Hierarchical Policy Gradient Algorithms

Mohammad Ghavamzadeh
University of Massachusetts - Amherst

Sridhar Mahadevan
University of Massachusetts - Amherst

Hierarchical Policy Gradient Algorithms

Mohammad Ghavamzadeh MGH@CS.UMASS.EDU
Sridhar Mahadevan MAHADEVA@CS.UMASS.EDU
Department of Computer Science, University of Massachusetts Amherst, Amherst, MA 01003-4610, USA

Abstract

Hierarchical reinforcement learning is a general framework which attempts to accelerate policy learning in large domains. On the other hand, policy gradient reinforcement learning (PGRL) methods have received recent attention as a means to solve problems with continuous state spaces. However, they suffer from slow convergence. In this paper, we combine these two approaches and propose a family of *hierarchical policy gradient* algorithms for problems with continuous state and/or action spaces. We also introduce a class of *hierarchical hybrid algorithms*, in which a group of subtasks, usually at the higher-levels of the hierarchy, are formulated as value function-based RL (VFRL) problems and the others as PGRL problems. We demonstrate the performance of our proposed algorithms using a simple taxi-fuel problem and a complex continuous state and action ship steering domain.

1. Introduction

Value function-based reinforcement learning (VFRL) has been extensively studied in the machine learning literature. However, there are only weak theoretical guarantees on the performance of these methods on problems with large or continuous state spaces.

An alternative approach to VFRL is to consider a class of parameterized stochastic policies, compute the gradient of a performance function with respect to the parameters, and improve the policy by adjusting the parameters in the direction of the gradient. This approach is known as policy gradient reinforcement learning (PGRL) (Marbach, 1998; Baxter & Bartlett, 2001), and have received much recent attention as a means to solve problems with continuous state spaces. The main motivations for this are : **1)** PGRL algorithms are theoretically guaranteed to converge to locally 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 real-world high-dimensional tasks, in which the performance function is parameterized using a large number of parameters, the PGRL methods might show poor performance by becoming stuck in local optima. Moreover, PGRL algorithms are usually slower than VFRL methods, due to the large variance of their gradient estimators. The above two reasons might make the application of these algorithms problematic in real-world domains. A possible solution is to incorporate prior knowledge and decompose the high-dimensional task into a collection of modules with smaller and more manageable state spaces and learn these modules in a way to solve the overall problem. Hierarchical value function-based RL methods (Parr, 1998; Dietterich, 1998; Sutton et al., 1999) have been developed using this approach, as an attempt to scale RL to large state spaces.

In this paper, we define a family of *hierarchical policy gradient* (HPG) algorithms, for scaling PGRL methods to high-dimensional domains. In HPG, subtasks involved in decision making (subtasks with more than one child), which we call *non-primitive* subtasks, are defined as PGRL problems whose solution involves computing a locally optimal policy. Each *non-primitive* subtask is formulated in terms of a parameterized family of policies, a performance function, a method to estimate the gradient of the performance function, and a routine to update the policy parameters using the performance gradient.

We accelerate the learning of HPG algorithms by formulating higher-level subtasks, which usually involve smaller and more manageable state and finite action spaces, as VFRL problems, and lower-level subtasks with infinite state and/or action spaces as PGRL problems (Morimoto & Doya, 2001). We call this family of algorithms *hierarchical hybrid* algorithms. The effectiveness of our proposed algorithms is demonstrated using a simple taxi-fuel problem as well as a more complex continuous state and action ship steering task.

2. Hierarchical Task Decomposition

We introduce the hierarchical task decomposition using a continuous state and action space ship steering problem (Miller et al., 1990, see Figure 1). 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.

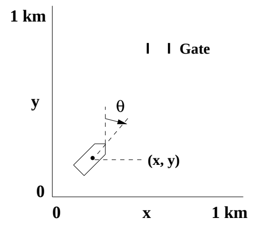

Figure 1. The ship steering domain.

Equation 1 gives the motion equations of the ship, where $T = 5$ is the time constant of convergence to desired turning rate, $V = 3m/sec$ is the constant speed of the ship, and $\Delta = 0.2sec$ is the sampling interval. There is a time lag between changes in the desired turning rate and the actual rate, modeling the effects of a real ship's inertia and the resistance of the water.

$$\begin{aligned} x[t+1] &= x[t] + \Delta V sin\theta[t] \\ y[t+1] &= y[t] + \Delta V cos\theta[t] \\ \theta[t+1] &= \theta[t] + \Delta\dot{\theta}[t] \\ \dot{\theta}[t+1] &= \dot{\theta}[t] + \Delta(r[t] - \dot{\theta}[t])/T \end{aligned} \tag{1}$$

At each time t, the state of the ship is given by its position $x[t]$ and $y[t]$, orientation $\theta[t]$ and actual turning rate $\dot{\theta}[t]$. The action is the desired turning rate of the ship $r[t]$. All four state variables and also the action are continuous and their range is shown in Table 1. The ship steering problem is episodic. In each episode, the goal is learning to generate sequences of actions that steer the center of the ship through the gate in the minimum amount of time. The sides of the gate are placed at coordinates (350,400) and (450,400). If the ship moves out of bound ($x < 0$ or $x > 1000$ or $y < 0$ or $y > 1000$), the episode terminates and is considered as a failure.

We applied both flat PGRL and actor-critic (Konda, 2002) algorithms to this task without achieving a good performance in reasonable amount of time (Figure 7). We believe this failure occurred due to two reasons,

State	x	0 to 1000 meters
	y	0 to 1000 meters
	θ	-180 to 180 degrees
	$\dot{\theta}$	-15 to 15 degrees/sec
Action	r	-15 to 15 degrees/sec

Table 1. Range of state and action variables for the ship steering task.

which make this problem hard for RL algorithms. First, since the ship cannot turn faster than 15 degrees/sec, all state variables change only by a small amount at each control interval. Thus, we need a high resolution discretization of the state space in order to accurately model state transitions, which requires a large number of parameters for the function approximator and makes the problem intractable. Second, 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 θ, which requires the controller to deal with long delays.

However, we successfully applied a flat policy gradient algorithm to simplified versions of this problem shown in Figure 2, when x and y change from 0 to 150 instead of 0 to 1000, the ship always starts at a fixed position with randomly chosen orientation and turning rate, and the goal is reaching a neighborhood of a fixed point. It indicates that this high-dimensional non-linear control problem can be learned using an appropriate hierarchical decomposition. Using this prior knowledge, we decompose the problem into two levels using the task graph shown in Figure 3. At the high-level, the agent learns to select among four diagonal and four horizontal/vertical subtasks. At the low-level, each low-level subtask learns a sequence of turning rates to achieve its own goal. We use symmetry and map eight possible subtasks of the *root* to only two subtasks at the low-level, one associated with four diagonal subtasks and one associated with four horizontal/vertical subtasks as shown in Figure 3. We call them *diagonal* subtask and *horizontal/vertical* subtask.

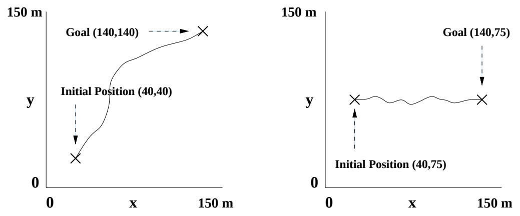

Figure 2. This figure shows two simplified versions of the ship steering task, used as low-level subtasks in the hierarchical decomposition of ship steering problem.

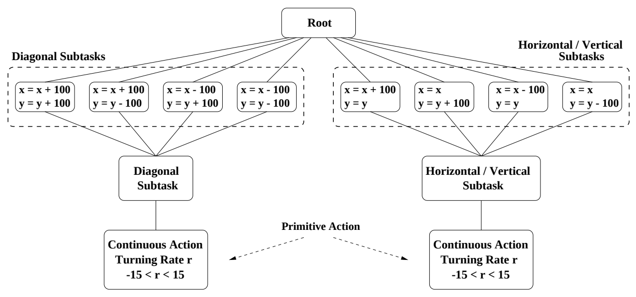

Figure 3. A task graph for the ship steering problem.

As illustrated above for the ship steering task, the designer of the system uses his/her domain knowledge and recursively decomposes the overall task into a collection of subtasks that are important for solving the problem. This decomposition is represented by a directed acyclic graph called *task graph*, as shown in Figure 3. Mathematically, the *task graph* is modeled by decomposing the overall task MDP M, into a finite set of subtask MDPs $\{M^0, \ldots, M^n\}$. Each subtask MDP M^i models a subtask in the hierarchy and M^0 is the *root* task and solving it solves the entire MDP M. Each non-primitive subtask i is defined over the MDP M^i using state space S^i, initiation set I^i, set of terminal states T^i, action space A^i, transition probability function P^i, and reward function R^i. 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. If we have a policy μ^i for each subtask i in this model, it gives us a *hierarchical policy* $\mu = \{\mu^0, \ldots, \mu^n\}$. A hierarchical policy is executed using a stack discipline, similar to subroutine calls in programming languages.

3. Policy Gradient Formulation

After decomposing the overall problem into a set of subtasks as described in section 2, we formulate each subtask as a PGRL problem. Our focus in this paper is on *episodic* problems, so we assume that the overall task (*root* of the hierarchy) is *episodic*.

3.1. Policy Formulation

We formulate each subtask i using a set of randomized stationary policies $\mu^i(\theta^i)$ parameterized in terms of a vector $\theta^i \in \Re^K$. $\mu^i(s, a, \theta^i)$ denotes the probability of taking action a in state s under the policy corresponding to θ^i. We make the following assumption about this set of policies.

A1: For every state $s \in S^i$ and every action $a \in A^i$, $\mu^i(s, a, \theta^i)$ as a function of θ^i, is bounded and has bounded first and second derivatives. Furthermore, we have $\nabla\mu^i(s, a, \theta^i) = \mu^i(s, a, \theta^i)\psi^i(s, a, \theta^i)$, where $\psi^i(s, a, \theta^i)$ is bounded, differentiable and has bounded first derivatives.

Since every time we call a subtask i in the hierarchy, it starts at one of its initial states ($\in I^i$) and terminates at one of its terminal states ($\in T^i$), we can model each instantiation of subtask i as an episode as shown in Figure 4. In this model, all terminal states ($s \in T^i$) transit with probability 1 and reward 0 to an absorbing state s^{*i}. We make the following assumption for every subtask i and its parameterized policy $\mu^i(\theta^i)$.

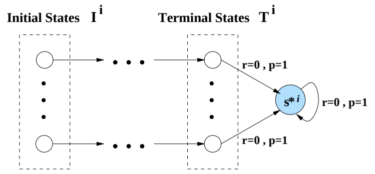

Figure 4. This figure shows how we model a subtask as an *episodic* problem under assumption A2.

A2 (Subtask Termination): There exists a state $s^{*i} \in S^i$ such that, for every action $a \in A^i$, we have $R^i(s^{*i}, a) = 0$ and $P^i(s^{*i}|s^{*i}, a) = 1$, and for all stationary policies $\mu^i(\theta^i)$ and all states $s \in S^i$, we have $P^i(s^{*i}, N|s, \mu^i(\theta^i)) > 0$, where $N = |S^i|$.

Under this model, we define a new MDP $M^i_{\bar{\pi}^i}$ for subtask i with transition probabilities

$$P^i_{\bar{\pi}^i}(s'|s, a) = \begin{cases} P^i(s'|s, a) & s \neq s^{*i} \\ \bar{\pi}^i(s') & s = s^{*i} \end{cases}$$

and rewards $R^i_{\bar{\pi}^i}(s, \theta^i) = R^i(s, \theta^i)$, where $\bar{\pi}^i(s)$ is the probability that subtask i starts at state s.

Let $\mathcal{P}^i_{\bar{\pi}^i}$ be the set of all transition matrices $P^i_{\bar{\pi}^i}(s'|s, \mu^i(\theta^i))$. We have the following result for subtask i.

Lemma 1: Let assumptions A1 and A2 hold. Then for every $P^i_{\bar{\pi}^i} \in \mathcal{P}^i_{\bar{\pi}^i}$ and every state $s \in S^i$, we have $\sum_{n=1}^N P^i_{\bar{\pi}^i}(s^{*i}, n|s, \mu^i(\theta^i)) > 0$, where $N = |S^i|$.

Lemma 1 is equivalent to assume that the MDP $M^i_{\bar{\pi}^i}$ is *recurrent*, i.e. the underlying Markov chain for every policy $\mu^i(\theta^i)$ in this MDP has a single recurrent class and the state s^{*i} is recurrent state.

3.2. Performance Measure Definition

We define *weighted reward-to-go* $\chi^i(\theta^i)$ as the performance measure of subtask i formulated by parameterized policy $\mu^i(\theta^i)$, and for which assumption A2 holds, as

$$\chi^i(\theta^i) = \sum_{s \in S^i} \bar{\pi}^i(s) J^i(s, \theta^i)$$

where $J^i(s, \theta^i)$ is the reward-to-go of state s,

$$J^i(s, \theta^i) = E_{\theta^i}\left[\sum_{k=0}^{T-1} R^i(s_k, \theta^i) | s_0 = s\right]$$

where $T = min\{k > 0 | s_k = s^{*i}\}$ is the first future time that state s^{*i} is visited.

3.3. Optimizing the Weighted Reward-to-Go

In order to obtain an expression for the gradient $\nabla\chi^i(\theta^i)$, we use MDP $M^i_{\bar{\pi}^i}$ defined in section 3.1. Using Lemma 1, MDP $M^i_{\bar{\pi}^i}$ is *recurrent.* For MDP $M^i_{\bar{\pi}^i}$, let $\pi^i_{\bar{\pi}^i}(s, \theta^i)$ be the steady state probability distribution of being in state s and let $E_{\bar{\pi}^i,\theta^i}[T]$ be the mean recurrence time, i.e. $E_{\bar{\pi}^i,\theta^i}[T] = E_{\bar{\pi}^i,\theta^i}[T | s_0 = s^{*i}]$. We also define $Q^i(s, a, \theta^i) = E_{\bar{\pi}^i,\theta^i}\left[\sum_{k=0}^{T-1} R^i_{\bar{\pi}^i}(s_k, \theta^i) | s_0 = s, a_0 = a\right]$, which is the usual action-value function.

Using MDP $M^i_{\bar{\pi}^i}$, we can derive the following proposition which gives an expression for the gradient of the *weighted reward-to-go* $\chi^i(\theta^i)$ with respect to θ^i.

Proposition 1: If assumptions A1 and A2 hold

$$\nabla\chi^i(\theta^i) = E_{\bar{\pi}^i,\theta^i}[T] \sum_{s \in S^i} \sum_{a \in A^i} \pi^i_{\bar{\pi}^i}(s, \theta^i) \nabla\mu^i(s, a, \theta^i) Q^i(s, a, \theta^i)$$

The expression for the gradient in proposition 1 can be estimated over a *renewal cycle*[1] as

$$F^i_m(\theta^i) = \sum_{n=t_m}^{t_{m+1}-1} \tilde{Q}^i(s_n, a_n, \theta^i) \frac{\nabla\mu^i(s_n, a_n, \theta^i)}{\mu^i(s_n, a_n, \theta^i)} \quad (2)$$

where t_m is the time of the mth visit at the recurrent state s^{*i} and $\tilde{Q}^i(s_n, a_n, \theta^i) = \sum_{k=n}^{t_{m+1}-1} R^i(s_n, a_n)$ is an estimate of Q^i.

From Equation 2, we obtain the following procedure to update the parameter vector along the approximate gradient direction at every time step.

$$z^i_{k+1} = \begin{cases} 0 & s_k = s^{*i} \\ z^i_k + \psi^i(s_k, a_k, \theta^i_k) & \text{otherwise} \end{cases} \quad (3)$$

$$\theta^i_{k+1} = \theta^i_k + \alpha^i_k R^i(s_k, a_k) z^i_{k+1}$$

[1]Cycle between consecutive visits to recurrent state s^{*i}.

where α_k is the step size parameter and satisfies the following assumptions.

A4: α_k's are deterministic, nonnegative and satisfy $\sum_{k=1}^{\infty} \alpha_k = \infty$ and $\sum_{k=1}^{\infty} \alpha_k^2 < \infty$.

A5: α_k's are non-increasing and there exists a positive integer p and a positive scalar A such that $\sum_{k=n}^{n+t}(\alpha_n - \alpha_k) \leq At^p\alpha_n^2$ for all positive integers n and t.

We have the following convergence result for the iterative procedure in Equation 3 to update the parameters.

Proposition 2: Let assumptions A1, A2, A4 and A5 hold, and let (θ^i_k) be the sequence of parameter vectors generated by Equation 3. Then, $\chi^i(\theta^i_k)$ converges and $lim_{k\to\infty}\nabla\chi^i(\theta^i_k) = 0$ with probability 1.

Equation 3 provides an unbiased estimate of $\nabla\chi^i(\theta^i)$. For systems involving a large state space, the interval between visits to state s^{*i} can be large. As a consequence, the estimate of $\nabla\chi^i(\theta^i)$ might have a large variance. Several approaches have been proposed to reduce the variance in these estimations and delivering faster convergence. For instance, in one approach, we can replace s^{*i} with S^{*i}, a subset of state space containing s^{*i}, and reset z^i when $s \in S^{*i}$. This method can be easily implemented for each subtask by defining $S^{*i} = T^i \bigcup \{s^{*i}\}$. Another approach uses a discount factor γ in reward-to-go estimation. However, these methods introduce a bias into the estimate of $\nabla\chi^i(\theta^i)$. For both approaches, we can derive a modified version of Equation 3 to incrementally update the parameter vector along the approximate gradient direction.

4. Hierarchical Policy Gradient Algorithms

After decomposing the overall task to a set of subtasks as described in section 2, and formulating each subtask in the hierarchy as an *episodic* PGRL problem as illustrated in section 3, we can use the update Equation 3 and derive a hierarchical policy gradient algorithm (HPG) to maximize the *weighted reward-to-go* for every subtask in the hierarchy. Algorithm 1 shows the pseudo code for this algorithm.

$G^i(s'|s, a)$ in lines 10 and 16 of the algorithm is the internal reward which can be used only inside each subtask to speed up its local learning and does not propagate to upper levels in the hierarchy. Lines $11-16$ can be replaced with any other policy gradient algorithm to optimize the *weighted reward-to-go*, such as (Marbach, 1998) or (Baxter & Bartlett, 2001). Thus, Algorithm 1 demonstrates a family of hierarchical policy gradient algorithms to maximize the *weighted reward-to-go* for every subtask in the hierarchy.

Algorithm 1 A hierarchical policy gradient algorithm that maximizes the *weighted reward-to-go* for every subtask in the hierarchy.

```
1: Function HPG(Task $i$, State $s$)
2: $\tilde{R} = 0$
3: if $i$ is a primitive action then
4:     execute action $i$ in state $s$, observe state $s'$ and reward $R(s'|s,i)$
5:     return $R(s'|s,i)$
6: else
7:     while $i$ has not terminated $(s \neq s^{*i})$ do
8:         choose action $a$ using policy $\mu^i(s, \theta^i)$
9:         $R$=HPG(Task $a$, State $s$)
10:        observe result state $s'$ and internal reward $G^i(s'|s,a)$
11:        if $s' = s^{*i}$ then
12:            $z^i_{k+1} = 0$
13:        else
14:            $z^i_{k+1} = z^i_k + \psi^i(s, a, \theta^i_k)$
15:        end if
16:        $\theta^i_{k+1} = \theta^i_k + \alpha^i_k (R + G^i(s'|s,a)) z^i_{k+1}$
17:        $\tilde{R} = \tilde{R} + R$
18:        $s = s'$
19:    end while
20: end if
21: return $\tilde{R}$
22: end HPG
```

The above formulation of each subtask has the following limitations: **1)** Compact (parameterized) representation of the policy limits the search for a policy to a set which is typically smaller than the set of all possible policies. **2)** Gradient-based optimization algorithms as a search method for a policy, find a solution which is locally, rather than globally, optimal. Thus, in general, the family of algorithms described above converges to a *recursively local optimal* policy. If the policy learned for every subtask in the hierarchy coincides with the best policies, then these algorithms converge to a *recursively optimal* policy.

Despite all the methods proposed to reduce the variance of gradient estimators in PGRL algorithms, these algorithms are still slower than VFRL methods, as we will show in the simple taxi-fuel experiment in section 5.1. One way to accelerate learning of HPG algorithms is to formulate those subtasks involving smaller state spaces and finite action spaces, usually located at the higher-levels of the hierarchy, as VFRL problems, and those with large state spaces and infinite action spaces, usually located at the lower-levels of the hierarchy, as PGRL problems. This formulation can benefit from the faster convergence of VFRL methods and the power of PGRL algorithms in domains with infinite state and/or action spaces at the same time. We call this family of algorithms, *hierarchical hybrid* algorithms and illustrate them in our ship steering experiment.

5. Experimental Results

In this section, we first apply the hierarchical policy gradient algorithm proposed in this paper to the well-known taxi-fuel problem (Dietterich, 1998) and compare its performance with MAXQ-Q, a value-based hierarchical RL algorithm, and flat Q-learning. Then we turn to a more complex continuous state and action ship steering domain, apply a hierarchical hybrid algorithm to this task and compare its performance with flat PGRL and actor-critic algorithms.

5.1. Taxi-Fuel Problem

A 5-by-5 grid world inhabited by a taxi is shown in Figure 5. There are four stations, marked as B(lue), G(reen), R(ed) and Y(ellow). The task is episodic. In each episode, the taxi starts in a randomly chosen location and with a randomly chosen amount of fuel (ranging from 5 to 12 units). There is a passenger at one of the four stations (chosen randomly), and that passenger wishes to be transported to one of the other three stations (also chosen randomly). The taxi must go to the passenger's location, pick up the passenger, go to the destination location and drop off the passenger there. The episode ends when the passenger is deposited at the destination station or taxi goes out of fuel. There are 8,750 possible states and seven primitive actions in the domain, four navigation actions, Pickup action, Dropoff action, and Fillup action (each of these consumes one unit of fuel). Each action is deterministic. There is a reward of -1 for each action and an additional reward of 20 for successfully delivering the passenger. There is a reward of -10 if the taxi attempts to execute the Dropoff or Pickup actions illegally and a reward of -20 if the fuel level falls below zero. The system performance is measured in terms of the average reward per step. In this domain, this is equivalent to maximizing the total reward per episode. Each experiment was conducted ten times and the results averaged.

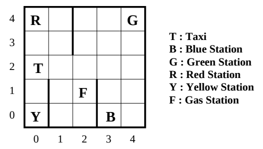

Figure 5. The Taxi-Fuel Domain.

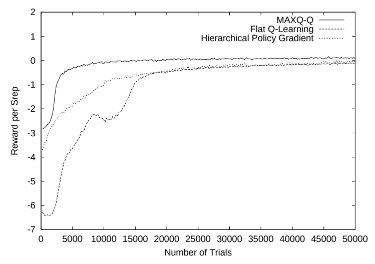

Figure 6. This figure compares the performance of the hierarchical PGRL algorithm proposed in this paper with MAXQ-Q and flat Q-learning on the taxi-fuel problem.

Figure 6 compares the proposed hierarchical policy gradient (HPG) algorithm with MAXQ-Q (Dietterich, 1998), a value-based hierarchical RL algorithm, and flat Q-learning. The graph shows that the MAXQ-Q converges faster than HPG and flat Q-learning, and HPG is slightly faster than flat Q-learning.

The hierarchical policy gradient algorithm used in this experiment is the one shown in Algorithm 1, with one policy parameter for each state-action pair (s, a).

As we expected, the HPG algorithm converges to the same performance as MAXQ-Q, but it is slower than its value-based counterpart. The performance of HPG can be improved by better policy formulation and using more sophisticated policy gradient algorithms for each subtask. The slow convergence of PGRL methods motivates us to use both value and policy based methods in a hierarchy and study how to define more expressive policies for each subtask. We address the former using the *hierarchical hybrid* algorithms in the next section and leave the latter for future work.

5.2. Ship Steering Problem

In this section we apply a *hierarchical hybrid* algorithm to the ship steering task described in section 2 and compare its performance with flat PGRL and actor-critic algorithms.

The flat PGRL algorithm used in this section uses Equation 3 and CMAC function approximator with 9 four dimensional tilings, dividing the space into $20 \times 20 \times 36 \times 5 = 72000$ tiles each. The actor-critic algorithm (Konda, 2002) also uses the above function approximator for its actor, and 9 five dimensional tilings of size $5 \times 5 \times 36 \times 5 \times 30 = 135000$ tiles, for its critic. The fifth dimension of critic's tilings is for continuous action.

In hierarchical hybrid algorithm, we decompose the task using the *task graph* in Figure 3. At the high-level, the learner explores in a low-dimensional subspace of the original high-dimensional state space. The state variables are only the coordinates of the ship x and y on the full range from 0 to 1000. The actions are four diagonal and four horizontal/vertical subtasks similar to those subtasks shown in Figure 2. The state space is coarsely discretized into 400 states. We use the value-based $Q(\lambda)$ algorithm with ϵ-greedy action selection and replacing traces to learn a sequence of diagonal and horizontal/vertical subtasks to achieve the goal of the entire task (passing through the gate). Each episode ends when the ship passes through the gate or moves out of bound. Then the new episode starts with the ship in a randomly chosen position, orientation and turning rate. In this algorithm, λ is set to 0.9, learning rate to 0.1 and ϵ starts with 0.1 remains unchanged until the performances of low-level subtasks reach to a certain level and then is decreased by a factor of 1.01 every 50 episodes.

At the low-level, learner explores local areas of the high-dimensional state space without discretization. When the high-level learner selects one of the low-level subtasks, the low-level subtask takes control and executes the following steps as shown in Figure 2. **1)** Maps the ship to a new coordinate system in which the ship is in position (40,40) for the diagonal subtask and (40,75) for the horizontal/vertical subtask. **2)** Sets the low-level goal to position (140,140) for the diagonal subtask and (140,75) for the horizontal/vertical subtask. **3)** Sets the low-level boundaries to $0 \leq x, y \leq 150$. **4)** Generates primitive actions until either the ship reaches to a neighborhood of the low-level goal, a circle with radius 10 around the low-level goal (success), or moves out of the low-level bounds (failure).

The two low-level subtasks use all four state variables, however the range of coordination variables x and y is 0 to 150 instead of 0 to 1000. Their action variable is the desired turning rate of the ship, which is a continuous variable with range -15 to 15 degrees/sec. The control interval is $0.6sec$ (three times the sampling interval $\Delta = 0.2sec$). They use the policy gradient learning algorithm in lines 11-16 of Algorithm 1 to update their parameters. In addition, they use a CMAC function approximator with 9 four dimensional tilings, dividing the space into $5 \times 5 \times 36 \times 5 = 4500$ tiles each. One parameter w is defined for each tile and the parameterized policy is a Gaussian:

$$\mu(s, a, W) = \frac{1}{\sqrt{2\pi}} e^{-\frac{A}{2}} \qquad , \qquad A = \frac{\sum_{i=0}^{N} w_i \phi_i}{\sum_{i=0}^{N} \phi_i}$$

where $N = 9 \times 4500 = 40500$ is the total number of tiles and ϕ_i is 1 if state s falls in tile i and 0 otherwise.

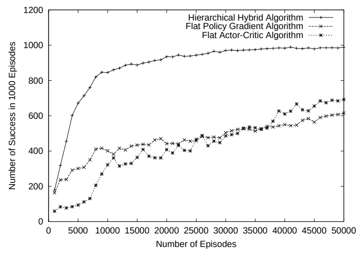

Figure 7. This figure shows the performance of the hierarchical hybrid, flat PGRL and actor-critic algorithms in terms of the number of successful trials in 1000 episodes.

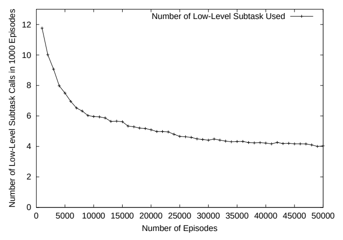

Figure 8. This figure shows the performance of the system in terms of number of low-level subtask calls.

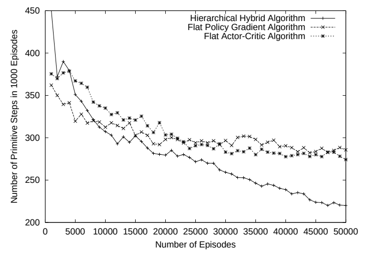

Figure 9. This figure shows the performance of the hierarchical hybrid, flat PGRL and actor-critic algorithms in terms of number of steps to pass through the gate.

The actual action is generated after mapping the value chosen by the Gaussian policy to the range from -15 to 15 *degrees/sec* using a sigmoid function.

In addition to the original reward of -1 per step, we define internal rewards 100 and -100 for low-level success and failure, and a reward according to the distance of the current ship orientation θ to the angle between the current position and low-level goal $\hat{\theta}$ given by

$$G = exp\left(-\frac{\|\theta - \hat{\theta}\|^2}{30 \times 30}\right) - 1$$

where 30(deg) gives the width of the reward function. When a low-level subtask terminates, the only reward that propagates to the high-level is the summation of all -1 rewards per step. In addition to reward received from low-level, high-level uses a reward 100 upon successfully passing through the gate.

We train the system for 50000 episodes. In each episode, the high-level learner (controller located at *root*) selects a low-level subtask, and the selected low-level subtask is executed until it successfully terminates (ship reaches the low-level goal) or it fails (ship goes out of the low-level bounds). Then control returns to the high-level subtask (*root*) again. The following results were averaged over five simulation runs.

Figure 7 compares the performance of the hierarchical hybrid algorithm with flat PGRL and actor-critic algorithms in terms of the number of successful trials in 1000 episodes. As this figure shows, despite the high resolution function approximators used in both flat algorithms, their performance is worse than hierarchical hybrid algorithm. Moreover, their computation time per step is also much more than the hierarchical hybrid algorithm, due to the large number of parameters to be learned.

Figure 8 demonstrates the performance of the system in terms of the average number of low-level subtask calls. This figure shows that after learning, the learner executes about 4 low-level subtasks (diagonal or horizontal/vertical subtasks) per episode.

Figure 9 compares the performance of the hierarchical hybrid, flat PGRL and actor-critic algorithms in terms of the average number of steps to goal (averaged over 1000 episodes). This figure shows that after learning, it takes about 220 primitive actions (turn actions) for hierarchical hybrid learner to pass the gate. Although flat algorithms should show a better performance than hierarchical algorithm in terms of the average number of steps to goal (flat algorithms should find the global optimal policy, whereas hierarchical hybrid algorithm converges just to recursive optimal solution), Figure 9 shows that their performance after 50000 episodes is still worse than the hierarchical hybrid algorithm.

Figures 10 and 11 show the performance of the diagonal and horizontal/vertical subtasks in terms of number of success out of 1000 executions, respectively.

Finally, Figure 12 demonstrates the learned policy for two sample initial points shown with big circles. The upper initial point is $x = 700$, $y = 700$, $\theta = 100$ and $\dot{\theta} = 3.65$ and the lower initial point is $x = 750$, $y = 180$, $\theta = 80$ and $\dot{\theta} = 7.9$. The low-level subtasks chosen by the agent at the high-level are shown by small circles.

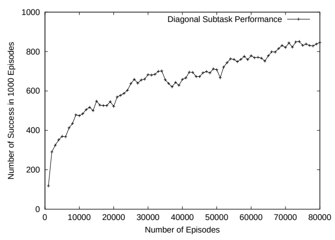

Figure 10. This figure shows the performance of the diagonal subtask in terms of the number of successful trials in 1000 episodes.

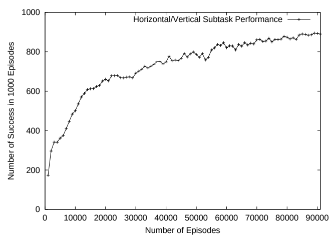

Figure 11. This figure shows the performance of the horizontal/vertical subtask in terms of the number of successful trials in 1000 episodes.

6. Conclusions and Future Work

This paper combines the advantages of hierarchical task decomposition and PGRL methods and describes a class of *hierarchical policy gradient* (HPG) algorithms for problems with continuous state and/or action spaces. To accelerate learning in HPG algorithms, we proposed *hierarchical hybrid* algorithms, in which higher-level subtasks are formulated as VFRL and lower-level subtasks as PGRL problems. The effectiveness of these algorithms was demonstrated by applying them to a simple taxi-fuel problem and a continuous state and action ship steering domain.

The algorithms proposed in this paper are for the case that the overall task is *episodic*. We also formulated these algorithms for the case that the overall task is *continuing*, but we do not include the results for space reasons. In this case, the *root* task is formulated as a *continuing* problem with the *average reward* as its performance function. Since the policy learned at *root* involves policies of its children, the type of optimality achieved at *root* depends on how we formulate other subtasks in the hierarchy. We investigated different notions of optimality in hierarchical average reward, reported in our previous work (Ghavamzadeh & Mahadevan, 2001; Ghavamzadeh & Mahadevan, 2002), for HPG algorithms with *continuing* root task.

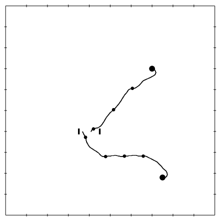

Figure 12. This figure shows the learned policy for two initial points.

Although the proposed algorithms give us the ability to deal with large continuous state spaces, they are not still appropriate to control real-world problems in which the speed of learning is crucial. However, policy gradient algorithms give us this opportunity to accelerate learning by defining more expressive set of parameterized policies for each subtask. The results of ship steering task indicate that in order to apply these methods to real-world domains, a more expressive representation of the policies is needed.

References

Baxter, J., & Bartlett, P. (2001). Infinite-horizon policy-gradient estimation. *Journal of Artificial Intelligence Research*, *15*, 319–350.

Dietterich, T. (1998). The MAXQ method for hierarchical reinforcement learning. *Proceedings of the Fifteenth International Conference on Machine Learning* (pp. 118–126).

Ghavamzadeh, M., & Mahadevan, S. (2001). Continuous-time hierarchical reinforcement learning. *Proceedings of the Eighteenth International Conference on Machine Learning* (pp. 186–193).

Ghavamzadeh, M., & Mahadevan, S. (2002). Hierarchically optimal average reward reinforcement learning. *Proceedings of the Nineteenth International Conference on Machine Learning* (pp. 195–202).

Konda, V. (2002). *Actor-Critic algorithms.* Doctoral dissertation, MIT.

Marbach, P. (1998). *Simulation-based methods for Markov decision processes.* Doctoral dissertation, MIT.

Morimoto, J., & Doya, K. (2001). Acquisition of stand-up behavior by a real robot using hierarchical reinforcement learning. *Robotics and Autonomous Systems*, *36*, 37–51.

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

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