Hierarchical Reinforcement Learning Based on Subgoal Discovery and Subpolicy Specialization

Bram Bakker[1,2] & Jürgen Schmidhuber[1]
[1] *IDSIA, Manno-Lugano, Switzerland, {bram,juergen}@idsia.ch*
[2] *IAS, University of Amsterdam, The Netherlands, bram@science.uva.nl*

Abstract. We introduce a new method for hierarchical reinforcement learning. High-level policies automatically discover subgoals; low-level policies learn to specialize on different subgoals. Subgoals are represented as desired abstract observations which cluster raw input data. High-level value functions cover the state space at a coarse level; low-level value functions cover only parts of the state space at a fine-grained level. Experiments show that this method outperforms several flat reinforcement learning methods in a deterministic task and in a stochastic task.

1 Introduction

Autonomous systems ("agents") are often difficult to program. Reinforcement learning (RL) is an attractive alternative, as it allows the agent to *learn* behavior on the basis of sparse, delayed reward signals provided only when the agent reaches desired goals. However, standard reinforcement learning methods do not scale well for larger, more complex tasks. One promising approach to scaling up RL is *hierarchical* reinforcement learning (HRL) [12, 7]. Low-level policies, which emit the actual, "primitive" actions at a fast timescale, solve only parts of the overall task. Higher-level policies solve the overall task, but they may consider only few abstract, high-level observations and actions (macro-actions, options), at a slower timescale. This reduces each level's search space and facilitates temporal credit assignment. Another advantage is that low-level policies can be re-used easily, either within the same task or in other tasks.

Most studies on HRL assume that the hierarchical structure itself is given by a designer, and they learn policies within this hardwired structure [3, 11, 4, 2]. Since an important goal of machine learning is to minimize the designer's role, it is desirable to learn the hierarchy itself. As Barto and Mahadevan [2] put it, "a key open question is how to form task hierarchies automatically". The present paper presents a step in that direction. It is based on the idea of letting high-level policies identify *subgoals* that precede overall goals. Simultaneously, low-level policies (subpolicies) learn to reach subgoals set by the higher level. Low-level policies also learn which subgoals they are capable of reaching, and in this way they learn to specialize.

Our algorithm is relatively substrate independent, in the sense that tables or various function approximators with or without short-term memory may be used to represent the value functions. For this reason, the work is applicable in principle to both MDPs (Markov Decision Processes, where memoryless policies and standard value function-based RL are sufficient) and POMDPs (partially observable MDPs, requiring policies with memory). In this paper we only consider MDPs and memoryless policies (but see [1]).

The next section describes our HRL algorithm and relates it to previous work. Section 3 presents experimental results in a simulated office navigation task, with comparisons to several variants of non-hierarchical, "flat" RL. First a deterministic version of the task is investigated and then a stochastic version. Section 4 concludes with a general discussion of results, limitations, and possible extensions.

2 The HASSLE algorithm

2.1 Intuition

In what follows, for simplicity we assume only two levels in the control hierarchy, and focus on reward-only-at-goal tasks. However, the algorithm can be extended to more levels (see Conclusions) and possibly non-episodic tasks and continual rewards.

In the HASSLE algorithm (Hierarchical Assignment of Subgoals to Subpolicies LEarning algorithm), each action of a higher-level policy π^H is the selection of a subgoal to be reached by a lower-level policy π^L. HRL should reduce the search space of the higher-level policy [3]. To achieve this, our possible subgoals are taken from a set o^H of abstract high-level observations which differs from (and is smaller than) the set o^L of "primitive" low-level observations; o^H is also the set of possible inputs to the higher-level policy (details and possible limitations of this approach will be discussed below). At any time step of its discrete time scale, the high-level policy π^H receives the current high-level observation o_s^H as its input; its action is the selection of another high-level observation o_g^H as the current subgoal. Essentially, it selects the high-level observation that it wants to see next.

It is the job of the low-level policies to reach the subgoal. There is a limited set of low-level policies π_i^L. None of them is initially associated with any of the subgoals o_g^H or any of the possible o_s^H. Instead, this association is learned. Every π_i^L contains a table of so-called C-values of (o_s^H, o_g^H) pairs, each of which represents the "Capability" of π_i^L to reach subgoal o_g^H (Goal) from high-level observation o_s^H (Start). Following a high-level observation change, one low-level policy is selected based on the C-values of all low-level policies for the current (o_s^H, o_g^H) pair. This one then attempts to reach the current subgoal o_g^H. If it does indeed reach o_g^H, its C-value for this (o_s^H, o_g^H) pair is increased, making future selection of this low-level policy in this context more likely. Otherwise the C-value is decreased. Furthermore, the low-level policy itself receives a positive reward if it reaches the subgoal, such that it becomes better at reaching it, or it gets zero reward otherwise. In either case the high-level policy can now select a new subgoal, and the process repeats itself.

Thus, a low-level policy may learn that from some start situation it can reach subgoal A but not subgoal B. Another low-level policy may learn the opposite. This realizes specialization. But a single low-level policy may also learn that its capability encompasses *multiple* (o_s^H, o_g^H) pairs. This realizes generalization. The idea is that the low-level policies will specialize when they have to, but generalize when they can. After all, one can easily imagine cases where very similar behaviors will work well for different start/goal combinations. For example, a "leave this room" low-level policy may work well for diverse rooms with diverse neighboring places: generalization. But the "make coffee" subtask should not interfere with the "leave this room" strategy: specialization.

To facilitate generalization and specialization, in the experiments we will use function approximators for low-level policies. In this way, each low-level policy can learn to focus on those parts of the low-level observation space which are relevant to its specialization [4], and learn to generalize within that specialization. The current (o_s^H, o_g^H) pair should be part of its

input vector, because even though it does not know the high-level policy's overall "plan" it should know the high-level policy's current "command" [3]. With the high-level command as part of the input vector, a low-level policy can learn somewhat flexible behavior for various situations in which it is the specialist. For example, a "leave this room" subpolicy may be usable in various rooms although behavior and value function may be somewhat room-dependent.

With sufficient exploration by low-level policies, the current observation of the high-level policy will change at some point, such that it can select a new subgoal. Even so, it is a good idea to use a time-out value [3], i.e., a maximum number of low-level actions that the low-level policy can execute before control returns to the higher level. The high-level policy then gets the opportunity to select the same or another subgoal (the previous one might have been unreachable), and another low-level policy may get a chance.

An attractive feature of HASSLE is that both high-level and low-level policies may use standard value function-based reinforcement learning algorithms such as Q-learning [12]. We use Advantage learning [6] on both levels, because it works well with function approximators. The higher level value function covers the complete state space at a coarse level. It is updated based on "real" rewards r, received via interaction with the environment. Low-level value functions cover only parts of the overall state space, at a fine-grained level. They are updated based on "local" rewards r^L provided by the high-level policy.

2.2 Learning rules

High-level policies and low-level policies use temporal difference learning at different temporal resolutions. A high-level time step (incrementing t^H by 1) corresponds to a high-level observation change (or time-out of the low-level policy); a low-level time step (incrementing t^L by 1) corresponds to a low-level observation change. At each t^L, the currently active low-level policy executes one learning step (described below), and it selects a new low-level, primitive action using standard Boltzmann exploration [7]. At each t^H, the active low-level policy receives a reward; next, the high-level policy performs one learning step, and it selects a new subgoal o_g^H using standard Boltzmann exploration. Subsequently, a new low-level policy is Boltzmann-selected, using the $C_i(o_s^H, o_g^H)$ from all π_i^L.

High-level policy. π^H can learn no matter whether the desired subgoal o_g^H or a different $o^H \neq o_g^H$ was reached. In the latter case it simply learns as if the reached o^H was the desired subgoal. This difference to standard RL reflects the fact that high-level "actions" are just "desired high-level observations" rather than traditional actions, and allows for efficient use of all experiences. In both cases let us denote the reached observation by o_r^H. If the high-level policy values are implemented using a table (as in the first experiment described below), the update of the Advantage value $A^H(o_s^H, o_r^H)$ corresponds to

$$\begin{aligned} &\Delta A^H(o^H_{s,t^H}, o^H_{r,t^H}) = \\ &\alpha^H[V^H(o^H_{s,t^H}) + \frac{r_{t^H} + \gamma_H V^H(o^H_{s,t^H+1}) - V^H(o^H_{s,t^H})}{\kappa^H} - A^H(o^H_{s,t^H}, o^H_{r,t^H})] \end{aligned} \tag{1}$$

where $V^H(o^H_{s,t^H}) = \max_{o_g^H} A^H(o^H_{s,t^H}, o_g^H)$, α^H is a learning rate parameter, γ_H is a discount parameter, and κ^H is a constant scaling the difference between values of optimal and suboptimal subgoals (if $\kappa^H = 1$, the equations reduce to Q-learning). In addition, if the subgoal was not reached,

$$\Delta A^H(o^H_{s,t^H}, o^H_{g,t^H}) = \alpha^H[0 - A^H(o^H_{s,t^H}, o^H_{g,t^H})]. \tag{2}$$

The rationale behind this update is that failure to reach the subgoal indicates that the subgoal may not be a good one at this moment (perhaps because it is unreachable), so it should be penalized.

Low-level policies. The learning rule for the low-level policies is even closer to standard Advantage learning. With low-level function approximators (as used in the experiments), the weight updates for the currently active low-level policy π_i^L are

$$\Delta w_{i,m} = \alpha^L [V_i^L(o_{t^L}^L) + \frac{r_{t^L}^L + \gamma_L V_i^L(o_{t^L+1}^L) - V_i^L(o_{t^L}^L)}{\kappa^L} - A_i^L(o_{t^L}^L, a_{t^L}^L)] \frac{\partial A_i^L(o_{t^L}^L, a_{t^L}^L)}{\partial w_{i,m}} \quad (3)$$

where $V_i^L(o_{t^L}^L) = \max_{a^L} A_i^L(o_{t^L}^L, a^L)$, and α^L, γ_L, and κ^L are constants. In the experiments we will set $r^L = 1$ if the subgoal is reached and $r^L = 0$ otherwise. We use simple linear function approximators: low-level policies compute $A_i^L(o_{t^L}^L, a_k^L)$ for each low-level action a_k^L as a weighted sum of all o_m^L, the elements of the current observation vector. This means that the partial derivative of the Advantage value with respect to the weight $w_{i,m}$ in eq. 3 equals o_m^L.

C-values. Just like the high-level Advantage values, C-values can be updated no matter whether the subgoal is reached or not. If low-level policy i was the active one, then

$$\Delta C_i(o_{s,t^H}^H, o_{r,t^H}^H) = \alpha_r^C [\gamma_C^{t_r^L - t_s^L} - C_i(o_{s,t^H}^H, o_{r,t^H}^H)] \quad (4)$$

where α_r^C is a learning rate, γ_C a parameter which specifies to what extent a subpolicy's capability of reaching a subgoal should be discounted with each additional time step needed to reach it, t_s^L is the low-level time step at which the current low-level policy was started, and t_r^L the one at which o_r^H was reached. In addition, if the subgoal was not reached,

$$\Delta C_i(o_{s,t^H}^H, o_{g,t^H}^H) = \alpha_n^C [0 - C_i(o_{s,t^H}^H, o_{g,t^H}^H)] \quad (5)$$

where α_n^C is another learning rate. In the experiments α_n^C will be an order of magnitude smaller than α_r^C, because in initial learning stages low-level policies rarely reach desired subgoals. If these learning rates were equal, it would be hard to selectively boost C-values of mediocre low-level policies that work just a little bit better than others on this subgoal, because the C-values of all low-level policies would be pulled down more often than up. This would make it more difficult for any of the low-level policies to become the specialist for this subgoal and to learn it effectively.

2.3 Producing high-level observations

So far we have ignored the issue of how to arrive at abstract high-level observations. Here HASSLE is not constrained to any particular method. The main requirement is that a clustering of primitive, low-level observations is accomplished such that neighboring low-level states tend to be clustered together. This allows the high-level policy to select meaningful subgoals, and the low-level policies to develop non-trivial behavior leading the system from subgoal to subgoal.

In the experiments we will use a simple unsupervised learning vector quantization technique called ARAVQ (Adaptive Resource Allocation Vector Quantization [8]). It adaptively allocates a new model vector to classify some continuous-valued input vector if the latter's Euclidean distance to any of the existing model vectors exceeds a parameter δ. The version used here is a simplified version without ARAVQ's original stage of determining whether a sequence of consecutive inputs is sufficiently stable, and without its online adaptation of existing model vectors.

2.4 Related work

HASSLE shares the idea of finding subgoals associated with observations with several other HRL algorithms which learn the hierarchical structure [5, 13, 10, 9]. These algorithms, however, associate subgoals with primitive, low-level observations rather than high-level observations. As explained above, there are important advantages to having the higher levels abstract in observation space and action space. Those HRL algorithms that do exploit the advantages of high-level observations (notably Feudal RL [3]) are limited to *predesigned* hierarchies.

Previous HRL approaches also differ in how they assign low-level policies to specific subgoals. For example, Feudal RL [3] distributes low-level policies evenly over the entire state space, such that at each level each possible start/subgoal pair has exactly one responsible low-level policy. One disadvantage is that this usually implies that many low-level policies and value functions need to be stored and learned. Other approaches identify a limited set of subgoals or subtasks in an initial design phase [11, 4] or learning phase [9], and assign one low-level policy to each subgoal or subtask. Thereafter low-level policies are used by high-level policies as though they were normal (but temporally extended) actions. However, this approach depends on significant knowledge of or initial experience with the overall task.

In contrast, HASSLE and HQ-learning [13] and the SSS algorithm [10] simultaneously learn subgoals and low-level policies specializing on certain subgoals (low-level observations for HQ and SSS but not for HASSLE). Only a limited set of low-level policies is available for the whole state space and all subgoals. However, HQ and SSS do not use local reward functions. Instead the value functions of their low-level policies strongly depend on those of subsequently invoked low-level policies. For this reason, HQ's low-level policies can be used only once per episode, and SSS's can be used multiple times only when the various trigger situations are very similar in value and appearance. This is regrettable as one would like to be able to reuse low-level policies, say "leave this room", everywhere in the environment, close to and far from the goal, etc. HASSLE can easily invoke its subpolicies again and again.

3 An illustrative experiment: office navigation

3.1 Task setup and comparison with flat methods

To demonstrate and test HASSLE we consider a navigation task in a simulated "office" grid world. The agent should learn to move from any possible start position to a fixed goal position (see Figure 1). Its possible orientations are north, east, south, west. It has three primitive low-level actions: make a step in the current direction, turn left 90°, turn right 90°. It can move through doors but not through walls. Actions and observations are deterministic (but see below). The number of states is 10,973. Each episode starts with a random position and orientation. It ends once the agent reaches the goal (where it receives the only reward $r = 4$), or after at most 20,000 low-level actions (episode time-out). Random walk finds the goal in around 24% of the episodes, within 6667 low-level time steps on average.

Four simulated directed "sonar" sensors (one for each direction relative to the current orientation) measure the current distance (in number of grid cells) to the nearest wall/door. Four additional directed binary sensors detect the presence of doors, and four more the presence of the goal, provided it is at most 8 grid cells away. The agent's orientation is part of the primitive low-level observation vector. With these sensors the task is an MDP, because the distance to walls or doors has a unique value for each position.

The high-level observations are produced online by applying ARAVQ to the 4-dimensional

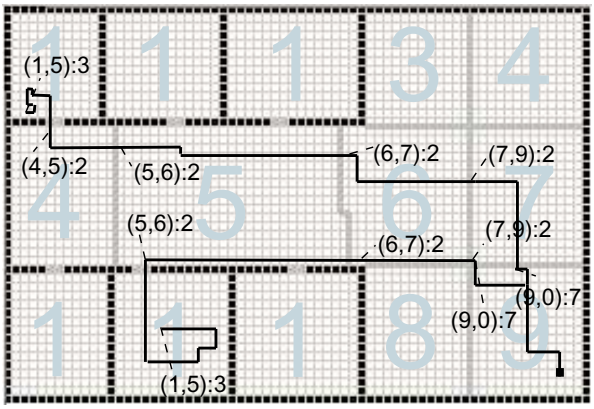

Figure 1: The office environment. The goal is the black dot in the lower right corner. There are 6 rooms, connected by doors to a central corridor. Large grey numbers indicate 9 possible high-level observations extracted by ARAVQ; 14 fields with grey boundaries mark areas in which high-level observations do not change.

output of the distance sensors, ignoring the current orientation, using $\delta = 32$. Figure 1 shows how ARAVQ distributes high-level observations over the state space (large grey numbers). There is also one high-level observation that corresponds to the goal, such that the high-level policy can learn to select it as a subgoal once the agent is close enough. This yields a total of 10 high-level observations.

Thus, π^H consists of a table of 10 (o_s^H) by 10 (o_g^H) A^H-values ($\alpha^H = .02$, $\gamma_H = .95$, and $\kappa^H = .2$). We use 8 π_i^L, each using a table of 10 by 10 C_i-values ($\alpha_r^C = .002$, $\alpha_n^C = .00004$, $\gamma_C = .99$) and a linear function approximator with 3 outputs encoding the A_i^L-values of the low-level actions, and a 36-element input vector (the 12 sensors described above, 4 bits to encode the agent's orientation, 10 bits to encode o_s^H, and 10 bits to encode o_g^H). We use $\alpha^L = .002$, $\gamma_L = .95$, $\kappa^L = .2$. The time-out value for low-level policies is 100, i.e. a low-level policy may execute at most 100 low-level actions to reach its assigned subgoal. All parameters here and below were found by a coarse search in parameter space.

We compare the HASSLE system to 4 flat RL systems. The first is a single linear function approximator similar to HASSLE's low-level policies, also trained using Advantage learning. Note that the nature of the primitive observations makes a table-based system inappropriate. This system's input and output vectors are like those for a HASSLE low-level policy (see above), except that the 20 input bits for encoding o_s^H and o_g^H are omitted. The learning parameters are $\alpha = .0001$, $\gamma = .99$, and $\kappa = .2$. One may argue that the task is just too complex for a simple linear function approximator. For this reason, we replace it with a nonlinear one: a multilayer feedforward neural network (MLFF) approximating the value function. The network is trained using the same learning algorithm and inputs and outputs as the linear function approximator. It has 15 hidden units and uses $\alpha = .002$, $\gamma = .99$, and $\kappa = .2$. Finally, we extend the input vectors of both flat systems by 10 additional bits encoding o_s^H as computed by ARAVQ, to investigate the utility of one HASSLE aspect, namely, spatial and temporal abstraction of observations, without interference from HASSLE's other features. Learning parameters are left unchanged.

3.2 Results

Figure 2a plots the average number of primitive actions needed to reach the goal as a function of the total number of low-level time steps. All curves are averages of 10 runs. HASSLE converges to roughly 90 low-level actions per episode within around 1 million low-level time

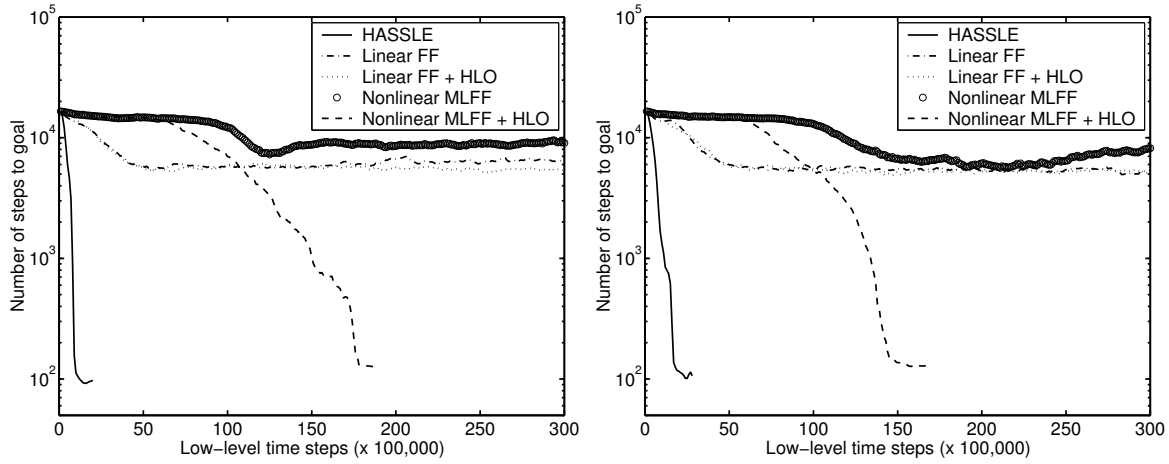

Figure 2: Average numbers of low-level actions (note the *log* scale) needed to reach the goal, as functions of learning time (low-level time steps). Fig. a (left). The deterministic task. Fig. b (right). The stochastic task. See text for details.

steps. Such policies are close to optimal, as determined by studying the agent's trajectories. Flat linear systems with and without high-level observations (HLOs) do achieve significant performance improvements compared to a random walk (they reach the goal in many more cases), but never come close to HASSLE. The first nonlinear system (MLFF) performs poorly, but the one *with* high-level observations (MLFF + HLO) does quite well and its final performance is only a bit worse than HASSLE's. It takes much longer, however. But this shows that a single memoryless policy can indeed learn the task fairly well, and that high-level observations are helpful even for an otherwise flat RL system, probably because they provide useful "global" information about the agent's position relative to the goal.

Figure 1 shows trajectories of a HASSLE agent after learning. The invocation of new subgoals and subpolicies is indicated by labels of the form (o_s^H, o_g^H): *subpolicy index*. For example, in the upper left room, $o_s^H = 1$, subgoal $o_g^H = 5$ is selected, and specialized subpolicy 3 is started. This subpolicy has learned to look for doors and go through them. Following the trajectories, one can see how the high-level policy selects proper subgoals following each high-level observation change, as well as examples of specialization and generalization of subpolicies. A single subpolicy always specialized on leaving rooms, and usually this was all it could do. Other specialists learned to move through the corridor, and usually there was a specialist for moving to the goal once it was close enough.

3.3 A stochastic variation of the office navigation task

Realistic autonomous systems (e.g. robots) typically do not have *deterministic* tasks such as the one above. The following experiment investigates the same office navigation task with *stochastic* observations and actions. Each of the 4 distance sensors now adds independent Gaussian noise (standard deviation .05) to the actual distance at each low-level time step, and with probability .01 one of the sensors detecting door or goal has the wrong value. With probability .05, a random high-level observation is perceived rather than the correct one. Furthermore, with probability .01 primitive turn left/right actions turn too far or fail to turn, and primitive move forward actions move in a random direction.

Architecture and parameters of HASSLE and the four flat RL methods remain unchanged. Figure 2b plots the average number of primitive actions needed to reach the goal as a function of the total number of low-level time steps. As before, HASSLE converges to near-optimal policies—it can deal with stochastic tasks as well. Presumably this is due to using, at both

levels in the hierarchy, standard RL algorithms, which are known to handle stochasticity well. Not surprisingly, the post-learning paths to the goal are a bit longer on average than in the deterministic case. Only slightly longer learning times are needed to converge. As before, HASSLE outperforms the flat systems. Again, the only flat system that converges to good solutions (after much longer training than HASSLE) is the nonlinear one aided by inclusion of the high-level observation in its input vector (MLFF + HLO).

4 Conclusions

HASSLE, our new hierarchical RL algorithm, outperformed variants of plain RL in deterministic and stochastic versions of a large MDP. It did so by learning to create both useful subgoals and the corresponding specialized subtask solvers. High-level value functions cover the state space at a coarse level; low-level value functions cover only parts of the state space at a fine-grained level. Current limitations of the system include the large number of parameters, the lack of strict convergence guarantees (although the system uses standard RL algorithms for which some limited convergence theorems exist), and the dependence on identifying reasonable high-level observations.

Ongoing work aims at *adapting* high-level observations online so as to improve their utility as subgoals. Furthermore, HASSLE can be used with policies with memory for the case of POMDPs [1]. Finally, in preliminary experiments the system was successfully extended by adding a still higher (third) level. This is straightforward as the task of reaching a goal (a real one or a higher level's subgoal) remains essentially the same for all levels.

5 Acknowledgments

The first author was funded in part by a CSEM research grant on "Learning robots" and in part by an NWO research grant on "Concept learning".

References

[1] B. Bakker and J. Schmidhuber. Hierarchical reinforcement learning based on subgoal discovery and subpolicy specialization: First experiments with the HASSLE algorithm. Technical report, IDSIA, 2003.

[2] A. G. Barto and S. Mahadevan. Recent advances in hierarchical reinforcement learning. *Discrete Event Systems, Special issue on reinforcement learning*, 13:41–77, 2003.

[3] P. Dayan and G. E. Hinton. Feudal reinforcement learning. In *Advances in Neural Information Processing Systems 5*, 1993.

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

[5] B. Digney. Emergent hierarchical control structures: Learning reactive/hierarchical relationships in reinforcement environments. In *Proc. Conf. Simulation of Adaptive Behavior*, 1996.

[6] M. E. Harmon and L. C. Baird. Multi-player residual advantage learning with general function approximation. Technical report, Wright-Patterson Air Force Base, 1996.

[7] L. P. Kaelbling, M. L. Littman, and A. W. Moore. Reinforcement learning: A survey. *Journal of Artificial Intelligence Research*, 4:237–285, 1996.

[8] F. Linåker and H. Jacobsson. Mobile robot learning of delayed response tasks through event extraction: A solution to the road sign problem and beyond. In *Proc. of IJCAI'2001*, 2001.

[9] A. McGovern and A. G. Barto. Automatic discovery of subgoals in reinforcement learning using diverse density. In *Proc. 18th International Conf. on Machine Learning*, 2001.

[10] R. Sun and C. Sessions. Self-segmentation of sequences: automatic formation of hierarchies of sequential behaviors. *IEEE Trans. on Systems, Man, and Cybernetics*, 30:403–418, 2000.

[11] R. S. Sutton, D. Precup, and S. Singh. Between MDPs and semi-MDPs: A framework for temporal abstraction in reinforcement learning. *Artificial Intelligence*, 112:181–211, 1999.

[12] C. Watkins. *Learning from delayed rewards*. PhD thesis, Cambridge University, 1989.

[13] M. Wiering and J. Schmidhuber. HQ-learning. *Adaptive Behavior*, 6:2, 1997.