Hierarchical Memory-Based Reinforcement Learning

Natalia Hernandez-Gardiol
Department of Computer Science
Michigan State University
East Lansing, MI 48824
hernan49@cse.msu.edu

Sridhar Mahadevan
Department of Computer Science
Michigan State University
East Lansing, MI 48824
mahadeva@cse.msu.edu

Abstract

A key challenge for reinforcement learning is how to scale up to large partially observable domains. In this paper, we show how a hierarchy of behaviors can be used to create and select among variable length short-term memories appropriate for a task. At higher levels in the hierarchy, the agent abstracts over lower-level details and looks back over a variable number of high-level decisions in time. We formalize this idea in a framework for solving partially observable, sequential decision tasks called Hierarchical Short-Term Memory (HSM). HSM uses a memory-based SMDP Q-learning method to rapidly propagate delayed reward across long decision sequences. We show that the HSM framework outperforms several related reinforcement learning techniques on a realistic corridor navigation task.

1 Introduction

Reinforcement learning encompasses a class of machine learning problems in which an agent learns from experience as it interacts with its environment. One fundamental challenge faced by reinforcement learning agents in real-world problems is that the state space can be very large, and consequently there may be a long delay before reward is received. Previous work has addressed this problem by breaking down a large problem into a hierarchy of subtasks or abstract behaviors. These approaches include using a behavior-based decomposition to accelerate learning in real robots [2], learning a policy for a task over a set of macro-actions [5], and using a top-down decomposition of tasks into subtasks [1]. In these approaches, each level of the hierarchy focuses only on the subset of the state space relevant to its activity.

Another difficult issue that frequently arises in real-world environments is the problem of perceptual aliasing: different world states often generate the same observations. One strategy to deal with perceptual aliasing is to add memory about past percepts. Short-term memory consisting of a linear (or tree-based) sequence of primitive actions has been shown to be a useful strategy [3]. However, considering short-term memory at a flat, uniform resolution of primitive actions would likely scale poorly to tasks with long decision sequences. Thus, just as spatio-temporal

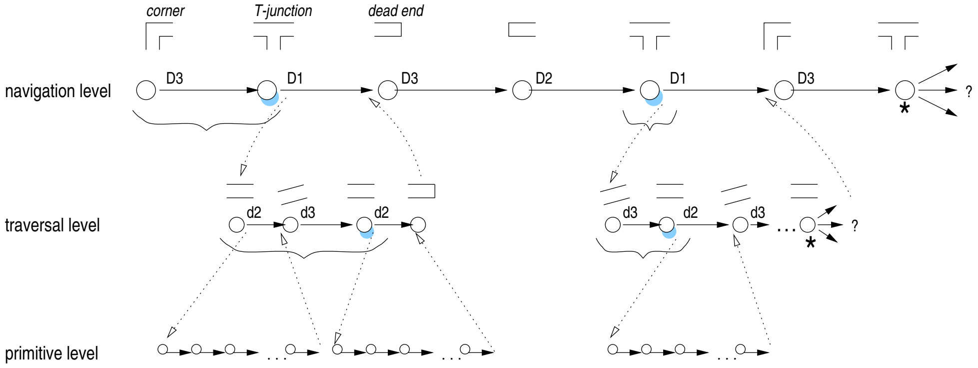

Figure 1: A hierarchical short-term memory structure for a navigation task. At the abstract (navigation) level, observations and decisions occur at intersections. At the lower (corridor-traversal) level, observations and decisions occur within the corridor. This figure illustrates memory-based decision making at two levels in the hierarchy. At each level, the current decision point is shown with a star. Each decision point examines its past experience to find states with similar history (shown with shadows).

abstraction of the state space improves scaling in completely observable environments, for large partially observable environments a similar benefit may result if we consider the space of *past experience* at variable resolution. Given a task, we want a hierarchical strategy for rapidly bringing to bear past experience that is appropriate to the grain-size of the decisions being considered.

Consider, for example, the task of navigating in a long corridor environment. A decision can be made at a given T-junction, say, to continue onward to the next intersection. However, the navigation decision at the next intersection might depend on the percepts at the last intersection and not the sequence of percepts seen while traversing the corridor. The issue then becomes how to choose memories with variable granularity. In this paper, we show that considering past experience at a variable, task-appropriate resolution can speed up learning and greatly improve performance under perceptual aliasing. The resulting approach, which we call Hierarchical Short-Term Memory (HSM), is a general technique for solving large, perceptually aliased tasks.

2 Hierarchical Short-Term Memory

By employing short-term memory over abstract decisions, each of which involves running a hierarchy of behaviors, we can apply memory at a *more informative* level of abstraction. An important side-effect is that the agent can look at a decision point many steps back in time while ignoring the exact sequence of low-level observations and actions that transpired. Figure 1 illustrates the HSM method and a more precise description is given below.

1. Given an abstraction level l and choice point s: for each potential future decision, d, examine the history at level l to find a set of past choice points that have executed d and whose incoming (suffix) history most closely matches that of the current point. Call this set of instances the "voting set" for decision d.

2. Choose d_t as the decision with the highest average discounted sum of reward over the voting set. Occasionally, choose d_t using an exploration strategy. Here, t is the event counter of the current choice point at level l.
3. Execute the decision d_t and record: o_t, the resulting observation; r_t, the reward received; and n_t, the duration of abstract action d_t (measured by the number of primitive environment transitions executed by the abstract action).

   Note that for every environment transition from state s_{i-1} to state s_i with reward r_i and discount γ, we accumulate any reward and update the discount factor: $r_t \leftarrow r_t + \gamma_t r_i \qquad \gamma_t \leftarrow \gamma \gamma_t$
4. Using the following SMDP Q-learning rule, update the Q-value of the current decision point. Also, update the Q-value analogously for each instance in the voting set, using the decision, reward, and duration values recorded along with the instance (β is the learning rate).

$$Q_l(s_t, d_t) \leftarrow (1 - \beta)Q_l(s_t, d_t) + \beta(r_t + \gamma_t \max_d Q_l(s_{t+n_t}, d))$$

To implement the hierarchy of behaviors, in principle any hierarchical reinforcement learning method may be used. For our implementation, we used the Hierarchy of Abstract Machines (HAM) framework proposed by Parr and Russell [4]. When executed, an abstract machine executes a partial policy and returns control to the caller upon termination. The HAM learns with a Q-learning rule modified for SMDPs. This Q-learning rule can be shown to converge to the optimal policy in the restricted space of policies consistent with the HAM decomposition.

HSM also requires a technique for short-term memory. A variety of alternatives are available. For simplicity, we implemented the Nearest Sequence Memory (NSM) algorithm proposed by McCallum [3]. NSM records each of its raw experiences as a linear chain. To choose the next action, the agent evaluates the outcomes of the k "nearest" neighbors in the experience chain. NSM evaluates the closeness between two states according to the match length of the suffix chain preceding the states. The chain can either be grown indefinitely, or old experiences can be replaced after the chain reaches a maximum length.

3 The Navigation Task

To test the HSM framework, we devised a navigation task in a simulated corridor environment (see Figure 2). The task is for the robot to find its way from the start, the center T-junction, to the goal, the four-way junction. The robot receives a reward at the goal intersection and and a small negative reward for each primitive step taken.

Since the agent is equipped with sonar and infrared sensors, the environment presents significant perceptual ambiguity. The robot's range-finding sensors may identify the goal, but they cannot unambiguously identify the location of the robot in the environment otherwise. Additionally, sensor readings can be noisy; even if the agent is at the goal or an intersection, it might not "see" it.

What makes the task difficult are the several activities that must be executed concurrently. Conceptually, there are two levels to our navigation problem. At the top, most abstract, level is the root task of navigating to the goal. At the lower level is the task of physically traversing the corridors, avoiding obstacles, maintaining alignment with the walls, etc.

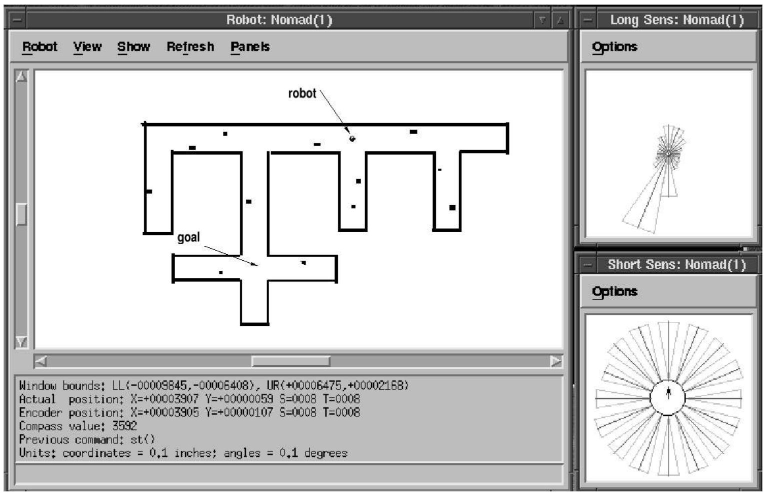

Figure 2: The corridor environment in the Nomad 200 robot simulator. The goal is the 4-way junction. The robot is shown at the middle T-junction. The robot is equipped with 16 short-range infrared and long-range sonar sensors. The other figures in the environment are obstacles the robot must maneuver around.

Our primary testbed was a simulated agent using a Nomad 200 robot simulator. This simulated robot is equipped with 20 bumper and 16 sonar and infrared sensors, arranged radially. The dynamics of the simulator are not "grid world" dynamics: the Nomad 200 simulator realistically represents continuous, noisy sensor input and the occasional unreliability of actuators. Note the size of the robot relative to the environment in Figure 2.

4 Implementation of the Learning Agents

In our experiments, we compared four learning agents: a basic HAM agent, two agents using HSM (one with short-term memory only at the most abstract level, and one with short-term memory at multiple levels in the hierarchy), and a "flat" NSM agent.

4.1 Design of the Behavioral Hierarchy

In order to build a set of behaviors for hallway navigation, we used a three-level hierarchy. The top abstract level is basically a choice state for choosing a hallway navigation direction (see Figure 3a). In each of the four nominal directions (front, back, left, right), the agent can make one of three observations: {wall, opening, unknown}. The agent must learn to choose among the four abstract machines to reach the next intersection. This top level machine has control initially, and it regains control at intersections.

The second level of the hierarchy contains the machines for traversing the hallway. The traversal behavior is shown in Figure 3b. Each of the four machines at this level executes a reactive strategy for traversing a corridor.

Finally, the third level of the hierarchy implements the follow-wall and avoid-obstacle strategies using primitive actions. Both the avoid-obstacle and the follow-

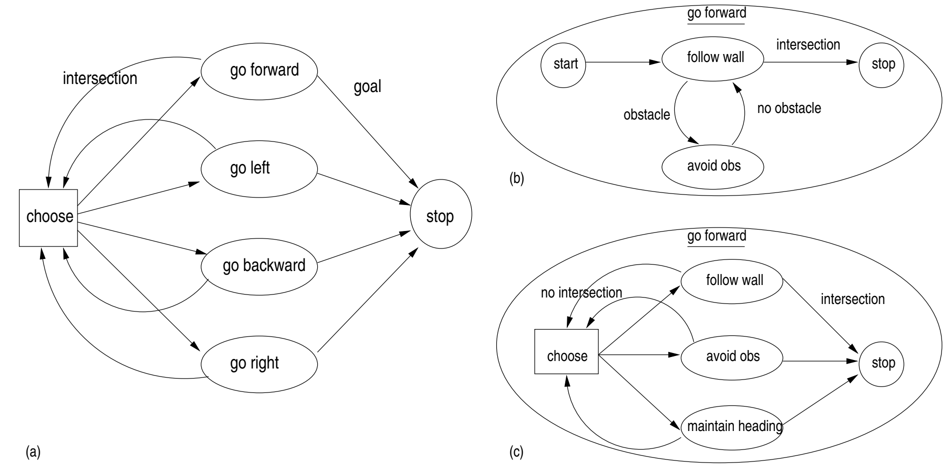

Figure 3: Hierarchical structure of behaviors for hallway navigation. Figure (a) shows the most abstract level – responsible for navigating in the environment. Figures (b) and (c) show two implementations of the hall-traversal machines. The machine in Figure (b) is reactive, and Figure (c) is a machine with a choice point.

wall strategies were themselves trained previously using Q-learning to exploit the power of reuse in the hierarchical framework.

4.2 Design of the HAM Agent

The HAM agent uses the three-level behavior hierarchy as described above. There is a single choice state, at the top level, and the agent learns to coordinate its choices by keeping a table of Q-values. The Q-value table is indexed by the current percepts and the chosen action (one of four abstract machines). The HAM agent uses a discount of 0.9, and a learning rate of 0.1. Exploration is done with a simple epsilon-greedy strategy.

4.3 Design of the HSM Agents

The first of two HSM agents uses short-term memory only at the most abstract level. It uses the same behavior hierarchy as the HAM. However, when making a decision at the abstract level, it extracts and updates the Q-values according to the method described in section 2. The HSM agent uses a history length of 1000, a k of 4, a discount of 0.9, and a learning rate of 0.1. Exploration was done with a simple epsilon-greedy strategy. The performance of this agent was studied as a control against the more complex multi-level memory agent described next.

The second of two HSM agents uses short-term memory both at the abstract navigation level and at the intermediate level. The behavior decomposition at the abstract navigation level is the same for the previous two agents. As in the above HSM agent, this agent must use short-term memory at the abstract level to learn a strategy for navigating the corridor. However, the traversal behavior is in turn composed of machines that must make a decision based on short-term memory. Each of the four machines at the traversal level uses short-term memory to learn to coordinate a strategy behaviors for traversing a corridor. The memory-based version of the traversal machine is shown in Figure 3c. The traversal machine uses

NSM as the short-term memory technique, maintaining a maximum chain length of 1000. Exploration is done with a simple epsilon-greedy strategy in all cases.

4.4 Design of the Flat Agent

Finally, we compare the HSM agents to a "flat" NSM agent designed to solve the same task. The flat agent must keep track of the following perceptual data: first, it needs the same perceptual information as the top-level HAM (so it can identify the goal); second, it needs the additional perceptual data for aligning to walls and for avoiding obstacles: {bumped, angle to the wall (binned into 4 groups of 45° each)}.

The flat agent chooses among four primitive actions: go-forward, veer-left, veer-right, and back-up. Not only must it learn to make it to the goal, it must simultaneously learn to align itself to walls and avoid obstacles. The NSM agent uses a history length of 1000 , a k of 4, a discount of 0.9, and a learning rate of 0.1. Exploration is done with a simple epsilon-greedy strategy.

5 Experimental Results

In Figure 4, we see the learning performance of each agent in the navigation task. The graphs show the performance advantage of both HSM agents over the non-HSM agents. In particular, we find that the flat memory-based agent does considerably worse than the other three, as expected. The flat agent must carry around the perceptual data to perform both high and low-level behaviors. From the point of view of navigation, this results in long strings of uninformative corridor states between the more informative intersection states. Since takes such an agent longer to discover patterns in its experience, it never quite learns to navigate successfully to the goal.

Next, both memory-based hierarchical agents outperform the HAM agent. The HAM agent does better at navigation than the flat agent since it abstracts away the perceptually aliased corridor states. However, it is unable to distinguish between all of the intersections. Without the ability to tell which T-junctions lead to the goal, and which to a dead end, the HAM agent does not perform as well.

Finally, the multi-level HSM agent outperforms the single-level HSM agent. The multi-level HSM agent can tune its traversing strategy to the characteristics of the cluttered hallway by using short-term memory at the intermediate level.

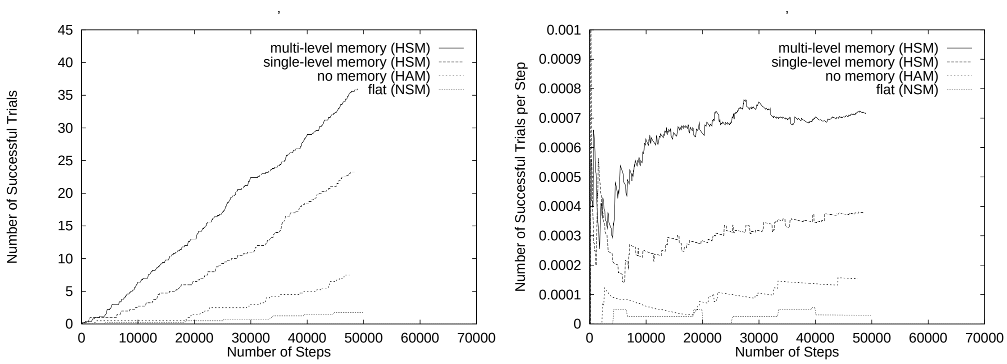

Figure 4: Learning performance in the navigation task. Each curve is averaged over five trials for each agent. A trial is "successful" when the agent reaches the goal.

6 Conclusions and Future Work

In this paper we proposed a framework for solving large perceptually aliased tasks through the Hierarchical Short-term Memory (HSM) method. This approach has several key advantages. The hierarchical behavioral structure gives the agent the ability to conduct a more efficient initial exploration of its environment. Rather than wasting time re-learning obstacle avoidance each time it is placed in a new environment, it re-uses previously learned hall-traversing strategies to direct its efforts to solving the navigation task. Second, organizing past experience hierarchically scales better to problems with long decision sequences than organizing past experience as a linear chain of primitive observations and actions. Without a hierarchical structure, the experiences an agent needs to solve the navigation task (namely, its observations and decisions at the intersections) are widely separated by long strings of experience during the traversal of a corridor. It is hard for the agent to effectively generalize across those experiences in order to learn to find the goal. We presented an experiment comparing four different learning methods, showing that hierarchical short-term memory produces overall the best performance in a perceptually aliased corridor navigation task.

The work in this paper can be extended in several directions. For the final version of this paper, we expect to include results on HSM with more sophisticated short-term memory techniques, such as multi-scale variants of Utile Suffix Memory [3].

One key limitation of the current HSM framework is that each abstraction level examines only the history at its own level. One extension is to permit interaction between the memory streams at each level of the hierarchy. Consider a navigation task in which the decision at a given intersection depends on an observation seen while traversing the corridor. In this case, the abstract level should have the ability to "zoom in" to inspect a particular low-level experience in greater detail. Conversely, a decision at the low-level may also depend on the context of the high-level behavior. We expect that pursuit of general techniques to manage past experience at variable granularity will lead to strategies for control that are able to gracefully scale to large, partially observable problems.

Acknowledgements

This research is supported in part by a Knowledge and Distributed Intelligence (KDI) grant from the National Science Foundation ECS-9873531.

References

[1] Thomas G. Dietterich. The MAXQ method for hierarchical reinforcement learning. In *Autonomous Robots Journal, Special Issue on Learning in Autonomous Robots*, 1998.

[2] S. Mahadevan and J. Connell. Automatic programming of behavior-based robots using reinforcement learning. *Artificial Intelligence*, 55(1):311–365, 1992.

[3] Andrew K. McCallum. *Reinforcement Learning with Selective Perception and Hidden State*. PhD thesis, University of Rochester, 1995.

[4] Ron Parr. *Hierarchical Control and Learning for Markov Decision Processes*. PhD thesis, University of California at Berkeley, 1998.

[5] R. Sutton, D. Precup, and S. Singh. Intra-option learning about temporally abstract actions. In *Proceedings of the 15th International Conference on Machine Learning*, pages 556–564, 1998.