Transfer in Variable-Reward Hierarchical Reinforcement Learning

Neville Mehta **Sriraam Natarajan** **Prasad Tadepalli** **Alan Fern**

School of Electrical Engineering and Computer Science
Oregon State University
Corvallis, OR 97333
{mehtane,natarasr,tadepall,afern}@eecs.oregonstate.edu

Abstract

We consider the problem of transferring learned knowledge among Markov Decision Processes (MDPs) that share the same transition dynamics but different reward functions. In particular, we assume that reward functions are described as linear combinations of reward features, and that only the feature weights vary among MDPs. We introduce Variable-Reward Hierarchical Reinforcement Learning (VRHRL), which leverages previously learned policies to speed-up learning in this setting. With suitable design of the hierarchy, VRHRL can achieve better transfer than its non-hierarchical counterpart.

1 Introduction

Most work in Reinforcement Learning (RL) addresses the problem of solving a single Markov Decision Process (MDP) defined by a state transition function and a reward function. The focus on solving single MDPs makes it difficult, if not impossible, to learn cumulatively, i.e., to transfer useful knowledge from one MDP to another. In this paper, we consider *variable-reward transfer learning* where the objective is to speed-up learning on a new MDP by transferring experience from previous MDPs that share the same dynamics but different reward functions. In particular, we assume that reward functions are weighted linear combinations of reward features, and that the "reward weights" vary across MDPs.

For such classes of MDPs, previous work [3] has shown how to leverage the reward structure in order to usefully transfer value functions, effectively speeding-up learning. In this paper, we extend this work to the hierarchical setting, where we are given a task hierarchy to be used across the entire variable-reward MDP family. The hierarchical setting provides advantages over the flat RL case, allowing for transfer at multiple levels of the hierarchy, which can significantly speed-up learning. We demonstrate our results in a simplified real-time strategy (RTS) game domain.

2 Variable-Reward Reinforcement Learning

A Semi-Markov Decision Process (SMDP) $\mathcal{M}$ is a tuple $\langle S, A, \mathrm{P}, r, t \rangle$, where S is a set of states, A is a set of temporally extended actions, and the transition function $\mathrm{P}(s'|s, a)$ gives the probability of entering state s' after taking action a in state s. The functions $r(s, a)$ and $t(s, a)$ are the expected reward and execution time respectively for taking action a in state s. In this work, we assume a linear reward function such that $r(s, a) = \sum_i w_i \, r_i(s, a)$, where the $r_i(s, a)$ are reward features, and the w_i are reward weights.

Given an SMDP, the average reward or *gain* ρ^π of a policy π is defined as the ratio of the expected total reward to the expected total time for N steps of the policy from any state s as N goes to infinity. In this work, we seek to learn policies that maximize gain. The *average-adjusted reward* of taking an action a in state s is defined as $r(s, a) - \rho^\pi t(s, a)$. The limit of the total expected average-adjusted reward starting from state s and following policy π is called its *bias* and denoted by $h^\pi(s)$. Importantly, for our linear reward setting, the gain and bias are linear in the reward weights $\vec{w}$, $\rho^\pi = \vec{w} \cdot \vec{\rho}_\pi$ and $h^\pi(s) = \vec{w} \cdot \vec{h}^\pi(s)$, where the i components of $\vec{\rho}_\pi$ and $\vec{h}^\pi(s)$ are the gain and bias with respect to $r_i(s, a)$ respectively.

We consider transfer learning in the context of families of MDPs that share all components except for the reward weights. After encountering a sequence of such MDPs, the goal is to transfer that experience to speed up learning in a new MDP given its reward weights. For example, in our RTS experimental domain, we would like to consider changing the relative reward weighting for bringing in units of wood, gold, and damaging the enemy, but still leverage prior experience.

A previous approach to this problem [3] is based on the following idea. Since the above value functions are linear in the reward weights, policies can be represented indirectly as a set of parameters of these linear functions. Thus, the set of optimal policies for different weights forms a convex and piecewise linear average reward and bias function. As long as the same policy is optimal for different sets of weights, the same parameters will suffice. Furthermore, if Π represents a set of all stored previous policies, then given a new weight vector $\vec{w}_{\text{new}}$, we might expect the policy $\pi_{\text{init}} = \operatorname{argmax}_{\pi \in \Pi} \{\vec{w}_{\text{new}} \cdot \vec{\rho}_\pi\}$ to provide a good starting point for learning. Thus, transfer learning is conducted by initializing the bias and gain vectors to those of π_{init} and then further optimizing via standard reinforcement learning. The newly learned bias and gain vectors were only stored in Π if the gain of the new policy with respect to $\vec{w}_{\text{new}}$ improved by more than δ. With this approach, if the optimal polices are the same or similar for many weight vectors, we will only store a small number of policies, and achieve significant transfer.

3 Variable-Reward Hierarchical Reinforcement Learning Framework

In HRL [1], the original MDP $\mathcal{M}$ is split into sub-SMDPs $\{\mathcal{M}_0, \mathcal{M}_1, \ldots, \mathcal{M}_n\}$, where each sub-SMDP represents a subtask (composite or primitive). Solving the root task $\mathcal{M}_0$ solves the entire MDP $\mathcal{M}$. The task hierarchy is represented as a directed acyclic graph known as the *task graph* that represents the task-subtask relationships. All primitive actions of the original MDP are represented as leaf nodes in the task graph. All subtasks except the root and the primitive actions have explicit termination conditions; the primitive actions terminate immediately and the root never terminates. The task hierarchy for the RTS domain is shown in Figure 1(b).

A local policy π_i for the subtask $\mathcal{M}_i$ is a mapping from the states to the child tasks of $\mathcal{M}_i$. A hierarchical policy π for the overall task is an assignment of a local policy π_i to each sub-MDP $\mathcal{M}_i$. Every subtask $\mathcal{M}_i$ is associated with an abstraction function B_i which abstracts the states into groups that have the same task-specific value function. The objective is to learn a recursively optimal policy that optimizes the policy for each subtask assuming that its children's policies are optimized.

For Variable-Reward HRL, in every subtask (but the Root) we store the total expected reward during that subtask, and the expected duration of the subtask for every state. Storing the parameters of the value function that are independent of the global average reward ρ allows for the transfer of any subtree of the task hierarchy across different MDPs. For action selection, the objective is to maximize the weighted average reward (the dot product of the average reward vector with the weight vector). The value function decomposition for a recursively optimal policy satisfies the following equations:

$$\begin{aligned} \vec{h}_i(s) &= \vec{r}(s) \quad \text{if } i \text{ is a primitive subtask} \\ &= 0 \quad \text{if } s \text{ is a terminal/goal state for } i \\ &= \vec{h}_j(\mathrm{B}_j(s)) + \sum_{s' \in \mathcal{S}} \mathrm{P}(s'|s,j) \cdot \vec{h}_i(s') \quad \text{otherwise,} \end{aligned}$$

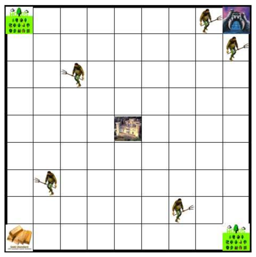
$$\text{where } j = \operatorname*{argmax}_a \left\{ \vec{w} \cdot \left(\vec{h}_a(\mathrm{B}_a(s)) - \rho \cdot t_a(\mathrm{B}_a(s)) + \mathrm{E}\left[\vec{h}_i(s') - \vec{\rho} \cdot t_i(s') \right] \right) \right\}$$

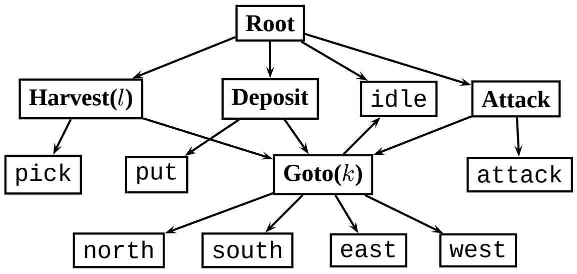
At Root, we only store the average adjusted reward because it never terminates:

$$\vec{h}_{Root}(s) = \max_j \left\{ \vec{w} \cdot \left(\vec{h}_j(\mathrm{B}_j(s)) - \vec{\rho} \cdot \vec{t}_j(\mathrm{B}_j(s)) + \sum_{s' \in \mathcal{S}} \mathrm{P}(s'|s,j) \cdot \vec{h}_j(s') \right) \right\}$$

Our model-based algorithm learns the transition models for each subtask and uses the above equations to update the task-specific value functions. The agent stores the newly learned hierarchical policy for a new weight vector w_{new} only if $w_{\text{new}} \cdot \rho - w_{\text{new}} \cdot \rho_{\text{initial}} > \delta$. Further, the value function for a subtask is stored only when at least one of its components for at least one of the states is different from previously stored versions of this subtask by more than ε; otherwise, a pointer to this previously stored subtask value function suffices.

4 Experimental Results


(a)



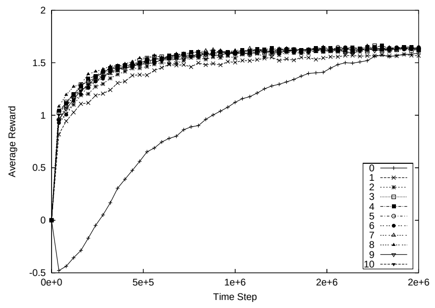
(b)

Figure 1: RTS domain and the corresponding task hierarchy.

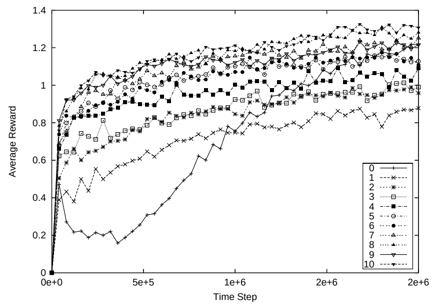
We consider a simplified RTS game shown in Figure 1(a). It is a grid world that contains peasants, the peasants' home base, resource locations (forests and goldmines) where the peasants can harvest wood or gold, and an enemy base which can be attacked. The primitive actions available to a peasant are moving one cell to the `north`, `south`, `east`, and `west`, `pick` a resource, `put` a resource, `attack` the enemy base, and `idle` (no-op). The following results are based on a single peasant game in a 25×25 grid with 3 forests cells, 2 goldmines, a home base, and an enemy base. The resources get regenerated stochastically. The enemy also appears stochastically and stays till it has been destroyed. Reward weights are generated randomly and dictate the relative value of collecting the various resources or attacking the enemy.

Figures 2(a) and 2(b) plot learning curves for a test reward weight after having seen zero through ten previous training weights, for both the flat and VRHRL learners (averaged over 10 training sets). The learning curves converge much faster for VRHRL. In the flat case we see “negative transfer” where learning based on one previous weight is worse than learning from scratch. This is unsurprising given that currently we always attempt to transfer past experience, even when experience is limited. Hierarchical RL seems to avoid such negative transfer by clustering experience into similar subtasks.

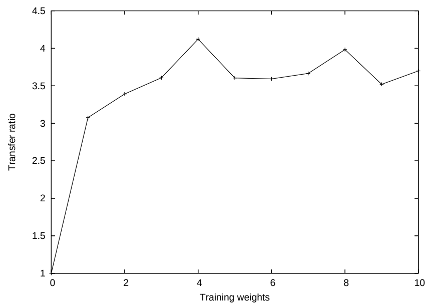
As another measure of transfer, let F_Y be the area between the learning curve and its optimal value for problem Y with no prior learning experience on X, and $F_{Y|X}$ be the area between the learning curve and its optimal value for problem Y given prior training on X. The transfer ratio is defined as $F_X/F_{Y|X}$. The transfer ratio is greater for the VRHRL learner than for the flat learner.



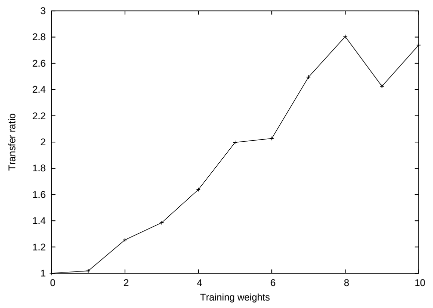
(a) Learning curves for the VRHRL learner.



(b) Learning curves for the flat learner.



(c) Transfer ratio for the VRHRL learner.



(d) Transfer ratio for the flat learner.

Figure 2: Experimental results.

5 Conclusions and Future Work

In this paper, we have shown that hierarchical task structure can accelerate transfer across variable-reward MDPs more so than in the non-hierarchical case. Extending these results to MDP families with slightly different dynamics would be interesting. Another possible direction is an extension to shared subtasks in the multi-agent setting [2].

References

[1] T. Dietterich. Hierarchical Reinforcement Learning with the MAXQ Value Function Decomposition. *Journal of Artificial Intelligence Research*, 9:227–303, 2000.

[2] N. Mehta and P. Tadepalli. Multi-Agent Shared Hierarchy Reinforcement Learning. *ICML Workshop on Richer Representations in Reinforcement Learning*, 2005.

[3] S. Natarajan and P. Tadepalli. Dynamic Preferences in Multi-Criteria Reinforcement Learning. In *Proceedings of ICML-05*, 2005.