Economic Hierarchical Q-learning

Citation	Schultink, Erik, Ruggiero Cavallo, and David C. Parkes. 2008. Economic hierarchical Q-learning. In Proceedings of the Twenty-third AAAI Conference on Artificial Intelligence and the Twentieth Innovative Applications of Artificial Intelligence Conference: July 13-17, 2008, Chicago, Illinois, ed. American Association for Artificial Intelligence, 689-695. Menlo Park, Calif.: AAAI Press.
Published Version	http://portal.acm.org/citation.cfm?id=1620163.1620179
Accessed	February 18, 2015 12:49:42 PM EST
Citable Link	http://nrs.harvard.edu/urn-3:HUL.InstRepos:4000334
Terms of Use	This article was downloaded from Harvard University's DASH repository, and is made available under the terms and conditions applicable to Other Posted Material, as set forth at http://nrs.harvard.edu/urn-3:HUL.InstRepos:dash.current.terms-of-use#LAA

Economic Hierarchical Q-Learning

Erik G. Schultink, Ruggiero Cavallo and David C. Parkes
School of Engineering and Applied Sciences, Harvard University
Cambridge, MA USA
{schultin,cavallo,parkes}@eecs.harvard.edu

Abstract

Hierarchical state decompositions address the curse-of-dimensionality in Q-learning methods for reinforcement learning (RL) but can suffer from suboptimality. In addressing this, we introduce the *Economic Hierarchical Q-Learning* (EHQ) algorithm for hierarchical RL. The EHQ algorithm uses subsidies to align interests such that agents that would otherwise converge to a recursively optimal policy will instead be motivated to act hierarchically optimally. The essential idea is that a parent will pay a child for the relative value to the rest of the system for "returning the world" in one state over another state. The resulting learning framework is simple compared to other algorithms that obtain hierarchical optimality. Additionally, EHQ encapsulates relevant information about value tradeoffs faced across the hierarchy at each node and requires minimal data exchange between nodes. We provide no theoretical proof of hierarchical optimality but are able demonstrate success with EHQ in empirical results.

Introduction

In Hierarchical Reinforcement Learning (HRL), a human programmer provides a task hierarchy in a domain, specifying how to decompose a task into subtasks. Given such a state decomposition, a reinforcement learning (RL) algorithm can exploit the structure of a problem domain to converge to a solution policy more quickly than RL with flat state spaces (Parr & Russell 1998; Dietterich 2000a). By piecing together policies to subtasks local to each piece of the hierarchy, HRL algorithms construct a solution policy for the entire problem.

But there is a tension. Not all HRL algorithms converge to optimal policies. Some algorithms, such as MAXQQ learning (Dietterich 2000a), reason only about the effect of an action on the local subtask. This leads to a policy that is *recursively optimal* (Dietterich 2000a), as the solution for each subtask can be thought of as locally optimal given the solutions to the other subtasks. For example, if you are passing a gas station on your way to the store but have enough gas to reach the store, it is not optimal to stop if you are considering only the subtask of 'driving to the store'. However, from the global perspective, if you will not have enough gas to reach a gas station from the store when you try to drive home, then it is globally optimal to stop.

HRL algorithms that represent and learn the impact of local actions on the entire problem can converge to *hierarchically optimal* policies (Dietterich 2000a; Andre & Russell 2002; Marthi, Russell, & Andre 2006). We seek to address the same problem with *Economic Hierarchical Q-learning* (EHQ), but seek greater simplicity and information locality by appealing to economically-inspired methods to align local and global interests. We use transfers of reward between agents solving different subtasks in the hierarchy to alter local behavior to promote hierarchically optimal policies. These transfers are the only information exchanged between agents during learning. Upon activating a subtask, the one or more agents representing a subtask *bid* for the right to control the world in performing that subtask.[1] This provides for a bottom-up flow of reward in the hierarchy. Parents also seek to modulate the local policies of children by providing incentive (or *subsidy*) payments to encourage the child to return the world in one state over another state in completing the subtask. This provides for a top-down flow of reward.

An apt metaphor is the notion of outsourcing– agents who must fulfill a contract either take primitive actions to fulfill that contract or else may offer sub-contracts, as specified by the hierarchy, to other agents. EHQ was inspired by the notion that multiple sub-contractors or child agents would bid to compete for the right to fulfill the contract. These bids provide a flow of rewards from children to parents. However, there might be several ways the child can satisfy a contract. If the child chooses the cheapest way from its perspective, the algorithm will be limited to a recursively optimal policy. By allowing the parent to subsidize the different (exit) states that satisfy the contract, the child can internalize the global effect of its local actions. These exit-state subsidies provide a flow of rewards from parents to children. With appropriate exit-state rewards, the locally optimal solution to the subtask will be hierarchically optimal. We provide no theoretical proof that EHQ is able to converge to prices with this property. But we are able to give empirical results that demonstrate its success in a large learning domain.

The philosophy of our approach owes a debt to Baum and Durdanovich's (1998) *Hayek* system, which itself is a variation on the Holland (1986) classifier system. Baum considered the metaphor of auctions with one agent "buying (control of) the world" from another agent and competing in an *evolutionary* setting to find complementary, effective sub-policies for solving larger, complex problems. We adapt and modify Baum's approach to that of HRL and do not consider models of evolutionary computation.

[1] In our current implementation, we have only one agent associated with each subtask and simply program this agent to bid its true expected value for receiving control, i.e. simulating a competitive economic system.

	HAMQ	MAXQQ	ALispQ HOCQ	EHQ
RO conv		✓		
HO conv	✓		✓†	✓†
Value decomp.		✓	✓	✓
State abstr.		✓	✓	✓
Decentralized				✓

Table 1: Summary of HRL algorithms († shown only empirically).

Motivating Example. The *Taxi* domain of Dietterich (2000a) takes place in a grid world and is illustrated in Figure 1. There are four locations, labeled R, B, G, and Y, where the passenger may appear. The goal is to pick the passenger up with the taxi and then drop the passenger off at the destination (also one of R, B, G, and Y). There are 4 primitive actions for movement (`north`, `east`, `south`, and `west`), each yielding -1 reward. If movement in the specified direction would cause the taxi to run into a wall (indicated with bold), the action is a `no-op` with -10 reward. The `pickup` action puts the passenger in the taxi if executed when the passenger and the taxi are in the same marked cell; otherwise `pickup` is a `no-op`. There is an analogous `putdown` primitive action. Both `pickup` and `putdown` yield -1 reward. Executing the `putdown` primitive when the passenger is in the taxi and the taxi is in the correct destination state yields reward of 100 and ends the episode. Each episode begins with the taxi positioned at location (0,1) in the grid-world and a passenger at a random marked state waiting to be taken to a random destination.

The *TaxiFuel* domain, a variation on the Taxi domain, adds a fuel constraint. Each movement primitive decreases the taxi's fuel by 1, unless the taxi has 0 fuel remaining, in which case there is a reward of -10. The taxi can be refueled to its maximum capacity of 10 units if the `fill-up` primitive is executed in location F. The initial fuel for the taxi is 10 units. This variation increases the state space size from $25 \times 5 \times 4 = 500$ to 5500 states (large for flat Q-learning to solve) and has distinct recursively optimal and hierarchically optimal policies.

Related Work

The hierarchical planning work of Dean and Lin (1995) first introduced the concept of breaking apart a Markov Decision Process (MDP) into subtasks. But their approach is about planning rather than learning. HAMQ (Parr & Russell 1998) was the first application of a hierarchical structure to reinforcement learning, and has a formal proof of convergence to a hierarchically optimal policy. But HAMQ uses no state abstraction or value decomposition and learns very slowly. Dietterich (2000a) subsequently introduced MAXQQ and the MAXQ value decomposition, which improved on HAMQ by allowing for state abstraction by decomposing the Q-values for the subtasks at each node, thereby improving learning speed. However MAXQQ is only recursively optimal.

Existing hierarchically optimal RL methods draw on ideas from both HAMQ and MAXQQ and include ALispQ (Andre & Russell 2002) and HOCQ (Marthi, Russell, & Andre 2006).[2] Table 1 summarizes the convergence properties of various HRL algorithms. Only HAMQ and Dietterich's MAXQQ algorithm are supported with formal proofs of convergence. We provide experimental support in this paper for the convergence of EHQ to transfers via bids and subsidies that provide the necessary incentives to induce a hierarchically optimal solution. But again, while RO convergence results are available for EHQ (by reducing to MAXQQ as subsidies converge) we leave theoretical analysis of the hierarchical optimality properties of EHQ for future work.

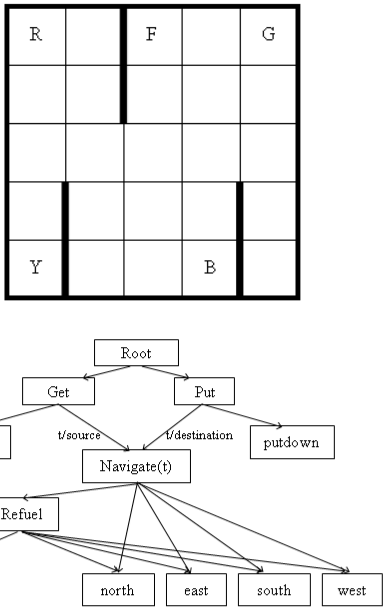

Figure 1: (a) Grid world for Taxi and TaxiFuel. (b) Hierarchy for TaxiFuel.

We believe that EHQ has several advantages over ALispQ and HOCQ. EHQ agents interact in simple, highly structured economic transactions, allowing for greater decoupling than ALispQ or HOCQ, both of which use extensive sharing of information amongst parts of the problem. These earlier methods require that agents have access to the decomposed value functions of all agents below them in the hierarchy, and observe the intrinsic reward that is accrued by the system when any agent below them takes a primitive action.

[2] ALispQ introduced a term, Q_E, representing the expected *non-local reward* for an action, and allowing an agent to reason about its effect on the entire problem and providing convergence to a hierarchically-optimal policy. However the use of Q_E severely limits the possibility of state abstraction and can cause slower convergence than MAXQQ because agents need state information relevant to the rest of the problem to learn Q_E. HOCQ uses the same value decomposition as ALispQ, but does without an explicit representation of Q_E. Agents in HOCQ will instead learn the conditional probability distribution for the exit-state of each subtask given an entry state. Using this probability, it is then possible to compute Q_E directly. Through the use of fairly complex state abstractions and approximations for this probability information, HOCQ can perform significantly better than ALispQ.

Thus, EHQ is not only conceptually simple, but generalizes to massively-distributed multi-agent architectures; this is what is intended by "decentralization" in Table 1.

Hierarchical Reinforcement Learning

In Hierarchical Reinforcement Learning (HRL), hierarchies are used to provide structure and information about a domain, ultimately permitting both temporal and state abstractions. For example, the hierarchy shown in Figure 1 depicts a possible hierarchy for the TaxiFuel domain. The hierarchy decomposes the `Root` task into `Get` and `Put` tasks, which are further broken down into `pickup` and `Navigate(source)`, and `putdown` and `Navigate(destination)`, respectively. All internal nodes represent subtasks and all leaves represent primitive actions in A. The nodes labeled `pickup` and `north` are examples of primitive actions shown in 1.

In our environment we associate an agent with each internal node and interchangeably adopt "node", "agent" and "subtask" in what follows. We represent hierarchies as directed acyclic graphs that decompose an MDP $M = (S, A, P, R)$, for state space S, action space $A(s)$ in state s, transition function $P(s, a, s') \in (0, 1)$ from state s to s' given action $a \in A(s)$, and reward $R(s, a)$ for action a in state s, into a set of *subtasks*, $\{M_0, M_1, ..., M_n\}$. Let $\gamma \in [0, 1)$ denote the discount factor. For consistency with Dietterich (2000a), whom our notation closely follows, we use the convention that M_0 is the `Root` subtask.

Definition 1. *(Dietterich 2000a) A* subtask *M_i is a 3-tuple, (T_i, A_i, R_i) defined as follows:*

- *T_i: a termination predicate, partitioning S into a set of active states S_i and a set of exit-states E_i.*
- *A_i: set of actions that can be performed to achieve M_i, that is $A_i(s) \subseteq A(s) \cup \{M_j : j \in Child(i)\}$ where $Child(i)$ denotes the children of node i.*
- *R_i: a local-reward function, in our setting defined in terms of reward for primitive actions and adjusted for bid and subsidy payments made between agents.*

Primitive actions in the hierarchy can be considered to be subtasks that execute immediately and provide reward $R_i(s, a) = R(s, a)$. Note that not every node need have access to every primitive action; indeed this will generally not be the case. We also adopt the term *macroaction* to refer to a subtask. The macroactions provided by the children of a node are available in every state. Moreover, these are the only macroactions available at this point in the control hierarchy. The hierarchy defines the control logic: upon activating a subtask this subtask can now take available primitive actions or call other subtasks. Eventually, upon completion of the subtask control returns to the calling node to select additional actions. In our setting, it is only the agents that are "*active*" and have control that gain reward from the world for taking primitive actions. Rewards only flow to other agents via bids and subsidies. Like Dietterich, we also allow for parameterized subtasks. The `Navigate(t)` subtask shown in 1 is an example of a parameterized subtask, with possible bindings of the formal parameter t to source and destination.

Policies and Optimality

The notion of a policy can be extended for an MDP M that has been decomposed into a set of subtasks.

Definition 2. *(Dietterich 2000a) A* hierarchical policy *$\pi = \{\pi_0, \pi_1, ..., \pi_n\}$ is defined such that each π_i is a mapping from a state s to either a primitive action a or a policy π_j, where π_j is the policy for a subtask M_j that is a child of the node corresponding to M_i in the hierarchy.*

Definition 3. *(Dietterich 2000a) A* hierarchically optimal *(HO) policy is one that selects the same primitive actions as the optimal policy in every state, except where prevented from doing so by the constraints imposed by the hierarchy.*

Definition 4. *(Dietterich 2000a) A policy is* recursively optimal *(RO) if, for each subtask M_i in the hierarchy, the policy π_i is optimal given the policies for the subtasks that correspond to children of the node associated with M_i, and also their descendents.*

The hierarchies that we consider always allow for the representation of globally-optimal policies. When this is not the case it is generally through bad design, although for certain domains, this may represent a tradeoff between optimality and allowing fast learning through aggressive control structure. To gain some intuition: a hierarchy cannot represent a globally optimal policy if it prevents the optimal sequencing of primitive actions, for instance by requiring a particular sequencing of subtasks each of which has access to only some primitive actions ("first I get dressed, then I make breakfast, then I read the paper,...").

Example 1. *The HOFuel domain, based on an example from Dietterich (2000a), illustrates the distinction between recursive and hierarchical optimality. HOFuel's grid-world and hierarchy are shown in Figure 2. Each movement primitive (`north`, `south`, `east`, and `west`) gives reward of -1 and reduces the fuel level by 1 unit. If the vehicle moves with fuel level of 0, there is additional reward of -10. The primitive `fill-up` can only be executed in the left room, gives reward -1, and fills the fuel level to 5. Reaching the goal gives a reward of 10. Colliding with a wall is a* no-op *with a reward of -10. We treat the doors as one-way passages to avoid infinite loops. The taxi begins with fuel level 5. HOFuel has **144 states** (making it solvable by flat-Q learning) and 5 actions. In HOFuel, the RO policy will exit the left room through the lower door without refueling, since the agent at `Leave left room` cannot see the penalties this will incur in the right room. In contrast, the HO policy is to refuel in the left room before exiting through the upper door. The RO policy yields total reward of -50 and the HO policy yields 3.*

Value Decomposition. Following MAXQ, it is useful to decompose the Q-value $Q(s, a)$ updated by node i during learning (and representing its current belief about its cumulative value for taking action a in state s and then following an optimal local policy, and given the rewards it is receiving via payments to/from other nodes) into the sum of two terms:

- $Q_V(i, s, a)$: the expected discounted reward to i from action a (which might be a macroaction, in which case

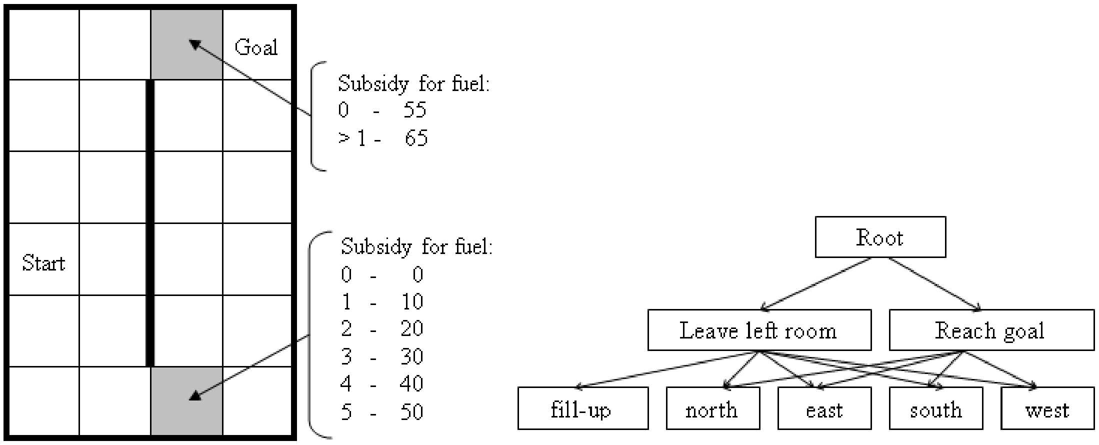

Figure 2: (a) The HOFuel domain, with subsidies assigned to the exit states, (b) A hierarchy for the HOFuel domain.

this will include the reward from the bid made by the child, and also be net any subsidy payment made to the child upon completion, discounted in this case by the number of periods that the macroaction takes to complete)

- $Q_C(i, s, a)$: the expected discounted total reward to i after a completes and until the subtask M_i is completed (this reward includes that achieved from taking primitive actions, and also due to bids and subsidy payments).

In ALispQ and HOCQ the MAXQQ value decomposition was extended to also include an "exit-value" term Q_E to capture the sum of reward to the rest of the system after the current subtask exits. But as discussed earlier, this explicit representation of Q_E (ALispQ), or computation of Q_E via access to nested information in other nodes (HOCQ), leads to complexity and loss of both state abstraction and decentralization. The innovation of EHQ is that Q_V and Q_C will *already reflect the value of a local agent's policy to the entire system* because of the payments that flow between agents.

State Abstractions. In practice it is possible to augment this HRL framework to allow for state abstractions. This provides considerable improvement in learning speed (Dietterich 2000a). The intuition for the usefulness of state abstractions is that not all state variables directly effect the value of a given state-action pair. The HRL framework makes the use of abstractions especially powerful and natural. For example, what you had for breakfast is not relevant to the decision of turning left or right at a particular intersection when you are driving to work and thus irrelevant if the subtask is about navigation rather than eating.

Dietterich (2000a) introduced the idea of abstract states wherein a node (or agent) associated with a subtask reasons about an abstracted state space in which each local state, $\tilde{s}_i$, represents a *subset* of the underlying state space of the MDP M. Everything else proceeds unchanged, just with local states replaced with local abstract states. The abstraction needs to be *safe* (Dietterich 2000b; Andre & Russell 2002). The appropriate notion of safety depends on whether the goal is one of hierarchical optimality or recursive optimality.

Definition 5. *(Dietterich 2000a) A state abstraction is hierarchically (recursively) safe given a hierarchy, if the hierarchically (recursively) optimal policy in the original state space is hierarchically (recursively) optimal in the abstract space.*

HOFuel is an example of a domain in which the appropriate notion of abstraction safety depends on whether the goal is hierarchical or recursive optimality. MAXQQ, which is only capable of learning a recursively optimal solution policy for HOFuel, can completely ignore the fuel level in its abstraction. Either way, it will be unable to learn the effects of the fuel level variable on its reward and will converge to the recursively optimal policy. If EHQ used this level of abstraction, it would similarly be limited to the recursively optimal policy, as it would be unable to reason about the penalty for running out of fuel.

In this paper, we develop our own abstractions where necessary but adopt the conditions set out by Dietterich (2000a) in ensuring that all of the state abstractions are hierarchically safe, and adopt the same abstractions within both the EHQ and MAXQQ algorithms (and noting that any abstraction that is hierarchically safe is trivially recursively safe.)

The EHQ Algorithm

For HO convergence, the rewards to an agent in one part of the hierarchy must represent the impact of local actions on the global solution quality (allowing for hierarchically-optimal policies elsewhere). An EHQ agent passes control of the world to a child agent directly below it in the hierarchy, receiving control back when the child satisfies its termination predicate.[3] In exchange for passing control to its child, the parent receives a transfer of reward equal to the *bid* by the child for the state.[4] The bid provides bottom-up

[3] The `Root` subtask has no predicate; in the episodic setting, it finishes only when the world reaches a terminal state, and in a problem with an infinite horizon it never completes.

[4] We use the term "bid" due to EHQ's inspiration from Hayek (Baum & Durdanovich 1998) and the possible extension

reward flow in the system and this reward is deducted from the child's reward and accrues to the parent. We define the child's bid to be its MDP value $V^*(i,s)$ for the current state of the world, where $V^*(i,s) = \max_{a \in A_i(s)}[Q_V(i,s,a) + Q_C(i,s,a)]$; i.e. simulating a competitive market. The above scheme alone can yield RO convergence. To obtain HO convergence, we also allow the parent in EHQ to pay the child agent in the form of a *subsidy* that depends on the quality of the exit-state the child obtained. The amount of this subsidy, which provides top-down reward flow in the system, is deducted from the parent's reward upon completion of the subtask and accrues to the child. This transfer of reward aligns the incentives of the child with the parent, encapsulating any possible trade-off between exit-states within the child's local value function.

Agents in the EHQ algorithm store Q_V and Q_C locally and update these values in an analogous manner to MAXQQ. This is presented in the pseudocode for EHQ. But the main difference between EHQ and MAXQQ is that the reward that is used for updates of Q_V and Q_C in the case of macroactions is not simply reward for primitive actions, but also includes these bid and subsidy payments. Specifically, the value received from a child as a bid encapsulates the expected reward for the composite of the actions the child will take in completing its subtask as well as the subsidy it expects to receive from the parent upon exiting.

Intuitively, the subsidy mechanism in EHQ allows the parent to lead the child to the exit-state that the parent prefers by placing an additional reward on that state. The hierarchy can be thought of as defining contractual relationships between nodes. For example, a contract may require that the child produce a car for some price. The parent exercises the contract when it calls the child. The child must then fulfill the contract by providing the car, receiving the transfer as compensation. However, the contract may not explicitly specify what color the car must be, leaving the child freedom to provide whichever it can as the lowest cost. The subsidy mechanism provides a way for the parent to express its preference over the color of the car: if the parent values red over blue, it can explicitly provide its value as a subsidy for receiving a red car. If the subsidy is more than the marginal cost to the child of providing a blue car, the child will provide a red car to receive the subsidy.

The parent's ability to subsidize the child is critical for HO convergence. More formally, we can assert that the parent agent should be willing to pay a child agent for returning the world in exit-state e' up to the marginal value of the parent for e' over the exit-state e in which the child would otherwise return the world without a subsidy. Consider an agent, $parent$, implementing a subtask M_i and another agent, $child$, implementing a subtask M_j that is a child of M_i. Define $SUBSIDY(parent, e)$, where e is an exit-state of subtask M_j ($e \in E_j \subseteq S_j$), to be the expected marginal value of e to the $parent$ relative to the **minimal** value across exit states:[5]

$$SUBSIDY(parent, e) = V^*_{\text{parent}}(e) - \min_{e' \in E_j}\left[V^*_{\text{parent}}(e')\right]$$

In practice, we found it to be extremely beneficial to limit the set of exit-states used for the purpose of defining the subsidy to the *reachable* exit-states. The subsidy is set to zero on all other exit-states. These states are not programmed but discovered by the parent during learning. In many domains, only a few states in E_j can be reached. For instance, HOFuel's `Leave left room` subtask has only 12 reachable exit-states out of a total of 72. Apart from reducing the complexity of the subsidy calculation and improving running time, this simplification appears to reduce volatility in the subsidies in large domains.

For an example of this subsidy policy, subsidies have been defined for several reachable exit-states in HOFuel for subtask `Leave left room` in Figure 2. These subsidies are sufficient to alter the locally optimal policy such that it will exit through the upper door with a fuel level greater than 1.

We provide the following pseudo-code for EHQ, called here for subtask M_i and with a list of subsidies $EXIT\_SUBSIDIES$ provided by the calling agent and depending on the exit state achieved. α_t is the learning rate (cooled over time t) and $\gamma \in [0,1)$ is the discount factor:

```
EHQ($M_i$, $EXIT\_SUBSIDIES$)
while $T_i$ not satisfied in current state $s$ do
  use $Q(i,s,a) = Q_V(i,s,a) + Q_C(i,s,a)$ to choose
  action $a$ from $A_i(s)$ (follow an $\epsilon$-greedy rule)
  if $a$ is a primitive action then
    take action $a$; observe reward $r$ and state $s'$
    observe $N = 1$ steps elapsed
  else
    EHQ($M_a$, $SET\_SUBSIDIES$($M_i$,$M_a$))
    observe $N$ steps elapsed during $M_a$, exit state $s'$
    $r$ := BID($M_a$,$s$) - $\gamma^N SUBSIDY(M_i, s')$
  end if
  if $T_i$ is satisfied in $s'$ then
    $r := r + \gamma^N EXIT\_SUBSIDIES(s')$
  end if
  $Q_V(i,s,a) := (1-\alpha_t)Q_V(i,s,a) + \alpha_t r$
  $Q_C(i,s,a) := (1-\alpha_t)Q_C(i,s,a)+$
    $\alpha_t \gamma^N \max_{a' \in A_i(s')} [Q_V(i,s',a') + Q_C(i,s',a')]$
end while
```

$SET\_SUBSIDIES(M_i, M_a)$ assigns subsidies from M_i to the exit-states of M_a based on the subsidy policy.

The update of Q_V and Q_C here is defined as in MAXQQ. We adopt $N \geq 1$ to denote the number of periods that the action a takes to execute. Note that the value from action a (Q_V) already includes some discounting in our setting, in

of the EHQ algorithm to have competing agents at each node that would enter their bids in an auction to win control from the parent. At present each child simply bids its expected discounted value forward from the state.

[5] We explored alternative definitions to adopting the worst-case exit state as the reference point (e.g., this could be defined relative to the maximal value across exit states, the value of an average exit state, the value of the exit state typically selected, etc.). All are possible because the absolute value does not matter since this part of the reward ultimately flows back to the parent via a child's bid. But we found that adopting the worst-case exit state as the reference point provides good learning speed and good robustness.

that the reward allows for the subsidy payments happening some number of periods after the current period in the case that a is a macroaction. The reward that accrues to this node due to the exit subsidy that the subtask receives upon completion is also similarly discounted. Finally, also observe that Q_C is defined to allow for the possibility that the current action may take $N > 1$ periods to complete (again, in the case that it is a macroaction).

It is possible in this scheme for the parent to pay a much larger subsidy than the reward the parent expects to receive in the future. This is OK, because the subsidy will propagate back through the child's Q_C values and thus into the child's bid, less whatever amount is necessary to change the child's behavior (assuming the subsidy is sufficiently large to affect a change).

It is also interesting to note that when all agents have converged, the transfer of the bid amount between the child and the parent cancels out the *expected* gain or loss of the child, leaving it with a net expected reward of 0. The root ultimately takes all of the surplus. This would be different in a system with actual competition between nodes, wherein each subtask is associated with multiple agents with the competency to perform the subtask and each node would be expected to bid not V^* but the minimal amount to "win (control of) the world."

Results

We present empirical results on the HOFuel and TaxiFuel domains. Following the existing literature we evaluate the performance of EHQ in terms of the number of primitive actions required to learn a good policy (Dietterich 2000a; Andre & Russell 2002; Marthi, Russell, & Andre 2006). The results show that EHQ, through the use of exit-state subsidies, can achieve hierarchically optimal policies, which is to say policies that are superior to those obtained by MAXQQ. We compare EHQ with MAXQQ and also flat Q-learning where possible.

For all three implementations, the learning rate α_t was cooled according to the expression $1/(1 + t/C_\alpha)$, where t is the number of time steps elapsed since the algorithm was initiated and C_α is a tuning parameter. All three used epsilon-greedy exploration policies, where ϵ is set according to $1/(1 + t/C_\epsilon)$, for parameter C_ϵ. All Q values were initialized to 0. The C_α and C_ϵ parameters were tuned for convergence.

EHQ converges to the same policy as MAXQQ in the Taxi domain. That this final policy is the same for EHQ and MAXQQ is unsurprising because the RO, HO, and globally optimal policies are equivalent in the Taxi domain. We present results for the TaxiFuel domain, which has been suggested by Dietterich (2000a) as an example in which the RO and HO policies are not the same, in Figure 3. The problem has 5500 states, making it outside of the scope of what we could achieve with flat Q-learning. In fact, we now believe that this domain has very little difference between HO and RO (and for several initial states there is no difference). The result is that neither trial really shows EHQ doing significantly better than MAXQQ. In both the Taxi and TaxiFuel domain we believe that the performance of EHQ is worse in terms of speed of convergence than MAXQQ because EHQ represents additional Q_V values, nodes do not directly observe the primitive reward accrued by children below them, and because a node's subsidies introduce additional non-stationarity.

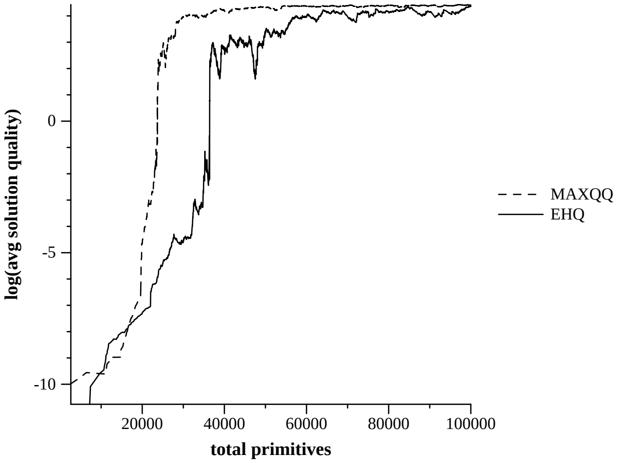

Figure 3: EHQ and MAXQQ convergence on TaxiFuel domain

Results on HOFuel, shown in Figure 4 ($C_\alpha = 2000$, $C_\epsilon = 400$, solution quality plotted is the average for last 30 trials) clearly demonstrate that EHQ and flat Q-learning converged to HO policies while MAXQQ was limited to the RO policy. EHQ was able to propagate information about the penalties in the right room into the value function of the agent in the left room. To illustrate this, we also present a graph showing the convergence of `root`'s EHQ subsidies to `Leave left room`. Note that the subsidies have not converged to quite the correct values, which would be omnisciently set as shown in Figure 2. For HO convergence, we need only that the subsidies be near enough to incentivize the child to choose the exit state that makes the right global trade-off. [6]

Future work should compare the performance of EHQ with HOCQ on the larger version of the Taxi domain (15500 states) considered by Marthi et al. (2006). A similar problem would arise as in our TaxiFuel domain, namely that both domains are too large to be solved by flat Q-learning. These earlier authors compare their algorithm with their own alternative RO and HO algorithms.

Conclusions

The EHQ algorithm provides the innovative contribution of using a hierarchical artificial economy to align local and global interests within a HRL setting. In a conceptually sim-

[6] That final subsidies are approximate is likely caused by the learning parameters being optimized for policy convergence speed rather than subsidy convergence. It may also be the result of interactions as the subsidies and policy are learned simultaneously by all agents in the hierarchy. Moreover, regions of the state space that were unpromising were not fully-explored, so the V^* for these states is less refined.

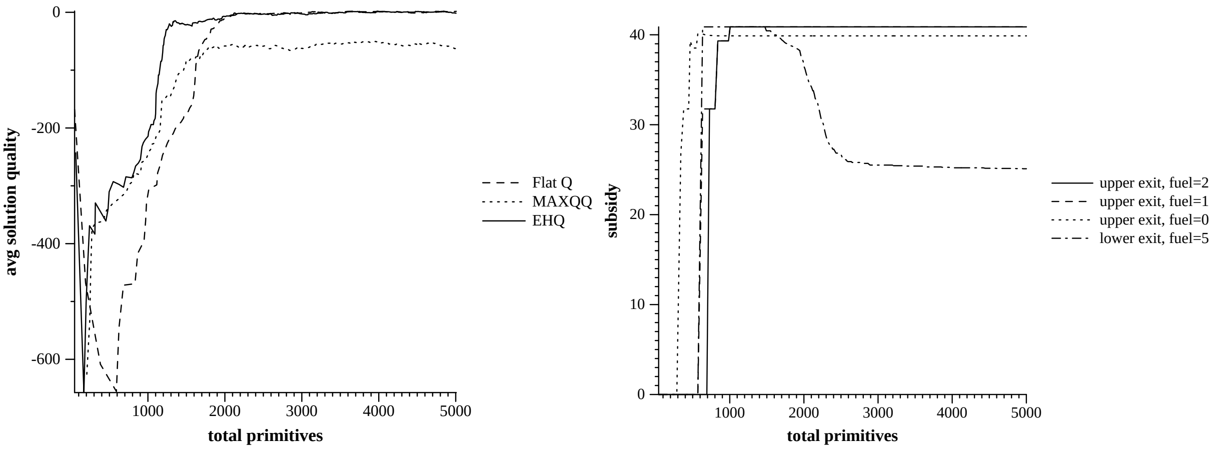

Figure 4: (a) Performance on HOFuel. (b) Exit-state subsidy convergence.

ple structure, an agent bids its expected reward for solving a subtask to provide a flow of rewards from primitive actions around the system. The parent can also subsidize the possible exit-states of the child in order to get the child to produce an exit-state it prefers. Thus, EHQ agents are dependent on their calling context, since a child's local value function incorporates the expected subsidy payment from its parent. The child ultimately returns the world in such a state if the subsidy exceeds the additional cost the child will incur. This simple incentive structure allows EHQ to converge in experiments to hierarchically optimal solution policies, aligning local and global interests and correcting the suboptimality of the recursively-optimal outcomes that can so easily arise.

The chief advantage of EHQ over earlier methods such as ALispQ and HOCQ is its simplicity and decentralization. All HRL algorithms that converge to hierarchically optimal policies need to model some equivalent of Q_E, i.e. the expected non-local reward for taking an action. ALispQ explicitly learns this value, but it often depends on a large number of state variables, which is reflects in ALispQ's slower convergence speed compared to MAXQQ. HOCQ learns the exit-state probability distribution, $P_e(s, a)$, for each node, from which it can compute Q_E given the parent's value function $V_p(e)$. The subsidies in EHQ are based on the parent's value function, and it is these subsidies that provide the equivalent of Q_E, directly via rewards that are incorporated into local values at a node for Q_V and Q_C. EHQ nodes do need not to have direct access to their parent's value function to compute the locally optimal action. The EHQ decomposition provides autonomy and localization: beyond its own action space and state abstraction, an agent need only know the children one level below it in the hierarchy. This gives great potential to use this approach in highly distributed or multi-agent settings, for which earlier methods are poorly motivated.

Acknowledgments

The first author thanks Avi Pfeffer and Rob Wood for agreeing to be readers of his senior thesis. We are also grateful for the constructive comments of three anonymous reviewers.

References

Andre, D., and Russell, S. 2002. State abstraction for programmable reinforcement learning agents. In *AAAI-02*. Edmonton, Alberta: AAAI Press.

Baum, E. B., and Durdanovich, I. 1998. Evolution of cooperative problem-solving in an artificial economy. *Journal of Artificial Intelligence Research*.

Dean, T., and Lin, S.-H. 1995. Decomposition techniques for planning in stochastic domains. In *IJCAI-95*, 1121–1127. San Francisco, CA: Morgan Kaufmann Publishers.

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

Dietterich, T. G. 2000b. State abstraction in MAXQ hierarchical reinforcement learning. *Advances in Neural Information Processing Systems* 12:994–1000.

Holland, J. 1986. Escaping brittleness: The possibilities of general purpose learning algorithms applied to parallel rule-based systems. In *Machine Learning*, volume 2. San Mateo, CA: Morgan Kaufmann.

Marthi, B.; Russell, S.; and Andre, D. 2006. A compact, hierarchically optimal Q-function decomposition. In *UAI-06*.

Parr, R., and Russell, S. 1998. Reinforcement learning with hierarchies of machines. *Advances in Neural Information Processing Systems* 10.