

# Autonomous Learning of Sequential Tasks: Experiments and Analyses

Ron Sun<sup>1,2</sup>, Todd Peterson<sup>2</sup>

<sup>1</sup> NEC Research Institute, 4 Independence Way, Princeton, NJ 08540

<sup>2</sup> The University of Alabama, Tuscaloosa, AL 35487

EMAIL: rsun, todd@cs.ua.edu

August 28, 1998

**KEYWORDS:** multi-strategy learning, hybrid models, reinforcement learning, rule extraction, navigation, Markov decision process, sequential decision making

**Acknowledgements:** This work is supported in part by Office of Naval Research grant N00014-95-1-0440. We thank Helen Gigley and Susan Chipman for the support. The simulator for the navigation task was provided by NRL (through Diana Gordon, Jim Ballas and Alan Schultz). Ed Merrill helped with related work. Thanks to Dave Waltz, Diana Gordon, and Marco Dorigo for their comments. Thanks to the anonymous reviewers for their suggestions.

## ABSTRACT:

This paper presents a novel learning model CLARION, which is a hybrid model based on the two-level approach proposed in Sun (1995). The model integrates neural, reinforcement, and symbolic learning methods to perform on-line, bottom-up learning (i.e., learning that goes from neural to symbolic representations). The model utilizes both procedural and declarative knowledge (in neural and symbolic representations respectively), tapping into the synergy of the two types of processes. It was applied to deal with sequential decision tasks. Experiments and analyses in various ways are reported that shed light on the advantages of the model.

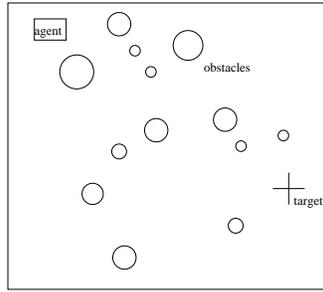


Figure 1: Navigating Through A Minefield

## 1 Introduction

This paper presents a model that unifies neural, symbolic, and reinforcement learning. It addresses the following three issues: (1) It deals with autonomous learning: It allows a situated agent to learn autonomously and continuously, from on-going experience in the world, without the use of preconstructed data sets or preconceived concepts (i.e., a priori knowledge). (2) The model learns not only procedural knowledge but also declarative knowledge (which is beyond traditional reinforcement learning algorithms, as will be discussed later). (3) The learning is bottom-up: declarative knowledge is acquired from an agent’s experience interacting with the world through the mediation of procedural knowledge. (This differs from top-down learning in which procedural knowledge is acquired through “compiling” externally given declarative knowledge (e.g., Anderson 1983).)

Sequential decision tasks involve selecting and performing a sequence of actions to accomplish an objective on the basis of moment-to-moment perceptual information. An abstract specification of sequential decision tasks is as follows: there is an agent (or a set of agents) that can select, from a finite set of actions, an action to perform at each time step. The selection decision is mainly based on the current state of the world. The world is presented to the agent, through sensory input, as a state vector and over time, as a sequence of state vectors. At certain points in a sequence, the agent may receive *payoffs* or *reinforcements* for their actions performed at or prior to the current state. Thus, the agent may need to perform *credit assignment* in learning, to attribute the payoffs/reinforcements to actions at various points in time (the temporal credit assignment problem), and in accordance to various aspects of a state (the structural credit assignment problem). There is in general no teacher providing additional input. The agent starts with little or no a priori knowledge. One example is learning to navigate through minefields (see Figure 1; more discussion later).

Although the essential motivation for this project is cognitive modeling (that is, understanding human learning in sequential decision making situations; see Sun 1997, Sun et al 1997), this paper will

focus instead on computational experiments and analyses of the model that we developed. The goal of this paper is to detail the specifics of the model and the rationales behind the design of the model, and then through experiments, to demonstrate various aspects of the model, especially its performance advantages (i.e., the synergy; more later). In the remainder of the paper, section 2 presents the model. Section 3 presents some experiments in navigation domains. Section 4 provides some further analyses, while section 5 relates the model to existing work. Section 6 ends the paper with concluding remarks.

## 2 A Model of Procedural and Declarative Knowledge

There has been a great deal of work demonstrating the difference between procedural knowledge and declarative knowledge: for example, Anderson (1983), Keil (1989), Damasio et al. (1990), and Sun (1994). It is believed that a balance between the two is essential to the development of complex cognitive agents. This is based on two lines of argument. First, there are ample psychological data that support the distinction between procedural and declarative knowledge and the need for both. Anderson (1983) initially proposed the distinction based on human data on problem solving. Keil (1989) and Sun (1994) subsequently made similar points also based on psychological data. Second, there are many philosophical arguments for making this distinction and achieving an appropriate balance between the two. Dreyfus and Dreyfus (1987) proposed the distinction between analytical and intuitive thinking. Smolensky (1988) proposed the distinction between conceptual (publicly accessible) and subconceptual processing. In addition, the distinction between conscious and subconscious processes, although controversial, is well known (cf. James 1890, Sun 1997). The inadequacy of “intelligent” systems that ignore these points in dealing with the full range and complexity of intelligent behaviors lends additional support for these arguments. We are not aiming to capture all of the above dichotomies. Denoting more or less the same thing, these dichotomies serve as justifications for our main distinction, between (conceptual) declarative knowledge and (subconceptual) procedural knowledge, which is distinguished by the difference in accessibility (declarative knowledge is easily accessible and linguistically expressible, procedural knowledge is not; Anderson 1983, 1993, Sun 1994).

Declarative knowledge has some advantages which make it indispensable to a learning agent despite the fact that procedural knowledge is more efficient or easier to learn. These advantages are:

- It helps to guide the exploration of new situations, and reduces the time necessary to develop skills in new situations. In other words, it helps the transfer of learned skill (as shown psychologically by Willingham et al 1989).

- Declarative knowledge can help to speed up learning. If properly used, explicit knowledge that is extracted on-line during procedural learning can help to facilitate the learning process itself (Willingham et al 1989, Stanley et al 1989).
- Declarative knowledge can also help in communicating learned knowledge and skills to other agents.

A two-level (or two-tiered) model provides the needed framework for representing both types of knowledge in an integrated model that encompasses both. Our model is named CLARION (i.e., *Connectionist Learning with Adaptive Rule Induction ON-line*), which is similar to the model proposed in Sun (1994, 1995) but is specifically designed for skill learning in sequential decision tasks. It consists of two levels: The bottom level contains procedural knowledge (Anderson 1983) and the top level contains declarative knowledge in the form of propositional rules.

Let us discuss the two levels in some more detail based on the idea of Sun (1994, 1995). There are indications that the difference between the two levels lies mainly in their representations (see Reber 1989 regarding available psychological evidence in this respect). First, to capture procedural knowledge, we prefer a subsymbolic distributed representation, such as that provided by a backpropagation network. This is because of the implicit nature (i.e., inaccessibility) of procedural knowledge (Anderson 1983) (and also because of the need for function approximators due to large state spaces; more later). In terms of learning, we prefer reinforcement learning (i.e., the temporal difference method, which can handle sequential decision situations; Sutton 1988). This is because in sequential decision tasks such as navigation, there is seldomly any uniquely correct action. In general, for each situation, there are various possible responses that are roughly equally good; thus, supervised learning procedures do not seem applicable. However, for each situation, there are indeed good actions and bad actions; we measure the goodness of actions through a payoff/reinforcement signal. An adjustment can be made to some parameters to increase the chance of selecting the actions that will lead to positive reinforcement and to reduce the chance of selecting the actions that will lead to negative reinforcement. Second, to capture declarative knowledge, we prefer a symbolic (or localist) representation, in which each unit has a clear conceptual meaning (or interpretation), because declarative knowledge is highly accessible and inferences are performed explicitly at the conceptual level (Smolensky 1988, Sun 1994, 1995). We focus on propositional rules here. In learning rules, we can make use of the other level — the network that is trained with reinforcement learning and embodies procedural knowledge. There is thus no need for a completely separate learning mechanism for the top level. We can extract information from the bottom-level and thereby form explicit rules. In addition, we should dynamically acquire rules

and modify rules subsequently when necessary. (As a comparison, human learning is often on-line and gradual, which is especially true in rule learning as specifically discussed by Dominowski (1975), Nosofsky et al (1994) and others.)

An overall sketchy pseudo-code algorithm that describes the operation of CLARION is as follows (we will explain the notions of Q-values, Q-learning, Rule-Extraction-Revision, and so on later):

1. Observe the current state  $x$ .
2. Compute in the bottom level the *values* of  $x$  associated with each of all the possible actions  $a_i$ 's:  
 $Q(x, a_1), Q(x, a_2), \dots, Q(x, a_n)$ .
3. Find out all the possible actions ( $b_1, b_2, \dots, b_m$ ) at the top level, based on the input  $x$  and the rules in place.
4. Choose an appropriate action  $b$ , considering the values of  $a_i$ 's and  $b_j$ 's.
5. Perform the action  $b$ , and observe the next state  $y$  and (possibly) the reinforcement  $r$ .
6. Update the bottom level in accordance with *Q-Learning*.
7. Update the top level with *Rule-Extraction-Revision*.
8. Go back to Step 1.

## 2.1 The Bottom Level

**Reinforcement learning.** A Q-value is an evaluation of the “quality” of an action in a given state:  $Q(x, a)$  indicates how desirable action  $a$  is in state  $x$  (which consists of sensory input). To acquire the Q-values, we use the *Q-learning* algorithm (Watkins 1989; a temporal difference algorithm). In the algorithm,  $Q(x, a)$  estimates the maximum discounted cumulative reinforcement that the agent will receive from the current state  $x$  on. The updating of  $Q(x, a)$  is based on minimizing the so called Bellman residual:

$$r + \gamma e(y) - Q(x, a)$$

where  $\gamma$  is a discount factor,  $e(y) = \max_a Q(y, a)$ , and  $y$  is the new state resulting from action  $a$ . Thus, the updating is based on the *temporal difference* in evaluating the current state and the action chosen. Through successive updates of the Q function, the agent can learn to take into account future steps in longer and longer sequences, notably without explicit planning (Watkins 1989).

**Implementation.** To implement Q-learning, we chose to use a four-layered network, in which the first three layers form a backpropagation network for computing Q-values and the fourth layer (with only one node) performs stochastic decision making. The output of the third layer (i.e., the output layer of the backpropagation network) indicates the Q-value of each action (represented by

an individual node), and the node in the fourth layer determines probabilistically the action to be performed based on a Boltzmann distribution (Watkins 1989):

$$p(a|x) = \frac{e^{Q(x,a)/\tau}}{\sum_i e^{Q(x,a_i)/\tau}}$$

Here  $\tau$  controls the degree of randomness (temperature) of the decision-making process. The training of the backpropagation network is based on the following error signals:

$$err_i = \begin{cases} r + \gamma e(y) - Q(x, a) & \text{if } a_i = a \\ 0 & \text{otherwise} \end{cases}$$

where  $i$  is the index for an output node representing the action  $a_i$  and  $a$  is the action performed. The backpropagation procedure is then applied as usual to adjust weights. <sup>1</sup>

## 2.2 The Top Level

**Rule encoding.** Declarative knowledge is captured in a simple propositional rule form. Although we can directly use a symbolic rule representation, to facilitate correspondence with the bottom level and to encourage uniformity and integration, we chose to use a localist connectionist model instead. Basically, we connect the nodes representing the conditions of a rule to the node representing the conclusion (Sun 1992, Towell and Shavlik 1993). That is, we directly translate the structure of a rule set to that of a network. Details will be discussed later.

**Rule learning.** We devised a novel rule learning algorithm in accordance with the task characteristics: concurrent (on-line) learning, reactivity, and the lack of teacher input and a priori knowledge. The basic idea is as follows: we perform rule learning (extraction and subsequent revision) at each step. If some action decided by the bottom level is successful (as measured by some criterion, which will be presented later) then the agent extracts a rule that corresponds to the decision and adds the rule to the rule network. Then, in subsequent interactions with the world, the agent verifies the extracted rule by considering the outcome of applying the rule: if the outcome is not successful (as measured by a criterion), then the rule should be made more specific and exclusive of the current case; if the outcome is successful, the agent may try to generalize the rule to make it more universal.

Let us look into a refined version of the overall algorithm that fleshes out some rule learning details (in steps 8 and 9).

---

<sup>1</sup>The lookup table implementation of Q-learning is usually not feasible because of the (likely) continuous input space and when discretized, the resulting huge state space (e.g., in the minefield navigation task there are more than  $10^{12}$  states). A function approximator has to be used (Sutton 1990, Lin 1992, Mahadevan and Connell 1992, Tesauro 1992) and as a result the convergence of learning is not guaranteed (Bertsekas and Tsitsiklis 1996).

Do the following for each decision step:

1. Observe the current state  $x$ .
2. Calculate the Q-values of all the actions in the current state:  $Q(x, a_1), Q(x, a_2), \dots, Q(x, a_n)$
3. Find all the rules that match the current state. Choose one matching rule (randomly) to apply.
4. Combine the outcomes of the previous two steps using a weighted sum (details later). Based on the combined values of actions, select an action  $b$  to be performed using the Boltzmann distribution.
5. Perform the action  $b$ .
6. Enter the new state  $y$  resulting from the action. Possibly receive a reinforcement  $r$ .
7. Calculate all the Q-values for the new state:  $Q(y, a_1), Q(y, a_2), \dots, Q(y, a_n)$
8. Update statistics for each rule and its variations (defined later). Check the current criterion for rule extraction and revision.
  - 8.1. If the result is successful according to the current criterion (to be defined later), and there is no rule matching that state and that action, then perform *extraction* of a new rule: state  $\rightarrow$  action. Add the extracted rule to the rule network.
  - 8.2. If the result is unsuccessful according to the current criterion (defined later), revise all the matching rules using *shrinking* and *deletion*.
    - 8.2.1. Remove the matching rules from the rule network.
    - 8.2.2. Add the revised versions of the rules into the rule network.
  - 8.3. If the result is successful according to the current criterion, then generalize the matching rules.
    - 8.3.1. Create new rules using *expansion*.
    - 8.3.2. Add the expanded rules to the rule network to replace the original rules.
9. Perform *merge* to combine existing rules.
10. Update the bottom level (as specified before).

**Details of Rule Encoding and Learning.** First, let us examine details of rule encoding. Assume that an input state  $x$  is made up of a number of dimensions (e.g.,  $x_1, x_2, \dots, x_n$ ). Each dimension can have a number of possible values (e.g.,  $v_1, v_2, \dots, v_m$ ).<sup>2</sup> Rules are in the following form: *current-state*  $\rightarrow$  *action*, where the left-hand side is a conjunction of individual elements each of which refers to a dimension  $x_i$  of the input  $x$ , specifying a value range or a value in the dimension (i.e.,  $x_i = v_i$  or  $x_i \in \{v_{i1}, v_{i2}, \dots\}$ ), and the right-hand side is an action recommendation  $a$ . Each element in the left-hand side is represented by an individual node. For each rule, a set of links are established, each of which connects a node representing an element in the left-hand side of a rule to the node representing the conclusion in the right-hand side of the rule. If an element is in a positive form, the link carries a positive weight  $w$  (which we set to 1); otherwise, it carries a negative weight  $-w$ . Sigmoidal functions

---

<sup>2</sup>Each dimension is either ordinal (discrete or continuous) or nominal. In the following discussion, we focus on ordinal values; nominal values can be handled similarly. A binary dimension is a special case of a discrete ordinal dimension.

are used for node activation (which is an obvious choice; other functions are also possible):

$$\frac{1}{1 + e^{\sum_i i_i w_i - \nu}} \quad (1)$$

The threshold  $\nu$  of a node is set to be  $n * w - \theta$ , where  $n$  is the number of incoming links (the number of the elements in the condition leading to the conclusion represented by this node), and  $\theta$  is a parameter, selected along with  $w$  to make sure that it is always the case that the node has activation above 0.9 when all the elements of its condition are satisfied, and has activation below 0.1 when some elements of its condition are not met. Activations above 0.9 are considered 1, and activations below 0.1 are considered 0. So rules are in fact discretized and thus crisp/binary.<sup>3</sup>

There are several different versions of rule learning for CLARION, including a deterministic version and two statistical versions. The criteria used in these versions measuring whether a step is successful or not (needed for rule extraction, shrinking, and expansion) are different, and some other details of the operations are also different. Let us focus here on a statistical version of CLARION (see Sun and Peterson 1997, 1998 for the other versions). In this version, the criteria for rule construction, shrinking, deletion, and expansion are mostly determined by an *information gain* measure (which basically compares the quality of two candidate rules). To calculate the information gain measure, we do the following. At each step, we examine the following information:  $(x, y, r, a)$ , where  $x$  is the state before action  $a$  is performed,  $y$  is the new state after an action  $a$  is performed, and  $r$  is the reinforcement received after action  $a$ . Based on that, we update the positive and negative match counts for each rule condition and each of its variations (i.e., the rule condition plus/minus one possible value in one of the input dimensions)  $C$ , with regard to the action  $a$  performed. That is, we update  $PM_a(C)$  (i.e., Positive Match) and  $NM_a(C)$  (i.e., Negative Match).  $PM_a(C)$  equals the number of times that an input matches the condition  $C$ , action  $a$  is performed, and the result is positive.  $NM_a(C)$  equals the number of times that an input matches the condition  $C$ , action  $a$  is performed, and the result is negative. Here, positivity/negativity is determined by the following inequality concerning Bellman residuals:  $\max_b Q(y, b) - Q(x, a) + r > threshold$ , which indicates whether or not the action is reasonably good (see the analysis next; see also Sun and Peterson 1998).<sup>4</sup>

---

<sup>3</sup>In addition, if there is more than one rule that leads to the same conclusion, an intermediate node is created for each such rule: all of the elements on the left-hand side of each rule are linked to the same intermediate node, and then all the intermediate nodes are linked to the node representing the conclusion. For more complex rule forms including predicate rules and variable binding, see Sun (1992). Such rules can be learned using more complex ILP techniques (see Lavrac and Dzeroski 1994).

<sup>4</sup>Each statistic is updated with the following formula:  $stat := stat + 1$  (where  $stat$  stands for  $PM$  or  $NM$ ). At the end of each episode, it is discounted by:  $stat := stat * 0.90$ . The results are time-weighted statistics, which are useful in nonstationary situations.

Based on these statistics, we calculate the information gain measure; that is,

$$IG(A, B) = \log_2 \frac{PM_a(A) + 1}{PM_a(A) + NM_a(A) + 2} - \log_2 \frac{PM_a(B) + 1}{PM_a(B) + NM_a(B) + 2}$$

where A and B are two different conditions that lead to the same action  $a$ .<sup>5</sup> The measure compares essentially the percentage of positive matches under different conditions A and B (with the Laplace estimator; Lavrac and Dzeroski 1994). If A can improve the percentage to a certain degree over B, then A is considered better than B. In the algorithm, if a rule is better compared with the match-all rule (i.e, the rule with the condition that matches all inputs), then the rule is considered successful (for the purpose of deciding on expansion or shrinking operations).

We decide on whether or not to construct a rule based on a simple success criterion which is fully determined by the current step  $(x, y, r, a)$ :

- *Extraction*: if the Bellman residual  $r + \gamma e(y) - Q(x, a) > threshold$ , where  $a$  is the action performed in state  $x$  and  $y$  is the resulting new state [**that is, if the current step is successful**], and if there is no rule that covers this step in the top level, set up a rule  $C \rightarrow a$ , where  $C$  specifies the values of all the input dimensions exactly as in  $x$ .

The criterion for applying the *expansion* and *shrinking* operators, on the other hand, is based on the afore-mentioned information gain measure. Expansion amounts to adding an additional value to one input dimension in the condition of a rule, so that the rule will have more opportunities of matching inputs, and shrinking amounts to removing one value from one input dimension in the condition of a rule, so that it will have less opportunities of matching inputs. Here are the detailed descriptions of these operators:

- *Expansion*: if  $IG(C, all) > threshold1$  and  $\max_{C'} IG(C', C) \geq 0$ , where  $C$  is the current condition of a matching rule, *all* refers to the match-all rule (with regard to the same action specified by the rule), and  $C'$  is a modified condition such that  $C' = C$  plus one value (i.e.,  $C'$  has one more value in one of the input dimensions) [**that is, if the current rule is successful and an expanded condition is potentially better**], then set  $C'' = \operatorname{argmax}_{C'} IG(C', C)$  as the new (expanded) condition of the rule. Reset all the rule statistics. Any rule covered by the expanded rule will be placed in its children list.<sup>6</sup>

---

<sup>5</sup>This is a widely used measure, for example, in many Inductive Logic Programming models, and well justified on the empirical ground. See e.g. Lavrac and Dzeroski (1994).

<sup>6</sup>The children list of a rule is created to keep aside and make inactive those rules that are more specific (thus fully covered) by the current rule. It is useful because if later on the rule is deleted or shrunk, some or all of those rules on its children list may be reactivated if they are no longer covered.

- *Shrinking*: if  $IG(C, all) < threshold2$  and  $\max_{C'} IG(C', C) > 0$ , where  $C$  is the current condition of a matching rule, *all* refers to the match-all rule (with regard to the same action specified by the rule), and  $C'$  is a modified condition such that  $C' = C$  minus one value (i.e.,  $C'$  has one less value in one of the input dimensions) [**that is, if the current rule is unsuccessful, but a shrunk condition is better**], then set  $C'' = \operatorname{argmax}_{C'} IG(C', C)$  as the new (shrunk) condition of the rule.<sup>7</sup> Reset all the rule statistics. Restore those rules in the children list of the original rule that are not covered by the shrunk rule.
- *Deletion*: included in *Shrinking*. If shrinking the condition makes it impossible for a rule to match any input state, delete the rule.
- *Merge*: when the conditions of two rules are close enough, the two rules may be combined: If one rule is covered completely by another, it is put on the children list of the other. If one rule is covered by another except for one dimension, produce a new rule that covers both.

Note that although the accumulation of statistics is gradual, the acquisition and refinement of rules are one-shot and all-or-nothing, in contrast to the bottom level.

### 2.3 The Whole Model

In the overall algorithm, step 4 is for making the final decision of which action to take by incorporating outcomes from both levels. Several different methods of combining outcomes from the two levels were tried. A good method is the *stochastic method*. In this method, we combine the corresponding values for an action from the two levels by a weighted sum; that is, if the top level indicates that action  $a$  has an activation value  $v$  (which should be 0 or 1 as rules are binary) and the bottom level indicates that  $a$  has an activation value  $q$  (the Q-value), then the final outcome is  $w_1 * v + w_2 * q$ . Stochastic decision making with Boltzmann distribution based on the weighted sums is then performed to select an action out of all the possible actions.

Relative weights of the two levels can be automatically set based on the *probability matching* of the relative performance of the two levels (which is commonly observed in animal behavior). That is, if the success rate of the decisions made by the top level is  $s_b$  and the success rate of the bottom level is  $s_t$ , then the weights are  $s_t / (s_b + s_t)$  for the top level and  $s_b / (s_b + s_t)$  for the bottom level.<sup>8</sup>

---

<sup>7</sup>Clearly, we should have  $threshold2 \leq threshold1$  to avoid oscillation.

<sup>8</sup>We have  $s_b = 1 - e_b$  and  $s_t = 1 - e_t$ ; in turn,  $e_b = E[(\max_b Q(y, b) - Q(x, a) + r)^2 * p(a|x)]$ , where  $x$  is the current state,  $a$  is the action chosen,  $y$  is the new state, and  $p(a|x) = \frac{e^{Q(x, a)/\tau}}{\sum_i e^{Q(x, a_i)/\tau}}$  is the probability of choosing action  $a$  in

## 2.4 Some Preliminary Analyses

We can contrast the characteristics of the two levels. The top level is discrete, all-or-nothing, rigorously verified (through statistics), and without random exploration, and it learns through trial-and-error in a one-shot fashion. The bottom level is continuous, graded, not rigorously verified, and with random exploration, and it learns in a gradual and cumulative fashion. Thus they complement each other. Note also that the generalization of rules complements the generalization in the bottom level: While the bottom level generalization is continuous/graded, rule generalization at the top level is discrete/crisp (in terms of results, not the process), thus capturing different kinds of regularities. Because they possess different characteristics, each level tends to learn differentially; thus a combination of the two, through stochastic “averaging”, is likely to result in improved performance. This is similar in a way to the basic motivation behind “stacking”, “bagging”, and other averaging techniques (Breiman 1996). It also bears some resemblance to work combining multiple heterogeneous learners to improve overall performance, such as Domingos (1996) and Giraud-Carrier and Martinez (1995).

The temporal difference in Q-values that determines rule learning (i.e., the Bellman residual  $\max_b Q(y, b) - Q(x, a) + r > threshold$ ) encourages taking the actions that may lead to an improvement (increase) of Q-values (i.e., lead to a new state with a higher Q-value and thus a positive temporal difference between the old and the new state). Naturally, these are actions that are worth exploring in their respective states, because they lead to the learning of better Q-values. When learning is on-going, although taking promising actions (i.e., actions with a large temporal difference) is needed, the agent should not ignore those actions that have already been proven to be effective (i.e., those that already have large Q-values). This is achieved through combining the outcomes from the rules and the Q-learning network stochastically: A high Q-value from the bottom level encourages the agent to take the corresponding action.

However, when learning is well under way, an actions leading to a positive temporal difference tend to be an above-average action in a given state, because a positive temporal difference means that the action leads to a new state with an above-average Q-value compared with other new states resulting from other actions. This is because each Q-value tends to be “averaged” by neighboring Q-values (for different actions and states), which is due to network generalization/approximation that tends to “smooth” the output value surface. Because of the selection of actions with above-average Q-values, the temporal difference criterion leads to rules with above-average performance, which means that they

---

state  $x$ ;  $e_t = E[(\max_b Q(y, b) - Q(x, a) + r)^2 * prob(a|x)]$ , where  $prob(a|x)$  is the percentage of the rules matching  $x$  that recommend action  $a$ . So the error rate not only takes into account the Bellman residuals, but also the probabilities of these residuals occurring.

are likely to be optimal or near optimal rules. These rules supplement the approximated Q-values (produced by a function approximating neural network) and compensate for the (over)generalization therein (that is, counterbalancing the “smoothing” tendency; see sections 3 and 4).<sup>9</sup>

Note that the rule learning part of the model is incomplete: it cannot by itself learn to perform temporal credit assignment. To perform temporal credit assignment, some mechanism comparable in power to Q-learning must be used. In CLARION, rule learning is based on Q-learning at the bottom level. But in turn, rules extracted from the bottom level help to improve its performance (as discussed next).

## 3 Experiments

### 3.1 The Task Setting

We tested CLARION on the on-line simulated navigation task as shown in Figure 1. The agent has to navigate an underwater vessel through a minefield to reach a target location. The agent receives information only from a number of instruments. As shown in Figure 2, the sonar gauge shows how close the mines are in 7 equal areas that range from 45 degrees to the left of the agent to 45 degrees to the right. The fuel gauge shows how much time is left before fuel runs out. The bearing gauge shows the direction of the target from the present direction of the agent. The range gauge shows how far the target is from the current location. Based only on such information, the agent decides on (1) how to turn and (2) how fast to move. The time allotted to the agent for each episode is 200 steps. The agent, within an allotted time period, can either (1) reach the target (a success), (2) hit a mine (a failure), or (3) run out of time (a failure again). Each episode starts with the agent on the one side of the minefield and the target on the other. An episode ends when (1) the target is reached, (2) the time runs out (200 steps), or (3) the agent hits a mine. A random mine layout is generated for each episode. The mines are randomly placed between the starting point of the agent and the target. On-line, real-time simulation was used for navigation and learning.

---

<sup>9</sup>However, a Q-value is the statistical average taking into account all the possible resulting states from an action. So a rule may be extracted erroneously due to a fortuitous state transition into one of the better outcome states (which leads to a large temporal difference also). But such rules will be deleted later on, because of the rule revision process using the information gain measure, which takes into consideration the frequencies of such state transitions.

## 3.2 The Model Setup

In CLARION, as input to the bottom level, each gauge was represented by a set of nodes. We tried both discrete and analog input values. In the case of discrete inputs, the following nodes are used for the inputs:

fuel	1 input node
range	2 inputs nodes
bearing	6 input nodes
sonar	4 X 7 input nodes

That is, one node is used for “fuel” (with two values: *a lot* and *a little*), six for “bearing” (including *far left*, *left*, *straight ahead*, *right*, *far right*, and *right behind*), four for each of the seven “sonars” (ranging from *very far* to *very close*), and one for “range” (with two values: *far* and *near*). There are 41 inputs and thus more than  $10^{12}$  states. In the case of analog inputs, each gauge is represented by one node, which takes on continuous values between the highest possible value and the lowest possible value, scaled to within  $[0, 1]$ .<sup>10</sup> We have to deal with the problem of high input dimensionality. A lookup table implementation for Q-learning at the bottom level is not possible, because of the high dimensionality (Tesauro 1992, Lin 1991). A function approximator, such as a backpropagation network, must be used (Bertsekas and Tsitsiklis 1996).

The action outputs in the bottom level consist of two clusters of nodes: one clusters of 5 nodes for 5 different values of “direction” (including *left*, *slightly left*, *straight*, *slightly right*, and *right*), and the other cluster of 5 nodes for 5 different values of “speed” (including *very fast*, *fast*, *normal*, *slow*, and *standstill*).

In the top level, as an alternative to using the same inputs as the bottom level, we also tried using some secondary (derived) features as inputs (cf. Zhang and Dietterich 1995). These features include:

LeftLeastDense	1 input node
CenterLeastDense	1 input node
RightLeastDense	1 input node
LeftFurthestMine	1 input node
CenterFurthestMine	1 input node
RightFurthestMine	1 input node
LeftAvgMineDistance	4 input nodes

---

<sup>10</sup>We compared the performance of these two types of inputs, and found no significant difference.

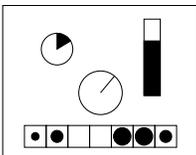


Figure 2: The Navigation Input

The display at the upper left corner is the fuel gauge; the vertical one at the upper right corner is the range gauge; the round one in the middle is the bearing gauge; the 7 sonar gauges are at the bottom.

CenterAvgMineDistance	4 input nodes
RightAvgMineDistance	4 input nodes

Each of the 3 nodes regarding *LeastDense* is binary. The same for the 3 nodes regarding *FurthestMine*. The nodes regarding average distance are calculated from averaging the sonar values on each of the three sides, which results in three node clusters each of which has 4 nodes similar to individual sonar gauge representations. These features were used along with “fuel”, “range” and “bearing” (as before).

11

CLARION’s internal parameters are set through trial-and-error optimization by hand. Seven hidden units are used in the backpropagation network, the momentum parameter is 0.7, network weights are randomly initialized between -0.01 and 0.01. The Q-value discount rate is 0.95. The temperature (randomness parameter) for stochastic decision making is set at 0.01. A schedule for the gradual lowering of learning rates is used:  $\alpha := \alpha_0 * e^{c*s}$ , where  $\alpha$  is the current learning rate,  $\alpha_0 = 0.03$  is the initial learning rate,  $c = 6.0$  is a constant, and  $s$  is the success rate (the percentage of successful episodes during the previous block of 20 episodes). The thresholds for rule extraction and revision are set as follows:  $threshold = 0.06$ ,  $threshold1 = 1.5$ , and  $threshold2 = 0.5$ . To limit the number of rules at the top level, we instituted an upper limit of 100 rules. When more rules are created than allowed, we purge “old” existing rules, according to the rank ordering of rules on the basis of their ages, where the age of a rule is defined to be the number of steps occurred since the last application of the rule. Fixed combination weights for the two levels are used:  $w_1 = 0.2$  and  $w_2 = 0.8$ .

The reinforcements for an agent are produced from two sources. One is the gradient reward, which is proportional to the change in the “range” readings (i.e, the change in the distance to the target).

<sup>12</sup> The other is the end reward, which is determined by how successful the agent is at the end of an

---

<sup>11</sup>Although the secondary feature set led to more comprehensible rules, they had no significant impact on learning performance. Therefore, in the following discussion of experiments, we do not distinguish between the primary and the secondary feature sets.

<sup>12</sup>When the agent is going toward the target, the reinforcement is  $gr = 1/c * ((x_2 - x_1)/x)^4$ , where  $c = 7.5$ ,  $x_2 - x_1$  is

episode. The end reward is 1 if the agent reaches the target within the allotted time, and is inversely proportional to the distance (from the target) if the agent runs out of time or gets blown up.<sup>13</sup>

### 3.3 Results

**Learning speed.** Figures 3, 4, and 5 show the data of CLARION (using the analog inputs in the bottom level and the secondary features in the top level). In terms of learning effectiveness, which is measured by the number of successful episodes out of a total of 1000 episodes of training (averaged over 10 runs), the “training” columns of these figures show the difference between CLARION and the bottom level alone (trained with pure Q-learning). It appears that at higher mine densities (that is, the more difficult settings), CLARION is significantly better compared with the bottom level alone. In the 30-mine and 60-mine cases, the superiority of CLARION (over the bottom level alone with Q-learning only) is statistically significant (with  $t$  tests,  $p < 0.05$ ). However, the performance is statistically undifferentiable in the 10-mine case. Figure 6 shows the learning curves during the course of training, in which each data point is the average of the percentages of successful episodes in a 20-episode block.

**Transfer.** The right three blocks of Figures 3, 4, and 5 show the transfer data, where transfer is measured by the percentage of successful episodes in the new setting by the trained models (each trained model is applied to minefields that contain a *different* number of mines for a total of 20 episodes; the data is averaged over 10 runs). The data generally follows the pattern that the higher the mine density is, the lower the success rate is. Moreover, the performance of a model is generally better if it is trained at a higher mine density (probably because if it is trained at a high mine density, it tends to learn strategies that are more suitable for high-density minefields). As also indicated by the tables, CLARION outperforms the bottom level alone (trained with Q-learning) in transfer at higher mine densities; the higher the mine density, the more pronounced the difference. The differences are statistically significant in the 30-mine and 60-mine cases (using  $t$  tests,  $p < 0.05$ ). Finally, comparing the transfer performance of the top level, the bottom level, and the whole system (after they are trained together), we notice that the whole system always performs much better than either level alone. There is definitely a synergy between the two levels (in the sense that the whole system performs better than either levels). Learning rules does help to improve the transfer performance.

---

the distance traveled in the target direction in one step, and  $x$  is the maximum distance possible (which is 40). When the agent is going away from the target, the reinforcement is  $gr' = -0.5gr$ .

<sup>13</sup>When the agent runs out of time, the reinforcement is  $er = 500/(500 + x) - 1$ , where  $x$  is the distance to the target. When the agent gets blown up, the reinforcement is  $er = 1/2 * 500/(500 + x) - 1$ .

Mine Density During Training: 10

Model	Training	Both	Bottom	Top	Both	Bottom	Top	Both	Bottom	Top
	10	10	10	10	30	30	30	60	60	60
CLARION	651.8	63.5	6.5	4.5	35.5	1.5	0.5	11.5	1.0	0.0
s.d.	31.3	34.4	4.5	6.9	21.0	2.3	1.5	9.5	2.0	0.0
Q	645.7		82.0			42.0			14.5	
s.d.	86.9		14.2			24.5			18.1	

Figure 3: Learning and transfer from 10-mine minefields.

$Q$  refers to the bottom level used alone with Q-learning as the sole learning method. *Training* indicates the total numbers of successful episodes during training. The next three blocks contain performance data (in percentage), in three different mine densities (10, 30, and 60) using the trained models with either the top level, the bottom level, or both together.

Mine Density During Training: 30

Model	Training	Both	Bottom	Top	Both	Bottom	Top	Both	Bottom	Top
	30	10	10	10	30	30	30	60	60	60
CLARION	663.8	89.0	5.0	1.0	75.0	7.0	0.0	47.5	2.5	0.0
s.d.	48.4	26.5	3.2	2.0	23.6	5.6	0.0	24.9	2.5	0.0
Q	539.1		77.0			68.0			35.5	
s.d.	105.6		17.5			20.4			20.7	

Figure 4: Learning and transfer from 30-mine minefields.

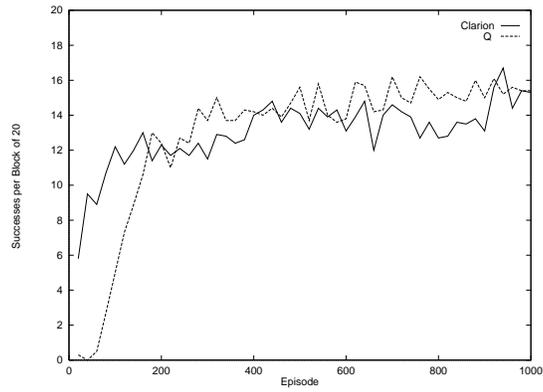
$Q$  refers to the bottom level used alone with Q-learning as the sole learning method. *Training* indicates the total numbers of successful episodes during training. The next three blocks contain performance data in percentage, in three different mine densities (10, 30, and 60) using the trained models with either the top level, the bottom level, or both together.

Mine Density During Training: 60

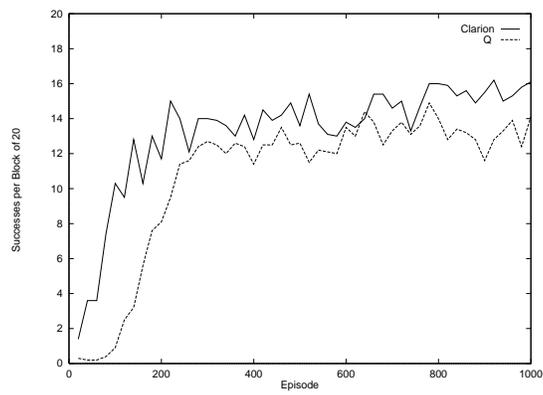
Model	Training	Both	Bottom	Top	Both	Bottom	Top	Both	Bottom	Top
	60	10	10	10	30	30	30	60	60	60
CLARION	581.4	99.5	9.5	2.0	96.0	8.5	0.0	76.0	6.0	0.0
s.d.	79.0	1.5	6.5	3.3	3.7	4.5	0.0	15.9	6.6	0.0
Q	495.8		71.5			67.5			47.5	
s.d.	137.9		11.6			16.8			24.3	

Figure 5: Learning and transfer from 60-mine minefields.

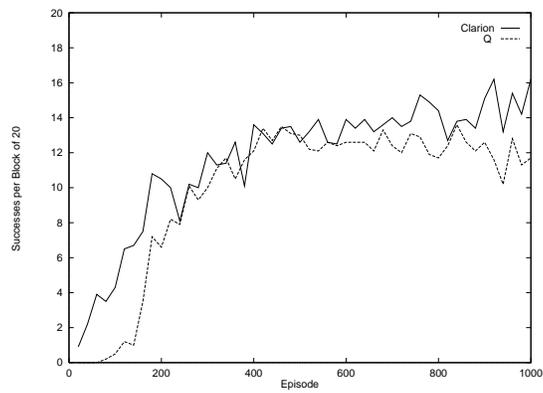
$Q$  refers to the bottom level used alone with Q-learning as the sole learning method. *Training* indicates the total numbers of successful episodes during training. The next three blocks contain performance data in percentage, in three different mine densities (10, 30, and 60) using the trained models with either the top level, the bottom level, or both together.



The 10-mine learning curves



The 30-mine learning curves



The 60-mine learning curves

Figure 6: Learning curves for 10-mine, 30-mine, and 60-mine settings.

**Trained performance.** The right three blocks of Figures 3, 4, and 5 also contain the trained performance data: The 10-mine block in Figure 3 shows the trained performance after training in 10-mine minefields. The 30-mine block in Figure 4 shows the trained performance after training in 30-mine minefields. The 60-mine block in Figure 5 shows the trained performance after training in 60-mine minefields. Trained performance is defined to be the percentage of successful episodes in the *same* setting as used in training by the trained models (each trained model is applied to the minefields for a total of 20 episodes; the data is averaged over 10 runs). At higher mine densities, we notice that the trained performance of CLARION is better than the bottom level alone (trained only with Q-learning). Comparing the performance of the whole system and the two levels separately after the two levels are trained together, we again notice that the whole system performs much better than the bottom level and the top level alone, which strongly suggests a synergy between the two levels. See Appendix for some examples of the rules learned at the top level.

### 3.4 Traces

For the minefield navigation task, we plotted a few snapshots of trajectories. Since random mine layouts were used during training (with repetitions though), the best way to show the progress of an agent through training is to select those episodes of the agent that operate on a same mine layout (which occurred actually relatively regularly during the training sessions). We thus selected a total 18 episodes of an agent on the same mine layout (see Figure 7, 8). Progressively, as shown in the figures, the agent went from being completely random, to following a reasonable path to the goal, to eventually following an almost optimal path to the goal. The progress, however, was not strictly monotonic. Regressions back to earlier worse behaviors sometimes occurred, but eventually a better behavior usually resulted from further training.

## 4 Analyses

### 4.1 Sources of Power

One possible objection to the apparent advantages of the model is that the improvements exhibited in experiments may be purely due to the high randomness in the Q-learning network at the bottom level. To counter the objection, we tried different temperatures ( $\tau = 0.1, 0.01, 0.001$ ). The results indicated that CLARION invariably outperformed the bottom level alone (with Q-learning) at high mine density. See Figure 9, which illustrated in one case the effect of different temperatures. Another point that

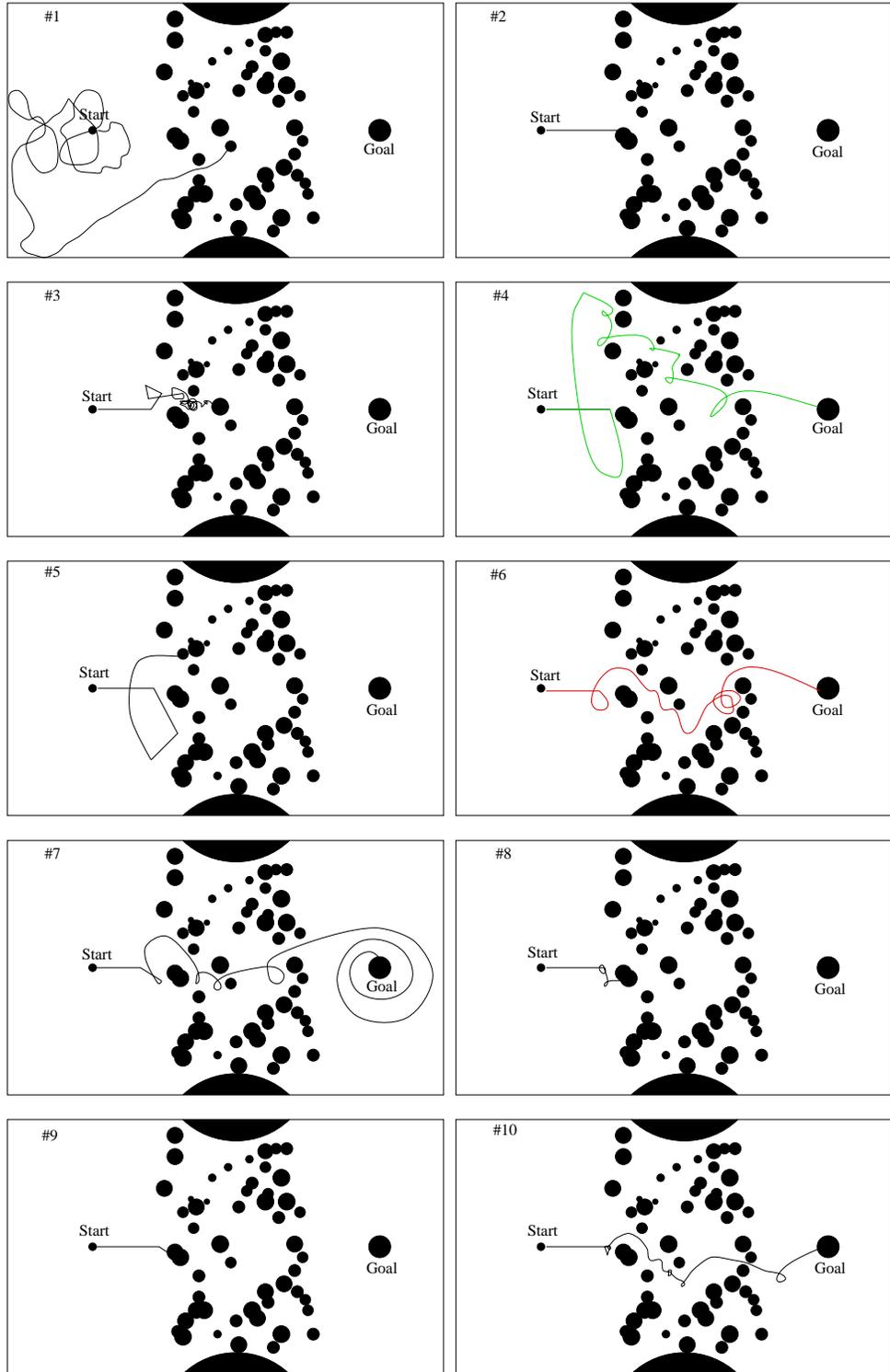


Figure 7: Sample trajectories during learning.

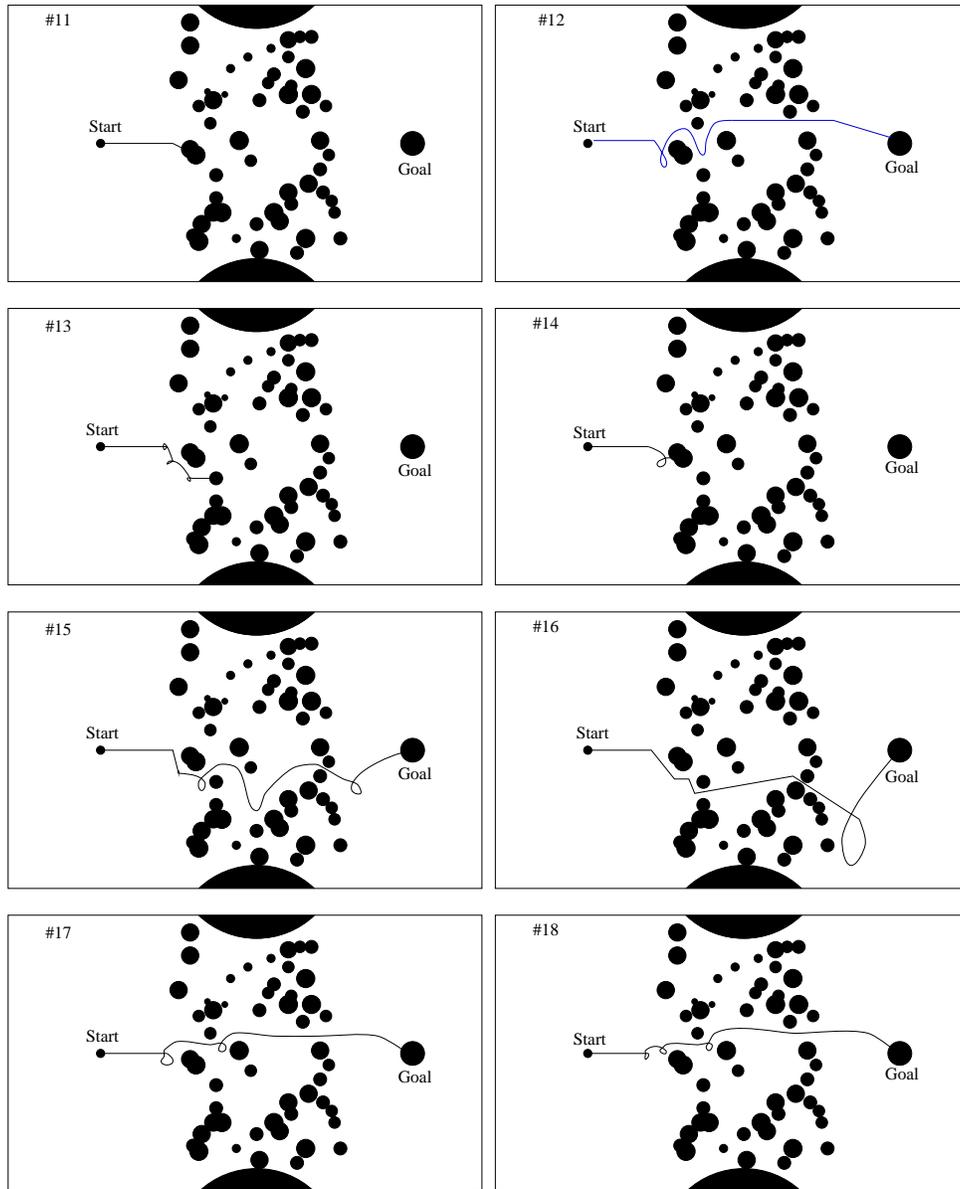


Figure 8: Sample trajectories during learning (continued).

temperature	Q performance	s.d.	CLARION performance	s.d.
0.05	50.8	69.3	361.8	56.9
0.01	439.8	137.9	581.4	79.0
0.001	77.2	114.3	423.3	56.2

Figure 9: Performance during training using different temperatures. Performance is measured by the number of successful episodes out of 1000 training episodes. The mine density is 60.

can be made is that it appeared that in this domain, stochastic policies (i.e., using a non-zero  $\tau$  value) were usually better than deterministic policies (i.e., choosing always the actions corresponding to the maximum Q-values instead of using a Boltzmann distribution). After training (with a non-zero temperature, which is a necessity for training), comparing the performance of the two types of policies, we found that on average stochastic policies outperformed deterministic policies.

Another possibility was that the “direct” action representation at the top level, which is supposedly different from the Q-value representation at the bottom level, might be the source of power. But this cannot be true because Q-value representation can be reduced to action representation if only one action has a maximum activation value and all other values are zero or near zero.

We tested to see if gradient reward given was of great help. This type of reward is to encourage the agent to go forward, rather than keeping going around in the same neighborhood. It turned out that it sped up learning slightly in most cases. The improvements, however, were not significant.

We wondered if the source of power of the model could be attributed to the absence of hidden units in the top level. We tested to see if removing hidden units at the bottom level would improve the performance. Our results indicated that, with hidden units removed from the bottom level, the model as a whole or the bottom level alone had great difficulty in learning the tasks (see also Tesauro 1992). Thus, simply removing hidden units from the bottom level cannot be the source of performance improvement. In relation to this, we also tested to see if making activation functions of the nodes at the bottom level more discrete (by increasing the steepness of the sigmoidal function) and adding a threshold term (as in the top level) could improve the performance. Our results indicated otherwise. On the other hand, when we added hidden units to the top level and made the nodes there less discrete (i.e., turning the top level into a backpropagation network), the performance of the model deteriorated, which indicated the importance of discrete localist representations at the top level. Taken together, the results indicated the need for complementary representations in the two levels.

We also explored the possibility that an insufficient number of hidden units used in the bottom level led to the improvement of performance when the top level was added, because the top level supplemented the bottom-level capacity. We tried different numbers of hidden units: 7, 30, 60, 120,

# of hidden units	Q performance	s.d.	CLARION performance	s.d.
7	362.0	84.6	566.8	49.5
15	281.2	100.2	564.0	44.2
30	223.2	113.9	605.8	34.9
60	202.2	80.9	609.2	23.9
120	314.2	114.6	641.6	36.8
180	261.8	151.4	640.8	53.3

Figure 10: Performance during training using different numbers of hidden units in the bottom level. Performance is measured by the number of successful episodes out of 1000 training episodes.

and 180. We found that with increasingly more hidden units, there was little performance improvement when the bottom level was used alone, and some improvement when the whole system was applied together. See Figure 10, which illustrated in one case the effect of different numbers of hidden units. This again confirmed the advantage of CLARION’s two-level architecture.

We tested to see if one-shot rule learning was the source of power. We compared different frequencies of rule learning. A readiness parameter *ready* determines how many times a rule extraction/revision criterion (as listed before) has to be satisfied before a rule can be extracted/revise. We varied *ready* from 0 through 4, for either extraction or revision (or both). The results indicated there was no significant performance difference. However, when we extended *ready* to a much higher value, the performance of the model deteriorated significantly. This indicated that one-shot rule learning might be a partial explanation for the model advantages.<sup>14</sup>

We also tested to see if the rule learning criterion was the source of the power. We tried two other criteria: the amount of direct reward received by an agent at each step (that is, “if  $r > \text{threshold}$ ”), and the maximum Q-value at a state (that is, “if  $Q(x, a) > \max_b Q(x, b) - \epsilon$ ”). The former criterion is an indication of whether or not an action taken in a given state is *directly* beneficial, but it fails to take into account sequences of actions. The latter criterion concerns whether the Q-value of an action is close enough to the maximum Q-value in that state, indicating the optimality of the action. Our experimental results showed that adopting either of these two criteria lead to significantly worse performance.

In sum, the power of the model can be attributed to the following factors: (1) the complementary representations of the two levels: discrete vs. continuous; (2) the complementary learning processes: one-shot rule learning vs. gradual Q-value tuning; and (3) the proper rule learning criterion based on

---

<sup>14</sup>Note that such rule learning cannot be done independently because the bottom level provides the criteria for rule learning based on temporal credit assignment and before proper discrete rules are learned, it also provides an approximate guide for actions (resulting from generalization in the backpropagation network), which is necessary given the huge state space.

version	performance	s.d.
CLARION	581.4	79.0
CLARION with EP	476.9	96.9
Q	495.8	137.9
Q with EP	239.6	123.0

Figure 11: Performance using playback. Performance is measured by the number of successful episodes out of 1000 training episodes. The mine density is 60.

version	performance	s.d.
CLARION	581.4	79.0
CLARION with EM	633.8	170.7
Q	495.8	137.9
Q with EM	575.2	34.4

Figure 12: Performance using episodic memory. Performance is measured by the number of successful episodes out of 1000 training episodes. The mine density is 60.

Bellman residuals as used in CLARION.

## 4.2 Additional Techniques

We tested a number of other techniques in the literature. Episode playback (Sutton 1990, Lin 1992) involves training the Q-learning network with previously encountered episodes, played in a backward direction in their entirety, done between running new episodes. The reason for doing this is to fully utilize information available from previous experience, to speed up learning. Playback was tested using an exponentially decreasing updating rate in the backward direction; in a way, this is similar to the idea of TD( $\lambda$ ) (updating using eligibility traces; Sutton 1988 and Singh and Sutton 1996). Playback was also tested with a constant updating rate. We found that in either case, it did not seem to be of help. See Figure 11.

Episodic memory is an alternative technique for speeding up learning, whereby at each step a set of previous decision steps (each including the state, the action, the new state, and the reinforcement) are randomly selected and used to train the Q-learning network at the bottom level (Sutton 1990, Lin 1992). Episodic memory did speed up learning, when applied either to the bottom level or to the CLARION model. CLARION still outperformed the bottom level at higher mine densities (more difficult settings). See Figure 12.

As an alternative to the “rule plus neural network” architecture of CLARION, we tried to combine a decision tree and a Q-learning network. That is, instead of rule learning at the top level, we used C4.5 (Quinlan 1986) to learn a decision tree, which was then combined with the output of the bottom level (as before). We periodically re-learned the decision tree using a set of newly accumulated instances of

mine density	C4.5+Q performance	s.d.	CLARION performance	s.d.
60	252.0	102.0	581.4	79.0
30	403.6	148.9	663.8	48.4
10	553.8	83.5	651.8	31.3

Figure 13: Performance of combining decision trees and Q-learners. The performance is measured by the number of successful episodes out of the 1000 training episodes.

decision steps (each including the state, the action, the new state, and the reinforcement, randomly selected). These instances were first determined as either positive or negative, depending on the positivity measure (as explained before). Then C4.5 was applied to the positive instances to learn a tree (using actions as classifications; Quinlan 1986). We varied the parameters, including frequency of tree induction, number of instances used, instance selection criterion, and combination parameters. The results of this alternative architecture were invariably worse than CLARION. See Figure 13 for a set of data. One explanation for the worse performance is that while a decision tree is forced to cover all possible cases (states and actions) (Quinlan 1986), rule learning in CLARION is limited to those cases for which the model has sufficient experience (which leaves the bottom level to handle the other cases using its generalization ability). Thus rules in CLARION are more reliable, which lead to better performance. Evidently, generalization in C4.5 is not well suited for sequential decision tasks, compared with that in backpropagation networks (used in the bottom level).

We also tried partitioning methods in which a state space is divided into several regions each of which is handled by a different learner. Gating (Jacobs et al 1991, Jordan and Jacobs 1994) is one of these techniques, popular in supervised learning. It involves using a linear gating network to assign inputs to different learners. We adapted it to sequential decision tasks (reinforcement learning). The result we obtained showed that its performance was no better than CLARION. Another possibility is to combine the results of multiple Q-learners each of which is trained over the entire state space. The combination can be either through straight averaging or weighted averaging, whereby weights are determined by performing (on-line) gradient descent on the error of the combined outcome. These learners can be diversified through using different parameter values (such as different initial weights, different number of hidden units, different learning rates, etc.), which has been shown to have some advantages in terms of obtaining better combined outcomes (Breiman 1996, Sun and Peterson 1997, 1998). Our results showed that while averaging and weighted averaging achieved better average performance than the bottom level alone (with Q-learning), performance was not better than CLARION. We also tried a form of the pocket algorithm (Gallant 1988) and did not observe significant improvement in performance.

In all, compared with these existing techniques reported in the literature, CLARION is no worse

than any of them, and often better.

### 4.3 Analyzing the Role of Rules

To see how rules and Q-values combine and complement each other, let us look into a set of input states and their corresponding rules and Q-values (after 1000 episodes of training). (Because of the high dimensionality of the input space, we cannot possibly examine all the states; that is why we choose a few representative states to illustrate our points.)

As shown in Figure 14, in some runs, it turned out that the Q-values were grossly inadequate in guiding the behavior of the agents. For almost all of the states shown in the figure, the same action was recommended from the Q-values, because the same action had the highest Q-value throughout these states. The recommended action was “go straight”, which was certainly incorrect in most of these states (that is, it was far from being optimal in these states), as was verified by trying out the action in the corresponding input states (for a description of these input states, see Figure 16). This suboptimal and uniform action recommendation was the result of function approximation in the backpropagation network used for computing the Q-values. Due to over-generalization, the same action recommendation was produced corresponding to all of these input states, which was certainly inappropriate.

However, somewhat surprisingly, the whole system behaved essentially correctly, with a success rate of 60%. The key to resolving this paradox lied in the rules learned by the model, which produced correct action recommendations for many of these states. So when the outcomes from the rules (marked as R-values in Figure 14) were combined with Q-values, correct action recommendations were produced for most of these input states (as was verified by trying out the actions in the corresponding input states; see Figure 16). This led to the 60% success rate by the whole system.

As shown in Figure 15, however, on a different run (with different parameter settings), the Q-values were basically correct for most of the states shown in the figure (although some of them might not be optimal). In this case, good rules were also learned. They were not complete though; that is, they did not cover many of the input states (as shown in Figure 15). This was because there was no need for a more complete coverage by rules, since Q-values were adequate for most of these states. When the outcomes from the rules were combined with Q-values, the results were correct action recommendations for most of these states. Comparable performance (around 60%) was produced by this run too.

Overall, these two cases demonstrated two possible scenarios with regard to the combination of

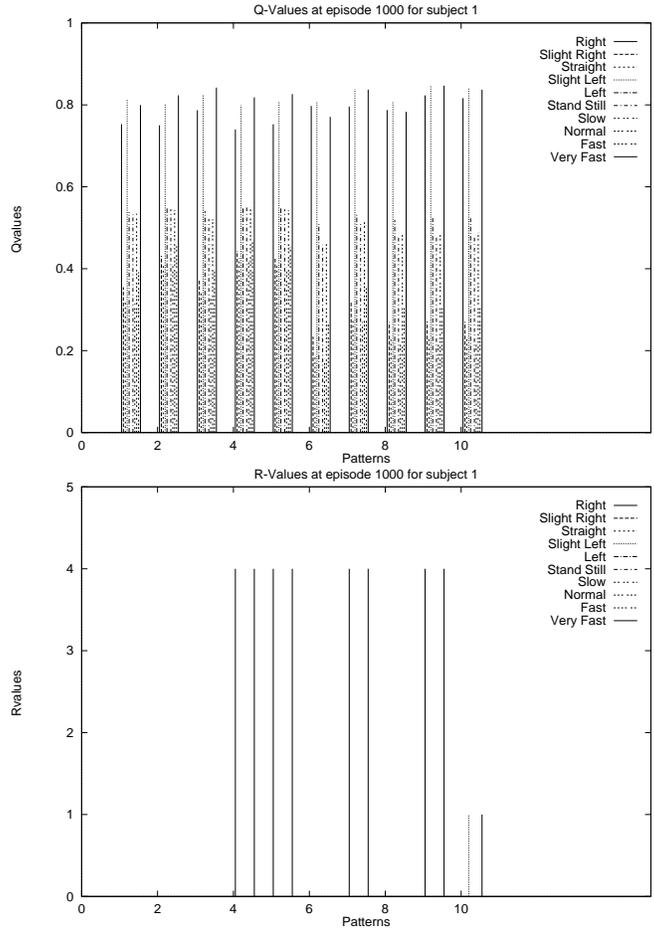


Figure 14: Q-values and R-values: complementarity.

where R-values refer to the outputs (action recommendations) of rules, with each value measuring the number of rules making a particular action recommendation. For the specification of the input states, see Figure 16.

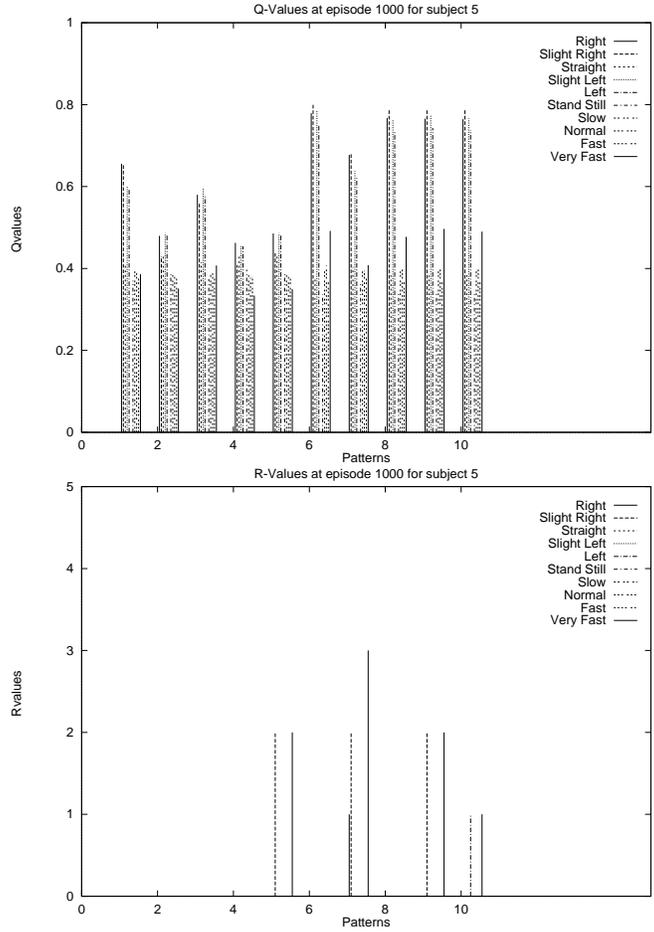


Figure 15: Q-values and R-values: an incomplete rule set.

where R-values refer to the outputs (action recommendations) of rules, with each value measuring the number of rules making a particular action recommendation. For the specification of the input states, see Figure 16.

Input State	Bearing	Sonar Values
1	9	31 35 60 90 95 99 99
2	11	52 49 34 30 27 26 27
3	11	71 76 81 78 67 50 46
4	12	05 05 05 05 09 09 86
5	12	40 41 41 48 40 35 33
6	6	99 99 99 99 99 99 99
7	1	20 99 99 99 99 99 99
8	9	99 99 99 99 99 99 99
9	12	92 94 98 99 99 99 98
10	11	99 99 99 99 99 99 99

Figure 16: Descriptions of some input states.

Bearing values use clock positions. Sonar values are as follows: 99 = open; 0 = mines at the current position.

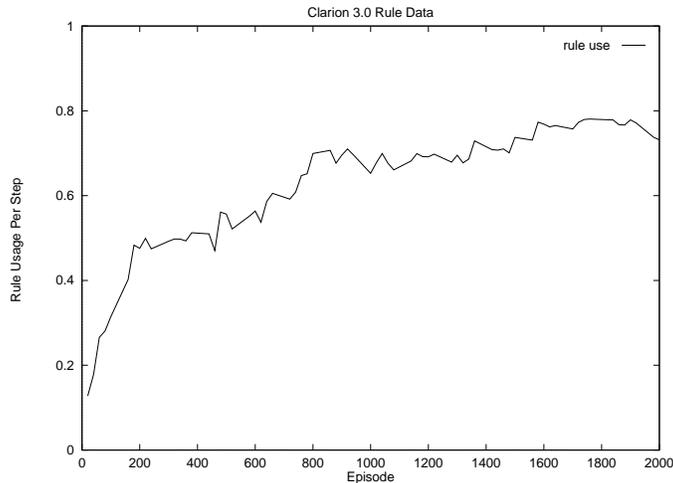


Figure 17: The increase of rule use over the course of learning.

Q-values and rules: they can be either complementary to each other as in the first case, or they can be redundant with both accomplishing the same result, as in the second case. In general, we should expect a combination of the two cases.

#### 4.4 Analysis of Rule Uses

The use of rules increases over the course of learning, as indicated by Figure 17. Here the rule use is defined to be the percentage of steps in which there exists a rule at the top level that matches the action decision made by the whole CLARION model at that step. As shown in the figure, this measure increases steadily, which indicates the gradually increasing role the top level plays over the course of learning, and to some extent, the usefulness of the top-level rules. Our data also indicates the stabilization of rules at the top level: while the use of rules increases, the rates of rule changes (i.e., the extraction, deletion, expansion and shrinking of rules) decrease steadily, as shown in Figure 18. This result points to an eventually relatively stable set of rules at the top level of CLARION, although clearly the contents of the eventual rule sets vary from one run to another (see the analysis in the previous subsection).

In an effort to further verify the role of rules in CLARION, after the training, we tested the resulting rules (at the top level) and the accumulated instances of decision steps (accumulated during training, each including the state, the action, the new state, and the reinforcement), using a decision tree algorithm C4.5 (Quinlan 1986). We induced one decision tree based on all the rules in CLARION,<sup>15</sup> and another based on a subset of randomly selected instances of decision steps (half of all the

<sup>15</sup>In inducing the tree, rules were viewed as classifications, with recommended actions being viewed as class labels.

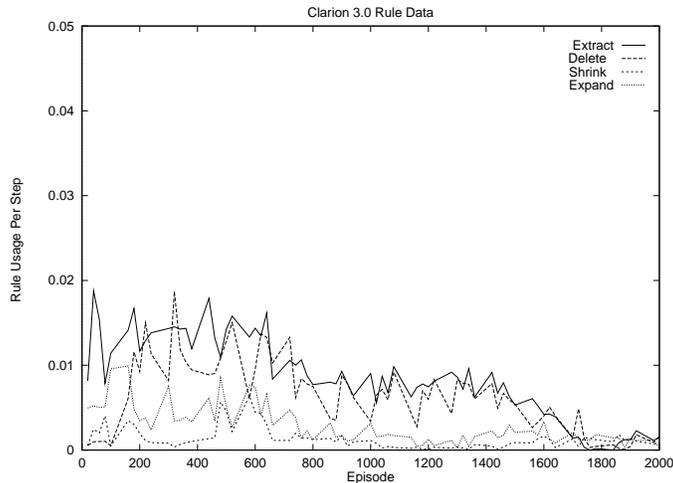


Figure 18: The gradual decrease of rule extractions and revisions (expansion, shrinking, and deletion) over the course of learning.

source of tree	tree size (# of leafs)	fit (correct classification)	validation fit
rules	25	94.84%	89.15%
instances	123	64.26%	91.13%

Figure 19: Comparisons of two trees resulting from rules and instances respectively.

accumulated instances).<sup>16</sup> We then applied both trees to the remaining half of the accumulated instances for validation. As shown in Figure 19, we found that rules led to better trees, in that the tree resulting from rules was more compact, and fit better the data from which it was generated. We also found that during the validation test (that is, the test on the remaining instances), the two trees produced comparable results, even though the tree resulting from the rules was much smaller. This result suggests that the learned rule set (in an informal sense) has a higher informational content (i.e., being more consistent). A sample set of rules are included in Appendix. A decision tree induced by C4.5 is also included in Appendix.

## 4.5 Complexity

Let us look into the time complexity of CLARION, in comparison to pure Q-learning at the bottom level. Assume that the size of the input state space is  $s$ , the size of the encoding of an input state (the number of bits) is  $i$  (which is  $\geq O(\log s)$ ), the size of the action space is  $a$ , and the number of hidden units is  $h$  (which can be safely assumed to be  $\leq i + a$ ). Assume that the maximum number

<sup>16</sup>The instances were first determined as either positive or negative, based on the positivity measure explained earlier. Then C4.5 was applied to the positive instances to induce a tree, using the actions performed as their respective classifications.

of rules is  $r$ , and the maximum number of premises in each rule is  $c$  (which is  $\leq i$ ). At each step, Q-value calculation takes approximately  $i * h + h * a$  steps (for activation propagation along two sets of weights), and thus is in the order of  $O(\max(i^2, a^2))$  in relation to the network sizes. On the other hand, at the top level, rule activation takes at most  $r * c$  steps (for comparing each premise of each rule to the input state) and thus is in the order of  $O(r * i)$ . Empirically we found that limiting  $r$  to be  $i^2$  is sufficient. Thus, rule activation takes polynomial time  $O(i^3)$ , close to Q-value calculation.

In terms of learning, at the bottom level, Q-learning (with backpropagation) takes approximately  $i * h + h * a$  steps (for updating each of all the weights), and thus is in the order of  $O(\max(i^2, a^2))$ , in relation to the network sizes. At the top level, for rule induction, updating all the relevant statistics takes  $\leq r * c$  steps. Revising rules based on these statistics at each step takes time linear with regard to  $r * c$ . So the total time taken for learning rule is  $O(r * c) = O(i^3)$ , which is close to Q-learning.

Space-wise, given the afore-mentioned assumptions, the top level takes  $r * c$  spaces for storing rules, and  $(r * c) * c$  spaces for storing (temporarily) all possible revisions of the currently active rules, and thus its space complexity is in the order of  $O(i^4)$ . The bottom level (the backpropagation network) takes  $i * h + h * a$  weight storage spaces (for storing two sets of weights), and thus its space complexity is in the order of  $O(\max(i^2, a^2))$ .

## 5 Related Work

**Hybrid Models.** Hybrid models (subsymbolic and symbolic, subconceptual and conceptual, reactive and deliberate, and so on) have become popular. These models include Hendler (1987), Gelfand et al. (1989), Miikkulainen and Dyer (1991), and Sun (1992, 1994, 1995). See Sun and Bookman (1994) for an overview.

Some hybrid connectionist models try to implement all types of knowledge (symbolic and subsymbolic) in one particular kind of network or another; for example, Shastri and Ajjanagadde (1990), Barnden (1988), and Sun (1992b) try to implement knowledge in localist networks, and Miikkulainen and Dyer (1991) and Hinton and Touretzky (1986) try to implement knowledge in distributed networks. CLARION, among others such as Sun (1995) and Gelfand et al (1989), takes a different tack and attempts to develop a principled dichotomy of the conceptual vs. the subconceptual in architectures. Among those models that incorporate such a dichotomy, some tend to simply juxtapose the two sides of the dichotomy. Instead, CLARION attempts to explore their synergy.

Some existing hybrid models do not or cannot perform learning, such as Shastri and Ajjanagadde

(1990), Sun (1992, 1995), and Barnden (1989), although their representations are more sophisticated. Others perform learning in a batch fashion, such as Miikkulainen and Dyer (1991), although their learning task is quite difficult. Instead, CLARION performs on-line (concurrent) learning (Gelfand et al 1989). CLARION is thus more cognitively plausible and more ecologically realistic in this regard (Nosofsky et al 1994). CLARION is also capable of integrated learning (that is, developing subsymbolic and symbolic representations along side of each other), which is unlike any of the existing models. In addition, most of the existing learning models explore mainly top-down learning (including advice taking), in which externally given declarative knowledge is turned into procedural knowledge through practice, such as Gelfand et al (1989), Maclin and Shavlik (1994), and Anderson (1983). Instead, CLARION explores bottom-up learning, to demonstrate how conceptual/symbolic knowledge can emerge in interacting with the world through the mediation of subconceptual/subsymbolic procedural knowledge.

**Rule Extraction.** Let us compare CLARION also with connectionist rule extraction algorithms. Fu (1991) proposed an exhaustive search based algorithm to extract conjunctive rules from perceptron networks. To find rules, the learner first searches for all the combinations of positive conditions that can lead to a conclusion; then, in the second phase, with a previously found combination of positive conditions, the learner searches for negative conditions that should be added to guarantee the conclusion. In the case of three-layered networks, the learner can extract two separate sets of rules, one for each layer, and then integrate them by substitution. Towell and Shavlik (1993) used rules of an alternative form, the *N-of-M* form: *If N of the M conditions,  $a_1, a_2, \dots, a_M$ , is true, then the conclusion b is true.* It is believed that some rules can be better expressed in such a form, which more closely resembles the weighted-sum computation in neural networks, in order to avoid the combinatorial explosion and to discern structures. A four-step procedure is used to extract such rules, by first grouping similarly weighted links, eliminating insignificant groups, and then forming rules from the remaining groups through an exhaustive search. However, these rule extraction algorithms are meant to be applied at the end of the training of a network. Once extracted, the rules are fixed; there is no modification on the fly, unless the rules are re-extracted (starting anew) after further training of the network. On the other hand, in CLARION, an agent can extract and modify rules dynamically. Connectionist reinforcement learning and rule learning work together simultaneously; thus we utilize the synergy of the two algorithms to improve learning. Extracting and modifying rules dynamically is computationally less expensive because it minimizes the search necessary; in fact, in CLARION, there is no separate search process in addition to neural network learning (such as in Towell and Shavlik 1993 and Fu 1991, the search processes in which are clearly exponential), although CLARION incurs slightly

more cost each step during learning (see the complexity analysis earlier).<sup>17</sup> It also helps the agent to adapt to changing environments by allowing the addition and the removal of rules at any time. In addition, we avoid examining the details of the network from which rules are extracted. Instead we focus on the *behavior* of the network and acquire rules on that basis.

**Rule Learning in AI.** Now let us turn to rule learning methods in traditional AI. Though CLARION learns rules, CLARION tackles tasks different from what is usually dealt with by traditional AI rule learning algorithms. Most of the supervised concept/rule learning algorithms (such as AQ and ID3; Michalski 1983, Quinlan 1986) require consistent data and pre-classification, which are not available to CLARION. As “batch” algorithms, they require the agent to obtain all data before learning starts, which means higher space complexity, slow start in learning, and no “drifting” (without extra mechanisms). They cannot be applied directly to our tasks also because they do not perform temporal credit assignment. There is also the incremental variety of supervised learning (e.g., the version space algorithm of Mitchell 1982 and also Utgoff 1989). Although incremental, they require labeled, complete, and consistent descriptions of instances, which are not available to CLARION. They do not handle sequences either.

Unsupervised rule/concept learning algorithms, such as Lebowitz (1987), Fisher (1987) and Stepp and Michalski (1986), are also unsuitable for our tasks, in that (1) in our tasks, there is feedback available (i.e., reinforcements), although there is no direct supervision; such feedback must be taken into consideration in order to achieve goals; (2) temporal credit assignment is necessary; (3) a complete description of instances on which a system can base its decisions is usually not available.

One might argue that the problem of sequences can be avoided by evaluating the values of states and/or actions statically (in isolation). However, such static evaluation relies on a great deal of a priori knowledge about the task (which changes the nature of learning). In contrast, we assume minimal a priori knowledge in the agent.

Some work, such as Giraud-Carrier and Martinez (1995) and Shen (1993), tried to incorporate a variety of learning methods, which is also what CLARION strives for. However, CLARION tried to incorporate different learning methods in a cognitively principled way (Anderson 1983, Sun 1994).

**Sequential decision making.** Let us review existing methods for sequential decision learning. A good method is reinforcement learning, as in e.g. Sutton (1988), Watkins (1989), Barto, Sutton,

---

<sup>17</sup>In CLARION, to reduce cost, one can stop neural network learning early and obtain a reasonably good (but not necessarily optimal) rule set. In Fu (1991) and Towell and Shavlik (1993), one may also shorten neural network learning, but one cannot shorten the exponential-cost search process for rule extraction.

and Watkins (1990), and Sutton (1990). There are many variations. It is possible that we can adopt some alternative reinforcement learning methods; however, much existing work shows that Q-learning is as good as any other method, if not better (Sutton 1990, Lin 1992, Mahadevan and Connell 1992).

However, reinforcement learning (including Q-learning) is problematic when the input space is continuous or otherwise large, in which case reinforcement learning using table lookup is no longer applicable and neural network implementations (function approximation) are not guaranteed successful learning (Lin 1991, Tesauro 1992). Our experiments and analyses suggested that CLARION helps in this regard, because rules complement the function approximator (i.e., the neural network) by detecting and correcting over-generalization (see section 4.3).

Some reinforcement learning systems, such as Tesauro (1992) and Zhang and Dietterich (1995), succeeded in large domains, comparable to or larger than the minefield navigation task. These models were aimed at practical applications, while our model was motivated by cognitive modeling considerations (i.e., to simulate and understand human cognitive processes, based on the dichotomy of procedural and declarative knowledge; Sun et al 1996, 1997). In addition, these models used some domain-specific techniques (thus not directly applicable to minefield navigation).

Clearly, there are a variety of techniques for speeding up or otherwise helping Q-learning, such as Sutton (1990), Lin (1992), Maclin and Shavlik (1994), McCallum (1996), Moore and Atkeson (1994), Singh (1994), Zhang and Dietterich (1995), and Singh and Sutton (1996). Such techniques include reuse of previous experience (Sutton 1990, Lin 1992), gradual enhancement of state spaces (McCallum 1996, Moore and Atkeson 1994), hierarchical learning (Singh 1994), and construction and use of high-level features (Zhang and Dietterich 1995). Some of these were experimentally tested in CLARION, such as reusing previous experience and constructing derived high-level features (see sections 3 and 4). Some others cannot be compared to CLARION, because they require a large amount of a priori knowledge which CLARION does not require (such as Maclin and Shavlik 1994), or because they are designed to deal with POMDP which is not the goal of CLARION (such as McCallum 1996). What is the relation between CLARION and the speedup techniques in general? The answer is that CLARION is a generic framework for extracting declarative knowledge (rules) from reinforcement learning networks (for the sake of improving overall performance). Thus, the CLARION method can be applied on top of any existing reinforcement learning techniques (including Sutton 1990, Lin 1992, McCallum 1996, and Singh and Sutton 1996, when they are implemented in neural networks) to extract rules (see section 4.2).

There has been some work in combining explanation-based learning and reinforcement learning.

In general, in explanation-based learning, the agent tries to extract specific rules regarding a situation by examining inferences regarding the situation and identifying relevant conditions in the inferences. Gordon and Grefenstette (1992) tried to explain and then generalize reactive behavior, based on a set of given rules (i.e., domain theories). Such models bear some remote resemblance to CLARION. Although they help to speed up task learning, they rely on a priori, externally given knowledge which CLARION does not require.

Let us examine alternatives to reinforcement learning. One alternative for handling sequences is explicit symbolic planning (from traditional AI). From an overall goal, the agent generates a number of subgoals that are to be accomplished in a certain order; then, from each of these subgoals, a number of sub-subgoals are generated, and so on, until each of these goals can be accomplished in one step. This approach is not suitable here, because (1) a substantial amount of a priori knowledge is required, which is not readily available to us; (2) it is unnatural for describing some simple sequential behaviors (Agre and Chapman 1990).

There are also some alternatives from the neural network literature, such as recurrent backpropagation networks. Such networks use hidden nodes as a memory, which represent a condensed record of past states. The hidden nodes are connected recurrently so that the previous states are taken into account in the present. The problem with such networks is that they require supervised learning and therefore are unsuitable to our tasks. There are techniques that get around the problem of requiring teacher input (supervision), but they are generally slower because of the lack of a proper temporal credit assignment mechanism.

Yet another alternative is the genetic algorithm (Grefenstette 1992, Schultz 1991, Meeden 1995). GA is a weak method for knowledge-lean heuristic search. It updates its knowledge mostly based on the acquired experience of an entire generation (each member of which goes through many episodes) at the end of all their trials, not on the individual experience of a step or an episode. Therefore the learning time required by the algorithm is expected to be longer than CLARION.

This work differs from our own previous work. Sun (1994, 1995) dealt with commonsense reasoning, but not sequential decision making. Sun et al (1996, 1997) dealt with the psychological issues in modeling human sequential decision making. Sun and Peterson (1997, 1998) used a different, more primitive algorithm for rule learning and presented experiments in a different domain.

## 6 Concluding Remarks

This paper presents the model CLARION that unifies connectionist, symbolic, and reinforcement learning methods to tackle the learning of sequential decision tasks. It is experience-driven, developing both procedural and declarative knowledge autonomously through exploring the world, in a bottom-up direction. Utilizing a principled dichotomy of procedural and declarative knowledge, CLARION is able, at least in some circumstances, to learn faster, perform better, and transfer more effectively than models that neglect such a dichotomy. Experiments and analyses demonstrate in the three aspects the potential for such advantages. They suggest that the combination of the two types of knowledge can yield synergistic results. Thus the model has a performance advantage, besides being cognitively more plausible.

## A Some Sample Rules

Due to lengths, we will only show a subset of rules (extracted using derived secondary features):

```
Bearing: Straight Ahead
LeastDense: Center
FurthestMine: Center
LeftAvgMineDistance: Close
CenterAvgMineDistance: Very Far
RightAvgMineDistance: Far
Direction: Go Straight, Speed: Very Fast
```

```
Bearing: Straight Ahead
LeastDense: Center and Right
FurthestMine: Center
LeftAvgMineDistance: Close
CenterAvgMineDistance: Very Far
RightAvgMineDistance: Very Far
Direction: Go Straight, Speed: Very Fast
```

```
Bearing: Straight Ahead
LeastDense: Right
FurthestMine: Right
LeftAvgMineDistance: Close
CenterAvgMineDistance: Far
RightAvgMineDistance: Far
Direction: Go Straight, Speed: Very Fast
```

```
Bearing: Straight Ahead, Right, Far Right, Right Behind or Far Left
LeastDense: Right
FurthestMine: Right
LeftAvgMineDistance: Very Close or Close
CenterAvgMineDistance: Close
RightAvgMineDistance: Far
Direction: Turn Right, Speed: Very Fast
```

```
Bearing: Far Left, Left or Right Behind
LeastDense: Right
FurthestMine: Right
```

```

LeftAvgMineDistance: Very Close or Close
CenterAvgMineDistance: Close
RightAvgMineDistance: Far
Direction: Turn Right, Speed: Standstill

Bearing: Straight Ahead
LeastDense: Right
FurthestMine: Right
LeftAvgMineDistance: Very Close
CenterAvgMineDistance: Very Close
RightAvgMineDistance: Very Close to Far
Direction: Turn Slightly Right, Speed: Very Fast

Bearing: Far Left
LeastDense: Left
FurthestMine: Left
LeftAvgMineDistance: Far
CenterAvgMineDistance: Close
RightAvgMineDistance: Close
Direction: Turn Left, Speed: Very Fast

Bearing: Far Right
LeastDense: Left
FurthestMine: Left
LeftAvgMineDistance: Far
CenterAvgMineDistance: Close
RightAvgMineDistance: Very Close
Direction: Turn Left, Speed: Very Fast

```

## B A Decision Tree

```

'RightFurthestMine' = '0':
| 'Bearing' = 'Far Left' : 'Turn Left'
| 'Bearing' = 'Left' : 'Turn Left'
| 'Bearing' = 'Straight' : 'Go Straight'
| 'Bearing' = 'Right' : 'Turn Left'
| 'Bearing' = 'Far Right': 'Turn Left'
| 'Bearing' = 'Behind' : 'Turn Left'
'RightFurthestMine' = '1':
| 'CenterAvgMineDistance' = 'Very Close': 'Turn Right'
| 'CenterAvgMineDistance' = 'Close' : 'Turn Slightly Right'
| 'CenterAvgMineDistance' = 'Average' : 'Turn Right'
| 'CenterAvgMineDistance' = 'Far':
| | 'Bearing' = 'Far Left' : 'Turn Right'
| | 'Bearing' = 'Left' : 'Turn Right'
| | 'Bearing' = 'Straight' : 'Turn Slightly Right'
| | 'Bearing' = 'Right' : 'Turn Right'
| | 'Bearing' = 'Far Right': 'Turn Right'
| | 'Bearing' = 'Behind' : 'Turn Right'
| 'CenterAvgMineDistance' = 'Very Far ':
| | 'LeftAvgMineDistance' = 'Very Close': 'Turn Slightly Right'
| | 'LeftAvgMineDistance' = 'Close' : 'Turn Slightly Right'
| | 'LeftAvgMineDistance' = 'Average' : 'Go Straight'
| | 'LeftAvgMineDistance' = 'Far' : 'Turn Slightly Right'
| | 'LeftAvgMineDistance' = 'Very Far' : 'Turn Right'

```

## References

- J. R. Anderson, (1983). *The Architecture of Cognition*, Harvard University Press, Cambridge, MA
- J. Anderson, (1993). *Rules of the Mind*. Lawrence Erlbaum Associates. Hillsdale, NJ.
- A. Barto, R. Sutton, and C. Watkins, (1990). Learning and sequential decision-making. In: M. Gabriel and J. Moors (eds.), *Learning and Computational Neuroscience*. MIT Press. Cambridge, MA.
- J. Barnden, (1988). The right of free association: relative-position encoding for connectionist data structures, *Proc.10th Conference of Cognitive Science Society*, 503-509, Lawrence Erlbaum Associates, Hillsdale, NJ.
- D. Bertsekas and J. Tsitsiklis, (1996). *Neuro-Dynamic Programming*. Athena Scientific, Belmont, MA.
- L. Breiman, (1996). Bagging predictors. *Machine Learning*, Vol.24, No.2, pp.123-140.
- A. Damasio et al, (1990). Neural regionalization of knowledge access. In: *Cold Spring Harbor Symp. on Quantitative Biology*, Vol.LV. CSHL Press.
- R. Dominowski, (1975). How do people discover concepts?
- P. Domingos, (1996). Unifying instance-based and rule-based induction. *Machine Learning*. 24, 141-168.
- G. Drescher, (1991). *Made-up Minds*. MIT Press. Cambridge, MA.
- H. Dreyfus and S. Dreyfus, (1987). *Mind Over Machine*, The Free Press, New York, NY.
- D. Fisher, (1987). Knowledge acquisition via incremental conceptual clustering. *Machine Learning*. 2, 139-172.
- L.M. Fu, (1991). Rule learning by searching on adapted nets, *Proc.of AAAI'91*, pp.590-595.
- S. Gallant, (1988). Connectionist expert systems. *Communications of the ACM*, 24(2), 152-169.
- J. Gelfand, D. Handelman and S. Lane, (1989). Integrating Knowledge-based Systems and Neural Networks for Robotic Skill Acquisition, *Proc.IJCAI*, pp.193-198. Morgan Kaufmann, San Mateo, CA.
- C. Giraud-Carrier and T. Martinez, (1995). An integrated framework for learning and reasoning. *Journal of Artificial Intelligence Research*. 3, 147-185.
- D. Gordon, et al. (1994). NRL task: navigation and collision avoidance. Naval Research Lab. Washington, DC.
- D. Gordon and J. Grefenstette, (1992). Explanations of Empirically derived reactive plans. *Proc.of Machine Learning Conference*. 198-203. Morgan Kaufmann, San Mateo, CA.
- J. Grefenstette, The evolution of strategies for multiagent environments. *Adaptive Behavior*. 1(1). 65-90.
- J. Hendler, (1987). Marker Passing and Microfeature, *Proc.10th IJCAI*, pp.151-154, Morgan Kaufmann, San Mateo, CA.
- H. Hirsh, (1994). Generalizing version spaces. *Machine Learning*, 17, 5-46.
- W. James, (1890). *The Principles of Psychology*. Dover, New York.
- F. Keil, (1989). *Concepts, Kinds, and Cognitive Development*. MIT Press. Cambridge, MA.

- M. Lebowitz, (1987). Experiments with incremental concept formation: UNIMEM. *Machine Learning*. 2, 103-138.
- L. Lin, (1992). Self-improving reactive agents based on reinforcement learning, planning, and teaching. *Machine Learning*. Vol.8, pp.293-321.
- R. Maclin and J. Shavlik, (1994). Incorporating advice into agents that learn from reinforcements. *Proc.of AAAI-94*. Morgan Kaufmann, San Mateo, CA.
- P. Maes and R. Brooks, (1990). Learning to coordinate behaviors, *Proc.of National Conference on Artificial Intelligence*. pp.796-802. Morgan Kaufmann, San Mateo, CA.
- S. Mahadevan and J. Connell (1992), Automatic programming of behavior-based robot with reinforcement learning. Vol.55, pp.311-365.
- A. McCallum, (1996). Learning to use selective attention and short-term memory in sequential tasks. *Proc. Conference on Simulation of Adaptive Behavior*. 315-324. MIT Press, Cambridge, MA.
- L. Meeden, (1995). An incremental approach to developing intelligent neural network controllers for robots. *Adaptive Behavior*.
- R. Michalski, (1983). A theory and methodology of inductive learning. *Artificial Intelligence*. Vol.20, pp.111-161.
- R. Miikkulainen & M. Dyer, (1991). Natural language processing with modular PDP networks and distributed lexicons. *Cognitive Science*. 15(3). pp.343-399.
- T. Mitchell, (1982). Generalization as search. *Artificial Intelligence*, 18, 203-226.
- A. Moore and C. Atkeson, (1994). The parti-game algorithm for variable resolution reinforcement learning in multidimensional state spaces. *Machine Learning*.
- R. Nosofsky, T. Palmeri, and S. McKinley, (1994). Rule-plus-exception model of classification learning. *Psychological Review*. 101 (1), 53-79.
- R. Quinlan, (1986). Inductive learning of decision trees. *Machine Learning*. 1, 81-106.
- P. Rosenbloom, J. Laird, A. Newell, and R. McCarl, (1991). A preliminary analysis of the SOAR architecture as a basis for general intelligence. *Artificial Intelligence*. 47 (1-3), 289-325.
- A. Schultz, (1991). Using a genetic algorithm to learn strategies for collision avoidance and local navigation. *Proc.of 7th International Symp. on Unmanned Untethered Submersible Technology*. 213-225. U. of New Hampshire, Durham.
- L. Shastri and V. Ajjanagadde, (1990). From simple association to systematic reasoning, Tech Report MS-CIS-90-05, University of Pennsylvania, Philadelphia, PA.
- W. Shen, (1993). Discovery as autonomous learning from the environment. *Machine Learning*. 12, 143-165.
- S. Singh and R. Sutton, (1996). Reinforcement learning with replacing eligibility traces. *Machine Learning*, 7, 1-37.

- P. Smolensky, (1988). On the proper treatment of connectionism. *Behavioral and Brain Sciences*, 11(1):1-74.
- R. Stepp and R. Michalski, (1986). Conceptual clustering. in: R. Michalski et al (eds.), *Machine Learning, II*. Morgan Kaufmann, Los Altos, CA.
- W. Stanley, R. Mathews, R. Buss, and S. Kotler-Cope, (1989). Insight without awareness: on the interaction of verbalization, instruction and practice in a simulated process control task. *Quarterly Journal of Experimental Psychology*. 41A (3), 553-577.
- R. Sun, (1992). On Variable Binding in Connectionist Networks, *Connection Science*, Vol.4, No.2, pp.93-124.
- R. Sun, (1994). *Integrating Rules and Connectionism for Robust Commonsense Reasoning*. John Wiley and Sons, New York, NY.
- R. Sun, (1995). Robust reasoning: integrating rule-based and similarity-based reasoning. *Artificial Intelligence*. 75, 2. 241-296.
- R. Sun, (1997). Learning, action, and consciousness: a hybrid approach towards modeling consciousness. *Neural Networks*, special issue on consciousness. 10 (7), pp.1317-1331.
- R. Sun and L. Bookman, (eds.) (1994). *Computational Architectures Integrating Neural and Symbolic Processes*. Kluwer Academic Publishers. Norwell, MA.
- R. Sun and T. Peterson, (1997). A hybrid model for learning sequential navigation. *Proc. of IEEE International Symposium on Computational Intelligence in Robotics and Automation (CIRA '97)*. Monterey, CA. pp.234-239. IEEE Press.
- R. Sun and T. Peterson, (1998). Some experiments with a hybrid model for learning sequential decision making. *Information Sciences*.
- R. Sun, T. Peterson, and E. Merrill, (1996). Bottom-up skill learning in reactive sequential decision tasks. *Proc. of 18th Cognitive Science Society Conference*, Lawrence Erlbaum Associates, Hillsdale, NJ. pp.684-690. 1996.
- R. Sun, E. Merrill, and T. Peterson, (1997). Skill learning using a bottom-up hybrid model. *Proc. of The First International Conference on Cognitive Science*, (Seoul, Korea. August 15-16, 1997.) pp.146-251.
- R. Sutton, (1990). Integrated architectures for learning, planning, and reacting based on approximating dynamic programming. *Proc. of Seventh International Conference on Machine Learning*. Morgan Kaufmann. San Mateo, CA.
- R. Sutton and A. Barto, (1981). Towards a modern theory of adaptive networks: expectation and prediction. *Psychological Review*, Vol.88, No.2, pp.135-170.
- T. Tesauro, (1992). Practical issues in temporal difference learning. *Machine Learning*. Vol.8, 257-277.
- D. Touretzky and G. Hinton, (1987). Symbols among neurons, *Proc. 9th IJCAI*, pp.238-243, Morgan Kaufman.
- G. Towell and J. Shavlik, (1993). Extracting Refined Rules from Knowledge-Based Neural Networks, *Machine Learning*. 13 (1), 71-101.
- P. Utgoff (1989). Incremental induction of decision trees. *Machine Learning*. Vol.4, 161-186.

C. Watkins, (1989). *Learning with Delayed Rewards*. Ph.D Thesis, Cambridge University, Cambridge, UK.

D. Willingham, M. Nissen, and P. Bullemer, (1989). On the development of procedural knowledge. *Journal of Experimental Psychology: Learning, Memory, and Cognition*. 15, 1047-1060.

J. Zhang and T. Dietterich, (1995). A reinforcement learning approach to job-shop scheduling. *Proc of International Joint Conference on AI*, 1114-1120. Morgan Kaufmann, San Francisco.