in S_i} F_i^{\mu}(x_2,N_2|x_1)L^{\mu}(Parent(i),(\omega\nearrow i,s_2),\mu_{parent(i)}(\omega\nearrow i,s_2))\right]. \tag{17}$$

The term $C^{\mu}(i,x,a)$ is the expected average-adjusted reward of completing subtask M_i after executing action a in state $x=(\omega,s)$. We call this term **completion function** after Dietterich (2000). The term $CE^{\mu}(i,x,a)$ is the expected average-adjusted reward received after subtask M_i terminates. We call this term **external completion function** after Andre and Russell (2002).

We can rewrite the definition of $\hat{H}$ as

$$\hat{H}^{\mu}(i,x) = \begin{cases} \hat{L}^{\mu}(i,x,\mu_i(x)) & \text{if } M_i \text{ is a non-primitive subtask,} \\ r(s,i)-g^{\mu} & \text{if } M_i \text{ is a primitive action,} \end{cases} \tag{18}$$

where $\hat{L}^{\mu}$ is the projected average-adjusted action-value function and can be written as

$$\hat{L}^{\mu}(i,x,a) = \hat{H}^{\mu}(a,(a\searrow\omega,s)) + C^{\mu}(i,x,a). \tag{19}$$

Equations 15 to 19 are the decomposition equations under a hierarchical policy μ. These equations recursively decompose the hierarchical average-adjusted value function for *root*, $H^{\mu}(0,x)$, into the projected average-adjusted value functions $\hat{H}^{\mu}$ for the individual subtasks, $M_1,\ldots,M_{m-1}$, in the hierarchy, the individual completion functions $C^{\mu}(i,x,a)$, $i=1,\ldots,m-1$, and the individual external completion functions $CE^{\mu}(i,x,a)$, $i=1,\ldots,m-1$. The fundamental quantities that must be stored to represent the hierarchical average-adjusted value function decomposition are the C and the CE values for all non-primitive subtasks, the $\hat{H}$ values for all primitive actions, and the *global gain g*. The decomposition equations can be used to obtain update equations for $\hat{H}$, C, and CE in this hierarchically optimal average reward model. Pseudo-code for the discrete-time *hierarchically optimal average reward RL* (HAR) algorithm is shown in Algorithm 1. In this algorithm, primitive subtasks update their projected average-adjusted value functions $\hat{H}$ (Line 5), while non-primitive subtasks update both their completion functions C (Line 17), and external completion functions CE (Lines 20 and 22). We store only one *global gain g* and update it after each non-random primitive action (Line 7). In the update formula on Line 17, the projected average-adjusted value function $\hat{H}(a^*,(a^*\searrow\omega,s'))$ is the average-adjusted reward of executing action a^* in state $(a^*\searrow\omega,s')$ and is recursively calculated by subtask M_{a^*} and its descendants using Equations 18 and 19. Notice that the hierarchical average-adjusted action-value function L on Lines 15, 19, and 20 is recursively evaluated using Equation 15.

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

$$\hat{H}_{t+1}(i,x) \leftarrow [1-\alpha_t(i)]\hat{H}_t(i,x) + \alpha_t(i)\left[k(s,i)+r(s,i)\tau(s,i)-g_t\tau(s,i)\right],$$
$$g_{t+1} = \frac{r_{t+1}}{t_{t+1}} = \frac{r_t+k(s,i)+r(s,i)\tau(s,i)}{t_t+\tau(s,i)},$$

where $\tau(s,i)$ is the time elapsing between state s and the next state, $k(s,i)$ is the fixed reward of taking action M_i in state s, and $r(s,i)$ is the reward rate for the time between state s and the next state.

Algorithm 1 : Discrete-time hierarchically optimal average reward RL (HAR) algorithm.

```
1:  Function HAR(Task $M_i$, State $x = (\omega, s)$)
2:  let $Seq$ = {} be the sequence of states visited while executing subtask $M_i$
3:  if $M_i$ is a primitive action then
4:      execute action $M_i$ in state $x = (\omega, s)$, observe state $x' = (\omega, s')$ and reward $r(s,i)$
5:      $\hat{H}_{t+1}(i,x) \leftarrow [1-\alpha_t(i)]\hat{H}_t(i,x) + \alpha_t(i)[r(s,i) - g_t]$
6:      if $M_i$ and all its ancestors are non-random actions then
7:          update the global gain    $g_{t+1} = \frac{r_{t+1}}{n_{t+1}} = \frac{r_t + r(s,i)}{n_t + 1}$
8:      end if
9:      push state $x_1 = (\omega \nearrow i, s)$ into the beginning of $Seq$
10: else
11:     while $M_i$ has not terminated do
12:         choose action (subtask) $M_a$ according to the current exploration policy $\mu_i(x)$
13:         let $ChildSeq$ = HAR($M_a$, $(a \searrow \omega, s)$), where $ChildSeq$ is the sequence of states visited
            while executing subtask $M_a$
14:         observe result state $x' = (\omega, s')$
15:         let $a^* = \arg\max_{a' \in A_i(s')} L_t(i, x', a')$
16:         for each $x = (\omega, s)$ in $ChildSeq$ from the beginning do
17:             $C_{t+1}(i,x,a) \leftarrow [1-\alpha_t(i)]C_t(i,x,a) + \alpha_t(i)\left[\hat{H}_t(a^*, (a^* \searrow \omega, s')) + C_t(i, x', a^*)\right]$
18:             if $s' \in T_i$ ($s'$ belongs to the set of terminal states of subtask $M_i$) then
19:                 $a'' = \arg\max_{a' \in A_{Parent(i)}} L_t(Parent(i), (\omega \nearrow i, s'), a')$
20:                 $CE_{t+1}(i,x,a) \leftarrow [1-\alpha_t(i)]CE_t(i,x,a) + \alpha_t(i) L_t(Parent(i), (\omega \nearrow i, s'), a'')$
21:             else
22:                 $CE_{t+1}(i,x,a) \leftarrow [1-\alpha_t(i)]CE_t(i,x,a) + \alpha_t(i) CE_t(i, x', a^*)$
23:             end if
24:             replace state $x = (\omega, s)$ with $(\omega \nearrow i, s)$ in the $ChildSeq$
25:         end for
26:         append $ChildSeq$ onto the front of $Seq$
27:         $x = x'$
28:     end while
29: end if
30: return $Seq$
31: end HAR
```

5.2 Recursively Optimal Average Reward RL

In the previous section, we introduced discrete-time and continuous-time hierarchically optimal average reward RL (HAR) algorithms. In HAR algorithms, we define only a *global gain* for the entire hierarchy to guarantee hierarchical optimality for the overall task. HAR algorithms find a hierarchical policy that has the highest *global gain* among all policies consistent with the given hierarchy. However, there may exist subtasks where their policies 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 (see Section 4.3). Thus, the goal at *root* is to maximize its gain given the policies for its descendants. The reason seeking 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, and therefore the learned subtask can be reused by other hierarchies. This leaves open the question of what local optimality criterion should be used for each subtask in a recursively optimal average reward RL setting.

One approach pursued by Seri and Tadepalli (2002) is to optimize subtasks using their expected total average-adjusted reward with respect to the *global gain*. Seri and Tadepalli introduced 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 *global gain* at each state. In this method, the projected average-adjusted value functions with respect to the *global gain* satisfy the following equations:

$$\hat{H}^{\mu}(i,s) = \begin{cases} r(s,i) - g^{\mu} & \text{if } M_i \text{ is a primitive action,} \\ 0 & \text{if } s \in T_i \ (s \text{ is a terminal state of subtask } M_i), \\ \max_{a \in A_i(s)} [\hat{H}^{\mu}(a,s) + \sum_{N, s' \in S_i} P_i^{\mu}(s', N|s,a) \hat{H}^{\mu}(i,s')] & \text{otherwise.} \end{cases} \tag{20}$$

The first term of the last part of Equation 20, $\hat{H}^{\mu}(a,s)$, denotes the expected total average-adjusted reward during the execution of subtask M_a (the projected average adjusted value function of subtask M_a), and the second term denotes the expected total average-adjusted reward from then on until the completion of subtask M_i (the completion function of subtask M_i after execution of subtask M_a). Since the expected average-adjusted reward after execution of subtask M_i is not a component of the average-adjusted value function of subtask M_i, this approach does not necessarily allow for hierarchical optimality, as we will show in the experiments of Section 6. Moreover, the policy learned for each subtask using this approach is not context free, since each subtask maximizes its average-adjusted reward with respect to the *global gain*. However, Seri and Tadepalli (2002) showed that this method finds the hierarchically optimal average reward policy when the *result distribution invariance* condition holds.

Definition 8 (Result Distribution Invariance Condition): For all subtasks M_i and states s in the hierarchy, the distribution of states reached after the execution of any subtask M_a (M_a is one of M_i's children) is independent of the policy μ_a of subtask M_a and the policies of M_a's descendants, that is, $P_i^{\mu}(s'|s,a) = P_i(s'|s,a)$. □

In other words, states reached after the execution of a subtask cannot be changed by altering the policies of the subtask and its descendants. Note that the *result distribution invariance* condition does not hold for every problem, and therefore HH-Learning is neither hierarchically nor recursively optimal in general.

Another approach is to formulate subtasks as continuing average reward problems, where the goal at each subtask is to maximize its gain given the policies of its children (Ghavamzadeh and Mahadevan, 2001). We describe this approach in detail in Sections 5.2.1 and 5.2.2. In Section 5.2.3, we use this method to find recursively optimal average reward policies, and present discrete-time and continuous-time *recursively optimal average reward RL* (RAR) algorithms. Finally, in Section 5.2.4, we investigate the conditions under which the policy learned by RAR algorithm at each subtask is independent of the context in which it is executed and therefore can be reused by other hierarchies.

5.2.1 ROOT TASK FORMULATION

In our recursively optimal average reward RL approach, we consider those problems for which Assumption 1 (*Continuing Root Task*) and the following assumption hold.

Assumption 3 (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[9]

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

where F_0^{μ} is the multi-step abstract transition probability function of *root* under the hierarchical policy μ described in Section 4.2, and $|S_0|$ is the number of states in the state space of *root*. $\square$

Assumption 3 is equivalent to assuming that the underlying Markov chain at *root* for every hierarchical policy μ has a single recurrent class, and state s_0^* is a recurrent state. If Assumptions 1 and 3 hold, the gain at the *root* task under the hierarchical policy μ, g_0^{μ}, is well defined for every hierarchical policy μ and does not depend on the initial state. When the state space at *root* is finite or countable, the gain at *root* can be written as[10]

$$g_0^{\mu} = \frac{\bar{F}_0^{\mu} r_0^{\mu}}{\bar{F}_0^{\mu} y_0^{\mu}},$$

where r_0^{μ} and y_0^{μ} are vectors with elements $r_0^{\mu}(s, \mu_0(s))$ and $y_0^{\mu}(s, \mu_0(s))$, for all $s \in S_0$. $r_0^{\mu}(s, \mu_0(s))$ and $y_0^{\mu}(s, \mu_0(s))$ are the expected total reward and the expected number of time steps between two decision epochs at *root*, given that the system occupies state s at the first decision epoch and the agent chooses its actions according to the hierarchical policy μ. The terms F_0^{μ} and $\bar{F}_0^{\mu} = \lim_{n\to\infty} \frac{1}{n} \sum_{t=0}^{n-1} (F_0^{\mu})^t$ are the transition probability matrix and the *limiting matrix* of the embedded Markov chain at *root* for hierarchical policy μ, respectively. The transition probability F_0^{μ} is obtained by marginalizing the multi-step transition probability P_0^{μ}. The term $F_0^{\mu}(s'|s, \mu_0(s))$ denotes the probability that the SMDP at *root* occupies state s' at the next decision epoch, given that the agent chooses action $\mu_0(s)$ in state s at the current decision epoch and follows the hierarchical policy μ.

5.2.2 SUBTASK FORMULATION

In Section 5.2.1, we described the average reward formulation for the *root* task of a hierarchical decomposition. In this section, we illustrate how we formulate all other subtasks in a hierarchy as average reward problems. From now on in this section, we use subtask to refer to non-primitive subtasks in a hierarchy except *root*.

In HRL methods, we typically assume that every time a subtask M_i is executed, it starts at one of its initial states ($\in I_i$) and terminates at one of its terminal states ($\in T_i$) after a finite number of time steps. Therefore, we can make the following assumption for every subtask M_i in the hierarchy.

9. Notice that the *root* task is represented as subtask M_0 in the HRL framework described in Section 4. Thus, we use index 0 to represent components of the *root* task.
10. When the underlying Markov chain at *root* for every hierarchical policy μ has a single recurrent class, $\bar{F}_0^{\mu}$ has equal rows, and the right hand side of the equation is a vector with elements all equal to g_0^{μ}.

Under this assumption, each subtask can be considered an episodic problem and each instantiation of a subtask can be considered an episode.

Assumption 4 (Subtask Termination): There exists a dummy state s_i^* such that, for every action $a \in A_i$ and every terminal state $s \in T_i$, we have

$$r_i(s,a) = 0 \qquad \text{and} \qquad P_i(s_i^*, 1|s,a) = 1$$

and for all hierarchical stationary policies μ and non-terminal states $s \in (S_i - T_i)$, we have

$$F_i^{\mu}(s_i^*, 1|s) = 0$$

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

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

where F_i^{μ} is the multi-step abstract transition probability function of subtask M_i under the hierarchical policy μ described in Section 4.2, and $|S_i|$ is the number of states in the state space of subtask M_i. □

Although subtasks are episodic problems, when the overall task (*root* of the hierarchy) is continuing as we assumed in this section (Assumption 1), they are executed an infinite number of times, and therefore can be modeled as continuing problems using the model described in Figure 4. In this model, each subtask M_i terminates at one of its terminal states $s \in T_i$. All terminal states transit with probability 1 and reward 0 to a dummy state s_i^*. Finally, the dummy state s_i^* transits with reward zero to one of the initial states ($\in I_i$) of subtask M_i upon the next instantiation of M_i. These are dummy transitions and do not add any time-step to the cycle of subtask M_i and therefore are not taken into consideration when the average reward of subtask M_i is calculated. It is important for the validity of the model to fix the value of dummy states to zero.

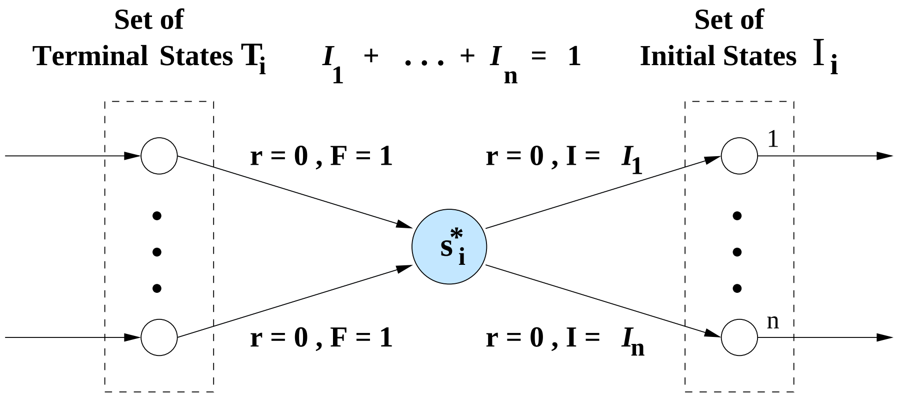

Figure 4: This figure shows how each subtask in a hierarchical decomposition of a continuing problem can be modeled as a continuing task.

Under this model, for every hierarchical policy μ, we define a new SMDP for each subtask M_i in the hierarchy with the following multi-step transition probabilities and rewards:

$$P^{\mu}_{I_i}(s',N|s,\mu_i(s)) = \begin{cases} P^{\mu}_i(s',N|s,\mu_i(s)) & s,s' \neq s^*_i \quad , \quad \forall N \in \mathbb{N}, \\ I_i(s') & s = s^*_i \quad , \quad N=1, \\ 1 & s' = s^*_i \quad , \quad s \in T_i \quad , \quad N=1, \\ 0 & \text{otherwise.} \end{cases}$$

$$r^{\mu}_{I_i}(s,\mu_i(s)) = \begin{cases} r^{\mu}_i(s,\mu_i(s)) & s \in (S_i - T_i), \\ 0 & s = s^*_i \quad \text{or} \quad s \in T_i. \end{cases} \tag{21}$$

where $I_i(s)$ is the probability that subtask M_i starts at state $s \in I_i$. The SMDP defined by Equation 21 has an embedded MDP with the following transition probability function:

$$F^{\mu}_{I_i}(s'|s,\mu_i(s)) = \begin{cases} F^{\mu}_i(s'|s,\mu_i(s)) & s,s' \neq s^*_i, \\ I_i(s') & s = s^*_i, \\ 1 & s' = s^*_i \quad , \quad s \in T_i, \\ 0 & s' = s^*_i \quad , \quad s \in (S_i - T_i). \end{cases} \tag{22}$$

Lemma 1: Let Assumption 4 (*Subtask Termination*) hold. Then, for every $F^{\mu}_{I_i}$ and every state $s \in S_i$, we have $\sum_{N=1}^{|S_i|} F^{\mu}_{I_i}(s^*_i,N|s) > 0$.[11] □

Lemma 1 is equivalent to assuming that for every subtask M_i in the hierarchy, the underlying Markov chain for every hierarchical policy μ has a single recurrent class and state s^*_i is its recurrent state. Under this model, the gain of subtask M_i under the hierarchical policy μ, g^{μ}_i, is well defined for every hierarchical policy μ and does not depend on the initial state. When the state space of subtask M_i is finite or countable, the gain of subtask M_i can be written as[12]

$$g^{\mu}_i = \frac{\bar{F}^{\mu}_{I_i} r^{\mu}_{I_i}}{\bar{F}^{\mu}_{I_i} y^{\mu}_{I_i}},$$

where $r^{\mu}_{I_i}$ and $y^{\mu}_{I_i}$ are vectors with elements $r^{\mu}_{I_i}(s,\mu_i(s))$ and $y^{\mu}_{I_i}(s,\mu_i(s))$, for all $s \in S_i$. $r^{\mu}_{I_i}(s,\mu_i(s))$ and $y^{\mu}_{I_i}(s,\mu_i(s))$ are the expected total reward and the expected number of time steps between two decision epochs of the SMDP defined by Equation 21 at subtask M_i, given that the system occupies state s at the first decision epoch and the agent chooses its actions according to hierarchical policy μ. The term $\bar{F}^{\mu}_{I_i} = \lim_{n\to\infty} \frac{1}{n} \sum_{t=0}^{n-1} (F^{\mu}_{I_i})^t$ is the *limiting matrix* of the Markov chain defined by Equation 22 at subtask M_i.

5.2.3 A RECURSIVELY OPTIMAL AVERAGE REWARD RL ALGORITHM

In this section, we present discrete-time and continuous-time recursively optimal average reward RL (RAR) algorithms using the formulation described in Sections 5.2.1 and 5.2.2. We consider

11. This lemma is a restatement of Lemma 5 on page 34 of Peter Marbach's thesis (Marbach, 1998).
12. When the underlying Markov chain for every hierarchical policy μ at subtask M_i has a single recurrent class, $\bar{F}^{\mu}_{I_i}$ has equal rows, and thus the right hand side of the equation is a vector with elements all equal to g^{μ}_i.

problems for which Assumptions 1, 3, and 4 (*Continuing Root-Task*, *Root-Task Recurrence*, and *Subtask Termination*) hold, *root* is modeled as an average reward problem as described in Section 5.2.1, and every other non-primitive subtask in the hierarchy is modeled as an average reward problem using the model described in Section 5.2.2. Under these assumptions, the average reward for every non-primitive subtask in the hierarchy including *root* is well defined for every hierarchical policy and does not vary with initial state. Since we are interested in finding a recursively optimal average reward policy, we do not need to include the contents of the Task-Stack as a part of the state space of the problem. We also replace the projected value and action-value functions in the hierarchical model of Section 4 with the projected average-adjusted value and projected average-adjusted action-value functions described in Section 3.1.

We show how the overall projected average-adjusted value function $\hat{H}^{\mu}(0,s)$ is decomposed into a collection of projected average-adjusted value functions of individual subtasks $\hat{H}^{\mu}(i,s), i = 1,\dots,m-1$, in RAR algorithm. The projected average-adjusted value function of hierarchical policy μ at subtask M_i is the average-adjusted (with respect to the local gain g_i^{μ}) sum of rewards earned by following policy μ_i and the policies of all descendants of subtask M_i starting in state s until M_i terminates. Now, let us suppose that the first action chosen by μ_i is executed for a number of primitive steps N and terminates in state s' according to multi-step transition probability $P_i^{\mu}(s',N|s,\mu_i(s))$. We can write the projected average-adjusted value function in the form of a Bellman equation as

$$\hat{H}^{\mu}(i,s) = r_i^{\mu}(s,\mu_i(s)) - g_i^{\mu} y_i^{\mu}(s,\mu_i(s)) + \sum_{N,s'\in S_i} P_i^{\mu}(s',N|s,\mu_i(s))\hat{H}^{\mu}(i,s'). \tag{23}$$

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

$$r_i^{\mu}(s,\mu_i(s)) = \begin{cases} \hat{V}^{\mu}(\mu_i(s),s) = \hat{H}^{\mu}(\mu_i(s),s) + g^{\mu}_{\mu_i(s)} y_i^{\mu}(s,\mu_i(s)) & \\ & \text{if } M_{\mu_i(s)} \text{ is a non-primitive subtask,} \\ \hat{V}^{\mu}(\mu_i(s),s) & \\ & \text{if } M_{\mu_i(s)} \text{ is a primitive action.} \end{cases}$$

By replacing $r_i^{\mu}(s,\mu_i(s))$ from the above expression, and the fact that $y_i^{\mu}(s,\mu_i(s))$ equals 1 when $M_{\mu_i(s)}$ is a primitive action, Equation 23 can be written as

$$\hat{H}^{\mu}(i,s) = \begin{cases} \hat{H}^{\mu}(\mu_i(s),s) - (g_i^{\mu} - g^{\mu}_{\mu_i(s)}) y_i^{\mu}(s,\mu_i(s)) + \sum_{N,s'\in S_i} P_i^{\mu}(s',N|s,\mu_i(s))\hat{H}^{\mu}(i,s') & \\ & \text{if } M_{\mu_i(s)} \text{ is a non-primitive subtask,} \\ \hat{V}^{\mu}(\mu_i(s),s) - g_i^{\mu} + \sum_{s'\in S_i} P_i^{\mu}(s'|s,\mu_i(s))\hat{H}^{\mu}(i,s') & \\ & \text{if } M_{\mu_i(s)} \text{ is a primitive action.} \end{cases} \tag{24}$$

We can restate Equations 24 for the projected action-value function as follows:

$$\hat{L}^{\mu}(i,s,a) = \begin{cases} \hat{H}^{\mu}(a,s) - (g_i^{\mu} - g_a^{\mu})y_i^{\mu}(s,a) + \sum_{N,s' \in S_i} P_i^{\mu}(s',N|s,a)\hat{L}^{\mu}(i,s',\mu_i(s')) \\ \qquad\qquad\qquad \text{if } M_a \text{ is a non-primitive subtask,} \\ \hat{V}^{\mu}(a,s) - g_i^{\mu} + \sum_{s' \in S_i} P_i^{\mu}(s'|s,a)\hat{L}^{\mu}(i,s',\mu_i(s')) \\ \qquad\qquad\qquad \text{if } M_a \text{ is a primitive action.} \end{cases} \tag{25}$$

By defining

$$C^{\mu}(i,s,a) = \begin{cases} -(g_i^{\mu} - g_a^{\mu})y_i^{\mu}(s,a) + \sum_{N,s' \in S_i} P_i^{\mu}(s',N|s,a)\hat{L}^{\mu}(i,s',\mu_i(s')) \\ \qquad\qquad\qquad \text{if } M_a \text{ is a non-primitive subtask,} \\ -g_i^{\mu} + \sum_{s' \in S_i} P_i^{\mu}(s'|s,a)\hat{L}^{\mu}(i,s',\mu_i(s')) \\ \qquad\qquad\qquad \text{if } M_a \text{ is a primitive action,} \end{cases} \tag{26}$$

we can express the average-adjusted action-value function $\hat{L}^{\mu}$ recursively as

$$\hat{L}^{\mu}(i,s,a) = \begin{cases} \hat{H}^{\mu}(a,s) + C^{\mu}(i,s,a) & \text{if } M_a \text{ is a non-primitive subtask,} \\ \hat{V}^{\mu}(a,s) + C^{\mu}(i,s,a) & \text{if } M_a \text{ is a primitive action,} \end{cases} \tag{27}$$

where

$$\hat{H}^{\mu}(i,s) = \hat{L}^{\mu}(i,s,\mu_i(s)). \tag{28}$$

We call $C^{\mu}(i,s,a)$ defined by Equation 26 **completion function**.

Equations 24 to 28 are the decomposition equations for the projected average-adjusted value and projected average-adjusted action-value functions. They can be used to obtain update formulas for $\hat{H}$ and C in this recursively optimal average reward model. Pseudo-code for the discrete-time *recursively optimal average reward RL* (RAR) algorithm is shown in Algorithm 2. In this algorithm, a gain is defined for every non-primitive subtask in the hierarchy and this gain is updated every time a subtask is non-randomly chosen. Primitive subtasks store their projected value functions, and update them using the equation on Line 5. Non-primitive subtasks store their completion functions and gains, and update them using equations on Lines 17, 19, and 23. The projected average-adjusted action-value function $\hat{L}$ on Lines 12, 17, and 19 is recursively calculated using Equations 26 to 28.

This algorithm can be easily extended to continuous-time (Ghavamzadeh and Mahadevan, 2001). In continuous-time RAR algorithm, in addition to visited state and reward, we need to insert the execution time of primitive actions τ into the sequence *Seq*. Therefore, $N = N+1$ on Line 15 of the algorithm is changed to $T = T + \tau$. We also need to change the update formulas for $\hat{V}$, C, and g_i on Lines 5, 17, 19, and 23 as

$$\hat{V}_{t+1}(i,s) \leftarrow [1-\alpha_t(i)]\hat{H}_t(i,s) + \alpha_t(i)\,[k(s,i) + r(s,i)\tau(s,i)]\,,$$

$$C_{t+1}(i,s,a) \leftarrow [1-\alpha_t(i)]C_t(i,s,a) + \alpha_t(i)[\hat{L}_t(i,s',a^*) - g_t(i)T],$$

$$C_{t+1}(i,s,a) \leftarrow [1-\alpha_t(i)]C_t(i,s,a) + \alpha_t(i)[\hat{L}_t(i,s',a^*) - (g_t(i) - g_t(a))T],$$

Algorithm 2 : Discrete-time recursively optimal average reward RL (RAR) algorithm.

```
1:  Function RAR(Task $M_i$, State $s$)
2:  let $Seq = \{\}$ be the sequence of (state visited, reward) while executing subtask $M_i$
3:  if $M_i$ is a primitive action then
4:      execute action $M_i$ in state $s$, observe state $s'$ and reward $r(s,i)$
5:      $\hat{V}_{t+1}(i,s) \leftarrow [1-\alpha_t(i)]\hat{V}_t(i,s) + \alpha_t(i)r(s,i)$
6:      push (state $s$, reward $r(s,i)$) into the beginning of $Seq$
7:  else
8:      while $M_i$ has not terminated do
9:          choose action (subtask) $M_a$ according to the current exploration policy $\mu_i(s)$
10:         let $ChildSeq = $ RAR($M_a$, $s$), where $ChildSeq$ is the sequence of (state visited, reward) while
            executing subtask $M_a$
11:         observe result state $s'$
12:         let $a^* = \arg\max_{a' \in A_i(s')} \hat{L}_t(i,s',a')$
13:         let $N = 0$;   $\rho = 0$;
14:         for each ($s$, $r$) in $ChildSeq$ from the beginning do
15:             $N = N + 1$;   $\rho = \rho + r$;
16:             if $a$ is a primitive action then
17:                 $C_{t+1}(i,s,a) \leftarrow [1-\alpha_t(i)]C_t(i,s,a) + \alpha_t(i)[\hat{L}_t(i,s',a^*) - g_t(i)N]$
18:             else
19:                 $C_{t+1}(i,s,a) \leftarrow [1-\alpha_t(i)]C_t(i,s,a) + \alpha_t(i)[\hat{L}_t(i,s',a^*) - (g_t(i) - g_t(a))N]$
20:             end if
21:         end for
22:         if $a$ and all its ancestors are non-random actions then
23:             update the gain of subtask $M_i$          $g_{t+1}(i) = \frac{r_{t+1}(i)}{n_{t+1}(i)} = \frac{r_t(i)+\rho}{n_t(i)+N}$
24:         end if
25:         append $ChildSeq$ onto the front of $Seq$
26:         $s = s'$
27:     end while
28: end if
29: return $Seq$
30: end RAR
```

$$g_{t+1}(i) = \frac{r_{t+1}(i)}{t_{t+1}(i)} = \frac{r_t(i) + \rho}{t_t(i) + T},$$

where $\tau(s,i)$ is the time elapsing between state s and the next state, $k(s,i)$ is the fixed reward of taking action M_i in state s, and $r(s,i)$ is the reward rate for the time between state s and the next state.

5.2.4 ANALYSIS OF THE RAR ALGORITHM

In this section, we study the optimality achieved by RAR algorithm. As described earlier, the expected average-adjusted sum of rewards after execution of subtask M_i is not a component of the average-adjusted value function of subtask M_i in RAR algorithm. Therefore, the algorithm fails to find a hierarchically optimal average reward policy in general, as was discussed in Seri and Tadepalli (2002) and will be demonstrated in the experiments of Section 6.

To achieve recursive optimality, the policy learned for each subtask must be context free, that is, each subtask should maximize its local gain given the policies of its descendants. In RAR algorithm, although each subtask maximizes its local gain given the policies of its descendants, the policy learned for each subtask is not necessarily context free, and as a result, the algorithm does not find a recursively optimal average reward policy in general. The reason is, the local gain g_i for each subtask M_i does not depend only on the policies of its descendants. The local gain g_i is the gain of the SMDP defined by Equation 21 and therefore depends on the initial state distribution $I_i(s)$. The initial state distribution $I_i(s)$, the probability of being in state s at the next instantiation of subtask M_i, depends not only on the policies of subtask M_i and all its descendants, but also on the policies of its ancestors. It makes the local gain g_i learned by RAR algorithm context dependent. However, the algorithm finds a recursively optimal average reward policy when the *initial distribution invariance* (IDI) condition holds. Under the IDI condition, the policy learned by RAR algorithm at each subtask is independent of the context in which it is executed and therefore can be reused by other hierarchies.

Definition 9 (Initial Distribution Invariance Condition): The initial state distribution for each non-primitive subtask in the hierarchy is independent of the policies of its ancestors. □

In other words, the initial state distribution for each non-primitive subtask cannot be changed by altering the policies of its ancestors. One special case that satisfies the IDI condition is when each non-primitive subtask in the hierarchy has only one initiation state, $|I_i| = 1, i = 1, \dots, m-1$, and M_i is a non-primitive subtask.

6. Experimental Results

The goal of this section is to show the type of optimality achieved by the *hierarchically optimal average reward RL* (HAR) and the *recursively optimal average reward RL* (RAR) algorithms proposed in Sections 5.1 and 5.2, as well as their performance and speed compared to other algorithms. We describe two sets of experiments. In Section 6.1, we apply five HRL algorithms to a simple discrete-time automated guided vehicle (AGV) scheduling problem. Since we use a hierarchical task decomposition in which the hierarchically and recursively optimal policies are different for this problem, our experimental results clearly demonstrate the difference between the optimality achieved by these algorithms. Then, we turn to a relatively large AGV scheduling task in Section 6.2. We model this AGV scheduling task as discrete time and continuous-time problems. In the discrete-time model, we compare the performance of HAR and RAR algorithms with a hierarchically optimal discounted reward algorithm and a recursively optimal discounted reward algorithm, as well as a non-hierarchical (flat) average reward algorithm. In the continuous-time model, we compare the performance of HAR and RAR algorithms with a recursively optimal discounted reward algorithm. We do not use pseudo-reward or reward shaping in the experiments of this section. The first problem is simple and can be solved easily without reward shaping. There are rewards associated with the terminal states of the subtasks in the original MDP of the second problem. Therefore, the agent can find out about the desirability of the terminal states upon completing the subtasks, without using pseudo-reward or reward shaping.

6.1 A Simple AGV Scheduling Problem

In this section, we apply the *discrete-time hierarchically optimal average reward RL* (HAR) algorithm described in Section 5.1, the *discrete-time recursively optimal average reward RL* (RAR) algorithm described in Section 5.2, and *HH-Learning*, the algorithm proposed by Seri and Tadepalli (2002), to a small AGV scheduling task. We also test MAXQ-Q, the recursively optimal discounted reward HRL algorithm proposed by Dietterich (2000), and a *hierarchically optimal discounted reward RL* algorithm (HDR) on this task. The HDR algorithm is an extension of MAXQ-Q using the three-part value function decomposition (Andre and Russell, 2002) described in Section 4.5.

A simple AGV domain is depicted in Figure 5. In this domain there are two machines $M1$ and $M2$ that produce parts to be delivered to the 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 is delivered to destination $G1$ and a reward of 1 when part 2 is delivered to destination $G2$. The AGV receives a reward of -1 for all other actions. Note that within subtasks "Go to Machine" and "Go to Door", the agent must choose which machine to go to, and which door to pass through, respectively. This task is deterministic and the state variables are AGV's *location* and *status* (empty, carry part 1, carry part 2), which is a total of $26 \times 3 = 78$ states. In all experiments, we use the task graph shown in Figure 5 and set the discount factor to 0.9 for the discounted reward algorithms. We tried several discounting factors and $\gamma = 0.9$ yielded the best performance. Using this task graph, hierarchically and recursively 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.

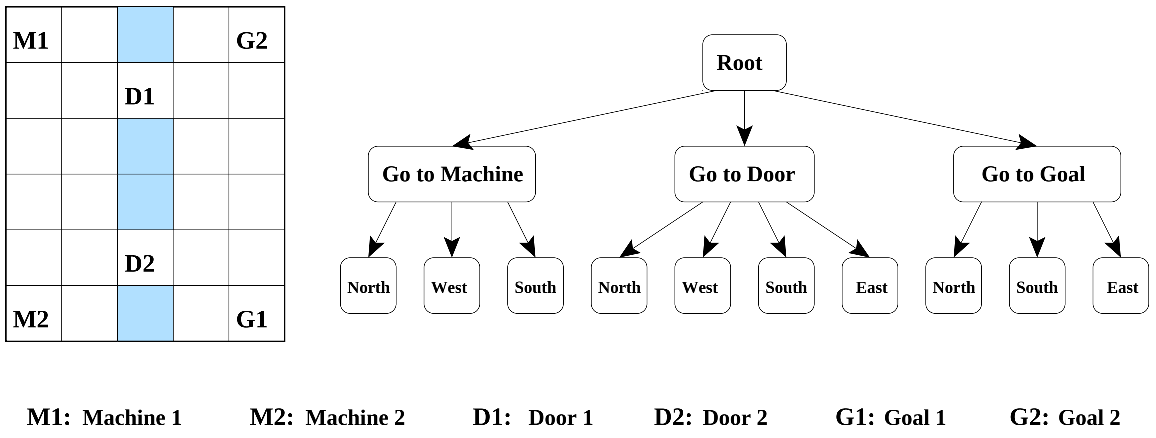

Figure 5: A simple AGV scheduling task and its associated task graph. Note that within subtasks "Go to Machine" and "Go to Door", the AGV must choose which machine to go to, and which door to pass through, respectively.

Among the algorithms we applied to this task, the hierarchically optimal average reward RL (HAR) and the hierarchically optimal discounted reward RL (HDR) algorithms find the hierarchically optimal policy, where the other algorithms only learn the recursively optimal policy. Figure 6 demonstrates the throughput of the system for the above algorithms. The hierarchically optimal algorithms learn more slowly than the recursively optimal algorithms due to more parameters to be learned. Since this problem is deterministic, the HH-Learning algorithm, which is the only model-based RL algorithm used in this experiment, learns the model of the environment quickly, and therefore converges much faster than the other algorithms. In this figure, the throughput of the system is the number of parts deposited at destination stations weighted by their rewards ($part1 \times 20 + part2 \times 1$) in 250 time steps. Each experiment was conducted twenty times and the results were averaged.

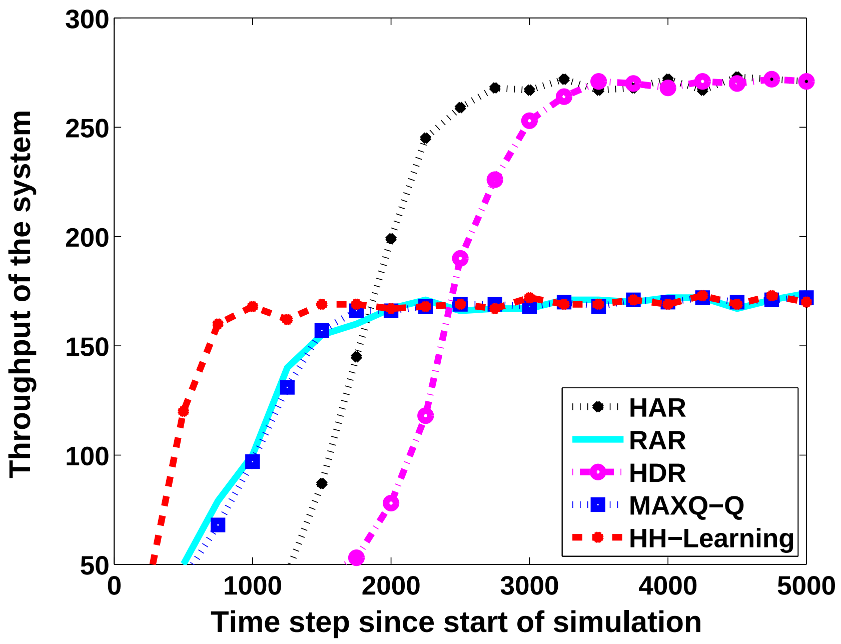

Figure 6: This figure shows that HDR and HAR algorithms (the two top curves) learn the hierarchically optimal policy while RAR, MAXQ-Q, and HH-Learning (the three bottom curves) only find the recursively optimal policy for the small AGV scheduling task.

6.2 AGV Scheduling Problem (Discrete and Continuous Time Models)

In this section, we describe two sets of experiments on the AGV scheduling problem shown in Figure 7. $M1$ to $M3$ are workstations in this environment. Parts of type i have to be carried to the drop-off station at workstation i (D_i), and the assembled parts brought back from pick-up stations of workstations (P_i's) to the warehouse. The AGV travel is unidirectional as the arrows show. The AGV receives a reward of 20 when it picks up a part at the warehouse, delivers a part to a drop-off station, picks up an assembled part from a pick-up station, or delivers an assembled part to the warehouse. It also gets a reward of -5 when it attempts to execute Put1–Put3, Pick1–Pick3, Load1–Load3, Unload, and Idle actions illegally. There is a reward of -1 for all other actions. We model this AGV scheduling task using both discrete-time and continuous-time models. In the discrete-time model, we show the performance of four HRL algorithms: *hierarchically optimal average reward RL* (HAR), *recursively optimal average reward RL* (RAR), *hierarchically optimal discounted reward RL* (HDR), and *recursively optimal discounted reward RL* (MAXQ-Q), as well as a *non-hierarchical average reward* algorithm. In the continuous-time model, we compare the performance of HAR and RAR algorithms with the continuous-time MAXQ-Q algorithm (Ghavamzadeh and Mahadevan, 2001). We use the task graph shown in Figure 8 in both experiments. Using this task graph, hierarchical and recursive optimal policies are the same, and therefore hierarchical and recursive optimal algorithms should converge to the same performance.

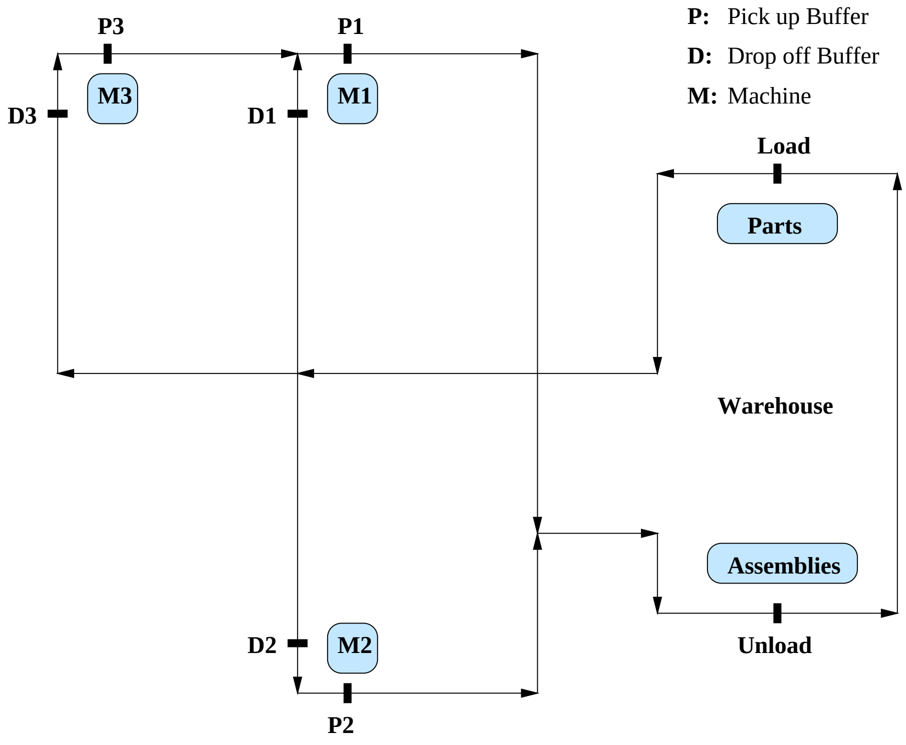

Figure 7: An AGV scheduling task. An AGV agent (not shown) carries raw materials and finished parts between machines (M1–M3) and warehouse.

The state of the environment consists of the number of parts in the pick-up and drop-off stations of each machine and whether the warehouse contains parts of each of the three types. In addition, the agent keeps track of its own location and status as a part of its state space. Thus, in the flat case, the state space consists of 33 locations, 6 buffers of size 2, 7 possible states of the AGV (carrying part1–part3, carrying assembly1–assembly3, empty), and 2 values for each part in the warehouse, that is, $33 \times 3^6 \times 7 \times 2^3 = 1,347,192$ states. Since there are 14 primitive actions (Left, Forward, Right, Put1–Put3, Pick1–Pick3, Load1–Load3, Unload, Idle) in this problem, the total number of parameters that must be learned (the size of the action-value function table) in the

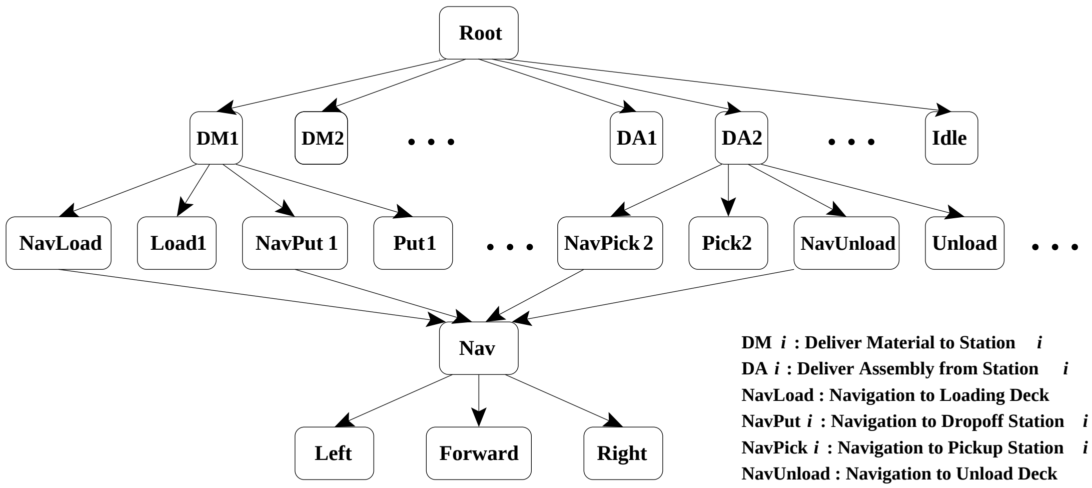
Root
DM1
DM2
DA1
DA2
Idle
NavLoad
Load1
NavPut 1
Put1
NavPick 2
Pick2
NavUnload
Unload
Nav
Left
Forward
Right
DM i : Deliver Material to Station i
DA i : Deliver Assembly from Station i
NavLoad : Navigation to Loading Deck
NavPut i : Navigation to Dropoff Station i
NavPick i : Navigation to Pickup Station i
NavUnload : Navigation to Unload Deck

Figure 8: Task graph for the AGV scheduling task.

flat case is $1,347,192 \times 14 = 18,860,688$. *State abstraction* helps in reducing the state space considerably. Only the relevant state variables are used while storing the value functions in each node of the task graph. For example, for the 8 *Navigation* subtasks, only the location state variable is relevant and each of these subtasks can be learned with only 33 values. Tables 1 and 2 show the relevant state variables and the number of relevant states for non-primitive and primitive subtasks in the AGV scheduling problem, respectively. These tables also contain the number of parameters that must be stored by these subtasks, that is, completion function values, C, and external completion function values, CE, for non-primitive subtasks, and V values for primitive actions. The number of parameters that must be stored by a subtask is its number of relevant states times its number of children. Using Tables 1 and 2, the total number of parameters that must be learned in hierarchically and recursively optimal algorithms for this problem are equal to $10,809,150$ and $10,834,890$, respectively.[13] Both these numbers are smaller than the number of parameters that must be learned in the flat case. This state abstraction gives us a compact way of representing the value functions and speeds up the hierarchical algorithms.

The discrete-time experimental results were generated with the following model parameters. The inter-arrival time for parts at the warehouse is distributed according to a Poisson distribution.[14] 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 Gamma random variables.[15] Since this is a discrete-time model for the AGV problem, we round the time x generated by these Gamma distributions to the nearest integer less than or equal to x. Table 3 shows the parameters of the

13. Note that in both recursively and hierarchically optimal algorithms, only one completion function needs to be defined at the *Root* of the hierarchy.
14. A random variable $x = 0, 1, 2, \ldots$ is said to be a Poisson random variable with parameter $\lambda > 0$, if $\Pr(x = n) = e^{-\lambda} \frac{\lambda^n}{n!}$. The mean and variance of the Poisson random variable x are both equal to λ.
15. A random variable $x \geq 0$ is said to have a Gamma distribution with parameters $(\kappa, \lambda),\ \kappa, \lambda > 0$, if its density function is given by $f(x) = \frac{\lambda e^{-\lambda x} (\lambda x)^{\kappa-1}}{\Gamma(\kappa)}$. The mean and variance of the Gamma random variable x are $\frac{\kappa}{\lambda}$ and $\frac{\kappa}{\lambda^2}$ respectively.

Subtask	**Relevant States**	**Num. of Relevant States**	**Num. of C (CE) Values**
Root	entire state space	$33 \times 3^6 \times 7 \times 2^3 = 1,347,192$	$1,347,192 \times 7 = 9,430,344$
DM*i*	AGV location, AGV status, status of input buffer *i*, whether part *i* exists in the warehouse	$33 \times 7 \times 3 \times 2 = 1,386$	$1,386 \times 4 = 5,544$
DA*i*	AGV location, AGV status, status of output buffer *i*	$33 \times 7 \times 3 = 693$	$693 \times 4 = 2,772$
Nav	AGV location	33	$33 \times 3 = 99$

Table 1: This table shows the relevant state variables, the number of relevant states, and the number of completion (external completion) function values C (CE) for non-primitive subtasks in the AGV scheduling problem.

Subtask	**Relevant States**	**Num. of Relevant States = Num. of V Values**
Left , Forward , Right	AGV location	33
Put*i* , Pick*i*	AGV location, AGV status, status of input/output buffer *i*	$33 \times 7 \times 3 = 693$
Load*i*	AGV location, AGV status, whether part *i* exists in the warehouse	$33 \times 7 \times 2 = 462$
Unload	AGV location, AGV status	$33 \times 7 = 231$
Idle	entire state space	$33 \times 3^6 \times 7 \times 2^3 = 1,347,192$

Table 2: This table shows the relevant state variables and the number of relevant states (which is equal to the number of V values) for primitive actions in the AGV scheduling problem.

discrete-time model. In these experiments, we used discount factors 0.9 and 0.95 for the discounted reward algorithms. Using the discount factor of 0.95 yielded a better performance.

Parameter	**Distribution**	**Mean** (steps)	**Var** (steps)
Assembly Time for Part1	Gamma ($\kappa = 180, \lambda = 3$)	60	20
Assembly Time for Part2	Gamma ($\kappa = 250, \lambda = 2.5$)	100	40
Assembly Time for Part3	Gamma ($\kappa = 288, \lambda = 2.4$)	120	50
Inter-Arrival Time for Parts	Poisson ($\lambda = 80$)	80	80

Table 3: 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 uniformly distributed. The inter-arrival time for parts at the warehouse is distributed according to a Poisson distribution. 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 Gamma random variables. Table 4 contains the parameters of the continuous-time model.

Parameter	**Distribution**	**Mean (sec)**	**Var (sec)**
Assembly Time for Part1	Gamma ($\kappa = 180, \lambda = 3$)	60	20
Assembly Time for Part2	Gamma ($\kappa = 250, \lambda = 2.5$)	100	40
Assembly Time for Part3	Gamma ($\kappa = 288, \lambda = 2.4$)	120	50
Inter-Arrival Time for Parts	Poisson ($\lambda = 80$)	80	80
Execution Time for Primitive Actions	Uniform ($6 < t < 14$)	10	5.33

Table 4: Parameters of the Continuous-Time Model

Figure 9 compares the performance of the discrete-time hierarchically (HAR) and recursively (RAR) optimal average reward algorithms with the performance of the discrete-time discounted reward hierarchically (HDR) and recursively (MAXQ-Q) optimal algorithms on the AGV scheduling problem. All these algorithms eventually converge to the same system performance. The hierarchically optimal algorithms learn slower than the recursively optimal algorithms due to more parameters to be learned. This figure also shows the performance of relative value iteration (RVI) Q-learning (Abounadi et al., 2001), a non-hierarchical average reward RL algorithm. As shown in this figure, RVI Q-learning does not converge to the optimal throughput after 10^5 time steps. Figure 10 shows the performance of RVI Q-learning for 3×10^6 time steps. The RVI Q-learning algorithm converges to the optimal performance after over 2×10^6 time steps, where the hierarchical algorithms converge to this performance in less than 10^5 time steps as shown in Figure 9. The difference in convergence speed between flat and hierarchical algorithms becomes more significant as we increase the number of states. All the graphs in these figures are averaged over twenty runs, except the RVI Q-learning graph, which is averaged over thirty runs.

With the inter-arrival time and assembly-time parameters used in this experiment, there are time steps in which there is no part left in the warehouse. This is when the AGV must learn to take the *idle* action and wait until new parts appear in the warehouse. At first, the AGV does not serve the machines properly, and therefore parts are accumulated in the warehouse. As the AGV learns to serve the machines, the system performance goes up until the parts accumulated in the warehouse at the first learning steps are all processed. Then, the system performance goes down and eventually converges to its optimal value. This is why in Figures 9 and 10, the performance of the algorithms reaches a peak before it converges to its optimal value.

Figure 11 compares the performance of the continuous-time hierarchically (HAR) and recursively (RAR) optimal average reward algorithms with the performance of continuous-time MAXQ-Q, a continuous-time recursively optimal discounted reward RL algorithm, first presented by Ghavamzadeh and Mahadevan (2001), on the AGV scheduling problem. All the algorithms converge to the same system performance. The discounted reward algorithm, continuous-time MAXQ-Q, learns faster than both the average reward algorithms, HAR and RAR. Moreover, the hierarchically optimal average reward algorithm (HAR) learns more slowly than the recursively optimal average reward algorithm (RAR) due to more parameters to be learned. All the graphs in this figure are averaged over fifty runs.

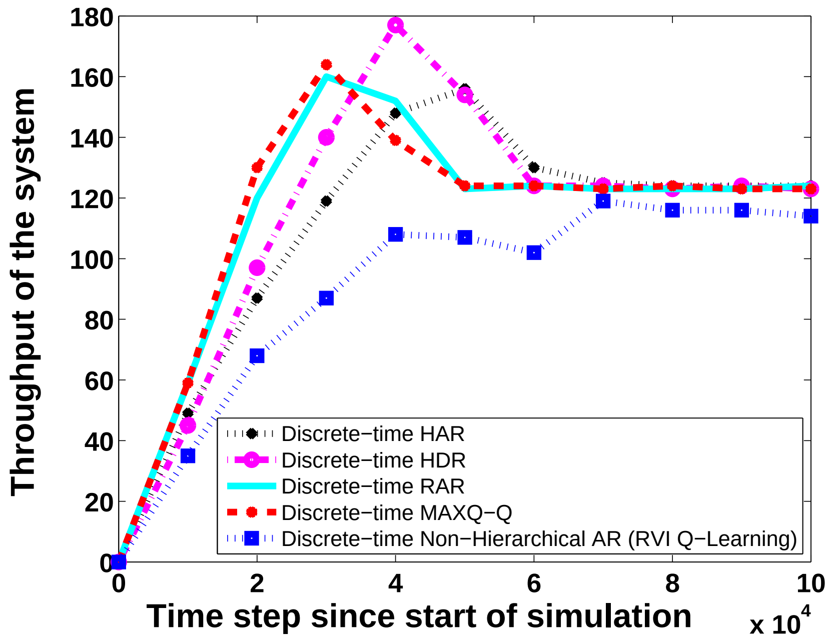

Figure 9: This figure compares the performance of the discrete-time hierarchically (HAR) and recursively (RAR) optimal average reward algorithms with the performance of the hierarchically (HDR) and recursively (MAXQ-Q) optimal discounted reward algorithms on the AGV scheduling problem. It also demonstrates the faster convergence of the hierarchical algorithms comparing to RVI Q-learning, a non-hierarchical average reward RL algorithm.

7. Conclusions and Future Work

Hierarchical reinforcement learning (HRL) is a general framework for scaling reinforcement learning (RL) to problems with large state spaces by using task (or action) structure to restrict the space of policies. Prior work in HRL, including hierarchies of abstract machines (HAMs) (Parr, 1998), options (Sutton et al., 1999; Precup, 2000), MAXQ (Dietterich, 2000), and programmable HAMs (PHAMs) (Andre and Russell, 2001; Andre, 2003), has been limited to the discrete-time discounted reward semi-Markov decision process (SMDP) model. These methods aim to find policies that maximize the long-term discounted sum of rewards. On the other hand, the average reward optimality criterion has been shown to be more appropriate for a wide class of continuing tasks than the more well-studied discounted formulation. A primary goal of continuing tasks, including manufacturing, scheduling, queuing, and inventory control, is to find policies that yield the highest expected payoff per step. Moreover, average reward optimality allows for more efficient state abstraction in HRL than the discounted reward optimality, as discussed in Section 5. Although average reward RL has been studied using both the discrete-time MDP model (Schwartz, 1993; Mahadevan, 1996; Tadepalli and Ok, 1996a,b, 1998; Marbach, 1998; Van-Roy, 1998) as well as the continuous-time SMDP

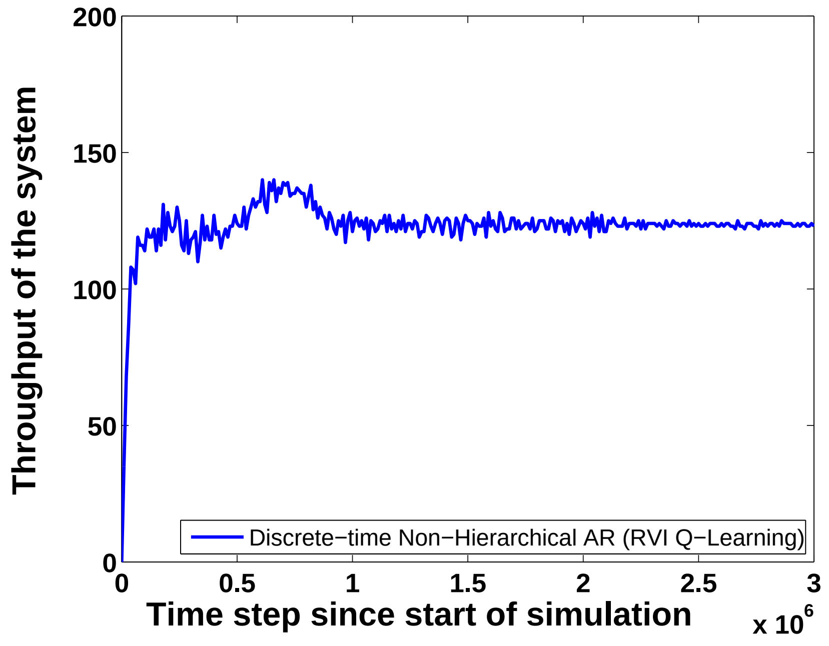

Figure 10: This figure shows the performance of RVI Q-learning, a non-hierarchical average reward algorithm, on the AGV scheduling problem. The RVI Q-learning algorithm converges to the optimal performance after over 2×10^6 time steps, where the hierarchical algorithms converge to this performance in less than 10^5 time steps as shown in Figure 9.

model (Mahadevan et al., 1997b; Wang and Mahadevan, 1999), prior work has been limited to *flat* policy representations.

In this paper, we extended previous work on HRL to the average reward setting, and presented new discrete-time and continuous-time *hierarchically optimal average reward RL* (HAR) and *recursively optimal average reward RL* (RAR) algorithms. These algorithms are based on the average reward SMDP model, and correspond to two notions of optimality in HRL: *hierarchical optimality* and *recursive optimality* (Dietterich, 2000). The HAR algorithm searches the space of policies defined by the hierarchical decomposition to find a hierarchical policy with maximum *global gain* (the gain of the Markov chain that results from flattening the hierarchy using a hierarchical policy). In the *recursively optimal average reward RL* setting, the formulation of learning algorithms directly depends on the local optimality criterion used for each subtask in the hierarchy. The RAR algorithm treats non-primitive subtasks as continuing average reward problems and solve them by maximizing their local gain given the policies of their children. We demonstrated that the policy learned for each subtask by the RAR algorithm is not necessarily context free, and as a result the algorithms do not find a recursively optimal average reward policy in general. However, we showed that the RAR algorithm finds a recursively optimal average reward policy when the *initial distribution invariance* condition holds. We used two automated guided vehicle (AGV) scheduling tasks as experimental testbeds to study the empirical performance of the proposed algorithms. The first problem is a relatively simple AGV scheduling task, in which the hierarchically and recursively optimal policies

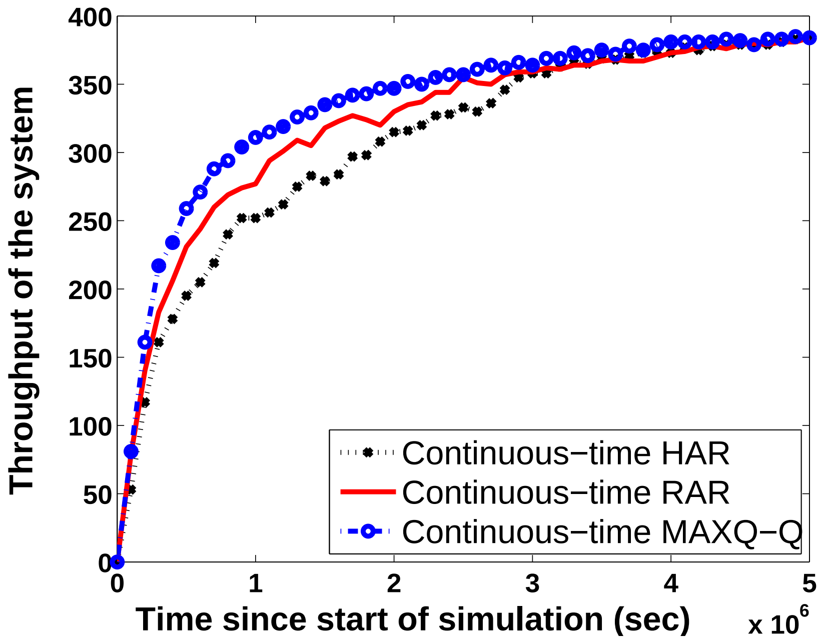

Figure 11: This figure compares the performance of the continuous-time hierarchically (HAR) and recursively (RAR) optimal average reward algorithms with the performance of continuous-time MAXQ-Q, a continuous-time recursively optimal discounted reward RL algorithm, on the AGV scheduling problem.

are different. We compared the proposed algorithms with three other HRL methods, including a hierarchically optimal discounted reward algorithm and a recursively optimal discounted reward algorithm on this problem. The results demonstrate the difference between the optimalities achieved by these algorithms. The second problem is a relatively larger AGV scheduling task. We modeled this problem using both discrete-time and continuous-time models. We used a hierarchical task decomposition with which the hierarchically and recursively optimal policies are the same for this problem. We compared the performance of the proposed algorithms with a hierarchically optimal discounted reward algorithm and a recursively optimal discounted reward algorithm, as well as a flat average reward algorithm in this problem. The results showed that the proposed hierarchical average reward algorithms converge to the same performance as their discounted reward counterparts.

There are a number of directions for future work. An immediate question that arises is proving the asymptotic convergence of the algorithms to hierarchically and recursively optimal policies. These results should provide some theoretical validity to the proposed algorithms, in addition to their empirical efficiency demonstrated in this paper. Studying other local optimality criteria for subtasks in a hierarchy is an interesting problem that needs to be addressed. It helps to develop more efficient *recursively optimal average reward RL* algorithms. It is also clear that our hierarchical average reward framework can be applied to many other manufacturing and robotics problems besides the AGV task.

Acknowledgments

We would like to thank Prasad Tadepalli, David Andre, Bernhard Hengst, and Balaraman Ravindran for their 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 in Autonomous Learning Laboratory in the Department of Computer Science at the University of Massachusetts Amherst under NASA contract No. NAg-1445 #1. Support for this research was also provided in part by the National Science Foundation under grants NSF IIS-0534999 and ECS 0218125. Part of this article was written while the first author was supported by iCORE Canada at the Department of Computing Science at the University of Alberta.

Appendix A. Index of Symbols

Here we present a list of the symbols used in this paper to provide a handy reference.

Notation	Definition
$\mathbb{R}$	set of real numbers
$\mathbb{N}$	set of natural numbers
$\mathbf{E}$	expected value
$\mathcal{M}$	a MDP model
$\mathcal{S}$	set of states of a SMDP
$\mathcal{A}$	set of actions of a SMDP
$\mathcal{P}$	multi-step transition probability function of a SMDP
$\mathcal{R}$	reward function of a SMDP
$r(s,a)$	reward of taking action a in state s
I	initial state distribution of a SMDP
μ	a policy
$\mu(a\|s)$	probability that policy μ selects action a in state s
μ^*	optimal policy
γ	discount factor
α	learning rate parameter
V^μ	hierarchical value function of hierarchical policy μ
$\hat{V}^\mu$	projected value function of hierarchical policy μ
V^*	optimal value function
Q^μ	hierarchical action-value function of hierarchical policy μ
$\hat{Q}^\mu$	projected action-value function of hierarchical policy μ
Q^*	optimal action-value function
g^μ	average reward or gain of policy μ
g^μ	global gain under hierarchical policy μ
g_i^μ	local gain of subtask M_i under hierarchical policy μ
g^*	optimal gain or gain of optimal policy
H^μ	average-adjusted value function of 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
H^*	optimal average-adjusted value function

Notation	Definition
L^μ	average-adjusted action-value function of 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
L^*	optimal average-adjusted action-value function
$P(s',N\|s,a)$	probability that action a will cause the system to transition from state s to state s' in N time steps
$F(s'\|s,a)$	probability that a SMDP occupies state s' at the next decision epoch given that the agent takes action a in state s at the current decision epoch
F^μ	transition probability matrix of the embedded Markov chain of a SMDP for policy μ
$\bar{F}^\mu$	limiting matrix of the embedded Markov chain of a SMDP for policy μ
$y(s,a)$	expected number of transition steps until the next decision epoch in a SMDP
$\mathcal{H}$	a hierarchy
M_i	subtask M_i in a hierarchy
S_i	set of states for subtask M_i in a hierarchy
$\|S_i\|$	cardinality of set of states S_i
A_i	set of actions for subtask M_i in a hierarchy
R_i	reward function for subtask M_i in a hierarchy
I_i	initiation set for subtask M_i in a hierarchy
T_i	termination set for subtask M_i in a hierarchy
μ_i	a policy for subtask M_i in a hierarchy
μ	a hierarchical policy
P_i^μ	multi-step transition probability function of subtask M_i
$P_i^\mu(s',N\|s)$	probability that action $\mu_i(s)$ causes transition from state s to state s' in N primitive steps under hierarchical policy μ
F_i^μ	multi-step abstract transition probability function of subtask M_i
$F_i^\mu(s',N\|s)$	probability of transition from state s to state s' in N abstract actions taken by subtask M_i under hierarchical policy μ
$F_i^\mu(s',1\|s)$	transition probability of the embedded Markov chain at subtask M_i under hierarchical policy μ (same as $F_i^\mu(s'\|s)$)
m^μ	transition probability function of the Markov chain that results from flattening the hierarchy using the hierarchical policy μ
$m^\mu(s'\|s)$	probability that hierarchical policy μ will cause the system to transition from state s to state s' at the level of primitive actions
m^μ	transition probability matrix of the Markov chain that results from flattening the hierarchy using the hierarchical policy μ
$\overline{m}^\mu$	limiting matrix of the Markov chain that results from flattening the hierarchy using the hierarchical policy μ
Ω	set of possible values for Task-Stack in a hierarchy
$X = \Omega \times S$	joint state space of Task-Stack values and states in a hierarchy
$x = (\omega, s)$	joint state value x formed by Task-Stack value ω and state value s in a hierarchy
$\omega \nearrow i$	popping subtask M_i off Task-Stack with content ω in a hierarchy
$i \searrow \omega$	pushing subtask M_i onto Task-Stack with content ω in a hierarchy
C^μ	completion function of hierarchical policy μ
CE^μ	external completion function of hierarchical policy μ

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*, 40:681–698, 2001.

D. Andre. *Programmable Reinforcement Learning Agents*. PhD thesis, University of California at Berkeley, 2003.

D. Andre and S. J. Russell. Programmable reinforcement learning agents. In *Proceedings of Advances in Neural Information Processing Systems 13*, pages 1019–1025. MIT Press, 2001.

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

A. Barto and S. Mahadevan. Recent advances in hierarchical reinforcement learning. *Discrete Event Systems (Special Issue on Reinforcement Learning)*, 13:41–77, 2003.

R. Bellman. *Dynamic Programming*. Princeton University Press, 1957.

S. Bradtke and M. Duff. Reinforcement learning methods for continuous-time Markov decision problems. In *Proceedings of Advances in Neural Information Processing Systems 7*, pages 393–400. MIT Press, 1995.

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

P. Dayan and G. Hinton. Feudal reinforcement learning. In *Proceedings of Advances in Neural Information Processing Systems 5*, pages 271–278, 1993.

T. 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.

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.

R. Howard. *Dynamic Programming and Markov Processes*. MIT Press, 1960.

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

L. Kaelbling. Hierarchical reinforcement learning: Preliminary results. In *Proceedings of the Tenth International Conference on Machine Learning*, pages 167–173, 1993a.

L. Kaelbling. Learning to achieve goals. In *Proceedings of the Thirteenth International Joint Conference on Artificial Intelligence*, pages 1094–1098, 1993b.

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

S. Mahadevan and J. Connell. Automatic programming of behavior-based robots using reinforcement learning. *Artificial Intelligence*, 55(2-3):311–365, 1992.

S. Mahadevan, N. Khaleeli, and N. Marchalleck. Designing agent controllers using discrete-event Markov models. In *Proceedings of the 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*, pages 182–190, 1997b.

P. Marbach. *Simulated-Based Methods for Markov Decision Processes*. PhD 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*, pages 278–287, 1999.

A. Ng, H. Kim, M. Jordan, and S. Sastry. Autonomous helicopter flight via reinforcement learning. In *Proceedings of Advances in Neural Information Processing Systems 16*. MIT Press, 2004.

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

D. Precup. *Temporal Abstraction in Reinforcement Learning*. PhD thesis, University of Massachusetts Amherst, 2000.

M. 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*, pages 298–305, 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.

S. Singh. Transfer of learning by composing solutions of elemental sequential tasks. *Machine Learning*, 8:323–339, 1992.

S. Singh and D. Bertsekas. Reinforcement learning for dynamic channel allocation in cellular telephone systems. In *Proceedings of Advances in Neural Information Processing Systems 9*, pages 974–980, 1996.

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. 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*, pages 471–479, 1996a.

P. Tadepalli and D. Ok. Auto-exploratory average reward reinforcement learning. In *Proceedings of the Thirteenth National Conference on Artificial Intelligence*, pages 881–887, 1996b.

P. Tadepalli and D. Ok. Model-based average reward reinforcement learning. *Artificial Intelligence*, 100:177–224, 1998.

G. Tesauro. TD-Gammon, a self-teaching backgammon program, achieves master-level play. *Neural Computation*, 6:215–219, 1994.

B. Van-Roy. *Learning and Value Function Approximation in Complex Decision Processes*. PhD 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*, pages 464–473, 1999.

W. Zhang and T. Dietterich. A reinforcement learning approach to job-shop scheduling. In *Proceedings of the Fourteenth International Joint Conference on Artificial Intelligence*, pages 1114–1120, 1995.