# Hierarchical Hybrid Reinforcement Learning Algorithms

**Mohammad Ghavamzadeh** MOHAMMAD.GHAVAMZADEH@INRIA.FR
INRIA Lille - Nord Europe, Team SequeL, France

## 1. Introduction

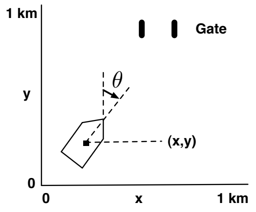
Hierarchical reinforcement learning (HRL) is a general framework which attempts to accelerate policy learning in large domains. The basic idea is to incorporate prior knowledge and decompose the task into a collection of subtasks with smaller and more manageable state and action spaces, and learn these subtasks in a way to solve the overall problem. The well-studied HRL frameworks: HAMs (Parr, 1998), options (Sutton et al., 1999), and MAXQ (Dietterich, 2000) are all based on value function-based RL (VFRL) methods. Although VFRL has been extensively studied in the machine learning literature, there are only weak theoretical guarantees on the convergence of these methods on problems with large or continuous state spaces. Moreover, application of these methods can be problematic in domains with large or continuous action spaces. This is because finding the greedy action, which is required at every step of these algorithms, can be very expensive in such problems.

An alternative approach to VFRL is policy gradient RL (PGRL). These algorithms have received much attention as a means to solve problems with continuous state and action spaces, mainly because 1) they are guaranteed to converge to local optimal policies, and 2) it is possible to incorporate prior knowledge into these methods via appropriate choice of the parametric form of the policy. However, in high-dimensional tasks, in which the policy is parameterized using a large number of parameters, the PGRL methods may show poor performance by becoming stuck in local optima. Moreover, they are usually slower than VFRL methods, due to the large variance of their gradient estimators.

A possible approach to control high-dimensional problems, including those in robotics, is to design algorithms that can take advantage of the strengths of all the above techniques: the modularity of HRL, the speed of VFRL algorithms, and the efficacy of PGRL methods in dealing with continuos state and action problems. In this work, we define a family of HRL algorithms in which VFRL methods are used to control high-level subtasks, which usually involve smaller and more manageable state and finite action spaces, and PGRL controllers are used for low-level subtasks, which usually have continuous state and/or action spaces. We call this family of algorithms *hierarchical hybrid RL*. In the next section, we use an example to briefly describe these algorithms; for more details please refer to Ghavamzadeh and Mahadevan (2003).

## 2. Hierarchical Hybrid RL

The ship steering problem (Miller et al., 1990) is a continuous 4-dim state 1-dim action problem (Fig. 1), in which a ship starts at a randomly chosen position, orientation, and turning rate, and is to be maneuvered at a constant speed through a gate placed at a fixed position. Flat PGRL algorithms without careful tuning do not achieve a good performance in a reasonable amount of time in this problem. We believe the failure is due to 1) Since the ship cannot turn faster than 15 deg/sec, all the state variables change only by a small amount at each control interval. Thus, a high resolution discretization of the state space is necessary in order to accurately model state transitions, which in turn means a large number of parameters for the approximator. 2) There is a time lag between changes in the desired turning rate $r$, and the actual turning rate $\dot{\theta}$, ship's position $x, y$ and orientation $\theta$, which requires the controller to deal with long delays.



| | | |
|---|---|---|
| State | $x$ | 0 to 1000 m |
| | $y$ | 0 to 1000 m |
| | $\theta$ | -180 to 180 deg |
| | $\dot{\theta}$ | -15 to 15 deg/sec |
| Action | $r$ | -15 to 15 deg/sec |

*Figure 1.* The ship steering domain.

However, the flat PGRL algorithms can be successfully applied to simplified versions of this problem shown in Fig. 2, when 1) the range of $x$ and $y$ is smaller, 2) the ship always starts at a fixed position with randomly chosen orientation and turning rate, and 3) the goal is a fixed point. It indicates that the control problem can be learned using an appropriate hierarchical decomposition (Fig. 3). At the high-level, the ship learns to select among four diagonal and four horizontal/vertical subtasks (see Fig. 2). At the low-level, each subtask learns a sequence of actions (turning rates) to achieve its goal. Using symmetry, the eight subtasks of the *root* can be mapped to only two subtasks, one corresponding to the four diagonal subtasks and one corresponding to the four horizontal/vertical subtasks.

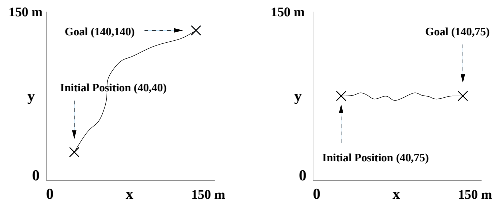


*Figure 2.* Simplified versions of the ship steering problem.

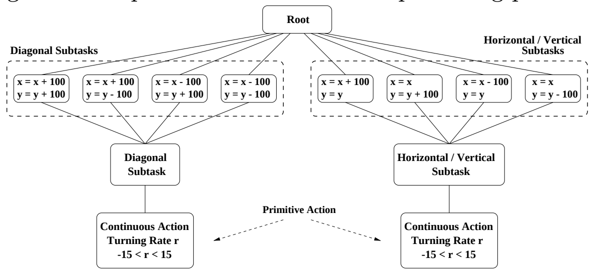


*Figure 3.* A task graph for the ship steering problem.

After decomposing the overall task into a collection of subtasks, each subtask is formulated either as a VFRL or a PGRL problem. At the high-level subtasks in which only a subset of the state variables is relevant (positions $x$ and $y$), and the action space is discrete (4 diagonal and 4 horizontal/vertical subtasks), the state space is coarsely discretized and a VFRL algorithm is used to learn a policy. At the low-level subtasks in which only a small range of the state variables is relevant ($0 \leq x, y \leq 150$) and the action space is continuous, the state space is finely discretized and a PGRL algorithm is used for learning. In these subtask, the performance measure optimized by the PGRL algorithm is *weighted reward-to-go* $\chi^\pi = \sum_s I^\pi(s) J^\pi(s)$, where $J^\pi(s)$ is the expected sum of rewards when starting in state $s$, and following policy $\pi$ until the subtask terminates, and $I^\pi(s)$ is the probability that the subtask starts in state $s$ under policy $\pi$. The overall learning algorithm is similar to MAXQ-Q (Dietterich, 2000). It is a recursive function that executes the current exploration policy starting from the *root*. When control is at a subtask, it performs an action until it reaches a terminal state, at which point it returns the sum of the rewards observed and the number of actions taken during its execution to its parent task. To execute an action, the algorithm calls itself recursively. When the recursive call returns, it updates its parameters either using a VFRL or a PGRL algorithm.

## 3. Discussion

Although the hybrid algorithms presented in this paper provide a framework for concurrent learning of high-level strategies and low-level behaviors, they may not yet be adequate for many problems in robotics. However, they have a number of desirable properties (discussed below) that give them the potential to be used along with other techniques in order to tackle robotics applications.

**1)** As we showed in the ship steering problem, different notions of similarity such as *symmetry* can be used to reduce the number of parameters to be learned.
**2)** Using state abstraction, only a subset of the state variables is usually relevant to control each subtask. State abstraction can also help to reduce the range of the state variables used by the subtasks.
**3)** Additional rewards to those of the original MDP can be used inside each subtask to accelerate its learning, but they should not be propagated to the higher levels of the hierarchy. This can be implemented using two value functions for each subtask: *internal value function* that is defined based on both the reward of the MDP and the additional reward, and is used only inside the subtask, and *external value function* that is only based on the reward of the MDP and is propagated to the higher levels of the hierarchy.
**4)** Learning in subtasks does not have to be started from scratch, subtasks can be initialized with policies acquired through imitation learning, practice, or some sort of bottom-up learning.
**5)** The policy learned for each subtask is without reference to the context in which it is executed. This property makes it easier to share and reuse subtasks.
**6)** Although the context-free property allows for subtask-sharing among problems, it generates non-smooth policies that may not be desirable in many applications, especially those in robotics. This is because each subtask terminates independent of the goal of the subtask that must be executed next. A possible solution to this problem is an alternative policy execution in the task graph, called *polling* (Dietterich, 2000). In polling, each action is chosen by starting at *root* (not at the current node in the hierarchy) and computing the path with the highest action-value function. The primitive action at the end of this path is then executed, and the process is repeated. If the learned policy is not optimal, then this one-step greedy policy will be closer to an optimal policy, because it corresponds to one step of policy improvement in policy iteration.

## References

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

Ghavamzadeh, M., & Mahadevan, S. (2003). Hierarchical policy gradient algorithms. *Proceedings of the Twentieth ICML* (pp. 226–233).

Miller, W., Sutton, R., & Werbos, P. (1990). *Neural netwroks for control.* MIT Press.

Parr, R. (1998). *Hierarchical control and learning for Markov decision processes.* Doctoral dissertation, University of California at Berkeley.

Sutton, R., Precup, D., & Singh, S. (1999). Between MDPs and semi-MDPs: A framework for temporal abstraction in reinforcement learning. *AI*, *112*, 181–211.