HQ-Learning

Adaptive Behavior 6:2, 1997)

Marco Wiering
marco@idsia.ch

Jürgen Schmidhuber
juergen@idsia.ch

IDSIA
Corso Elvezia 36
CH-6900 Lugano
Switzerland

http://www.idsia.ch

Abstract

HQ-learning is a hierarchical extension of Q(λ)-learning designed to solve certain types of partially observable Markov decision problems (POMDPs). HQ automatically decomposes POMDPs into sequences of simpler subtasks that can be solved by memoryless policies learnable by reactive subagents. HQ can solve partially observable mazes with more states than those used in most previous POMDP work.

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

1 Introduction

The problem. Sensory information is usually insufficient to infer the environment's state (*perceptual aliasing*, Whitehead 1992). This complicates goal-directed behavior. For instance, suppose your instructions for the way to the station are: "Follow this road to the traffic light, turn left, follow that road to the next traffic light, turn right, there you are.". Suppose you reach one of the traffic lights. To do the right thing you need to know whether it is the first or the second. This requires at least one bit of memory — your current environmental input by itself is not sufficient.

The most widely used reinforcement learning (RL) algorithms, such as Q-learning (Watkins 1989; Watkins and Dayan 1992) and TD(λ) (Sutton 1988), fail if the task requires creating short-term memories of relevant events to disambiguate identical sensory inputs observed in different states. They are limited to Markov decision problems (MDPs): at any given time the probabilities of the possible next states depend only on the current state and action.

Partially observable Markov decision problems (POMDPs, e.g., Littman 1996) — such as the traffic light problem — are more difficult than simple MDPs: the same observation may occur in more than one state of the environment, and different action responses may be required. POMDPs are generally considered difficult because of their particularly nasty temporal credit assignment problem: it is usually hard to figure out which prior observations are relevant, and how they should affect short-term memory contents. Even deterministic finite horizon POMDPs are NP-complete (Littman 1996) — general and exact algorithms are feasible only for small problems. This explains the recent interest in heuristic methods for finding good but not necessarily optimal solutions, e.g., Schmidhuber (1991c), McCallum (1993), Ring (1994), Cliff and Ross (1994), Jaakkola, Singh and Jordan (1995).

Unfortunately, however, previous methods do not scale up very well (Littman, Cassandra and Kaelbling 1995). This paper presents HQ-learning, a novel approach based on finite state memory implemented in a sequence of agents. HQ does not need a model of the POMDP and appears to scale up more reasonably than other approaches. For alternative approaches to larger scale POMDPs, see also Schmidhuber, Zhao and Wiering (1997b), Wiering and Schmidhuber (1996).

Inspiration. To select the optimal next action it is often not necessary to memorize the entire past (in general, this would be infeasible). A few memories corresponding to important previously achieved subgoals can be sufficient. To see this recall the traffic light scenario. While you are on your way, only a few memories are relevant, such as "I already passed the first traffic light". Between two such subgoals a memory-independent, *reactive policy* (RP) will carry you safely.

Overview. HQ-learning attempts to exploit such situations. Its divide-and-conquer strategy discovers a subgoal sequence decomposing a given POMDP into a sequence of *reactive policy problems* (RPPs). RPPs can be solved by RPs: all states causing identical inputs require the same optimal action. The only "critical" points are those corresponding to transitions from one RP to the next.

To deal with such transitions HQ uses multiple RPP-solving subagents. Each agent's RP is an adaptive mapping from observations to actions. At a given time only one agent can be active, and the system's only type of short-term memory is embodied by a pointer indicating which one. How many bits of information are conveyed by such a limited kind of short-term memory? The answer is: not more than the logarithm of the number of agents (the additional information conveyed by the system's RPs and subgoal generators tends to require many more bits, of course).

RPs 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 (subgoals are represented by desired inputs). Then it follows the policy embodied by its Q-function 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 adjusts both its RP and its subgoal. This is done by two learning rules that interact without explicit communication: (1) Q-table adaptation is based on slight modifications of Q(λ)-learning. (2) HQ-table adaptation is based on tracing successful subgoal sequences by Q(λ)-learning on the higher (subgoal) level. Effectively, subgoal/RP combinations leading to higher rewards become more likely to be chosen.

Although each agent's RP is represented by a memoryless lookup table, the whole system can solve "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, memoryless, stochastic policies for POMDPs with optimal, memory-based, deterministic solutions.

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 (up to 960 world states). They demonstrate HQ's ability to decompose POMDPs into several appropriate RPPs. Section 4 reviews related work. Section 5 summarizes HQ's advantages and limitations. Section 6 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 $t = 1, 2, 3, \ldots, T$, where $T < T_{max}$ if the agent solves the problem in fewer than T_{max} time steps. A 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 $B : S \rightarrow O$ is a many-to-one mapping of states to (ambiguous) observations, A is a finite set of actions, $R : S \times A \rightarrow \mathbb{R}$ 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 $D : S \times A \times S \rightarrow \mathbb{R}$ is a state transition function, where $p(S_{t+1}|S_t, A_t) := D(S_t, A_t \rightarrow S_{t+1}) = D(S_t, A_t, S_{t+1})$ denotes

the probability of transition to state S_{t+1} given S_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.

POMDPs as RPP sequences. The optimal policy of any deterministic POMDP with final goal state is decomposable into a finite sequence of optimal policies for appropriate RPPs, along with subgoals determining transitions from one RPP to the next. The trivial decomposition consists of single-state RPPs and the corresponding subgoals. In general, POMDPs whose only decomposition is trivial are hard — there is no efficient algorithm for solving them. HQ, however, is aimed at situations that require few transitions between RPPs. HQ's architecture implements such transitions by passing control from one RPP-solving subagent to the next.

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 control transfer 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. For each possible observation there is an HQ-table entry representing its estimated value as a subgoal. $HQ_i(O_j)$ denotes $\mathcal{C}_i$'s HQ-value (utility) of selecting O_j as its 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 variable $Active(t)$ represents the only kind of short-term memory in the system.

Architecture limitations. The sequential architecture restricts the POMDP types HQ can solve. To see this consider the difference between RL goals of (1) achievement and (2) maintenance. The former refer to single state goals (e.g., find the exit of a maze), the latter to maintaining a desirable state over time (such as keeping a pole balanced). Our current HQ-variant handles achievement goals only. In case of maintenance goals it will eventually run out of agents — there must be an explicit final desired state (this restriction may be overcome with different agent topologies beyond the scope of this paper).

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 $\mathcal{C}_i$. To explore different subgoal sequences we use the Max-Random rule: the subgoal with maximal HQ_i value is selected with probability p_{max}, a random subgoal is selected with probability $1 - p_{max}$. Conflicts between multiple subgoals with maximal HQ_i-values are solved by randomly selecting one. $\hat{O}_i$ denotes the subgoal selected by agent $\mathcal{C}_i$. This subgoal is only used in transfer of 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 actions according to the Max-Boltzmann distribution: with probability p_{max} take the action A_j with maximal $Q_i(O_t, A_j)$ value; with probability $1 - p_{max}$ select an action according to the traditional Boltzmann or Gibbs distribution, where the probability of selecting action A_j given observation O_t is:

$$p(A_j|O_t) = \frac{e^{Q_i(O_t,A_j)/T_i}}{\sum_{A_k \in A} e^{Q_i(O_t,A_k)/T_i}}.$$

The "temperature" T_i adjusts the degree of randomness involved in agent $\mathcal{C}_i$'s action selection in case the Boltzmann rule is used. Conflicts between multiple actions with maximal Q-values are solved by randomly selecting one.

Transfer of control. Control is transferred from one active agent to the next as follows. Each time $\mathcal{C}_i$ has executed an action, its control transfer 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 $t_1 \leftarrow 1$).

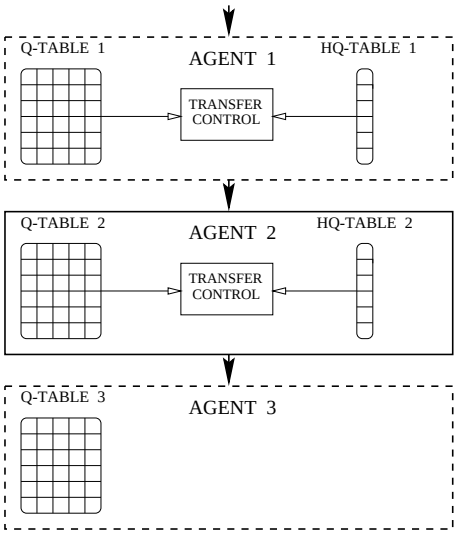

Figure 1: *Basic architecture. Three agents are connected in a sequential way. Each agent has a Q-table, an HQ-table, and a control transfer unit, except for the last agent which only has a Q-table. The Q-table stores estimates of actual observation/action values and is used to select the next action. 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 control transfer unit passes control to its successor.*

IF no goal 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 will use off-line learning for updating the tables — this means storing experiences and postponing learning until after trial end (no intra-trial parameter adaptation). In principle, however, online learning is applicable as well (see below). We will describe two HQ variants, one based on Q-learning, the other on Q(λ)-learning — Q(λ) overcomes Q's inability to solve certain RPPs. The learning rules appear very similar to those of conventional Q and Q(λ). 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 $Q_i(O_t, A_t)$ to approximate the system's expected discounted future reward for executing action A_t, given O_t. In the one-step lookahead case we have

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

where $P(S_j|O_t, \Theta, i)$ 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 belief vectors (Littman, 1996) or a world model (e.g., Moore 1993) might help to speed up learning. $V_i(O_t)$ 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 situations ($T \leq T_{max}$ denotes the total number of executed actions during the current trial, and α_Q is the learning rate):

Q.1 Let $\mathcal{C}_i$ and $\mathcal{C}_j$ denote the agents active at times t and $t+1$ — possibly $i = j$. If $t < T$ 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 $Q_i(O_T, A_T) \leftarrow (1 - \alpha_Q)Q_i(O_T, A_T) + \alpha_Q R(S_T, A_T)$.

Note that $R(S_T, A_T)$ is the final reward for reaching a goal state if $T < T_{max}$. A main difference with standard one-step Q-learning is that agents can be trained on Q-values which are not their own (see [Q.1]).

Learning the HQ-values: intuition. Recall the introduction's traffic light task. The first traffic light is a good subgoal. We want our system to discover this by exploring (initially random) subgoals and learning their HQ-values. The traffic light's HQ-value, for instance, should converge to the expected (discounted) future cumulative reinforcement to be obtained after it has been chosen as a subgoal. How? Once the traffic light has been reached and the first agent passes control to the next, the latter's own expectation of future reward is used to update the first's HQ-values. Where do the latter's expectations originate? They reflect its own experience with final reward (to be obtained at the station).

More formally. In the optimal case we have

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

where $\mathcal{E}$ denotes the average over all possible trajectories. $R_i = \sum_{t=t_i}^{t^{i+1}-1} \gamma^{t-t_i} R(S_t, A_t)$, $\mathcal{C}_i$'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 $HV_i := \max_{O_l \in O}\{HQ_i(O_l)\})$ is the estimated discounted cumulative reinforcement to be received by $\mathcal{C}_i$.

We adjust only HQ-values of agents active before trial end (N denotes the number of agents active during the last trial, 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 $\mathcal{C}_N$ has learned a (possibly high) value for a subgoal that is unachievable due to subgoals selected by previous agents.

HQ(λ)-learning: motivation. Q-learning's lookahead capability is restricted to one step. It cannot solve all RPPs because it cannot properly assign credit to different actions leading to identical next states (Whitehead 1992). For instance, suppose you walk along a wall that looks the same everywhere except in the middle where there is a picture. The goal is to reach the left corner where there is reward. This RPP is solvable by an RP. Given the "picture" input, however, Q-learning with one step look-ahead would assign equal values to actions "go left" and "go right" because they both yield identical "wall" observations.

Consequently HQ-learning may suffer from Q-learning's inability to solve certain RPPs. To overcome this problem, we augment HQ by TD(λ)-methods for evaluating and improving policies in a manner analogous to Lin's offline Q(λ)-method (1993). TD(λ)-methods integrate experiences from several successive steps to disambiguate identical short-term effects of different actions. Our experiments indicate that RPPs are solvable by Q(λ)-learning with sufficiently high λ.

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

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 $i = N, \ldots, 1$:
$HQ'_N(\hat{O}_N) \leftarrow R_N$
$HQ'_{N-1}(\hat{O}_{N-1}) \leftarrow R_{N-1} + \gamma^{t_N - t_i} R_N$
$HQ'_i(\hat{O}_i) \leftarrow R_i + \gamma^{t_{i+1} - t_i}((1 - \lambda)HV_{i+1} + \lambda HQ'_{i+1}(\hat{O}_{i+1}))$

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

In principle, online Q(λ) may be used as well. See, e.g., Peng and Williams (1996), or Wiering and Schmidhuber's fast Q(λ) implementation (1997). Online Q(λ) should not use "action-penalty" (Koenig and Simmons 1996), however, because punishing varying actions in response to ambiguous inputs will trap the agent in cyclic behavior.

Combined dynamics. Q-table policies are *reactive* and learn to solve RPPs. HQ-table policies are *metastrategies* for composing RPP sequences. Although Q-tables and HQ-tables do not explicitly communicate they influence each other through simultaneous learning. Their cooperation results in complex dynamics quite different from those of conventional Q-learning.

Utilities of subgoals and RPs are estimated by tracking how often they are part of successful subgoal/RP combinations. Subgoals that never or rarely occur in solutions become less likely to be chosen, others become more likely. In a certain sense subtasks compete for being assigned to subagents, and the subgoal choices "co-evolve" with the RPs. Maximizing its own expected utility, each agent implicitly takes into account frequent decisions made by other agents. Each agent eventually settles down on a particular RPP solvable by its RP and ceases to adapt. This will be illustrated by Experiment 1 in Section 3.

Estimation of average reward for choosing a particular subgoal ignores dependencies on previous subgoals. This makes local minima possible. If several rewarding suboptimal subgoal sequences are "close" in subgoal space, then the optimal one may be less probable than suboptimal ones. We will show in the experiments that this actually can happen.

Exploration issues. Initial choices of subgoals and RPs may influence the final result - there may be local minimum traps. Exploration is a partial remedy: it encourages alternative competitive strategies similar to the current one. Too little exploration may prevent the system from discovering the goal at all. Too much exploration, however, prevents reliable estimates of the current policy's quality and reuse of previous successful RPs. To avoid over-exploration we use the Max-Boltzmann (Max-Uniform) distribution for Q-values (HQ-values) (for discussions of exploration issues see, e.g., Fedorov 1972; Schmidhuber 1991a; Thrun 1992; Cohn 1994; Caironi and Dorigo 1994; Storck, Hochreiter and Schmidhuber 1995; Wilson 1996, Schmidhuber 1997). These distributions also make it easy to reduce the relative weight of exploration (as opposed to exploitation): to obtain a deterministic policy at the end of the learning process, we increase p_{max} during learning until it finally achieves a maximum value.

Selecting actions according to the traditional Boltzmann distribution causes the following problems: (1) It is hard to find good values for the temperature parameter. (2) The degree of exploration depends on the Q-values: actions with almost identical Q-values (given a certain input) will be executed equally often. For instance, suppose a sequence of 5 different states in a maze leads to observation sequence $O_1 - O_1 - O_1 - O_1 - O_1$, where O_1 represents a single observation. Now suppose there are almost equal Q-values for going west or east in response to O_1. Then the Q-updates will hardly change the differences between these Q-values. The resulting random walk behavior will cost a lot of simulation time.

For RP training we prefer the Max-Boltzmann rule instead. It focuses on the greedy policy and only explores actions competitive with the optimal actions. Subgoal exploration is less critical

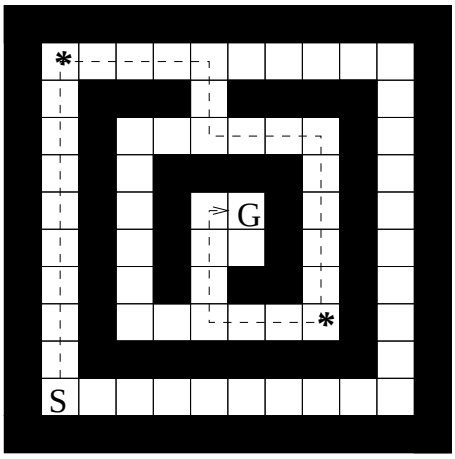

Figure 2: *A partially observable maze (POM). The task is to find a path leading from start S to goal G. The optimal solution requires 28 steps and at least three reactive agents. The figure shows a possible sub-optimal solution that costs 30 steps. Asterisks mark appropriate subgoals.*

though. The Max-Random subgoal exploration rule may be replaced by the Max-Boltzmann rule or others.

3 Experiments

We tested our system on two tasks in partially observable environments. The first task is comparatively simple — it will serve to exemplify how HQ discovers and stabilizes appropriate subgoal combinations. It requires finding a path from start to goal in a partially observable 10×10-maze, and can be collectively solved by three or more 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 agents) costs 83 steps.

3.1 Learning to Solve a Partially Observable Maze

Task. The first experiment involves the partially observable maze shown in figure 2. 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.* 16 possible observations are computed by adding the "field values" of blocked fields next to the agent's position, where the field value of the west, north, east, and south field is 1, 2, 4, and 8, respectively — 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. Stochastic memory-free policies will also perform poorly. For instance, input 5 stands for "fields to the left and to the right 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 reactive agents are necessary to solve this

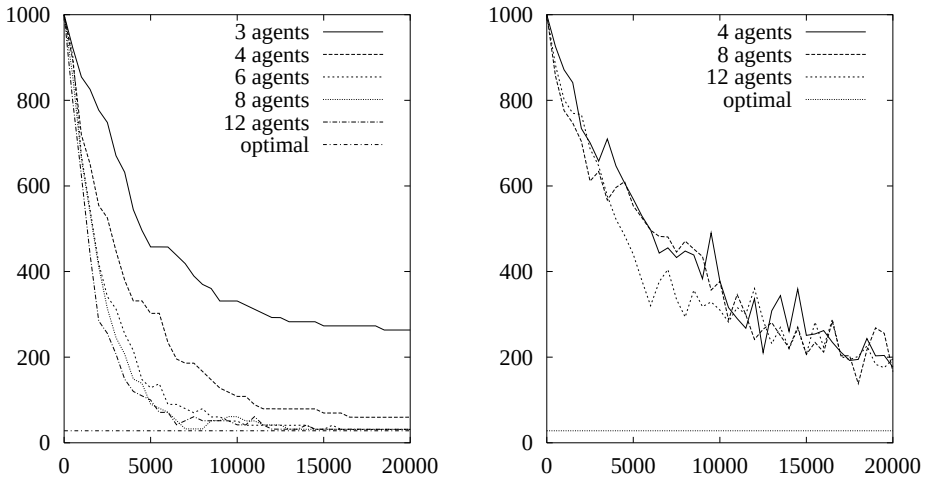

Figure 3: *left: HQ-learning results for the partially observable maze, for 3, 4, 6, 8, and 12 agents. We plot average test run length against trial numbers (means of 100 simulations). The system almost always converges to near-optimal solutions. Using more than the required 3 agents tends to improve performance. Right: 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.*

POMDP.

Reward function. Once the system hits the goal it receives a reward of 100. Otherwise the reward is zero. The discount factor $\gamma = 0.9$.

Parameters and experimental set-up. We compare systems with 3, 4, 6, 8, and 12 agents and 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 (p_{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. p_{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. We also tried Q(λ)-learning augmented as follows: the current input is the Cartesian product of the current observation and the last observation that is different from the current observation. At least in theory this Q(λ) variant might be able to solve the problem.

Results. Augmented Q(λ) failed to find stable solutions. HQ worked well, though. Figure 3A plots average test run length against trial numbers. Within 20,000 trials all systems almost always find near-optimal deterministic policies.

Consider Table 1. The largest systems are always able to decompose the POMDP into a sequence of RPPs. The average number of steps is close to optimal. In approximately 1 out of 8 cases, the optimal 28-step 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 appropriate subgoal combinations), 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 do not help though. All systems perform significantly

Table 1: *HQ-learning results for random actions replacing the selected actions with probability 0% and 10%. All table entries refer to the final test trial. The 2nd column lists average trial lengths. The 3rd column lists goal hit percentages. The 4th column lists average path lengths of solutions. The 5th column lists percentages of simulations during which the optimal path is found.*

System	Av. steps	(%) Goal	Av. sol.	(%) Optimal
3 agents	263	76	30	3
4 agents	60	97	31	6
6 agents	31	100	31	14
8 agents	31	100	31	12
12 agents	32	100	32	6
4 agents 10% noise	177	86	43	2
8 agents 10% noise	166	87	41	2
12 agents 10% noise	196	84	43	0
Random	912	19	537	0

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 3B). In the final trial of each simulation, systems with 4, 8, and 12 agents find the goal with probabilities of 86, 87, and 84 percent, respectively. 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. Less agents tend to give 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.

Experimental analysis. How does the system discover and stabilize subgoal combinations (SCs)? The only optimal 28-step solution uses observation 2 as the first subgoal (5th top field) and observation 9 as the second (southwest inner corner). There are several 30-step solutions, however — e.g., SCs (3, 12), (2, 12), (10, 12).

Figure 4 shows how SCs evolve by plotting them every 10 trials (observation 16 stands for an unsolved subgoal). The first 10,000 SCs are quite random, and the second agent often is not able to achieve its subgoal at all. Later, however, the system gradually focuses on successful SCs. Although useful SCs are occasionally lost due to exploration of alternative SCs, near simulation end the system converges to SC (3, 12).

The goal is hardly ever found prior to trial 5200 (the figure does not show this). Then there is a sudden jump in performance — most later trials cost just 30 steps. From this moment on observation 12 is used as second subgoal in more than 95% of all cases, and the goal is found in about 85%. The first subgoal tends to vary among observations 2, 3 and 10. Finally, around 16,000 trials, the first subgoal settles down on observation 3, although observation 2 would work as well.

The reasons for faster stabilization of the second subgoal may be its proximity to the final goal and the larger number of successful subgoal combinations in which it participates. Once the second and third agents have learned RPs leading from the second subgoal to the goal, the second subgoal's HQ-value will have increased dramatically and dominate the alternatives. Only once the second subgoal is firmly established can a similar effect help to stabilize the first. Subgoals tend to get fixed in reverse order of their online generation.

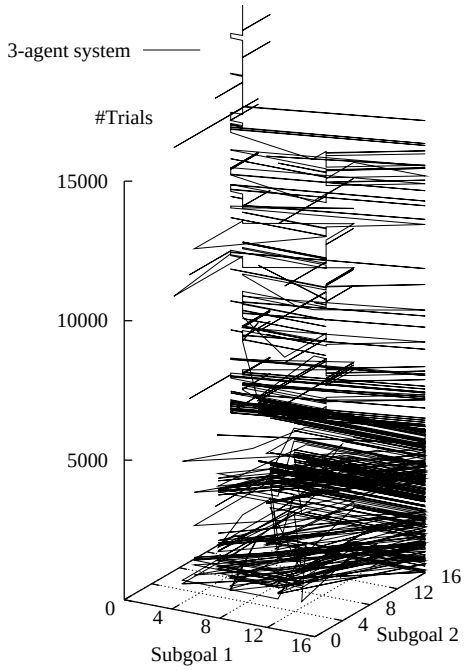

Figure 4: *Subgoal combinations (SCs) generated by a 3-agent system, sampled at intervals of 10 trials. Initially many different SCs are tried out. After 10,000 trials HQ explores less and less SCs until it finally converges to SC (3, 12).*

3.2 The Key and the Door

Task. The second experiment involves the 26×23 maze shown in figure 5. 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 key (door) is observed as a free field (wall). 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%). $\alpha_Q = .05$, $\alpha_{HQ} = .01$ $\forall i : \; T_i = .2$. p_{max} is linearly increased from .4 to .8. Again, $\lambda = .9$ for both HQ-tables and Q-tables, and all table entries are zero-initialized. One simulation consists of 20,000 trials.

Results. We first ran 20,000 thousand-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: goal discovery within 1000 steps (and without "action penalty" through negative reinforcement signals for each executed action) is very unlikely

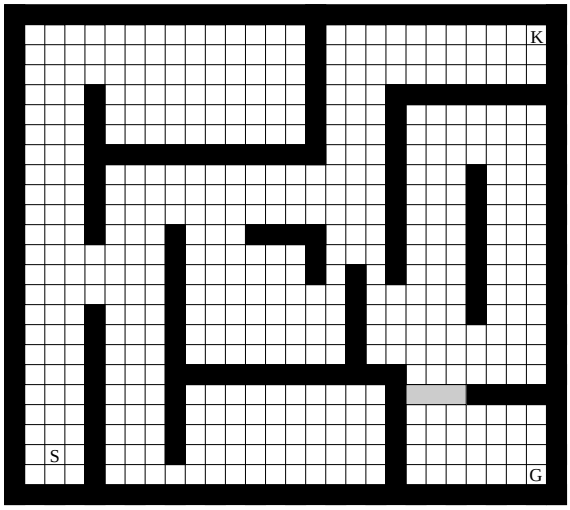

Figure 5: *A partially observable maze containing a key K and a door (grey area). Starting at S, the system first has to find the key to open the door, then proceed to the goal G. The shortest path costs 83 steps. This optimal solution requires at least three reactive agents. The number of possible world states is* 960.

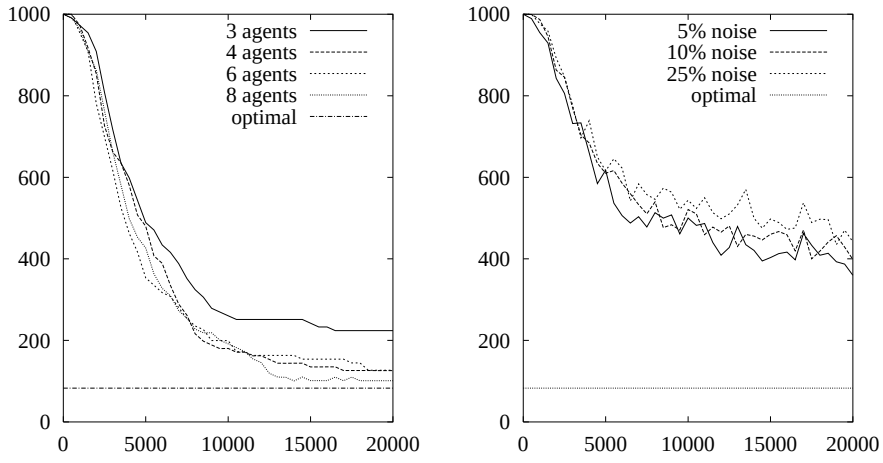

Figure 6: *left: HQ-learning results for the "key and door" problem. We plot average test run length against trial number (means of 100 simulations). Within 20,000 trials systems with 3 (4, 6 and 8) agents find good deterministic policies in 85% (96%, 96% and 99%) of the simulations. Right: HQ-learning results with an 8 agent system whose actions are replaced by random actions with probability 5%, 10%, and 25%.*

to happen.

Figure 6A 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. During the few runs that did not lead to good solutions the goal was rarely found at all. This reflects a general problem: in the POMDP case exploration issues are trickier than in the MDP case — much remains to be done to better understand them.

Table 2: *Results of 100 HQ-learning simulations for the "key and door" task. All table entries refer to the final test trial. The second column lists average trial lengths. The third lists goal hit percentages. The fourth lists average path lengths of solutions. The fifth lists percentages of simulations during which the optimal 83 step path was found. HQ-learning could solve the task with a limit of 1000 steps per trial. Random search needed a 10,000 step limit.*

System	Av. steps	(%) Goal	Av. sol.	(%) Optimal
3 agents	224	85	87	8
4 agents	126	96	90	9
6 agents	127	96	91	8
8 agents	101	99	92	6
8 agents (5% noise)	360	92	304	0
8 agents (10% noise)	399	90	332	0
8 agents (25% noise)	442	84	336	0
Random	*9310	19	6370	0

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.

4 Previous Work

Other authors proposed hierarchical reinforcement learning techniques, e.g., Schmidhuber (1991b), Dayan and Hinton (1993), Moore and Atkeson (1993), Tham (1995), Sutton (1995). Their methods, however, have been designed for MDPs. 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. Schmidhuber (1991c) uses 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 (this extends 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 attempts at overcoming this problem (e.g., Schmidhuber 1992; Hihi and Bengio 1996; Hochreiter and Schmidhuber 1997, and references therein), there are no reinforcement learning applications yet.

Belief vectors. Kaelbling, Littman and Cassandra (1995) hierarchically build policy trees to calculate optimal policies in stochastic, partially observable environments. For each possible environmental state the "belief vector" provides the agent's estimate of the probability of currently being in this state. After each observation the belief vector is updated using action/observation models and Bayes' formula. This compresses the history of previous events into a probability distribution. Based on this "belief state" an optimal action can be chosen (Sondik 1971). Dynamic programming algorithms are used to compute optimal policies based on the belief states. Problems with this approach are that the nature of the underlying MDP needs to be known, and that it is computationally very expensive. Methods for speeding it up focus on constructing more

compact policy trees. For instance, Littman (1996) uses the "witness algorithm" to accelerate policy tree construction. Zhang and Liu (1996) propose a scheme which (a) speeds up the dynamic programming updates and (b) uses an oracle providing additional state information to decrease uncertainty.

Boutilier and Poole (1996) use Bayesian networks to represent POMDPs, and use these more compact models to accelerate policy computation. Parr and Russell (1995) use gradient descent methods on a continuous representation of the value function. Their experiments show significant speed-ups on certain small problems.

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. An advantage of the other methods, however, is that they can deal with very noisy perceptions and actions.

A possible HQ extension could use belief vectors to assign selection probabilities to each agent and to weigh their Q-values. In very noise environments this may work better than simple HQ.

Hidden Markov Models. McCallum's utile distinction memory (1993) is an extension of Chrisman's perceptual distinctions approach (1992), which combines Hidden Markov Models (HMMs) and Q-learning. The system 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.

Memory bits. Littman (1994) uses branch-and-bound heuristics to find suboptimal memoryless policies extremely quickly. To handle 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.

Cliff and Ross (1994) use Wilson's (1994) classifier system (ZCS) for POMDPs. There are memory bits which can be set and reset by actions. ZCS is trained by bucket-brigade and genetic algorithms. The system is reported to work well on 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 (1995) 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 see how well this system can use memory for solving POMDPs.

One problem with memory bits is that tasks such as those in section 3 require (1) switching on/off memory bits at precisely the right moment, and (2) keeping them switched on/off for long times. During learning and exploration, however, each memory bit will be very unstable and change all the time — algorithms based on incremental solution refinement will usually have great difficulties in finding out when to set or reset it. Even if the probability of changing a memory bit in response to a particular observation is low it will eventually change if the observation is made frequently. HQ-learning does not have such problems. Its memory is embodied solely by the active agent's number, which is rarely incremented during a trial. This makes it much more stable.

Program evolution with memory cells (MCs). Certain techniques for automatic program synthesis based on evolutionary principles can be used to evolve short-term memorizing programs that read and write MCs during runtime (e.g., Teller, 1994). A recent such method is Probabilistic Incremental Program Evolution (PIPE — Salustowicz and Schmidhuber, 1997). PIPE iteratively

generates successive populations of functional programs according to an adaptive probability distribution over all possible programs. On each iteration it uses the best program to refine the distribution. Thus it stochastically generates better and better programs. An MC-based PIPE variant has been successfully used to solve tasks in partially observable mazes. Unlike the memory bit approach mentioned in the previous paragraph, population-based approaches will not easily unlearn programs that make good use of memory. On serial machines, however, their evaluation tends to be computationally much more expensive than HQ.

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 (see Ron, Singer and Tishby 1994) 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. McCallum's recent experiments demonstrate the algorithm's ability to learn reasonable policies in large state spaces.

One problem with Ring's and McCallum's approaches is that they depend 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 his method is that the system has no means of remembering and using any information other than that immediately perceivable. HQ-learning, however, can profit from remembering previous events for very long time periods.

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.

Success-Story Algorithm. Wiering and Schmidhuber (1996) also extend LS to obtain an incremental method for generalizing from previous experience ("adaptive LS"). To guarantee that the lifelong history of policy changes corresponds to a lifelong history of reinforcement accelerations, they use the success-story algorithm (SSA, e.g., Schmidhuber, Zhao and Schraudolph 1997a; Zhao and Schmidhuber 1996, Schmidhuber, Zhao and Wiering 1997b). This can lead to further significant speed-ups. SSA 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, SSA is able to solve huge POMDPs with more than 10^{13} states (Schmidhuber et al. 1997a). It may be

possible to combine SSA with HQ-learning in an advantageous way.

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.

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 simple maze task. It remains to be seen, however, whether the system can reliably learn to decompose solutions of complex problems into stable skills.

5 HQ's Advantages and Limitations

HQ's advantages.

1. Most POMDP algorithms need *a priori* information about the POMDP, such as the total number of environmental states, the observation function, or the action model. HQ does not.

2. Unlike "history windows", HQ-learning can in principle handle arbitrary time lags between events worth memorizing. To focus this power on where it is really needed, short history windows may be included in the agent inputs to take care of the shorter time lags. This, however, is orthogonal to HQ's basic ideas.

3. To reduce memory requirements, HQ does not explicitly store all experiences with different subgoal combinations. Instead it estimates the average reward for choosing particular subgoal/RP combinations, and stores its experiences in a single sequence of Q- and HQ-tables. These are used to make successful subgoal/RP combinations more likely. HQ's approach is advantageous in case the POMDP exhibits certain regular structure: if one and the same agent tends to receive RPPs achievable by similar RPs then it can "reuse" previous RPP solutions.

4. HQ-learning can immediately generalize from solved POMDPs to "similar" POMDPs containing more states but requiring identical actions in response to inputs observed during subtasks (the RPPs remain invariant).

5. Like Q-learning, HQ-learning allows for representing RPs and subgoal evaluations by function approximators other than look-up tables.

HQ's current limitations.

1. An agent's current subgoal does not uniquely represent previous subgoal histories. This means that HQ-learning does not really get rid of the "hidden state problem" (HSP). HQ's HSP is not as bad as Q's, though. Q's is that it is impossible to build a Q-policy that reacts differently to identical observations, which may occur frequently. Appropriate HQ-policies, however, do exist.

   Still, HQ's remaining HSP may prevent HQ from *learning* an optimal policy. To deal with this HSP one might think of using subgoal trees instead of sequences. All possible subgoal sequences are representable by a tree whose branches are labeled with subgoals and whose nodes contain RPs for solving RPPs. Each node stands for a particular history of subgoals

and previously solved subtasks — there is no HSP any more. Since the tree grows exponentially with the number of possible subgoals, however, it is practically infeasible in case of large scale POMDPs. Perhaps it will be possible to find a reasonable compromise between simple linear sequences and full-fledged trees.

2. In case of noisy observations transfer of control may happen at inappropriate times. A remedy may be to use more reliable inputs combining successive observations.

3. In case of noisy actions, an inappropriate action may be executed right before passing control. The resulting new subtask may not be solvable by the next agent's RP. A remedy similar to the one mentioned above may be to represent subgoals as pairs of successive observations.

4. If there are many possible observations then subgoals will be tested infrequently. This may delay convergence. To overcome this problem one might either try function approximators instead of look-up tables or let each agent generate a set of multiple, alternative subgoals. In the latter instance, once a subgoal in the set is reached, control is transferred to the next agent.

5. Some parameters, such as the maximal number of agents and the maximal runtime, need to be set in advance. The former is not critical — it may be large since storage requirements are low. The latter, however, should be as low as possible to avoid wasting time on "cycling" between states.

6 Conclusion

Summary. HQ-learning is a novel method for reinforcement learning in partially observable environments. "Non-Markovian" tasks are automatically decomposed into subtasks solvable by memoryless policies, 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 subgoal. At each time step, the only type of memory is carried by the "name" of the agent that is active. Our experiments involve (model-free, deterministic) 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.

Future work. The current HQ version is restricted to learning single 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. In our point of view, however, the most challenging problem is exploration: "destructive" exploration rules will unlearn good subgoal sequences. How to improve POMDP exploration is still an open question.

7 Acknowledgments

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

References

Boutilier, C. and Poole, D. (1996). Computing optimal policies for partially observable decision processes using compact representations. In *AAAI-1996: Proceedings of the Thirteenth National Conference on Artificial Intelligence*, pages 1168–1175, Portland, OR.

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

Chrisman, L. (1992). Reinforcement learning with perceptual aliasing: The perceptual distinctions approach. In *Proceedings of the Tenth International Conference on Artificial Intelligence*, pages 183–188. AAAI Press, San Jose, California.

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.

Hihi, S. E. and Bengio, Y. (1996). Hierarchical recurrent neural networks for long-term dependencies. In Touretzky, D. S., Mozer, M. C., and Hasselmo, M. E., editors, *Advances in Neural Information Processing Systems 8*, pages 493–499. MIT Press, Cambridge MA.

Hochreiter, S. and Schmidhuber, J. (1997). Long short-term memory. *Neural Computation*, 9:1681–1726.

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.

Koenig, S. and Simmons, R. G. (1996). The effect of representation and knowedge on goal-directed exploration with reinforcement learnign algorithm. *Machine Learning*, 22:228–250.

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. (1996). *Algorithms for Sequential Decision Making*. PhD thesis, Brown University, Providence, RI.

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, D. and Widrow, B. (1989). The truck backer-upper: An example of self learning in neural networks. In *Proceedings of the International Joint Conference on Neural Networks*, pages 357–363. IEEE Press.

Parr, R. and Russell, S. (1995). Approximating optimal policies for partially observable stochastic domains. In *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI-95)*, pages 1088–1094. Morgan Kaufmann.

Peng, J. and Williams, R. (1996). Incremental multi-step Q-learning. *Machine Learning*, 22:283–290.

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

Ron, D., Singer, Y., and Tishby, N. (1994). Learning probabilistic automata with variable memory length. In Aleksander, I. and Taylor, J., editors, *Proceedings Computational Learning Theory*. ACM Press.

Sałustowicz, R. P. and Schmidhuber, J. (1997). Probabilistic incremental program evolution. *Evolutionary Computation*, 5(2):123–141. See ftp://ftp.idsia.ch/pub/rafal/PIPE.ps.gz.

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). Learning to generate sub-goals for action sequences. In Kohonen, T., Mäkisara, K., Simula, O., and Kangas, J., editors, *Artificial Neural Networks*, pages 967–972. Elsevier Science Publishers B.V., North-Holland.

Schmidhuber, J. (1991c). 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. (1992). Learning complex, extended sequences using the principle of history compression. *Neural Computation*, 4(2):234–242.

Schmidhuber, J. (1997). What's interesting? Technical Report IDSIA-35-97, IDSIA.

Schmidhuber, J., Zhao, J., and Schraudolph, N. (1997a). Reinforcement learning with self-modifying policies. In Thrun, S. and Pratt, L., editors, *Learning to learn*. Kluwer. in press.

Schmidhuber, J., Zhao, J., and Wiering, M. (1997b). Shifting inductive bias with success-story algorithm, adaptive Levin search, and incremental self-improvement. *Machine Learning*, 26:105–130.

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.

Sondik, E. J. (1971). *The Optimal Control of Partially Observable Markov Decision Processes*. PhD thesis, Unpublished doctoral thesis, Stanford University, CA.

Storck, J., Hochreiter, S., and Schmidhuber, J. (1995). Reinforcement driven information acquisition in nondeterministic environments. In *Proceedings of the International Conference on Artificial Neural Networks*, 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.

Sutton, R. S. (1995). TD models: Modeling the world at a mixture of time scales. In Prieditis, A. and Russell, S., editors, *Machine Learning: Proceedings of the Twelfth International Conference*, pages 531–539. Morgan Kaufmann Publishers, San Francisco, CA.

Teller, A. (1994). The evolution of mental models. In Kenneth E. Kinnear, J., editor, *Advances in Genetic Programming*, pages 199–219. MIT Press.

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, Oxford.

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.

Wiering, M. A. and Schmidhuber, J. (1997). Fast online Q(λ). Technical Report IDSIA-21-97, IDSIA. In preparation.

Wilson, S. (1994). ZCS: A zeroth level classifier system. *Evolutionary Computation*, 2:1–18.

Wilson, S. (1995). Classifier fitness based on accuracy. *Evolutionary Computation*, 3(2):149–175.

Wilson, S. (1996). 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.

Zhang, N. L. and Liu, W. (1996). Planning in stochastic domains: Problem characteristics and approximation. Technical Report HKUST-CS96-31, Hong Kong University of Science and Technology.

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.