HQ-LEARNING: DISCOVERING MARKOVIAN SUBGOALS FOR NON-MARKOVIAN REINFORCEMENT LEARNING

TECHNICAL REPORT IDSIA-95-96

Marco Wiering Jürgen Schmidhuber
IDSIA, Corso Elvezia 36, CH-6900-Lugano, Switzerland
marco,juergen@idsia.ch - http://www.idsia.ch

October 9, 1996

Abstract

To solve partially observable Markov decision problems, we introduce HQ-learning, a hierarchical extension of Q-learning. HQ-learning is based on an ordered sequence of subagents, each learning to identify and solve a Markovian subtask of the total task. Each agent learns (1) an appropriate subgoal (though there is no intermediate, external reinforcement for "good" subgoals), and (2) a Markovian policy, given a particular subgoal. Our experiments demonstrate: (a) The system can easily solve tasks standard Q-learning cannot solve at all. (b) It can solve partially observable mazes with more states than those used in most previous POMDP work. (c) It can quickly solve complex tasks that require manipulation of the environment to free a blocked path to the goal.

Keywords: reinforcement learning, hierarchical Q-learning, POMDPs, non-Markovian interfaces, subgoal learning.

1 Introduction

The problem. If a learner's optimal next action always depends only on its current input, we speak of a *Markovian interface* between learner and environment. The most widely used reinforcement learning (RL) algorithms, such as TD(λ) (Sutton 1988) and Q-learning (Watkins 1989; Watkins and Dayan 1992), depend on Markovian interfaces; they fail if the problem requires memory of previous events. Such non-Markovian interfaces, however, are common in the real world — even Markovian environments may appear non-Markovian to the learner due to a lack of perfect information about the current environmental state. The problem of controlling a system in such partially observable environments can be cast in the *partially observable Markov decision problem* (POMDP) framework. There are a number of algorithms for POMDPs, e.g., Schmidhuber (1991), McCallum (1993), Ring (1994), Kaelbling et al. (1995), Jaakkola et al. (1995), Littman (1995), though most of them are feasible only for small problems. This paper presents a novel, quite different approach which appears to scale far more reasonably. For alternative approaches to larger scale POMDPs, see also Schmidhuber et al. (1996), Wiering and Schmidhuber (1996), and Zhao and Schmidhuber (1996).

Basic idea. In realistic environments some memory of previous events is required to select the optimal next action. Often, however, it is not necessary to memorize the entire past (in general this would be quite infeasible) — a few memories corresponding to important previously achieved subgoals can be sufficient. For instance, suppose your instructions for the way to the station were this: "Follow this road to the traffic light, turn left, follow that road to the next traffic light, turn right, there you are.". While you are on your way, only few memories corresponding to the important subgoals are relevant, such as "I already passed the first traffic light". In-between two subgoals a reactive, memory-independent strategy will carry you safely. This idea is incorporated in HQ-learning, a novel, hierarchical extension of Watkins' Q-learning. HQ-learning's divide-and-conquer strategy learns subgoals to decompose a possibly non-Markovian task into simpler, Markovian subtasks. The system uses multiple subagents. Each agent's *policy* is a mapping from states to actions. At a given time, the system's only type of short-term memory is embodied by a pointer indicating which agent is active. Each agent learns a context-specific strategy for solving its subgoal. Policies of different agents are combined in a way learned by the agents themselves. The first active agent uses a subgoal table (its *HQ-table*) to generate a subgoal for itself (for instance, subgoals can be represented as desired inputs). Then it follows the policy embodied by its Q-table until it achieves its subgoal. Then control is passed to the next agent, and the procedure repeats itself. After the overall goal is achieved or a time limit is exceeded, each agent uses a novel learning procedure (to be described in section 2) to adjust its policy and its subgoal. Although each agent learns only "Markovian" subproblems, the whole system can learn "non-Markovian" tasks impossible to learn with single lookup tables. Unlike, e.g., Singh's system (1992) and Lin's hierarchical learning method (1993), ours does not depend on an external teacher who provides *a priori* information about "good" subtasks. Unlike Jaakkola et al.'s method (1995), ours is not limited to finding *suboptimal* stochastic policies for POMDPs with an *optimal* deterministic solution.

Outline. Section 2 describes HQ-learning details, including learning rules for both Q- and HQ-tables. Section 3 describes experiments with relatively complex partially observable mazes. In the first experiment the system solves a POMDP that standard Q-learning cannot solve by automatically decomposing it into three appropriate Markovian subtasks. In the second experiment it solves a complex POMDP (with 960 world states) that requires finding a key to open a door blocking the path to the goal. Section 4 briefly reviews related work that has mainly been tested on small problems, as most previous methods do not scale up very well (Littman 1995). Section 5 concludes and lists directions for future research.

2 HQ-LEARNING

POMDP specification. System life is separable into "trials". A trial consists of at most T_{max} discrete time steps $t = 1, 2, 3, \ldots$,T. The POMDP is specified by $\mathcal{Z} =< S, S_1, O, B, A, R, \gamma, D >$, where S is a finite set of environmental states, $S_1 \in S$ is the initial state, O is a finite set of observations, the function $B : S \rightarrow O$ maps states to (ambiguous) observations, A is a finite set of actions, $R : S \times A \rightarrow I\!R$ maps state-action pairs to scalar reinforcement signals, $0 \leq \gamma \leq 1$ is a discount factor which trades off immediate rewards against future rewards, and $D : S \times A \rightarrow S$ is a state transition function. Though the framework can be extended to non-deterministic worlds, we focus on deterministic state transition functions for simplicity: $S_{t+1} := D(S_t, A_t)$, where $S_t \in S$ is the environmental state at time t, and $A_t \in A$ is the action executed at time t. The system's goal is to obtain maximal (discounted) cumulative reinforcement during the trial.

Architecture. There is an ordered sequence of M agents $\mathcal{C}_1$, $\mathcal{C}_2$, ... $\mathcal{C}_M$, each equipped with a Q-table, an HQ-table, and a transfer control unit, except for $\mathcal{C}_M$, which only has a Q-table (see figure 1). Each agent is responsible for learning part of the system's policy. Its Q-table represents its local policy for executing an action given an input. It is given by a matrix of size $|O| \times |A|$, where $|O|$ is the number of different possible observations and $|A|$ the number of possible actions. $Q_i(O_t, A_j)$ denotes $\mathcal{C}_i$'s Q-value (utility) of action A_j given observation O_t. The agent's current subgoal is generated with the help of its HQ-table, a vector with $|O|$ elements. $HQ_i(O_j)$ denotes $\mathcal{C}_i$'s HQ-value (utility) of selecting O_j as its subgoal. For each possible observation there is a HQ-table entry representing its estimated value as a subgoal.

The system's current policy is the policy of the currently *active* agent. If $\mathcal{C}_i$ is active at time step t, then we will denote this by $Active(t) := i$. The information about which agent is active represents the only kind of short-term memory in the system.

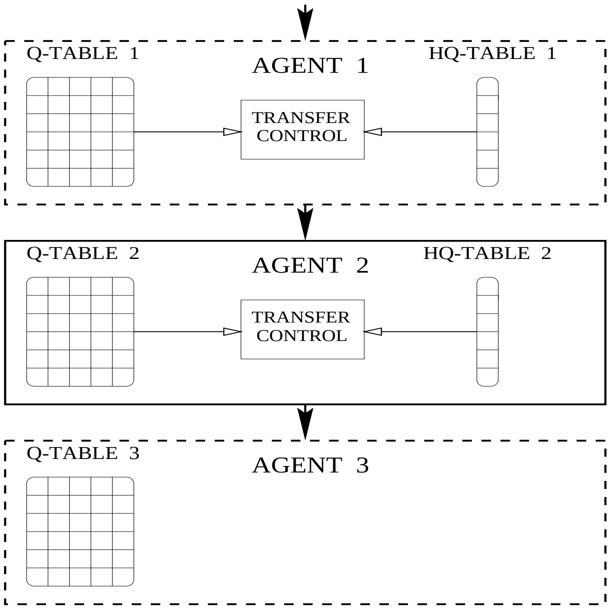

Figure 1: *Basic architecture. Three agents are connected in a sequential way. Each agent has a Q-table, an HQ-table, and a transfer control unit, except for the last agent which only has a Q-table. The Q-table stores estimates of actual observation/action values and is used to select the next action. The HQ-table stores estimated subgoal values and is used to generate a subgoal once the agent is made active. The solid box indicates that the second agent is the currently active agent. Once the agent has achieved its subgoal, the transfer control unit passes control to its successor.*

Selecting a subgoal. In the beginning $\mathcal{C}_1$ is made active. Once $\mathcal{C}_i$ is active, its HQ-table is used to select a subgoal for $\mathcal{C}_i$. Subgoal O_j is chosen with probability $P^{HQ_i}(O_j)$ according to the Max-Uniform distribution:

$$P^{HQ_i}(O_j) := Pr_{max} \frac{Max_i(O_j)}{\sum_{O_k \in O} Max_i(O_k)} + \frac{1 - Pr_{max}}{|O|}. \tag{1}$$

Here Pr_{max} denotes the probability of using the "max-choice rule" (which chooses a subgoal with maximal HQ-value) for subgoal selection. $Max_i(O_j)$ returns 1 if $\forall O_k \in O: \ HQ_i(O_k) \leq HQ_i(O_j)$, and 0 otherwise. $\hat{O}_i$ denotes the subgoal selected by agent $\mathcal{C}_i$. This subgoal is only used in transfer control as defined below and should not be confused with an observation.

Selecting an action. $\mathcal{C}_i$'s action choice depends only on the current observation O_t. During learning, at time t, the active agent $\mathcal{C}_i$ will select action A_j with probability $P_t^{Q_i}(A_j)$ according to the Max-Boltzmann distribution:

$$P_t^{Q_i}(A_j) := Pr_{max} \frac{Max_i(O_t, A_j)}{\sum_{A_k \in A} Max_i(O_t, A_k)} + (1 - Pr_{max}) \frac{e^{Q_i(O_t, A_j)/T_i}}{\sum_{A_k \in A} e^{Q_i(O_t, A_k)/T_i}}.$$

The function $Max_i(O_t, A_j)$ returns 1 if $\forall A_k \in A: \ Q_i(O_t, A_k) \leq Q_i(O_t, A_j)$, and 0 otherwise. The "temperature" T_i adjusts the degree of randomness involved in agent $\mathcal{C}_i$'s action selection (as long as $Pr_{max} < 1$).

Why use the Max-Boltzmann (Max-Uniform) distribution for Q-values (HQ-values)? Because these distributions can prevent over-exploration, which sometimes makes policies unstable (for discussions of exploration issues see, e.g., Fedorov 1972; Schmidhuber 1991; Thrun 1992; Cohn 1994; Caironi and Dorigo 1994; Storck et al. 1995; Wilson 1996b). These distributions also make it easy to reduce the relative weight of exploration (as opposed to exploitation). During learning we can increase Pr_{max} until it finally becomes 1 to obtain a deterministic policy at the end of the learning process. Schraudolph et al. (1994) also used these mixture distributions to train their TD-Go networks (Schraudolph, personal communication, 1996).

Transfer control. Transfer of control from one active agent to the next is implemented as follows. Each time $\mathcal{C}_i$ has executed an action, its transfer control unit checks whether $\mathcal{C}_i$ has reached the goal. If not, it checks whether $\mathcal{C}_i$ has solved its subgoal to decide whether control should be passed on to $\mathcal{C}_{i+1}$. We let t_i denote the time at which agent $\mathcal{C}_i$ is made active (at system start-up, we set $t_1 \leftarrow 1$).

IF no absorbing state reached AND current subgoal $= \hat{O}_i$
AND $Active(t) < M$ AND $B(S_t) = \hat{O}_i$
THEN $Active(t+1) \leftarrow Active(t) + 1$ AND $t_{i+1} \leftarrow t+1$

2.1 LEARNING RULES

We use off-line learning for updating the tables (no intra-trial changes). The learning rules appear very similar to those of conventional Q-learning. One major difference though is that each agent's prospects of achieving its subgoal tend to vary as various agents try various subgoals.

Learning the Q-values. We want $Q_i(O_t, A_j)$ to approximate the system's expected discounted future reward for executing action A_j, given O_t. In the optimal case we have

$$Q_i(O_t, A_j) = \sum_{S_j \in S} P_t(S_j | O_t, \Theta, i)(R(S_j, A_j) + \gamma V_{Active(t+1)}(B(D(S_j, A_j)))),$$

where $P_t(S_j|O_t, \Theta, i)$ denotes the probability that the system is in state S_j at time t given observation O_t, all architecture parameters denoted Θ, and the information that $i = Active(t)$. HQ-learning does not depend on estimating this probability (although a world model might help to speed up learning, e.g., Moore 1993). $V_i(O_t)$ is the utility of observation O_t according to agent $\mathcal{C}_i$, which is equal to the Q-value for taking the best action: $V_i(O_t) := Max_{A_j \in A}\{Q_i(O_t, A_j)\}$. Q-value updates are generated in two different cases ($T \leq T_{max}$ denotes the total number of executed actions during the current trial, and α^Q is the learning rate):

Q.1 If $\mathcal{C}_i$ is active at time t, and $\mathcal{C}_j$ is active at time $t+1$, and $t < T$ then
$Q_i(O_t, A_t) \leftarrow (1 - \alpha^Q) Q_i(O_t, A_t) + \alpha^Q (R(S_t, A_t) + \gamma V_j(O_{t+1}))$.

Q.2 If agent $\mathcal{C}_i$ is active at time T, and the final action A_T has been executed, then
$Q_i(O_T, A_T) \leftarrow (1 - \alpha^Q) Q_i(O_T, A_T) + \alpha^Q R(S_T, A_T)$.

As mentioned above, the update rules resemble normal one-step Q-learning. A main difference is that agents can be trained on Q-values which are not their own (see [Q.1]).

Learning the HQ-values. We want the HQ-values $HQ_i(O_j)$ to converge to the expected (discounted) future cumulative reinforcement given subgoal O_j and current system policy. In the optimal case we have

$$HQ_i(O_j) = \mathcal{E}(R_i) + \gamma^{t_{i+1} - t_i} HV_{i+1},$$

where $R_i = \sum_{t=t_i}^{t_{i+1}-1} \gamma^{t-t_i} R(S_t, A_t)$, $\mathcal{C}_i$'s discounted cumulative reinforcement during the time it will be active (note that this time interval and the states encountered by $\mathcal{C}_i$ depend on $\mathcal{C}_i$'s subtask), and where $HV_i := Max_{O_l \in O}\{HQ_i(O_l)\}$), the estimated discounted cumulative reinforcement to be received by $\mathcal{C}_i$.

In a given trial, we adjust only HQ-values of agents active during that trial. HQ-table updates resemble Q-table updates (α^{HQ} denotes the learning rate, and $\hat{O}_i$ the chosen subgoal for agent $\mathcal{C}_i$):

HQ.1 If $\mathcal{C}_i$ is invoked before agent $\mathcal{C}_{N-1}$, then we update according to
$HQ_i(\hat{O}_i) \leftarrow (1-\alpha^{HQ})HQ_i(\hat{O}_i) + \alpha^{HQ}(R_i + \gamma^{t_{i+1}-t_i} HV_{i+1})$.

HQ.2 If $\mathcal{C}_i = \mathcal{C}_{N-1}$, then $HQ_i(\hat{O}_i) \leftarrow (1-\alpha^{HQ})HQ_i(\hat{O}_i) + \alpha^{HQ}(R_i + \gamma^{t_N - t_i} R_N)$.

HQ.3 If $\mathcal{C}_i = \mathcal{C}_N$, and $i < M$, then $HQ_i(\hat{O}_i) \leftarrow (1-\alpha^{HQ})HQ_i(\hat{O}_i) + \alpha^{HQ} R_i$.

The first and third rules resemble traditional Q-learning rules. The second rule is necessary if agent $\mathcal{C}_N$ learned a (possibly high) value for a subgoal that is unachievable due to subgoals selected by previous agents.

Comment. Although Q-tables and HQ-tables do not explicitly communicate they influence each other. This results in complex dynamics quite different from those of conventional Q-learning.

TD(λ)-modification. To speed up learning we may use the TD(λ)-method to modify the learning rules above in a manner analogous to Lin's (1993). This changes update details as follows:

Q(λ).1 For the Q-tables we first compute desired Q-values $Q'(O_t, A_j)$ for $t = 1, \ldots, T$:
$Q'(O_T, A_T) \leftarrow R(S_T, A_T)$
$Q'(O_t, A_t) \leftarrow R(S_t, A_t) + \gamma((1-\lambda)V_{Active(t+1)}(O_{t+1}) + \lambda Q'(O_{t+1}, A_{t+1}))$

Q(λ).2 Then we update the Q-values, beginning with $Q_N(O_T, A_T)$ and ending with $Q_1(O_1, A_1)$, according to
$Q_i(O_t, A_t) \leftarrow (1-\alpha^Q)Q_i(O_t, A_t) + \alpha^Q Q'(O_t, A_t)$

HQ(λ).1 For the HQ-tables we also compute desired HQ-values $HQ'_i(\hat{O}_i)$ for $i = 1, \ldots, N$:
$HQ'_N(\hat{O}_N) \leftarrow R_N$
$HQ'_{N-1}(\hat{O}_{N-1}) \leftarrow R_{N-1} + \gamma^{t_N - t_i} R_N$
$HQ'_i(\hat{O}_i) \leftarrow R_i + \gamma^{t_{i+1}-t_i}((1-\lambda)HV_{i+1} + \lambda HQ'_{i+1}(\hat{O}_{i+1}))$

HQ(λ).2 Then we update the HQ-values for agents $\mathcal{C}_1, \ldots, \mathcal{C}_{Min(N,M-1)}$ according to
$HQ_i(\hat{O}_i) \leftarrow (1-\alpha^{HQ})HQ_i(\hat{O}_i) + \alpha^{HQ} HQ'_i(\hat{O}_i)$

3 Experiments

We test our system on two tasks involving non-Markovian interfaces between learner and environment. The first task is to find a path from start to goal in a partially observable 10 × 10-maze. This POMDP can be collectively solved by three or more "Markovian" agents. We study system performance as more agents are added. The second, quite complex task involves finding a key which opens a door blocking the path to the goal. The optimal solution (which requires at least 3 "Markovian" agents) takes 83 steps.

3.1 Learning to Solve a Partially Observable Maze

Task. The first experiment involves the partially observable maze shown in figure 2A. The system has to discover a path leading from start position S to goal G. There are four actions with obvious semantics: *go west, go north, go east, go south.* There are 16 possible observations: the agent can only "see" which of the 4 adjacent fields are blocked. Although there are 62 possible agent positions, there are only 9 highly ambiguous inputs. (Not all of the 16 possible observations can occur in this maze; this means that the system may occasionally generate unsolvable subgoals, such that control will never be transferred to another agent.) There is no deterministic, memory-free policy for solving this task. For instance, input 5 stands for "fields to the left and to the right

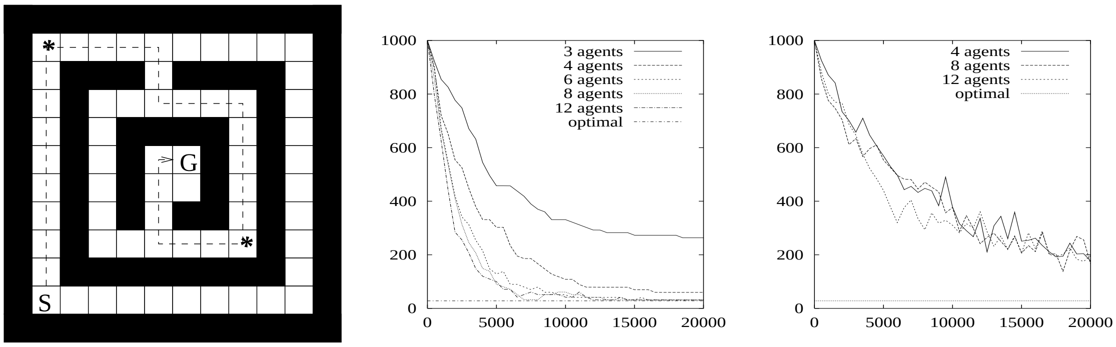

Figure 2: *(A) A partially observable maze (POM). The task is to find a path leading from start S to goal G. Although there are 62 possible agent positions, there are only 9 different, highly ambiguous inputs — some kind of memory is necessary to disambiguate them. That's why conventional Q-learning fails to solve this problem. The optimal solution requires 28 steps and at least three "Markovian" agents. The figure shows a possible solution that costs 30 steps. Asterisks mark appropriate "Markovian" subgoals. (B) HQ-learning results for the partially observable maze, for 3, 4, 6, 8, and 12 agents. We plot average test run length against trial numbers (means of 100 simulations). The system almost always converges to near-optimal solutions. Using more than the required 3 agents tends to improve performance. (C) Results for 4, 8, and 12 agents whose actions are corrupted by 10% noise. In most cases they find the goal, although noisy actions decrease performance.*

of the agent are blocked". The optimal action in response to input 5 depends on the subtask: at the beginning of a trial, it is "go north", later "go south", near the end, "go north" again. Hence at least three "Markovian" agents are necessary to solve this POMDP.

Reward function. Only if the system hits the goal, it receives a reward of 100. Otherwise the reward is zero. The discount factor $\gamma = .9$.

Parameters and experimental set-up. We compare systems with 3, 4, 6, 8, and 12 agents for noise-free actions. We also compare systems with 4, 8, and 12 agents whose actions selected during learning/testing are replaced by random actions with probability 10%. One experiment consists of 100 simulations of a given system. Each simulation consists of 20,000 trials. T_{max} is 1000. After every 500th trial there is a test run during which actions and subgoals with maximal table entries are selected (Pr_{max} is set to 1.0). If the system does not find the goal during a test run, then the trial's outcome is counted as 1000 steps.

After a coarse search through parameter space, we use the following parameters for all experiments: $\alpha^Q = .05$, $\alpha^{HQ} = .2$, $\forall i: \ T_i = .1$, $\lambda = .9$ for both HQ-tables and Q-tables. Pr_{max} is set to .9 and linearly increased to 1.0. All table entries are initialized with 0.

For purposes of comparison, we also ran 20,000 trials during which at most 1000 actions were picked randomly.

Results. Figure 2B plots average test run length against trial numbers. Within 20,000 trials all systems almost always find near-optimal deterministic policies. (Q-learning by itself fails miserably, of course.)

Consider Table 1. The largest systems are always able to decompose the POMDP into Markovian subtasks. The average number of steps is close to optimal. In approximately 1 out of 8 cases, the optimal path is found. In most cases one of the 30-step solutions is found. Since the number of 30-step solutions is much larger than the number of 28-step solutions (there are many more possible subgoal sequences), this result is not surprising.

Systems with more than 3 agents are performing better — here the system profits from having more free parameters. More than 6 agents don't help though. All systems perform significantly

System	Av. steps	(%) Found Goal	(%) Optimal
3 agents	263	76	3
4 agents	60	97	6
6 agents	31	100	14
8 agents	31	100	12
12 agents	32	100	6
4 agents 10% noise	177	86	2
8 agents 10% noise	166	87	2
12 agents 10% noise	196	84	0
Random	912	19	0

Table 1: *HQ-learning results for random actions replacing the selected actions with probability 0% and 10%. The 2nd column lists average numbers of steps required to find the goal. The 3rd column lists numbers of simulations during which the goal is found in the final trial. The 4th column lists numbers of simulations during which the optimal path is found in the final trial.*

better than the random system, which finds the goal in only 19% of all 1000 step trials.

In case of noisy actions (the probability of replacing a selected action by a random action is 10%), the systems still reach the goal in most of the simulations (see figure 3C). In the final trial of each simulation, systems with 4 (8, and 12) agents find the goal with probability 86% (87%, and 84%). There is no significant difference between smaller and larger systems.

We also studied how the system adds agents during the learning process. The 8-agent system found solutions using 3 (4, 5, 6, 7, 8) agents in 8 (19, 16, 17, 21, 19) simulations. Using more agents tends to make things easier. During the first few trials 3 agents were used on average. During the final trials 6 agents were used on average. Less agents tend to lead to better results, however. Why? Systems that fail to solve the task with few subgoals start using more subgoals until they become successful. But the more subgoals there are, the more possibilities to compose paths, and the lower the probability of finding a shortest path in this maze.

3.2 The Key and the Door

Task. The second experiment involves the 26×23 maze shown in figure 3A. Starting at S, the system has to (1) fetch a key at position K, (2) move towards the "door" (the shaded area) which normally behaves like a wall and will open (disappear) only if the agent is in possession of the key, and (3) proceed to goal G. There are only 11 different, highly ambiguous inputs — the task is a difficult POMDP. The optimal path takes 83 steps.

Reward function. Once the system hits the goal, it receives a reward of 500. For all other actions there is a reward of -0.1. There is *no* additional, intermediate reward for taking the key or going through the door. The discount factor $\gamma = 1.0$.

Parameters. The experimental set-up is analogous to the one in section 3.1. We use systems with 3, 4, 6 and 8 agents, and systems with 8 agents whose actions are corrupted by different amounts of noise (5%, 10%, and 25%). $\alpha^Q = .05$, $\alpha^{HQ} = .01$ $\forall i: T_i = .2$. Pr_{max} is linearly increased from .4 to .8. Other parameters are the same as in section 3.1. One simulation consists of 20,000 trials.

Results. We first ran 20,000 1000 step trials of a system executing random actions. It never found the goal. Then we ran the random system for 3000 10,000 step trials. The shortest path ever found took 1,174 steps. We observe: finding the goal at all without any negative reinforcement signals is extremely difficult.

Figure 3B and Table 2 show HQ-learning results for noise-free actions. Within 20,000 trials good, deterministic policies are found in almost all simulations. Optimal 83 step paths are found with 3 (4, 6, 8) agents in 8% (9%, 8%, 6%) of all simulations.

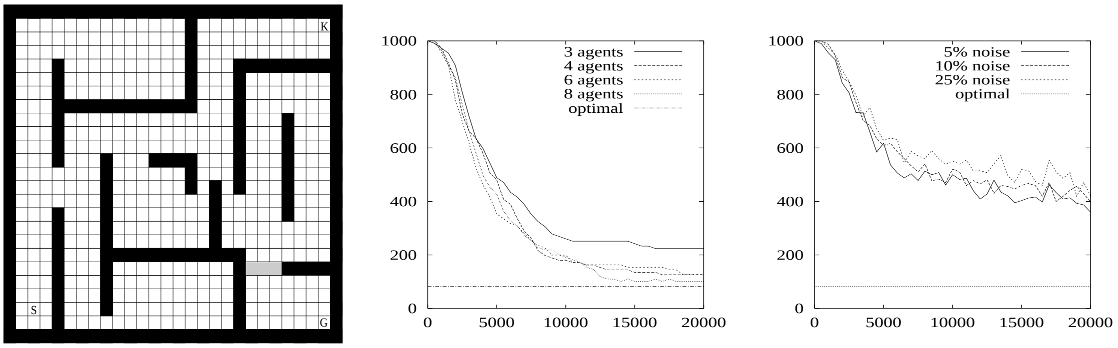

Figure 3: *(A) A partially observable maze containing a key K and a door (grey area). Starting at S, the system first has to find the key to open the door, then proceed to the goal G. The shortest path costs 83 steps. This optimal solution requires at least three "Markovian" agents. The number of possible world states is 960, which is much higher than in most POMDPs studied by other authors. (B) HQ-learning results for this "key and door problem". We plot average test run length against trial number (means of 100 simulations). Within 20,000 trials systems with 3 (4, 6 and 8) agents find good deterministic policies in 85% (96%, 96% and 99%) of the simulations. (C) HQ-learning results with an 8 agent system whose actions are replaced by random actions with probability 5%, 10%, and 25%.*

If random actions are taken in 5% (10%, 25%) of all cases, the 8 agent system still finds the goal in 92% (90%, 84%) of the final trials (see table 2). In many cases long paths (300 — 700 steps) are found. The best solutions use only 84 (91, 118) steps, though. Interestingly, a little noise (e.g. 5%) does decrease performance, but much more noise does not lead to much worse results. We do not yet know whether this is due to HQ-learning or the task set-up or both.

System	Av. steps	(%) Found Goal	(%) Optimal
3 agents	224	85	8
4 agents	126	96	9
6 agents	127	96	8
8 agents	101	99	6
8 agents (5% noise)	360	92	0
8 agents (10% noise)	399	90	0
8 agents (25% noise)	442	84	0
Random	*9310	19	0

Table 2: *Results of 100 HQ-learning simulations for the "key and door" task. The second column lists average numbers of steps required to find the goal. The third lists numbers of simulations during which the goal was found in the final trial. The fourth lists numbers of simulations during which the optimal path of 83 steps was found in the final trial. HQ-learning could solve the task with a limit of 1000 steps per trial. Random search needed a 10,000 step limit.*

4 Previous Work

Other authors proposed hierarchical reinforcement learning techniques to improve performance on Markov decision problems (MDPs), e.g., Dayan and Hinton (1993), Moore (1993), Tham (1995). However, their methods are based on the Markov assumption. Since the focus of our paper is on POMDPs, this section is limited to a brief summary of previous POMDP approaches with specific advantages and disadvantages.

Recurrent neural networks. There are two interacting, gradient-based recurrent networks. The "model network" serves to model (predict) the environment, the other one uses the model net to compute gradients maximizing reinforcement predicted by the model (Schmidhuber 1991; extending ideas by Nguyen and B. Widrow 1989; and Jordan and Rumelhart 1990). To our knowledge this work presents the first successful reinforcement learning application to simple non-Markovian tasks (e.g., learning to be a flipflop). Lin (1993) also uses combinations of controllers and recurrent nets. He compares time-delay neural networks (TDNNs) and recurrent neural networks. Despite their theoretical power, standard recurrent nets run into practical problems in case of long time lags between relevant input events. Although there are recent approaches to overcome this problem (Hochreiter and Schmidhuber 1996), there are no reinforcement learning applications yet.

Belief vectors etc. Kaelbling et al. (1995) hierarchically build policy trees to calculate optimal policies in stochastic, partially observable environments. For each possible environmental state, a "belief vector" represents the agent's estimate of the probability of currently being in this state. The belief vector is updated after each observation. Operation research algorithms are used to compute optimal actions by dynamic programming. Problems with this approach are that the nature of the underlying MDP needs to be known, and that it is computationally very expensive.

McCallum's utile distinction memory (1993) combines Hidden Markov Models (HMMs) with Q-learning. It is able to solve simple POMDPs (maze tasks with only a few fields) by splitting "inconsistent" HMM states whenever the agent fails to predict their utilities (but instead experiences quite different returns from these states). One problem of the approach is that it cannot solve problems in which conjunctions of successive perceptions are useful for predicting reward while independent perceptions are irrelevant. HQ-learning does not have this problem — it deals with perceptive conjunctions by using multiple agents if necessary.

Littman et al. (1995) compare different POMDP algorithms using belief vectors. They report that "small POMDPs" (with less than 10 states and few actions) do not pose a very big problem for most methods. Larger POMDPs (50 to 100 states), however, cause major problems. This indicates that the problems in the current paper (which involve 62 and 960 states) can hardly be solved by such methods. HQ-learning, by contrast, is neither computationally complex nor requires knowledge of the underlying MDP. In absence of prior knowledge this can be a significant advantage.

Memory bits. Littman (1994) uses branch-and-bound heuristics to find suboptimal memoryless policies extremely quickly. To deal with mazes for which there is no safe, deterministic, memoryless policy, he replaces each conventional action by two actions, each having the additional effect of switching on or off a "memory bit". Good results are obtained with a toy problem. The method does not scale though, due to search space explosion caused by adding memory bits. By contrast, HQ-learning does not depend on finding optimal memory bit settings with branch-and-bound techniques but uses an incremental learning method instead.

Cliff and Ross (1994) describe a classifier system (ZCS) for POMDPs which is trained by bucket-brigade and genetic algorithms. They also use memory bits, to be set and reset by actions. The system is reported to work well for small problems but to become unstable in case of more than one memory bit. Also, it is usually not able to find optimal deterministic policies. Wilson (1996a) recently described a more sophisticated classifier system which uses prediction accuracy for calculating fitness, and a genetic algorithm working in environmental niches. His study shows that this makes the classifiers more general and more accurate. It would be interesting to test whether his system can use memory for solving POMDPs.

One problem with memory bits (we tried them, too) is that tasks such as those in section 3 require long traces of memory bit resets. Memory bits are critical and must be turned on/off at precisely the right moment. For instance, suppose that the probability of turning on a memory bit in response to a particular observation is indeed low, but that the agent makes this observation very often. Eventually the memory bit won't remain switched off. Q-learning, for example, tends to fail to reliably set memory bits because learning the Q-values for changing a bit depends on luck (L. Kaelbling, personal communication, 1995). HQ-learning, however, does not depend on long traces of memory bit resets. Its memory is embodied solely in the active agent number, which is rarely incremented during a trial. This makes it much more stable.

Multiple Q-learners. Like HQ-learning, Humphrys' W-learning (1996) uses multiple Q-learning agents. A major difference is that his agents' skills are prewired — different agents focus on different input-features and receive different rewards. "Good" reward functions are found by genetic algorithms. An important goal is to learn which agent to select for which part of the input space. Eight different learning methods implementing cooperative and competitive strategies are tested in a rather complex dynamic environment, and seem to lead to reasonable results. Possibly W-learning and HQ-learning can be combined in an advantageous way.

Digney (1996) describes a nested Q-learning technique based on multiple agents learning independent, reusable skills. To generate quite arbitrary control hierarchies, simple actions and skills can be composed to form more complex skills. Learning rules for selecting skills and for selecting actions are the same, however. This may make it hard to deal with long reinforcement delays. In experiments the system reliably learns to solve a small maze-task. It remains to be seen, however, whether the system can reliably learn to decompose solutions of complex problems into stable skills.

Learning control hierarchies. Ring's system (1994) constructs a bottom-up control hierarchy. The lowest level nodes are primitive perceptual and control actions. Nodes at higher levels represent sequences of lower level nodes. To disambiguate inconsistent states, new higher-level nodes are added to incorporate information hidden "deeper" in the past, if necessary. The system is able to quickly learn certain non-Markovian maze problems but often is not able to generalize from previous experience without additional learning, even if the optimal policies for old and new task are identical. HQ-learning, however, can reuse the same policy and generalize well from previous to "similar" problems.

McCallum's U-tree (1996) is quite similar to Ring's system. It uses prediction suffix trees in which the branches reflect decisions based on current or previous inputs/actions. Q-values are stored in the leaves, which correspond to clusters of instances collected and stored during the entire learning phase. Statistical tests are used to decide whether instances in a cluster correspond to significantly different utility estimates. If so, the cluster is split. The method may be viewed a decision tree with reinforcement learning additions. McCallum's recent experiments demonstrate the algorithm's ability to improve in comparatively large state spaces. Its problem is that it depends on the creation of an n-th order Markov model, where n is the size of the "time window" used for sampling observations. Hence for large n the approach will suffer from the curse of dimensionality.

Consistent Representations. Whitehead (1992) uses the "Consistent Representation (CR) Method" to deal with inconsistent internal states which result from "perceptual aliasing" due to ambiguous input information. CR uses an "identification stage" to execute perceptual actions which collect the information needed to define a consistent internal state. Once a consistent internal state has been identified, a single action is generated to maximize future discounted reward. Both identifier and controller are adaptive. One limitation of the method is that the agent must have access to the external Markov model; this is not necessary for HQ-learning.

Levin Search. Wiering and Schmidhuber (1996) use Levin search (LS) through program space (Levin 1973) to discover programs computing solutions for large POMDPs. LS is of interest because of its amazing theoretical properties: for a broad class of search problems, it has the optimal order of computational complexity. For instance, suppose there is an algorithm that solves a certain type of maze task in $O(n^3)$ steps, where n is a positive integer representing the problem size. Then LS will solve the same task in at most $O(n^3)$ steps. Wiering and Schmidhuber show that LS may have substantial advantages over other reinforcement learning techniques, provided the algorithmic complexity of the solutions is low.

Meta-Reinforcement Learning. Wiering and Schmidhuber (1996) also extend LS to obtain an incremental method for generalizing from previous experience. To guarantee that the lifelong history of policy changes corresponds to a lifelong history of reinforcement accelerations, a novel reinforcement learning paradigm called "Meta-reinforcement learning" (MRL, Schmidhuber et al. 1996) is combined with LS. It is shown that this can lead to further significant learning speed-ups. MRL is actually not LS-specific, but a general approach that allows for plugging in a great variety of learning algorithms. For instance, in additional experiments with a "self-referential" system that embeds its policy-modifying method within the policy itself, MRL is able to solve huge POMDPs with more than 10^{13} states (Schmidhuber et al. 1996). We believe that we will be able to combine MRL with HQ-learning in an advantageous way.

5 Conclusion

Summary. We introduced HQ-learning, a novel method for reinforcement learning in partially observable environments. "Non-Markovian" tasks are automatically decomposed into Markovian subtasks without intermediate external reinforcement for "good" subgoals. This is done by an ordered sequence of agents, each discovering both a local control policy *and* an appropriate "Markovian" subgoal. Our experiments involve POMDPs with many more states than most POMDPs found in the literature. The results demonstrate HQ-learning's ability to quickly learn optimal or near-optimal policies. We believe that currently there is no other reinforcement learning method for solving similar POMDPs in comparable time.

Limitations and future work. The current version is restricted to linearly ordered subgoal sequences. For very complex POMDPs, generalized HQ-architectures based on directed acyclic (or even recurrent) graphs may turn out to be useful. This is left for future research.

6 Acknowledgments

Thanks for valuable comments and discussions to Marco Dorigo, Nic Schraudolph, Luca Gambardella, Rafał Sałustowicz, Jieyu Zhao, Cristina Versino.

References

Caironi, P. V. C. and Dorigo, M. (1994). Training Q-agents. Technical Report IRIDIA-94-14, Université Libre de Bruxelles.

Cliff, D. and Ross, S. (1994). Adding temporary memory to ZCS. *Adaptive Behavior*, 3:101–150.

Cohn, D. A. (1994). Neural network exploration using optimal experiment design. In Cowan, J., Tesauro, G., and Alspector, J., editors, *Advances in Neural Information Processing Systems 6*, pages 679–686. San Mateo, CA: Morgan Kaufmann.

Dayan, P. and Hinton, G. (1993). Feudal reinforcement learning. In Lippman, D. S., Moody, J. E., and Touretzky, D. S., editors, *Advances in Neural Information Processing Systems 5*, pages 271–278. San Mateo, CA: Morgan Kaufmann.

Digney, B. (1996). Emergent hierarchical control structures: Learning reactive/hierarchical relationships in reinforcement environments. In Maes, P., Mataric, M., Meyer, J.-A., Pollack, J., and Wilson, S. W., editors, *¿From Animals to Animats 4: Proceedings of the Fourth International Conference on Simulation of Adaptive Behavior, Cambridge, MA*, pages 363–372. MIT Press, Bradford Books.

Fedorov, V. V. (1972). *Theory of optimal experiments.* Academic Press.

Hochreiter, S. and Schmidhuber, J. (1996). Long short term memory. *Submitted to Neural Computation.*

Humphrys, M. (1996). Action selection methods using reinforcement learning. In Maes, P., Mataric, M., Meyer, J.-A., Pollack, J., and Wilson, S. W., editors, *¿From Animals to Animats 4: Proceedings of the Fourth International Conference on Simulation of Adaptive Behavior, Cambridge, MA*, pages 135–144. MIT Press, Bradford Books.

Jaakkola, T., Singh, S. P., and Jordan, M. I. (1995). Reinforcement learning algorithm for partially observable Markov decision problems. In Tesauro, G., Touretzky, D. S., and Leen, T. K., editors, *Advances in Neural Information Processing Systems 7*, pages 345–352. MIT Press, Cambridge MA.

Jordan, M. I. and Rumelhart, D. E. (1990). Supervised learning with a distal teacher. Technical Report Occasional Paper #40, Center for Cog. Sci., Massachusetts Institute of Technology.

Kaelbling, L., Littman, M., and Cassandra, A. (1995). Planning and acting in partially observable stochastic domains. Technical report, Brown University, Providence RI.

Levin, L. A. (1973). Universal sequential search problems. *Problems of Information Transmission*, 9(3):265–266.

Lin, L. (1993). *Reinforcement Learning for Robots Using Neural Networks.* PhD thesis, Carnegie Mellon University, Pittsburgh.

Littman, M. (1994). Memoryless policies: Theoretical limitations and practical results. In D. Cliff, P. Husbands, J. A. M. and Wilson, S. W., editors, *Proc. of the International Conference on Simulation of Adaptive Behavior: From Animals to Animats 3*, pages 297–305. MIT Press/Bradford Books.

Littman, M., Cassandra, A., and Kaelbling, L. (1995). Learning policies for partially observable environments: Scaling up. In Prieditis, A. and Russell, S., editors, *Machine Learning: Proceedings of the Twelfth International Conference*, pages 362–370. Morgan Kaufmann Publishers, San Francisco, CA.

McCallum, R. A. (1993). Overcoming incomplete perception with utile distinction memory. In *Machine Learning: Proceedings of the Tenth International Conference.* Morgan Kaufmann, Amherst, MA.

McCallum, R. A. (1996). Learning to use selective attention and short-term memory in sequential tasks. In Maes, P., Mataric, M., Meyer, J.-A., Pollack, J., and Wilson, S. W., editors, *¿From Animals to Animats 4: Proceedings of the Fourth International Conference on Simulation of Adaptive Behavior, Cambridge, MA*, pages 315–324. MIT Press, Bradford Books.

Moore, A. and Atkeson, C. G. (1993). Prioritized sweeping: Reinforcement learning with less data and less time. *Machine Learning*, 13:103–130.

Nguyen and Widrow, B. (1989). The truck backer-upper: An example of self learning in neural networks. In *IEEE/INNS International Joint Conference on Neural Networks, Washington, D.C.*, volume 1, pages 357–364.

Ring, M. B. (1994). *Continual Learning in Reinforcement Environments.* PhD thesis, University of Texas at Austin, Austin, Texas 78712.

Schmidhuber, J. (1991a). Curious model-building control systems. In *Proc. International Joint Conference on Neural Networks, Singapore*, volume 2, pages 1458–1463. IEEE.

Schmidhuber, J. (1991b). Reinforcement learning in Markovian and non-Markovian environments. In Lippman, D. S., Moody, J. E., and Touretzky, D. S., editors, *Advances in Neural Information Processing Systems 3*, pages 500–506. San Mateo, CA: Morgan Kaufmann.

Schmidhuber, J., Zhao, J., and Wiering, M. (1996). Simple principles of metalearning. Technical Report IDSIA-69-96, IDSIA.

Schraudolph, N. N., Dayan, P., and Sejnowski, T. J. (1994). Temporal difference learning of position evaluation in the game of go. In Cowan, J. D., Tesauro, G., and Alspector, J., editors, *Advances in Neural Information Processing Systems*, volume 6, pages 817–824. Morgan Kaufmann, San Francisco.

Singh, S. (1992). The efficient learning of multiple task sequences. In Moody, J., Hanson, S., and Lippman, R., editors, *Advances in Neural Information Processing Systems 4*, pages 251–258, San Mateo, CA. Morgan Kaufmann.

Storck, J., Hochreiter, S., and Schmidhuber, J. (1995). Reinforcement driven information acquisition in non-deterministic environments. In *Proceedings of the International Conference on Artificial Neural Networks, Paris*, volume 2, pages 159–164. EC2 & Cie, Paris.

Sutton, R. S. (1988). Learning to predict by the methods of temporal differences. *Machine Learning*, 3:9–44.

Tham, C. (1995). Reinforcement learning of multiple tasks using a hierarchical CMAC architecture. *Robotics and Autonomous Systems*, 15(4):247–274.

Thrun, S. (1992). Efficient exploration in reinforcement learning. Technical Report CMU-CS-92-102, Carnegie-Mellon University.

Watkins, C. (1989). *Learning from Delayed Rewards*. PhD thesis, King's College London.

Watkins, C. J. C. H. and Dayan, P. (1992). Q-learning. *Machine Learning*, 8:279–292.

Whitehead, S. (1992). *Reinforcement Learning for the adaptive control of perception and action*. PhD thesis, University of Rochester.

Wiering, M. and Schmidhuber, J. (1996). Solving POMDPs with Levin search and EIRA. In Saitta, L., editor, *Machine Learning: Proceedings of the Thirteenth International Conference*, pages 534–542. Morgan Kaufmann Publishers, San Francisco, CA.

Wilson, S. (1996a). Explore/exploit strategies in autonomy. In Meyer, J. A. and Wilson, S. W., editors, *Proc. of the Fourth International Conference on Simulation of Adaptive Behavior: From Animals to Animats 4*, pages 325–332. MIT Press/Bradford Books.

Wilson, S. (1996b). Generalization in XCS. In *ICML'96 Workshop on Evolutionary Computing and Machine Learning*. Morgan Kaufmann Publishers, San Francisco, CA.

Zhao, J. and Schmidhuber, J. (1996). Incremental self-improvement for life-time multi-agent reinforcement learning. In Maes, P., Mataric, M., Meyer, J.-A., Pollack, J., and Wilson, S. W., editors, *From Animals to Animats 4: Proceedings of the Fourth International Conference on Simulation of Adaptive Behavior, Cambridge, MA*, pages 516–525. MIT Press, Bradford Books.