City University

MSc in Artificial Intelligence

Project Report

2005

Associative Reinforcement Learning

Niclas Kjäll-Ohlsson

Supervised by: Dr. Eduardo Alonso

September 30, 2005

Abstract

A gap has been identified between Reinforcement Learning, which is based on early theories of animal conditioning, and current theories of conditioning from the field of Associative Learning in Psychology. In this thesis an attempt is made to improve an instance of Reinforcement Learning, called Q-Learning, by decreasing this theoretical gap on some accounts. Specifically, the Rescorla-Wagner equation for classical conditioning is utilized and combined with the functional framework of the Q-learning algorithm in order to achieve generalization.

Acknowledgement

Thanks to Dr. Esther Mondragón, expert on animal learning and behavior, at University College London, for providing help on associative learning theories and experimental design.

Contents

Chapter 1

Introduction

1.1 Hypothesis and aim

The aim of this project is to improve an instance of reinforcement learning, called Q-learning, on some accounts by incorporating theories of classical and instrumental learning from Psychology, by extending the underlying framework (Markov Decision Processes) and amending the way in which learning is taking place in such a framework. The hypothesis is thus stating that Q-learning can be improved by incorporating theories of animal learning from Psychology in the framework in which it is functioning.

1.2 Motivation

In order to set the stage for the motivation behind the aim, the identified problems have been briefly outlined below. It is not the purpose to explain these problems in detail in the introduction, as a further and elaborate account of the problems will be given in the literature survey. However, the reader will at least know what to expect from subsequent chapters, as the motivation behind the aim is manifested in the problems that the work behind this report have been trying to solve. In the general case, these problems refer to the goal of maximizing a reward over time by exhibiting sequences of actions that result as a consequence of this goal. Additionally, this problem carries with it other problems that are due to long-term vs. short-term reward maximization, generalization of learned action sequences to other environments, and learning in large and complex environments. Despite the apparent success of reinforcement learning techniques in many domains (e.g. [Samuel 1967, Tesauro 1995, Singh & Bertsekas 1997]), the problems remain and prohibit their application to large state spaces [Alonso & Mondragón 2005]. The following problems have not been extensively remedied in a consistent manner:

1. **Exploitation-exploration equilibrium:** Finding the right balance between the exploitation of actions that are known to yield reward, versus the exploration of

actions that might turn out to give a higher reward. This is a problem of keeping an ”open mind” towards potentially better strategies.

2. **Temporal discounting:** Modelling long term goals versus short term goals through use of reward discounting. How long into the future should an agent consider its well being? If long term goals are most important then this might slow down learning because of the many possible action sequences. Conversely, the outweighing of long term goals in favor of short term goals, may result in suboptimal strategies.

3. **Generalization:** Learning is dependent on the reward structure, and so therefore the learning has to start over when presented with a new problem. Additionally, situations, (also known as stimuli), are considered as irreducible entities in reinforcement learning, which hinder the transition of learning to similar situations.

4. **Large sized problems:** In order to learn the best strategy in an environment, all situations have to be visited repeatedly *ad infinitum*. Convergence to an optimal strategy is prohibited by large state spaces, due to the requirement of repetitive visits to all situations. This is also a problem of generalization. Quite often will different, but similar situations reoccur at later times in an environment. By merely considering situations as irreducible entities, the knowledge will fail to transfer to these new, but similar situations.

Reinforcement learning is based on early theories of instrumental learning, specifically the ”Law of effect” as brought forward by [Thorndike 1911]. This theory states that an association between a stimulus, (a.k.a situation/state), and an action is strengthened if followed by a positive outcome. As this theory has been proven inadequate [Mackintosh 1983], because it fails to account for the fact that the animal is expecting the outcome, there seems to be room for improvement in the reinforcement learning framework [Alonso & Mondragón 2004].

1.3 Objectives

Specifically, the following objectives have been identified as crucial in answering to the aim and making the hypothesis testable, as well as possibly helping to solve the aforementioned problems.

1. To incorporate psychological bias in the architecture of the agent that will guide its learning process. Internal drives will be introduced, that will make the agent approach appetitive stimuli and avoid aversive stimuli. Additionally, exploration will be introduced as a specific type of drive. This is an idea of abandoning the ”tabula rasa” approach of learning which is employed in reinforcement learning.

2. To endow the agent with the ability to form other types of association S-R associations. Associative theory envisages three types of association:

    - Stimulus-Response (S-R) associations.

- Stimulus-Stimulus (S-S) associations.
- Response-Outcome (R-O) associations.

3. To take into account associative theory's conception of event representation.
4. To redefine outcomes as comprising sensorial and motivational elements.
5. To take into account the fundamental conditions of association formation proposed by associative theory.

1.4 Research questions

It is contended that the aforementioned objectives will help solve the problems that have been identified with reinforcement learning. The research questions are thus specified in light of these problems:

1. **Exploration-exploitation equilibrium:** By considering exploration as a drive, S-S associations will form, allowing the agent to form a model of the environment, which in turn will help to discard unsuccessful exploratory policies.
2. **Temporal discounting:** Factors other than the immediate temporal contiguity between events or actions and their outcomes are integrated in the learning structure, and modulate the absolute value of the reinforcers.
3. **Generalization:** Associative theory treats stimuli as compounds of elements, each of which has an associative strength. It follows that two stimuli compounds which share some of these elements thus share some of their associative strength. Generalization follows directly from this analysis.
4. **Large sized problems:** All the factors included in the structure of learning will reduce the processing required. Based on stimulus generalization, new and larger environments will share elements and relationships with smaller ones, facilitating learning and reducing computational complexity.

1.4.1 Experiments to test hypothesis and objectives

Three types of experiments will be employed to test the relative merits of the resulting learning model with regards to Q-learning, and provide results to answer the research questions, support or undermine the hypothesis, and justify the objectives:

- **Convergence:** It will be tested whether the algorithms converge for scenarios of different complexity. Additionally, the speed of convergence to optimality will be tested for the same scenarios. This will provide results to discuss research questions 1and 2.
- **Generalization:** Experiments involving scenarios of different complexity will be employed to test whether the algorithms are able to use learning experience from one situation to a similar situation. This will provide results to discuss research questions 3 and 4

- **Avoidance of aversive situations:** An experiment will test whether the algorithms are able to avoid situations of low utility. This will provide results to discuss research question 1.

1.4.2 Evaluation criteria

The resulting algorithm has been tested and evaluated against the Q-learning algortihm in scenarios of various degrees of complexity. Algorithm performance has been measured according to the following criteria:

1. Eventual convergence to optimality (that is, provable guarantee of asymptotic convergence to optimal behavior).
2. Speed of convergence to optimality.
3. Generalization. Assessment has been carried out on how experience from one environment configuration can be successfully utilized in a different but similar environment configuration.

1.5 Project beneficiaries

The research undertaken during this project is likely to be of interest to the machine learning community, as well as to the associative learning community in the field of Psychology. A synthesis of reinforcement learning theories with associative learning theories from Psychology is attempted. This will provide the machine learning community with possible indications of solutions to the problems identified in the motivation section. Additionally, the associative learning community will gain a computational model incorporating a small subset of their theories.

1.6 Structure of the report

The project work underlying this report has a design-and-build component in that a new algorithm has been constructed and implemented in a learning simulator environment, which itself has been implemented from scratch. However, this is only the supporting framework upon which the aim and objectives are made possible and testable. The output from the simulator, in the form of statistics and convergence plots, is used as the supporting evidence to prove the new learning model which has been produced. In setting the context of the project, the literature survey provides an overview of reinforcement learning in itself, as well as of classical and instrumental conditioning. The methods chapter takes the role of explaining the work process during the requirements gathering, analysis, design and implementation of the new algorithm, as well as the design of experiments that will help answer the research questions. Additionally, the methods chapter presents the different components of the new learning framework. Carrying on from the methodology chapter, the results chapter presents the output from the different experiments and a statistical analysis on generalization. In the discussion

chapter a formal analysis is made on the basis of the experiment results. This analysis gives an explanation of the results, and why the new learning framework works for the tested scenarios. Finally, the last chapter will conclude to which degree the aim has been met by reference to the results analysis from the discussion chapter. Additionally the work process will be evaluated and future work will be suggested.

Chapter 2

Literature Survey

2.1 Reinforcement Learning

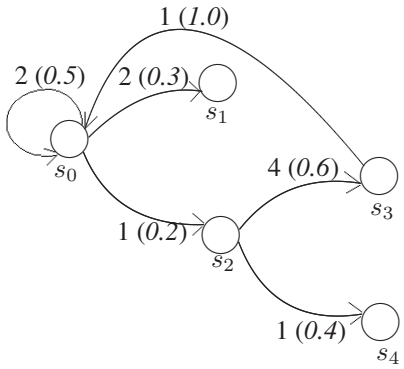
[Sutton & Barto 1998] present an introduction to the field of Reinforcement Learning (RL). In their book, the reader is acquainted with the history of Reinforcement Learning, with its roots being twofold. First, the RL field is based upon Psychological theories of learning by trial-and-error, that is, instrumental learning as described in early experiments by [Thorndike 1911] and his "Law of Effect" (more on this in section 2.3). Secondly, techniques for solving RL problems computationally are based on theories from "optimal control" arriving in the 1950's, the most influential being Dynamic Programming [Bellman 1957*a*]. Psychology and Computer Science have different aims when it comes to the underlying theory of RL. Whereas Psychology aims to describe observable learning phenomena in animals (and humans), Computer Science aims to exploit the models describing these phenomena in order to construct more effective algorithms in terms of adaptive systems and optimal control of computational processes.

Reinforcement Learning assumes the existence of an underlying framework called a Markov Decision Process (MDP) [Bellman 1957*b*, Russell & Norvig 2003]. A MDP contains elements that are essential for the functioning of the different techniques in RL. These elements are [Sutton & Barto 1998]; a set of states S; a set of actions available in each state $\mathcal{A}(s)$; a return function specifying the expected immediate return when executing an action in a specific state $\Re^a_{ss'}$; and a set of transition probabilities $\mathcal{P}^a_{ss'}$ for each state. State transitions and returns are provided by the environment, and the set of possible actions are known *a priori* and intrinsically to the learning system (agent). The transition probabilities specify a model of the dynamics of the environment in which the agent is situated and interacts with. A return can take on any real number, and therefore the agent is "wired" to prefer larger returns over smaller ones. Given this preference the long term-goal of the learning system is to maximize the return it receives over time. This implies that each action should be chosen so as to achieve this goal. More formally:

$$\forall a, a_{greedy} \in \mathcal{A}(s) : a_{greedy} \succ a \Leftrightarrow \Re^{a_{greedy}}_{ss'} > \Re^a_{ss''} \tag{2.1}$$

Statement (2.1) only specifies the preference over immediate returns, that is, it facil-

Figure 2.1: A MDP showing; states (circles), arrows (transitions),rewards (number above transition), and transitional probabilities (number in italic and parentheses above transition)



A deterministic policy π_{det}:

s	$a \rightarrow s'$
s_0	a_{s_2}
s_1	-
s_2	a_{s_3}
s_3	a_{s_0}
s_4	-

itates greedy action selection. However, this tells the agent little about the long-term effects of taking an action in a specific state. Therefore, what is needed, is a preference over state-values, where each state-value specifies the long-term effects of ending up in that state given some policy π. A policy π defines a mapping from states to action, i.e. it specifies the probability of taking an action in a specific state, written as $\pi(a, s)$. In a finite horizon environment, i.e. one in which a terminal state is always reached, the state-value under a given policy is specified by additive returns [Russell & Norvig 2003]:

$$V^{\pi}(s) = \sum_{t=0}^{terminal} R(s_t) \tag{2.2}$$

Conversely, in an infinite horizon environment, where it is uncertain whether a terminal state is ever reached, the state-value is specified by discounted additive returns utilizing an exponentially decreasing discount factor, γ, indexed by time [Russell & Norvig 2003]:

$$V^{\pi}(s) = \sum_{t=0}^{\infty} \gamma^t R(s_t), \gamma \in [0, 1] \tag{2.3}$$

Including the discount factor has the effect of paying less attention to the value of future states, given a sequence generated by a stochastic policy π. Of course, the importance of future returns is modulated by the discount factor, where if $\gamma = 0$ the agent will be immediately greedy, whereas if $\gamma > 0$ the agent will prefer accumulated returns. In Reinforcement Learning, the task is to find the optimal policy. The value of a policy is the expected sum of discounted returns for all state sequences generated by the policy,

the optimal policy π^* being the one having maximum value [Russell & Norvig 2003]:

$$\pi^* = \underset{\pi}{argmax} E\left[\sum_{t=0}^{\infty} \gamma^t R(s_t)\right] \tag{2.4}$$

However, equation 2.4 does not take into account the transitional model of the world. The Maximum Expected Utility (MEU) principle, described in [Russell & Norvig 2003], and with foundations in early philosophical theories by [Arnauld 1662], does consider both the value of an outcome and the probability of that outcome occurring, e.g. winning the grand prize in a lottery, which has high value but very low probability. The value of an optimal policy can thus be restated in light of the MEU principle [Russell & Norvig 2003]:

$$\pi^* = \underset{\pi}{argmax} \sum_{s'} \mathcal{P}^a_{ss'} V(s') \tag{2.5}$$

Similarly the state value function can be specified in light of equation 2.5 in what is known as the Bellman optimality equation [Bellman 1957*a*]. The insight of Bellman was that the value of a state can be expressed in terms of the immediate return in that state and the value of its neighboring state, given the action that leads to the state with highest utility. This concept is called bootstrapping, that is, specifying the value of a state based on the values of successive states. The equation thus has a recursive definition [Bellman 1957*a*, Sutton & Barto 1998]:

$$V^*(s) = \underset{a}{max} \sum_{s'} \mathcal{P}^a_{ss'} \left[R^a_{ss'} + \gamma V^*(s')\right] \tag{2.6}$$

As can be deduced from equation 2.6 a "lookahead"-model is needed in order to find the optimal action, *i.e* an optimal action is defined in terms of the optimal next state that it leads to. Therefore equation 2.6 implies no explicit preference over actions. However, the Bellman Optimality equation can similarly be defined for state-action pairs (termed Q-values), thereby discarding the need for a "lookahead"-model [Bellman 1957*a*, Sutton & Barto 1998]:

$$Q^*(s,a) = \sum_{s'} \mathcal{P}^a_{ss'} \left[R^a_{ss'} + \gamma \underset{a}{max}\, Q^*(s',a')\right] \tag{2.7}$$

The problem of reinforcement learning is one of finding the optimal policy, i.e. the one that yields the highest return, which implies finding the optimal action for each state; the action that leads to the state with greatest opportunity for future returns. Due to the complexity and the size of the state space of many environments, it is not possible to conclude a preferential order over all possible state sequences for every policy combination. All reinforcement learning methods therefore employ some form of search in the space of all policies. In general, the state-value functions, (or state-action value functions), provide the overall utility of a policy, and therefore the RL problem can be stated in terms of two interacting processes [Sutton & Barto 1998]:

1. Policy evaluation: making the current state-values consistent with the current policy, (that is, provide its true value)

2. Policy improvement: Improving the policy based on the current state-value functions.

Because an action that seems immediately worse than another, might be better in the long run, there will always be the dilemma whether to exploit the current policy or explore alternative actions in order to improve the policy. Hence, there is a tradeoff to be made between exploration and exploitation.

2.1.1 The concept of state and the Markov property

In Reinforcement Learning a state is anything the agent can perceive in the environment at any time-step, i.e. a signal from which the agent makes a decision as to which action is most appropriate in a particular situation. The reason for merely making a decision based on the current state signal, and not a conjunction of the preceding state signals leading up to it, (the history), is the assumption of the state having the Markov property. A state signal which encapsulates the history of states, actions, and rewards leading up to it is said to have the Markov property, and can therefore be used solely as a decision cue. The importance of the Markov property is due to the transitional model, which defines a conditional probability distribution for every state $s \in S$ given preceding sequences of states, actions, and rewards. Every state $s \in S$ must have the Markov property, for the transitional model to be able to provide a unique sequential sequence of transitions. More formally [Sutton & Barto 1998]:

$$\forall s_{t+1}, \forall r_{t+1}, \forall (s_t, a_t, r_t, s_{t-1}, a_{t-1}, \ldots, r_1, s_0, a_0) : Markov(s_t) \Leftrightarrow$$
$$Pr\{s_{t+1}, r_{t+1} | s_t, a_t, r_t, s_{t-1}, a_{t-1}, \ldots, r_1, s_0, a_0\} = Pr\{s_{t+1}, r_{t+1} | s_t, a_t\}$$

2.1.2 Dynamic Programming

For an environment with n states there are n Bellman equations, and these provide a unique solution for the optimal policy. i.e. the optimal policy is found by solving the set of non-linear optimality equations given by equation 2.6 for all n states. Because of the non-linear nature of these equations they cannot be solved simultaneously, the max-operator being non-linear [Russell & Norvig 2003]. For this reason several approximate techniques exist that are able to solve these equations iteratively, based on initial estimates of both optimal policy π^* and state values. Provided in algorithm 1 is pseudocode for one of these techniques, namely Value Iteration, in which an initial estimate of the value functions is used to iteratively improve both the value functions and implicitly the policy (as given in [Sutton & Barto 1998]). The value iteration algorithm will converge in the limit of θ, provided that the environment is finite or $\gamma < 1$.

Through the definition of Dynamic Programming, due to the Bellman optimality functions for the optimal policy and the optimal value functions, the techniques used to solve these equations require full models of the environment in which they are working [Sutton & Barto 1998]. The provision of complete transitional models over environments is somewhat of a luxury, and although approximations of these models can be learned from experience, it would generate extreme computational costs for large state spaces. Additionally, as can be seen from the recursive definition of the Bellman equation 2.6, Dynamic Programming performs full backups in order to compute

Algorithm 1 The Value Iteration algorithm

Initialize V arbitrarily, e.g., $V(s) = 0$, for all $s \in S$

Repeat
  $\Delta \leftarrow 0$
  For each $s \in S$:
    $v \leftarrow V(s)$
    $V(s) \leftarrow R(s) + \underset{a}{max} \sum_{s'} \mathcal{P}^a_{ss'} \gamma V(s')$
    $\Delta \leftarrow \max(\Delta, |v - V(s)|)$
until $\Delta < \theta$ (a small positive number)

Output a deterministic policy, π, such that
$\pi(s) = \underset{\pi}{argmax} \sum_{s'} \mathcal{P}^a_{ss'} \gamma V(s')$

value functions. For environments with large state spaces this yields combinatorial explosion, (for stochastic policies), [Sutton & Barto 1998], again adding to the computational costs and memory requirements. For these reasons Dynamic Programming can only provide techniques that have practical utility in off-line scenarios, i.e. in which the optimal policy is computed before it is actually utilized.

2.1.3 Monte Carlo methods

As described in the previous section, Dynamic Programming yields extreme computational costs and memory requirements in large state spaces, due to its requirement of full state value backups and a complete transitional model of the environment. Unlike Dynamic Programming, Monte Carlo methods do not require complete knowledge about the environment and performs only sample state-value backups (albeit full), due to the fact that these methods utilize sample interactions with the environment to estimate state-values [Sutton & Barto 1998]. From this incomplete, but often representative experience, it is possible to solve the Reinforcement Learning problems by averaging sample returns. As is clear from the recursive definition of the Bellman optimality equation (2.6), Dynamic Programming "bootstraps", that is, computes state value functions based on estimates of other state value functions. Monte Carlo methods, on the other hand, compute state value functions based on sample returns generated by interaction with a subset of the environment. Considering this sampled experience, a model of the environment is not needed. It is possible to evaluate any policy π by utilizing Monte Carlo sampling methods. The evaluated policy is then used to generate state-action sequences, which in turn yields return sequences. A representative Monte Carlo method for policy evaluation is First-Visit MC [Sutton & Barto 1998] as demonstrated in algorithm 2.

Monte Carlo methods have several advantages over Dynamic Programming, mainly in terms of savings in computational costs. Optimal policies can be learnt from direct interaction with the environment, without any explicit transitional model. This allows a focus to be made on a subset of the state space of the environment, which actually may

Algorithm 2 First-visit MC

Initialize:
  $\pi \leftarrow$ policy to be evaluated
  V $\leftarrow$ an arbitrary state-value function
  $Returns(s) \leftarrow$ an empty list, for all $s \in S$
Repeat forever:
  (a) Generate an episode using π
  (b) For each state s appearing in the episode:
    $R \leftarrow$ return following the first occurrence of s
    Append R to $Return(s)$
    $V(s) \leftarrow$ average($Return(s)$)

prove to be a representative sample. Because of the sampled episodic returns on which Monte Carlo methods are based, the First visit policy evaluation algorithm (alg. 2) makes no use of the Bellman equation, and hence does not have the strict requirement that every state must have the Markov property. For strict policy evaluation, as in algorithm 2, MC will converge to the true state-values provided that an infinite amount of visits is made to all states [Sutton & Barto 1998].

2.1.4 Temporal Difference Learning

Temporal difference (TD) learning is a hybrid of Dynamic Programming and Monte Carlo methods. First, as Dynamic Programming does, it "bootstraps", that is, estimate state-values based on successive state-values. Secondly, it approximates average returns based on sample interactions with the environment, like Monte Carlo does. Because TD-learning uses sample interactions to estimate state-values, it does not require a transitional model of the dynamics in the environment. In other words the transition probabilities are provided implicitly by the sampled state-transition sequences, and consequently TD-learning accumulates average return, or in other words an estimate of the Bellman optimality equation (2.6), given the optimal policy [Sutton & Barto 1998]. Contrarily to Monte Carlo methods, which perform full backups at the end of an episode, TD-learning estimates the current state-value only on the basis of the successive state-value, like Dynamic Programming. However, unlike Dynamic Programming, TD-learning updates its state-value estimates in increments, of which the step-size is decided by a learning-rate parameter α [Sutton & Barto 1998]:

$$V(s_t) \leftarrow V(s_t) + \alpha\big[r_{t+1} + \gamma V(s_{t+1}) - V(s_t)\big] \tag{2.8}$$

Given a finite set of episode sequences, which are iteratively used to update the state-value estimates by applying equation 2.8, the state-values will converge to the Maximum-likelihood estimate of the underlying Markov Decision process [Sutton 1988]. In practice this means that the policy has stabilized with regards to the state-values, or in other words the state values, and implicitly the action preferences for each state s, represent the parameter that has the highest probability of generating the episode sequences.

TD-learning has close connections with the Rescorla-Wagner model of classical conditioning [Rescorla & Wagner 1972, Sutton & Barto 1990], being a time-derivative model of classical conditioning. It has shown the ability to model learning phenomena from animal learning theories including conditioned inhibition and second order conditioning [Sutton & Barto 1990] (more on this in section 2.3).

2.1.5 Q-Learning

As described above TD-learning can be used to learn from sampled experience, but without having to wait until the end of the episode to perform backups. TD-learning estimates state-values from immediate backups based on successive state-values. This makes TD-learning suitable as an online control algorithm, where the sampled state transition sequences and the state-values are experienced and taken into use directly. The difference between this form of control algorithm and regular TD-learning is due to the explicit policy specification in TD-learning, whereas an online control algorithm derives the policy from the value functions. One example of an online control algorithm is Q-learning [Watkins 1989, Sutton & Barto 1998]. Q-learning uses off-policy value function estimation, or, more specifically, it has a different behavioral policy than the optimal policy π^* being improved. The algorithm approximates equation 2.7, and thereby works with state-action pairs, (Q-values), instead of state-values. Algorithm 3 [Sutton & Barto 1998] shows the working of Q-learning.

Algorithm 3 Q-learning

Initialize $Q(s, a)$ arbitrarily
Repeat (for each episode):
  Initialize s
  Repeat (for each step of episode):
    Choose a from s using policy derived from Q
    Take action a, observe r, s'
    $Q(s, a) + \alpha\big[r + max_{a'}Q(s', a') - Q(s, a)\big]$
    $s \leftarrow s'$
  until s is terminal

Q-learning works by bringing the behavior policy closer to the policy being estimated. Therefore, the algorithm can be seen as performing policy evaluation and policy improvement simultaneously. The Q-learning algorithm will converge with probability 1 to the true Q-values $Q(s, a)$, under the condition that the step-size parameter α is gradually decreased. Otherwise, under a constant step-size, it converges in the mean of α. In order to illustrate the sequential progress of Q-learning, figure 2.2 presents a simple non-exhaustive example of Q-value updates. From the point of view of Psychology, online TD-Learning, and thus Q-learning, explains the phenomena of second-order conditioning, or the anticipation of future reward, by predicting the accumulated reward that follows a state given a specific policy [Dayan & Abbot 2001], (more on this in section 2.3). This is evident from figure 2.2 as the reward received at the goal state is propagated gradually backwards to the start state. The example in figure 2.2 employs

an ϵ-greedy exploration policy, where $\epsilon \in [0, 1]$. Under such a policy the non-greedy action is selected with a probability of ϵ. Another explorative policy is Softmax, which computes the probability of selecting an action a in a state s by the following equation [Sutton & Barto 1998]: $\pi(s, a) = \frac{e^{Q(s,a)/\tau}}{\sum_{b=1}^{n} e^{Q(s,b)/\tau}}$, where $\tau > 0$. High values of τ will result in nearly equal probabilities for all actions, whereas low values will favor greedy action selection. A third way of ensuring exploration is to initialize Q-values according to a positive estimate, that is, to make the algorithm falsely believe that previously unvisited states are better than other states. This in turn causes exploration based on deceitful estimates.

Q-learning has traditionally been the most popular reinforcement learning technique [Alonso & Mondragón 2005], due to its simplicity. First, it can be defined almost completely in terms of the Q-value equations. Secondly, these equations apply only to states at the time they are visited, making it suitable as an online control algorithm. Thirdly, the two processes of the reinforcement learning problem, policy evaluation and policy improvement, are closely intertwined in an abstract way, only controlled by an exploration parameter, (*eq* by ϵ in an ϵ-greedy policy). Additionally, Q-learning is a so called model-free learning algorithm [Kaelbling, Littman & Moore 1996], in that it does not require any transitional model of the environment. Finally, the policy which is being evaluated and improved is not stored explicitly, it is derived from the Q-values. In sum, Q-learning exhibits its simplicity because both the transitional model and the optimal policy is a mirror-image of the Q-values, thereby cutting memory requirements. Q-learning is *exploration insensitive* in that it always estimates the current greedy policy regardless of the behavior policy being followed, and therefore the Q-values will converge to the optimal values. This is the most important reason for the success and popularity of Q-learning [Kaelbling et al. 1996].

2.1.6 Problems with Reinforcement Learning

[Alonso & Mondragón 2005, Kaelbling et al. 1996] have identified several problems with temporal difference learning methods, Q-learning being one instance:

Exploration-exploitation equilibrium: In order to converge with a probability of 1 to the true state-value functions $V(s)$ for a policy π, temporal difference learning methods have to decrease the step-size parameter α according to the true sample return average $V(s_t) = \frac{1}{t+1}\big[r_{t+1} - V(s_t)\big]$ [Sutton & Barto 1998], the fraction being the step-size parameter, i.e. $\alpha_t(s) = \frac{1}{t_s}$. However, this requires a stationary environment, one in which the underlying dynamics does not change over time. Stationary environments are not common in real world problems [Sutton & Barto 1998]. Therefore, to cater for stochastic environment dynamics, in which the policy has to be constantly reevaluated, the step-size parameter α is being held constant. Thus, in such a case, TD-learning will converge in the mean of α, but will continue to vary in response to the most recent rewards. The problem of exploration-exploitation balance is hence, (although not exclusively), one of readjusting the policy to a non-stationary environment. As have been discussed earlier in section 2.1, exploration-exploitation balance is also important for speed of convergence. Once a policy has been evaluated, policy

Figure 2.2: An example of Q-learning working in a grid environment

$$\forall s \in S : \forall a \in A(s) : a_{ss'} \wedge s' \neq s \wedge s' \neq s_{goal} \rightarrow r_{ss'} = 1$$
$$\forall s \in S : \forall a \in A(s) : a_{ss'} \wedge s' \neq s \wedge s' \equiv s_{goal} \rightarrow r_{ss'} = 100$$
$$\forall s \in S : \forall a \in A(s) : a_{ss'} \wedge s' \equiv s \wedge s' \neq s_{goal} \rightarrow r_{ss'} = 0$$

$$s_{goal} = (2, 2)$$
$$s_{start} = (0, 0)$$
$$\forall s \in S, \forall a \in A(s) : Q(s, a) = 0$$

$$\forall s \in S : A(s) = North, East, South, West$$

Discount factor $\gamma = 0.9$
Step-size parameter $\alpha = 0.1$

The initial Q-values for each state s in the grid:

0	0	0
0	0	0
0	0	0

The resulting Q-values after applying the policy $\pi_{\epsilon-greedy}$ for one episode:

0.1	0.1	0
0	0.1	10
0	0	0

Action sequence taken from s_{start}: East,South,East,South
Affected Q-values:
$Q((0,0), East) \leftarrow 0 + \alpha\big[1 + \gamma 0 - 0\big]$
$Q((1,0), South) \leftarrow 0 + \alpha\big[1 + \gamma 0 - 0\big]$
$Q((1,1), East) \leftarrow 0 + \alpha\big[1 + \gamma 0 - 0\big]$
$Q((2,1), South) \leftarrow 0 + \alpha\big[100 + \gamma 0 - 0\big]$

The resulting Q-values after applying the policy $\pi_{\epsilon-greedy}$ for a second episode:

0.199	0.199	0
0	1.09	19
0	0	0

Action sequence taken from s_{start}: East,South,East,South
Affected Q-values:
$Q((0,0), East) \leftarrow 0.1 + \alpha\big[1 + \gamma 0.1 - 0.1\big]$
$Q((1,0), South) \leftarrow 0.1 + \alpha\big[1 + \gamma 0.1 - 0.1\big]$
$Q((1,1), East) \leftarrow 0.1 + \alpha\big[1 + \gamma 10 - 0.1\big]$
$Q((2,1), South) \leftarrow 10 + \alpha\big[100 + \gamma 0 - 100\big]$

improvement can start. In fact, the latter does not have to wait for policy evaluation to complete. In order to perform policy improvement, an agent will have to explore new actions.

Temporal discounting: In reinforcement learning a discount factor, ($0 \geq \gamma \geq 1$), is utilized to modulate the importance of future reinforcement. The discount factor is most readily seen as a mathematical trick to bound the infinite sum [Kaelbling et al. 1996], in cases where the environment is non-episodic. A large discount factor approaching 1 will make the agent prefer long-term additive returns, whereas a small discount factor will make the agent greedy. The problem is then, as argued by [Alonso & Mondragón 2005], that a large discount factor will slow down learning, and contrarily a small discount factor might cause the value functions to converge to a sub-optimal policy. The agent does not know the length of an episode, it only knows the discount factor, which has to be set by the algorithm designer. It can be very difficult to set the discount factor correctly in order to guide the agent towards a goal, because in some environments it can be hard to actually define the goal, or there can be multiple goals with varying state sequence trajectories and different definitions of success. This is why a hierarchical approach has been suggested [Singh 1992], in which sub-goals are trained separately. An example of this is soccer, where the ultimate goal of scoring is a highly complex task dependent on many sub-goals such as; dribbling an opponent, passing the ball, kicking the ball in the right direction, *etc*. In other words, teaching an agent how to achieve a goal based on delayed rewards is not straightforward, and depending on the complexity of the environment would still require significant engineering, despite the simple definition of many reinforcement learning algorithms.

Generalization: The problem of generalization is generally due to the consideration of states as irreducible entities, and additionally because of the way in which state-values are considered and learned [Alonso & Mondragón 2005]. In the tabular case it is quite easy to realize that once a reward structure has been learned, there will be no generalization if state similarity is not considered. In other words there will be no transfer of learning from one state to another, even though the two might be similar. How to measure similarity, however, is a whole different problem. To define a similarity function which measures similarity on the basis of the dynamics of the environment would seem to be extremely difficult, because it should ideally take into account the actual distance between states, in terms of required changes in the environment to reach one state from the other, which in turn calls for a model of the environment. The issue with generalization and transfer of learning, is also due to the associations being formed [Alonso & Mondragón 2005]. In RL an association between a state and an action is formed, discarding to take into account the actual outcome of taking an action in a state. A direct consequence of this is that the policy has to be relearned if the reward structure changes, even though the transitional model might be static. Learning in RL is completely dependent on the reward structure.

Large sized problems: Many environments have prohibitively large state spaces. For instance the game of chess has been estimated to have $10^{10^{56}}$ possible game se-

quences [Weisstein 2005]. It is therefore not possible to visit all possible states, let alone generate all sequences. Even if it was, it would still not be possible to use a tabular approach to store state-values for each state, due to finite computer memory. This is why function approximation (FA) has been introduced in reinforcement learning. Although there are many ways to use function approximation, it can be used to generalize the value of one state to the value of a similar state. One example is a linear function, consisting of a parameter vector, $\vec{\theta}$, having as many elements as the state signal vector s. The value of a state can then be calculated as the dot product of the two vectors [Russell & Norvig 2003]:

$$V_{\vec{\theta}}(\vec{s}) = \sum_{i=0}^{n} \theta_i s_i$$

If there is a prediction error, that is, the function gave the wrong expected return, the parameter vector can be moved in the opposite direction of the error, i.e. gradient descent with regards to the prediction error [Russell & Norvig 2003]:

$$R = \sum_{t=0}^{\infty} \gamma^t r_t$$

$$\vec{\theta} \leftarrow \vec{\theta} + \alpha(R - V_{\vec{\theta}}(\vec{s})) \frac{\partial V_{\vec{\theta}}(\vec{s})}{\partial \vec{\theta}}$$

There are limits as to what a simple linear function approximator , (as in the previous example), can learn, and especially if there is no correlation between similarity of states and their value. In fact, linearity in the state features with regards to the parameter vector is a prerequisite for this simple FA to work. Additionally, if state features are correlated, conjunctive features needs to be defined [Sutton & Barto 1998]. Apart from simple linear FAs, other function approximators such as neural networks with backpropagation learning rule and non-linear activation functions have been used to learn an approximate state-value function in the game of backgammon (TD-Gammon: [Tesauro 1995]). However, success with non-linear FAs is the exception, rather than the rule. Combining FA and error correction rules like the one used in temporal difference learning (eg. $V(s) \leftarrow V(s_t) + \alpha\big[r_{t+1} + V(s_{t+1}) - V(s_t\big]$), can lead to uncontrollable error propagation. More specifically, FA tries to minimize a mean return prediction error over all states, and thus requires a static training set where the correct return is known for each state, (a.k.a. supervised learning). Formally, the parameter vector would be optimized in terms of minimizing a mean squared error over all states [Sutton & Barto 1998, p.195]:

$$MSE(\vec{\theta_t}) = \sum_{s \in S} \big[V^{\pi}(s) - V_t(s)\big]^2$$

It is evident that if a local error is reduced by changing the FA parameters, other state-values will change as well, and thereby adding another search process to the problem, i.e. the one of tuning the FA to provide the true value function with regards to a policy. If the policy is constantly changing, hence resulting in a non-static training set, there will be no convergence to the correct value function because the MSE will have no

basis of comparison from one policy to the other. However, this is not a problem in off-line learning where the policy is fixed and gradient descent can take place over a subset of the state space. The FA will then converge to a local optimum, (i.e. the smallest possible MSE), under assumptions of a decreasing step-size parameter. This local optimum may be a close match to the correct value function [Sutton & Barto 1998]. A fixed policy requires previous learning, and a stationary environment, both of which are uncommon for large state spaces.

2.2 Credit assignment problem

The credit assignment problem is one of establishing the level of responsibility of a state or action in generating a reward. As have been discussed earlier, a policy defines a mapping from states to actions, and an optimal policy maximizes the reward signal received over time. Thus, the state-value or state-action-value functions can be seen as providing an indicator of how much responsibility a particular action, (or an action leading to a beneficial state), has in maximizing the reward signal, i.e. the expected reward. In a long chain of alternating state-action sequences, only the last action may bring about reward. It would then seem logical to credit actions less and less, the further away from the actual reward they are, because there are usually costs involved with reaching the actual reward. However, they might still be crucial in bringing about the actual reward in the end. Reinforcement learning, as demonstrated in Q-learning (see fig. 2.2), propagates a reward signal backwards through the state-action sequence leading up to the reward. The credit thus represent the additive expected return by following the optimal policy from that state and onwards. This is the solution that reinforcement learning brings to the temporal credit assignment problem, i.e. how to credit each action in a sequence of actions leading up to a reward in the end. If an action does not lead to reward, then it will be discredited. In temporal difference learning, prior to convergence, the credit of each state is changed according to a prediction error (eq. 2.8), so for a state sequence vector the error correcting rule is moving towards the real credit in terms of the correct return [Sutton 1988]. In sum, reinforcement learning is solving the temporal credit assignment problem by means of return predictions.

Psychology has a slightly different way of looking at the temporal credit assignment problem. [Mackintosh 1983] describes the association between a stimulus (state) and a reinforcer (return) as being modulated by the temporal contiguity between the two events. It is established that the association grows more rapidly if the stimulus is presented in close temporal proximity with the reinforcer, and the learning rate decreases as the time interval increases. What is more important, however, is whether a second stimulus is presented in between the two events separated by this interval. If a second stimulus is presented after the first stimulus and before the reinforcer, this second stimulus will become associated with the reinforcer, while the association between the first stimulus and the reinforcer will be attenuated. In other words, the stimulus most close in temporal proximity with the reinforcer will become most strongly associated with the reinforcer. To explain this it is assumed that the predicting event, (e.g. event 1 predicts event 2), leaves a trace which gradually decreases with time [Gabriel & Moore 1990, Kehoe, Schreurs & Graham 1987]. This trace is thus working as a

modulator of the gain in association between event 1 and 2.

Additionally, the credit assignment problem has a structural element [Kaelbling et al. 1996]. A state signal having the Markov property, incorporates a lot of information, ideally the whole history of state-action-reward sequences leading up to the state. In reinforcement learning the state signal is seen as an irreducible entity. The agent does not reason explicitly with the information provided in the state signal. It only takes into account the state value, provided by the expected additive returns. The additive return is an abstraction of the actual outcome, allowing the different reinforcement learning techniques to be applied to a whole range of problems without taking into account any information provided by the state signal. It is evident though, [Kaelbling et al. 1996], that different elements of the state signal might have different levels of importance to the problem being solved by the agent. This is known as the structural credit assignment problem, "the problem of deciding how the different aspects of an input affect the value of the output" [Kaelbling et al. 1996].

An implication of the structural credit assignment problem, is the spatial contiguity of events. As is described in [Mackintosh 1983], if two events are presented in close spatial proximity, they tend to be more easily associated than if they were presented far away from each other. This effect can be seen as a consequence of the number of stimuli present. However, in Psychological theories of learning, when several stimuli are present in the current state, they are described as competing for the gain in associative strength with the succeeding event. In other words, they all gain associative strength relative to their attentional value [Gabriel & Moore 1990]. As have been described earlier, states are considered as irreducible entities in a Markov Decision Process. Therefore there is never more than one stimulus present in any state. Additionally, there is never more than one time-step between any two successive states, and the time-step is constant for the tabular case. Still, the discriminatory power of a state is dependent on the granularity of the state space, i.e. the number of spatially adjacent states surrounding a state. Thus, in reinforcement learning with irreducible state signals, the structural credit assignment problem is reduced to spatial contiguity in time, whereas associative learning considers several stimuli simultaneously.

2.3 Associative Learning

Associative learning is a field of Psychology concerned with the study and the construction of models to explain the acquisition and extinction of associations between different stimuli, and stimuli and responses, in animals[1]. A stimulus is anything present in the environment that the animal can perceive, and that elicits a response of some sort from the animal. In any situation the animal finds itself in, several stimuli may be present, implying the ability to discriminate or generalize over stimuli, in order to determine what is important in the environment in terms of survival. During experimental studies Psychologists control the presentation of stimuli over trials and measure

[1] A different ontology is used in Psychology to describe elements; a state is known as a Stimulus (S) (or a stimuli compound), an action is known as a Response (R), and the new state s' that appears after a has been performed in state s is known as the Outcome (O), (which is also a Stimulus or a stimuli compound) [Dayan & Abbot 2001].

the associations being formed by gathering statistics from animal responses. Traditionally the animals are trained to form associations between a stimulus that they have no instinctive aversion or excitation towards, (i.e. a neutral stimulus), hence called the conditioned stimulus (CS), and contrarily a stimulus that they naturally want or are afraid of, termed the unconditioned stimulus (US). The US is thus utilized to condition the animal in forming an association between a CS and the same US, where the association is measured by the animal's conditioned response to the CS. The conditioned response is of the same nature as the unconditioned response elicited by the animal when presenting the US alone, and the animal has thus generalized the CS with the US to some degree.

There are two main threads in associative learning [Mackintosh 1983]; classical (Pavlovian) conditioning and instrumental conditioning. The two differ in the responses that animals evoke during formation of associations between a CS and a US. In classical conditioning the response has no effect on the environment, i.e. the animal responds passively and has no influence over whether the US will be brought forward or not. This paradigm was first described by [Pavlov 1927] in his book "Conditioned reflexes". On the other hand, in instrumental conditioning the animal learns to associate specific stimuli with responses that provide excitatory outcomes. The animal actively tries to change the environment to bring forward something that it wants. Theories of instrumental conditioning were brought to light by [Thorndike 1911].

2.3.1 Classical conditioning

One of the early examples of classical conditioning was presented by [Pavlov 1927] in his notoriously famous experiment wherein dogs were conditioned to associate a tone with food. Initially the dogs have no natural excitation towards the tone, and therefore the tone is a neutral stimulus. During the conditioning the tone is sounded in close temporal proximity before the presentation of food. As opposed to the tone, which is termed the Conditioned Stimulus (CS), the food is something the dogs naturally want and that cause them to salivate. The food is termed the Unconditioned Stimulus (US) and is said to elicit an Unconditioned Response (UR), more specifically the salivation. As the tone is presented with the food an association is said to form between the tone and the food (CS$\rightarrow$US) and after several trials the CS alone will cause the dogs to elicit a Conditioned Response (CR), which is the same as the UR, that is, salivation. In sum, the mere sounding of the tone will cause salivation in the dogs, due to the association between the tone and the food. Two explanations have been suggested in terms of the procedural implications of classical conditioning. First it can be said to be the process by which learns a causal model of the environment, i.e. causal relationships between stimulus events [Sutton & Barto 1990]. Secondly, a related interpretation is that "classical conditioning is a manifestation of the animal's attempt to predict the US from cues provided by CSs" [Sutton & Barto 1990]. Either way it can be seen as providing a predictive model of the environment. It is important to stress that under classical conditioning, animal responses do not have any control over the delivery of reinforcers. Their responses are simply innate preparatory or consummatory reflexes, enabling them to better exploit or avoid the US.

2.3.2 Instrumental conditioning

Early theories of instrumental conditioning focused on the formation of associations between a conditioned stimulus and a response strengthened by a reinforcer, that is, S→R associations [Mackintosh 1983]. Even though S→R associations do form, and explain the formation of habits, they fail to explain the expectation of an outcome. According to [Mackintosh 1983] associations are formed between representations of events which are in fact related in the real world. S→R associations are only implicitly connected with the outcome, through some derived value of the outcome. However, [Rescorla 1992] notes that both S→R and S→O associations form in instrumental conditioning, the role of the CS merely being one of activating a representation of the response or outcome. [Mackintosh 1983] provides a more convincing account of instrumental conditioning:

> "If learning is viewed as the acquisition of knowledge about the world, then classical conditioning is a matter of learning what external events predict the occurrence of a reinforcer, while instrumental conditioning involves organisms learning which of their actions are responsible for the occurrence of a reinforcer." [Mackintosh 1983, p.80]

From this it is understood that response-outcome associations are formed. However, [Mackintosh 1983] also highlights that these associations might only arise, depending on the circumstances, in the presence of an antecedent stimulus, (the CS), thus working as a discriminatory cue. This cue is also known as an occasion setter as described in [Rescorla 1992]. The R→O association implies an expectation towards an outcome, which in turn implies the ability to derive a motivational value from the outcome. This ability is essential if the animal is to influence its own fate. In other words, the animal must know what it wants or does not want. The value of an outcome is thus decided by the animal's motivational state.

An important notion in instrumental learning is that of operant behavior. If two responses produce the same outcome, under similar conditions, they are essentially the same. That is, the outcome of a response determines the similarity of the response to other responses. [Mazur 1998] describes an experiment in which rats had to wade through a maze filled with a few inches of water in order to reach a reward. Subsequently, after the rats had learned this task, the water-level in the maze was increased and the rats were placed at the beginning of the maze again. Surprisingly, the rats did not have to relearn a new set of behaviors. Instantaneously, they swam through the maze, following the same path as they had learned by wading in the lower water-level. Thus, the rats learn a sequence of turns, not linked to any particular set of behavior. It is partly this sort of adaptability and flexibility that makes animal learning superior to current models of computational reinforcement learning.

2.3.3 The Rescorla-Wagner model for classical conditioning

The Rescorla-Wagner model [Rescorla & Wagner 1972] is a trial-level model for predicting the formation of associations between a CS and a US. In this model the association between a CS and a US is presented as the amount of associative strength,

written as V, that a CS compound manifests with regards to a subsequent US. The change in associative strength is adjusted from trial to trial until reaching an asymptote λ. Thus, the change in associative strength at a given trial, ΔV, is changed according to the discrepancy between the current level of associative strength in the CS, (V), and the asymptote (the maximum associative strength λ), as determined by the US. The actual change is modulated by two learning rates, α and β, specific to the CS and the US respectively. The most notable feature of the Rescorla-Wagner model is that the change in associative strength at a given trial is determined not only by the discrepancy between a single CS and the US, but by the sum of associative strengths of all present stimuli. Put formally, the following equation is the basis of the model [Rescorla & Wagner 1972]:

$$\Delta V_{ij} = \alpha_i \beta_j (\lambda_j - \sum_{i=0}^{n} V_{ij}) \tag{2.9}$$

In equation (2.9) the subscript $_j$ refers to the US, whereas the subscript $_i$ refers to one of the n present CSs, (stimuli compound). As mentioned earlier α_i refers to a learning-rate specific to one of the present CSs, whereas $beta_j$ refers to the US-specific learning-rate. These are both in the range $[0, 1]$, and are determined by the strength of the respective stimuli. The asymptote λ_j, represents the strength of the US. During real experiments with conditioning of animals, the conditioned response (CR) is a direct measure of the associative strength. In the case of Pavlov's dogs, the change in associative strength was measured by the increase in salivation.

A number of different learning phenomena can be explained by the Rescorla-Wagner model, each of which will be explained in the following paragraphs.

Acquisition: Acquisition is the iterative gain in associative strength between a CS compound and a US, when λ is greater than zero. Acquisition is also known as excitatory conditioning. Each CS in the compound will gain associative strength with the US according to the current discrepancy of the asymptote and the sum of associative strengths of the compound, multiplied by the learning-rate for the CS in question, and the US-specific learning rate. The associative strength of the compound will converge when the discrepancy is zero, whereupon no change will occur. In the beginning, when the associative strength of the stimuli compound is zero, the discrepancy will be large, and therefore the change in associative strength will be large. As the discrepancy decreases the change decreases as well. This is the idea of surprise; if the US is completely unexpected, more will be learned than if the US was expected.

Extinction: The opposite process of acquisition is extinction, (or inhibitory conditioning). If after a stimuli compound has been paired with an US, and the associative strength of the compound has reached a certain level, the US is removed at subsequent trials. In the Rescorla-Wagner model this is achieved by setting the asymptote to zero. The change in associative strength will then be negative, because the discrepancy is below zero. Conversely to the process of acquisition, extinction will drive the associative strength to zero.

Blocking: If a US is already fully expected, that is, the associative strength of a CS has reached the asymptotic level, then if another stimulus is added to the compound, this new stimulus will not gain any associative strength. In other words, it has been blocked by the already present stimuli. An example will make things clearer: Suppose CS_1 and CS_2 has been paired with US_1, until the sum of associative strength in the compound, $(V_{CS_1} + V_{CS_2})$, has reached asymptotic level, i.e. the discrepancy is zero. Then another stimulus, CS_3, is added to the compound. The associative strength of this stimulus is zero ($V_{CS_3} = 0$) Because CS_1 and CS_2 already fully predict US_1, the associative strength of CS_3, V_{CS_3}, will remain zero, and nothing will be learned about the new CS in terms of US prediction.

Conditioned inhibition: When considering the blocking example above, and taking into account what happens if extinction occurs after blocking, the blocked stimulus will become a conditioned inhibitor. The sum of associative strength in the two stimuli, CS_1 and CS_2, have already reached asymptote, they fully predict the US. Now, CS_3 is presented, but because CS_1 and CS_2 already predict the US, the associative strength of CS_3 will remain at zero. If the US is then removed, extinction will occur, and all stimuli in the compound will lose associative strength. Because the associative strength of CS_3 is already zero, V_{CS_3} will gain negative strength, and thus become a conditioned inhibitor. This is a consequence of calculating the discrepancy based on the sum of associative strength in the compound. As can be deduced from this, the conditioned inhibitor will, upon reaching a certain negative level, cancel the associative strength of other stimuli, thereby hindering their decrement to no associative strength.

Overshadowing: As mentioned earlier, different stimuli present in a compound may have different strength, and thus their learning-rates will vary. From this it can be seen that more salient stimuli will condition faster, than less salient stimuli. If a CS is much stronger than the other CSs in the compound, it will overshadow the other stimuli during learning. In other, words it will gain more associative strength and predict the US better than the other stimuli. A stimuli with relatively higher learning-rate, α_i, will more rapidly gain associative strength.

The Rescorla-Wagner model is a trial-level model, which effectively means that time is not taken into account. Variables such as inter-stimulus-interval (ISI), (the time interval from the offset of the CS to the onset of US), and inter-trial-interval (ITI), (the time between each trial), are not considered explicitly [Mazur 1998]. It has been contended that the inter-stimuli-interval can affect the associations being formed [Mazur 1998]. As stated by the principle of contiguity, two events occurring closer in time are more readily associated [Mazur 1998]. However, it has been established that "the temporal contiguity between a CS and a reinforcer is neither necessary nor sufficient to ensure conditioning" [Mackintosh 1983, p. 173]. The Rescorla-Wagner model takes this into account by considering stimuli compounds. If a US is already fully predicted by another CS, a newly presented CS will be uninformative, if it does not "signal a change in the probability of a reinforcer, or provide new information about its occurrence" [Mackintosh 1983, p. 173]. [Mazur 1998] notes that time itself can be considered as a stimulus contained in the stimulus compound; if a US is presented 10

seconds after a CS, then the compound will consist of the CS and the delay stimulus.

2.3.4 Hierarchical modulatory relations

[Rescorla 1992] provides an alternative view of conditioned inhibition. As described above in section (2.3.3) regarding the Rescorla-Wagner model, in which associations are formed between stimuli compounds and a US. In such a compound each member CS reflects a binary connection with the US, and the sum of the associative strength of all connections constitutes the overall associative strength between the compound and the US. A conditioned inhibitor, i.e. a CS which has negative associative strength, effectively cancels the associations of the other CSs in the compound with the US. As [Rescorla 1992] describes, an alternative view is that the conditioned inhibitor CS is not at all part of the compound, but is instead associated with the compound-US association on a hierarchical level. That is, the compound-US association is seen as a unit with which other stimuli can form associations. It is then suggested that the CS, also called occasion-setter, acts as a modulator of the associative strength within the unit. The conditions under which a CS takes on a modulatory role instead of becoming directly associated with the US, is said to be determined by the temporal relationship between the CS, the compound and the US. If the CS is presented simultaneously with the compound, it too enters into a direct binary connection with the US. On the other hand, if the CS is presented before the compound, and its predictive strength of the US is low or non-existent, it may enter into an association with the compound-US unit [Rescorla 1992]. Hierarchical conditioning in Pavlovian learning is thus when a CS becomes a modulator of a $CS_1 \rightarrow$US association, e.g. $CS_2 \rightarrow (CS_1 \rightarrow US)$. Under this interpretation the occasion setter, CS_2, is acting as a discriminatory cue stating whether CS_1 will be followed by the US or not. This notion of hierarchical relationships has been used to explain the role of contextual stimuli and their control of animal behavior.

For instrumental conditioning, the occasion setter seems a natural element signalling a hierarchical unit, because occasion setters logically play a role as a precondition of a response if that response is to lead to the desired outcome. Normally, associations between response and outcome (R→O) are considered, but these associations are typically cued by a preceding stimulus, i.e. the occasion setter CS [Rescorla 1992]. The animals learn to associate certain responses with certain outcomes, but a discriminatory cue can signal in which situations a response will lead to the desired outcome. This is known as a three-term-contingency [Mazur 1998]. Hence, the three-term-contingency consists of [Mazur 1998, p. 137]: ”(1) the context or situation in which a response occurs (that is, those stimuli that precede the response), (2) the response itself, and (3) the stimuli that follow the response (that is, the reinforcer)”. According to [Rescorla 1992], the context, (or occasion setter), serves the role of controlling which R→O associations are functional. It is thus understood that animals form R→O associations, which are modulated by occasion setters.

2.3.5 Second order conditioning and response chains

Under Pavlovian learning theories, it seems rather restrictive that animals merely learn to associate neutral CSs with USs on a first-order level. The same can be said for in-

strumental conditioning, where responses become associated with USs (more on this in the next paragraph). Indeed it has been shown experimentally that animals do form associations between stimuli that they have no innate aversion or excitation towards [Pearce 1997]. This process is known as second-order conditioning; the formation of associations between two stimuli, "even when neither of them has any unconditioned properties" [Pearce 1997, p. 42]. An example of second-order conditioning is when a light signals a tone which in turn signals food. In this case only the food is a US. However, by way of being associated with the US the tone attains associative properties, so that it can, by itself, be associated with the signal.

For instrumental conditioning a similar process to second order conditioning occurs [Mazur 1986]. The general principle, as in second order conditioning, is that otherwise neutral stimuli becomes conditioned reinforcers. The example given by [Mazur 1986] is that of a procedure called backward chaining, in which animals are conditioned to perform a complex response chain. First, the animal is conditioned to pair a response with an unconditioned stimulus, such as pressing a lever to get food. When the association between the response and the outcome is strengthened, the mere sight of the lever will act as an occasion setter, (or a discriminatory stimulus), for the response. Because the occasion setter has been paired with food, it is now said to be a conditioned reinforcer, as opposed to the food which is an unconditioned reinforcer. The conditioned reinforcer will then attract responses by itself, resulting in a new R→O association. Going further back a step, another stimulus can become a discriminatory stimulus for the new R→O association. A response chain is said to have formed. Obviously, learning will be more rapid the nearer the unconditioned reinforcer one gets, but the reinforcing properties of the final outcome will propagate backwards through the chain over trials, and consequently a complex sequence of responses can be learned gradually. In the words of [Mazur 1986, p. 135]: "Each stimulus in the middle of the chain serves as a conditioned reinforcer for the previous response and as a discriminative stimulus for the next response. [Dayan & Abbot 2001] describes an experiment where a rat is placed in a water maze, filled with blurred milky water, the aim being to find a platform at the other end of the maze, (a goal that rats will exhibit under these circumstances due to their natural aversion of water). At first the rats will make many mistakes, but as they are allowed several trials, errors will reduce near to the platform, (an error being that of walking a longer path than necessary). The errors then reduce gradually over trials backwards through the maze, until the rats have finally learned the shortest path. In this example each crossroad in the maze acts as discriminatory stimulus for the response of taking a certain direction. Additionally, the crossroad acts as a conditioned reinforcer to the previous response in the chain.

2.4 The concept of stimulus

In Psychology a stimulus consists of a set of elements, where an element is some property of the stimulus such as, in the case of *e.g.* light; strength, intensity, frequency, *etc* [Pearce & Bouton 2001]. On the other hand a set of stimuli is called a compound. Generalization is the transfer of associations to new, but similar situations. [Mazur 1998, p. 321] notes that "transfer is most likely to be found when two tasks involve

similar or identical movements in response to a similar stimulus situation". In classical conditioning, when a CS has been paired with a US, similar CSs will predict the same US with associative strength according to the similarity. That is, CSs that are similar to the original CS will elicit a CR which is similar to the response elicited by the original CS [Mazur 1998]. For a computational model, the measurement of similarity between two stimuli is a problem of choosing the stimuli representation. If the stimulus is represented as a vector, where each element represent some property of the stimuli, then these properties constitute modalities, each possibly having some continuous value. By far the simplest account of a stimulus is where it is represented by a single property, and where the stimulus vector represent different values of this property. Because values are limited, there will be a maximum and minimum value. Generalization to similar stimuli can then be determined by the absolute distance between property values. Because animal learning theories say little about the representation of stimuli, a computational model have quite a lot of freedom in terms of representation. This creates many open questions. Of course, stimulus representation is dependent on the environment and the problem being solved. The need for engineering of the representation will therefore remain.

2.4.1 The role of attention in learning

Associative learning is based on the associations being formed between events, such as CS and US. Depending on the salience of the stimuli, they have different learning rates, and therefore different rates of conditioning. The initial associativity of a stimulus is determined by its salience [Rescorla & Wagner 1972], but during the course of learning the associativity will change [Pearce & Bouton 2001]. Experimental evidence by [Lubow & Moore 1959] showed that non-reinforced exposure of a CS caused its associativity to decline, which was illuminated by the slowing down of conditioning on subsequent reinforced trials. Several theories aim to explain and model how this change in associativity takes place.

[Wagner 1981] explains the change in associativity by his theory of Standard Operating Procedures in memory (SOP). Under this model of learning, a stimulus can be in three states; A1, A2 and inactive. When a stimulus is in the A1 state, it is said to be the focus of attention, whereas an A2 state is at the margin of attention [Pearce & Bouton 2001]. Stimuli in the inactive state do not participate in conditioning. The only transition from inactive state to the A1 state is by direct perception of a stimulus. If there is an association between the perceived stimulus CS_{A1} and a successive stimulus, the successive stimulus will enter into the A2 state. Alternatively, a stimulus in the A1 state will eventually decay to the A2 state. Stimuli in the A2 state can only move into the inactive state. The point of the SOP theory is that only stimuli in the A1 state can gain associative strength. When a stimulus is repeatedly presented with no subsequent reinforcer, it is assumed that the context, (also a stimulus), in which the stimulus is presented enters into an association with that stimulus (context $\rightarrow$ CS). Thus the CS will be in the A2 state by way of prediction from the context, and association with a subsequent US is therefore retarded.

[Mackintosh 1975] postulates that the associability of a stimulus is determined by the discrepancy of its associative strength and the asymptote of the reinforcer, $(\lambda - V_A)$,

which is compared with the discrepancy of the associative strength of the compound, (except stimulus A), and the reinforcer, ($\lambda - V_{compound}$). Stimulus A is considered a good predictor of the reinforcer if the discrepancy of stimulus A is less than that of the compound. If the discrepancy of stimulus A is greater than or equal to that of the compound, it is considered a poor predictor, and consequently its associability will decrease [Pearce & Bouton 2001]. During non-reinforced trials, the stimuli that accompany stimulus A, will predict the absence of a reinforcer equally well, and therefore there will be loss of associability in stimulus A.

In Pearce & Hall's [1980] theory it is proposed that the associability of a stimulus A on trial t, is determined by the absolute value of the discrepancy, (on trial $t-1$), between the asymptote of the reinforcer and the sum of associative strength of all stimuli in the compound, including stimulus A. The associability of a stimulus will thus be high when it was succeeded by an unexpected reinforcer on the previous trial, and conversely it will be low when succeeded by an expected reinforcer. More formally [Pearce & Bouton 2001]:

$$\alpha_{A_t} = \mid \lambda - \sum_{i=0}^{n} V_i \mid_{t-1}$$

2.4.2 Elemental or configural associations

As have been mentioned earlier, animal learning theories consider states as consisting of compounds of stimuli. Specifically, several CSs can signal a US, because there will be several stimuli present in the environment at any time. According to the Rescorla-Wagner model [Rescorla & Wagner 1972], each element (stimulus) of the compound enters into an association with the US, and thus acquire associative strength according to the CSs and USs salience. This is a so-called elemental theory of associative learning [Pearce & Bouton 2001], where each element (stimulus) competes for the associative strength available [Gabriel & Moore 1990], as determined by the asymptote λ. A purely elemental associative learning model suffers from the inability to solve the XOR problem [Pearce & Bouton 2001]. In Artificial Intelligence, the XOR problem is manifested by the limitation of the linear Perceptron, which cannot discriminate a conjunction of elements as signalling the opposite of each element by itself [Negnevitsky 2002]. More formally, the Perceptron is a linear function of n variables, n corresponding weights, and a hard limiter θ [Negnevitsky 2002]:

$$f(x_1, x_2, \ldots, x_n) = \sum_{i=1}^{n} x_i w_i - \theta$$

$$output(f(x_1, x_2, \ldots, x_n)) = \begin{cases} 1 & \text{if } f(x_1, x_2, \ldots, x_n) \geq 0 \\ 0 & \text{if } f(x_1, x_2, \ldots, x_n) < 0 \end{cases}$$

Comparing the Perceptron to the Rescorla-Wagner error correction rule (2.9), the variables in the preceding equation correspond to binary CSs (denoting presence), and the weights determine the associative strength of each CS with regards to the US asymptote. If each individual CS input has a positive weight, and thus cause positive output,

then the Perceptron will always have positive output for any combination of the individual CSs. It cannot encode weights so that a compound of CSs will cause negative output, if each member of the compound already causes positive output. This is the XOR problem. In Psychology, the XOR problem is known as negative patterning. In negative patterning, two stimuli, A and B, are succeeded by a US, when presented alone, and hence they both acquire positive associative strength. Then, the two stimuli are presented as a compound, AB, but no US succeeds the compound. According to the Rescorla-Wagner equation (2.9, this would decrease the associative strength of both stimuli. However, this is incorrect as noted in [Pearce & Bouton 2001]. Therefore, configural theories postulate that the compound creates a new unique element, entering into an association with the US on its own. The "configuration" thus acquires negative associative strength, and does not interfere with any of the compound elements when they are presented by themselves. Similarly, a pair of Perceptrons can solve the XOR problem:

$$x_1, x_2, \ldots, x_n \in \{0, 1\}, w_1, w_2, \ldots, w_n \in \{1\}$$

$$f_1(x_1, x_2, \ldots, x_n) = \sum_{i=1}^{n} x_i w_i - \theta - f_2, \theta = 0.5$$

$$f_2(x_1, x_2, \ldots, x_n) = \sum_{i=1}^{n} x_i w_i - \theta, \theta = \big(\sum_{i=1}^{n} w_i\big) - 0.5$$

The output of a Perceptron can be seen as the strength of the conditioned response. When the compound is presented, it is cancelled by f_2, which serves as a NAND function of the compound elements.

2.5 Inconsistencies between RL and AL

The preceding paragraphs show evidence of a theoretical and practical gap between Reinforcement learning and associative theories of animal learning. The following paragraphs serves to recapitulate the inconsistencies as described above.

2.5.1 States and generalization

It dares to be repeated that Reinforcement learning places no particular emphasis on the analysis of what is included in the state signal, i.e. the representation of states. The definition of Q-learning leaves representational issues to the application designer. In associative learning theories utilizing the elemental approach, the state is a compound of stimuli, each attaining or losing associative strength over trials. In contrast to reinforcement learning, where the state as a whole, (i.e. the compound is considered as a configuration), enters into an association with the succeeding state, associative learning divides the state signal into stimuli, each entering into an association with the succeeding stimuli (there can be more than one). There will thus be transfer of associative strength between different states, to the degree which they share stimuli, thereby creating a savings effect.

2.5.2 Nature of reinforcers

In reinforcement learning reinforcers/inhibitors are not considered to be part of the environment [Sutton & Barto 1998]. Rather they are presented to the agent in order to guide its behavior, that is, as rewards or punishments[2]. In that sense they are separated from the resulting state s' after performing an action in state s. The performed action in state s is reinforced or suppressed upon reaching the resulting state s'. In Psychological terms an association between the state s and the action a is formed, whereas the desired state s' acts as a reinforcer or inhibitor. There is no direct equivalent to the concept of return (from RL) in Psychology; a return is just another form of stimulus having the quality of a return [Alonso & Mondragón 2004]. Hence Psychology makes no distinction between returns and Outcomes, whereas in Reinforcement Learning these are separate entities [Sutton & Barto 1998].

2.5.3 Types of learning

According to [Alonso & Mondragón 2005] the inconsistency between RL and AL lies in the types of associations that their respective models describe, as well as the abstraction of returns from the environment. As is shown in experiments from instrumental conditioning, animals form "S→R"-, "S→O"-, and "R→0"-associations. Additionally, according to classical conditioning, animals also form "S→S"-associations without regards to any instrumental action. As a result of the definition of Markov Decision Processes, and the way in which an optimal policy is defined, a RL agent merely forms "S→R"-associations, which, as is hypothesized by [Alonso & Mondragón 2005], restricts the effectiveness of learning.

2.6 Former integrations of Reinforcement Learning and Associative Learning

As described earlier, reinforcement learning is based on theories from associative learning, so there is no reason why this synthesis of computer science and psychology should not be taken further. [Sutton & Barto 1990] define their temporal difference model on the basis of Pavlovian conditioning, and in fact it is a time-derivative model of the Rescorla-Wagner model [Rescorla & Wagner 1972]. Later, little has been done to extend the framework upon which reinforcement learning is dependent on, MDPs, but some theories from associative learning has been applied in robotics. For example [Touretzky, Daw & Tira-Thompson 2002] take into consideration problems of inter stimulus intervals during learning. Also, [Saksida, Raymond & Touretzky 1997] have used principles from instrumental learning to shape robot behavior. Thirdly, neuroscience often use associative models of learning to confirm their brain-level learning models. It is believed that if a neuronal model can exhibit the same learning phenomena as associative models, it has some objective basis of support. [Dayan & Abbot 2001]

[2]This implies an external teacher to define the amount of reinforcement given upon entering a state, not an easy task in itself, because it effectively means that the external teacher needs to have enough knowledge about the structure of the environment to be able to guide the behavior in the most effective way.

provides an extensive account of computational neuroscience. The synthesis of reinforcement learning and associative learning, which is aimed at in this project, is not trying to bridge analytical gaps in the theory of associative learning. The intention is to use animal learning models to overcome problems in reinforcement learning. This view has been proposed by [Alonso & Mondragón 2005].

Chapter 3

Methods

3.1 Methodology

In what can be described as a search for psychological models of learning that can be exploited in order to decrease the theoretical gap between Reinforcement Learning and Associative Learning, requirements have been gathered in light of the objectives of this project. The model for association formation in Reinforcement Learning is based exclusively on early theories of instrumental conditioning, namely Thorndike's "Law of effect" [Thorndike 1911][1]. It is evident that Psychology has made a lot of progress since these early theories appeared. Hence there is plenty of theory to draw from in order to extend the Reinforcement Learning model with novel theories from Associative Learning. In this chapter the work process that have been carried out in order to gather requirements and design a new learning algorithm will be described, as well as the experiment design carried out to test the merits of the new model relative to the Q-learner agent. The following list briefly summarizes the work process:

- Requirements gathering: Meetings with associative learning expert and supervisor. Reading literature on conditioning in order to understand psychological models of learning. Gathering requirements for a new (extended) computational model of learning.
- Analysis of requirements: Understanding how associative theories of learning can improve reinforcement learning (specifically Q-learning).
- Design of algorithm and model according to requirements analysis. Planning of integration of model with Grid world testing environment.
- Implementation of new learning model, Q-learner, and Grid world testing environment in Java.
- Planning of testing and experiments; procedures to verify the model.

[1] The "Law of effect" states that an association between a response and a preceding stimulus will be strengthened or weakened if succeeded by an excitatory or aversive outcome respectively.

3.1.1 Requirements of model

The requirements of the new learning model are loosely given by the objectives of this project. Therefore, when approaching the problem, the objectives have served as a guideline of where to apply associative theories of learning. Following this line the objectives are again presented here, and discussed briefly. The following objectives constitute the high-level requirements of the new algorithm. In the next section the requirements will be analyzed in order to derive a realistic model.

Incorporating psychological bias: To incorporate psychological bias in the architecture of the agent that will guide its learning process. This is a rather vague objective, which can imply many things. However, in this case it means that internal drives will be introduced, that will make the agent approach appetitive stimuli and avoid aversive stimuli. Additionally, exploration will be introduced as a specific type of drive. This is an idea of abandoning the ”tabula rasa” approach of learning which is employed in reinforcement learning. In short, this means that the agent will have some form of representation of stimuli that are aversive or appetitive from the very instantiation of the agent.

New types of associations: To endow the agent with the ability to form other types of associations other than S→R associations. Associative theory envisages three types of association:

- Stimulus-Response (S→R) associations.
- Stimulus-Outcome (S→O) associations.
- Response-Outcome (R→O) associations.

This objective is closely tied with the previous in that the expectance of a particular outcome can be explicitly anticipated when the actual outcome is part of the association. The internal drives introduced in the previous paragraph thus serve as a means to reevaluate canonical outcomes, i.e. ”naturally” aversive/appetitive stimuli.

Event representation: To take into account associative theory's conception of event representation. An event is anything entering into an association with something else; a stimulus, a response, or an outcome. The question is then how to represent stimuli and responses.

Redefine outcomes: To redefine outcomes as comprising sensorial and motivational elements. As have been discussed in the literature review, there are two approaches to stimuli representation; one being elemental and the other being configural. Reinforcement learning (and Q-learning) uses a configural representation of outcomes by default, the states being considered irreducible entities. Associative learning, on the other hand, favors an elemental representation. This means that outcomes will consist of several stimuli, each of which can be either sensorial (neutral) or motivational (instinctively aversive/appetitive).

Amend the conditions of association formation: To take into account the fundamental conditions of association formation proposed by associative theory. Under the elemental approach of association learning there are other rules governing the acquisition of associative strength (Rescorla-Wagner update rule), and the change in associability of stimuli over time. These have been described in the literature review. More specifically, the Rescorla-Wagner error correction rule (equation 2.9), will be incorporated in the model, and also the change in associability as a derivative of the surprisingness of the outcome (see equation 2.10).

Requirements of testing environment

In addition to the requirements of the learning model, a number of requirements have been set for the learning simulator, i.e. the testing environment; the Grid world. The Grid world is a n x n board of squares (also known as spatial locations). At any moment the agent is situated in one of the squares. To move from square to square, the agent has a repertoire of 8 possible intrinsic responses, namely the movements going out from the current spatial location: East, South East, South, South West, West, North West, North, and North East. A response leading out of the Grid world, i.e. at one of the edges, will result in no movement. The Grid world is episodic, meaning that there is a start location and a goal location. Upon reaching the goal location, the agent will be moved back to the start location automatically.

Usually, in a Grid World environment, the x,y-coordinates are used as the state description. That is, if the agent is at spatial location (3,2), then the x,y-values will be used as the state description. This ensures that the Markov property holds, but does not say anything more about the actual state than its actual location. In associative learning, the state consists of stimuli, each of which can be anything perceivable by the agent, and the spatial location itself is not taken into account explicitly. There is therefore a need to define stimuli in spatial locations, implying an interface to the Grid World making this possible. This also means that the agent will perceive the stimuli compound present in the current spatial location, instead of the x,y-coordinates.

3.1.2 Analysis and implications of requirements

The high-level requirements presented in the previous section provide a guideline for the analysis in this section. A dissection of the requirements is therefore given below, which will describe the thought-process behind the actual design of the learning model. Questions that were posed include:

- What do the requirements mean?
- What do they imply?
- How do they translate into a realistic and doable design?

The most important question, however, is how the requirements can improve Q-learning.

Incorporating psychological bias

The psychological bias aimed at being introduced concerns the definition of drives, motivational state, exploration, and how appetitive or aversive a stimulus is. In the following list, these concepts are analyzed.

- Drives (Unconditional stimuli): A drive is something that causes behavior in the agent, and that is built in to the agent. In the simple definition of the Grid world, the only behavior known to the agent is movement. This is no different from the definition of a Reinforcement Learning agent; it also has built in actions/responses. It does not have drives, however. A Reinforcement Learning only seeks to maximize a reward signal, and consequently it does not really know what it wants. The reward signal is an abstraction of the outcome defined by the environment. On the other hand, a drive is something specific. It represents something that the agent wants or does not want; a goal or a danger. Because of the trivial set of responses available to the agent in the Grid World, the only behavior that these drives can provoke is approach or avoidance. The agent will approach appetitive stimuli and avoid aversive ones. A drive is thus an unconditional stimulus.

- Motivational state: Even though the agent knows intrinsically that certain stimuli are appetitive or aversive, it is not certain that this will cause behavior, because some need might recently have been satisfied. Without getting too speculative, a comparison to an animal might be appropriate. After an animal has eaten enough to be full, it will have no need to eat more until it gets hungry again. Similarly an agent which knows that certain stimuli are appetitive (i.e. cause approach), might have collected enough of these stimuli to satisfy its current need, and consequently will not be motivated to approach these stimuli until it has consumed those previously collected. However, this mechanism is beyond the scope of this project. For the simple model presented in this thesis, the motivational state is constant. The agent will always be motivated to approach appetitive stimuli and avoid aversive ones.

- Exploration: Closely intertwined with motivational state is exploration. In reinforcement learning, exploration is needed in order to improve the current policy, or in other words to collect the most rewards with the least amount of effort. In associative learning, the concept of behavior optimization is not explicitly analyzed. This is in fact why reinforcement learning was conceived. [Sutton & Barto 1990] provide a real-time model of associative learning, where the goal is to maximize associative strength. Associative strength is viewed as an expectation of reward. In the Grid World the distribution of associative strength over stimuli will always be greatest over the shortest path. Therefore the agent should prefer behavior which minimizes aversive outcomes and which lead to appetitive behavior with the least amount of movement. Exploration can be defined in two ways, (not exclusively), for an associative learner. E.g.:

  - A possibility is to use an explorative policy from reinforcement learning like e.g. ϵ-greedy

  - Alternatively, the agent's drive to explore can be modulated by the increase and decrease in the quantity of appetitive outcomes, as well as the increase in aversive outcomes. If the agent encounters many aversive outcomes, it should try to change its situation by adjusting its behavior; exploring new responses. On the same theme, if the agent is already doing quite well by approaching many appetitive outcomes, there is always the possibility that a change in behavior might lead to a better situation. Therefore the agent should always maintain some exploration to keep an "open mind" towards better policies.

- An implication of the maximization of appetitive outcomes, is that the avoidance of appetitive stimuli is considered aversive. Conversely the avoidance of aversive stimuli is considered appetitive.

New types of associations

- S→R associations, as in RL, fails to anticipate the actual outcome. These associations merely anticipate a reward. If the outcome loses its value, there is noting to tell the agent explicitly of this. In RL, the derived reward value has to change. Thus the responsibility is placed on the environment to reevaluate the outcome.

- S→O and R→O associations predict the actual outcome, and due to the agent's internal drives and motivational state, the agent itself is able to place a value on the outcome. There is a shift in responsibility. The agent has to reason about outcomes, i.e. derive the value of outcomes instead of being provided with rewards.

Event representation

In reinforcement learning the event representation, whether that is a state or a response, is some symbolic value facilitating function or table lookup. For instance with Q-values the state and response are used as indices for function or table lookup in order to store or retrieve the Q-value. When changing the event representation one needs to consider what the changes imply for the design of the model. The response representation remains unchanged for the Grid world environment; it is a numerical value denoting one of the possible movements. State representation on the other hand, will be completely redefined. The reinforcement learning agent will still depend on a unit representation of each state, whereas the associative learning agent will be given a list of stimuli upon entering a spatial location. After consulting literature from associative learning, it has been found that a stimulus can be represented as some modality having a strength property. A modality is a symbolic value, and the strength property of the stimulus denotes the salience of the stimulus, e.g. modality 4 with strength 9. A stimuli compound found in a spatial location is then a list of modalities with corresponding strengths. For the associative learning agent, each element in the list is considered as an event, whereas for the reinforcement learning agent, the whole list is considered as one event. The modality of a stimulus in this thesis is merely a numerical abstraction; it can represent anything. Contrarily, the strength property of the stimulus is important for generaliza-

tion and transfer of associative strength between similar stimuli, where two stimuli are considered similar if they are of the same modality and have near equal strength.

Redefine outcomes

Outcomes are redefined as consisting of both motivational and sensorial stimuli. In reinforcement learning, rewards define the value of an outcome. Conversely, in associative learning, the agent itself will define the value of an outcome. As described above, this value will be derived on the basis of internal drives. Stimuli encountered will therefore be compared with the internal drives, and the similarity will be used as the basis of value. Some stimuli will match the drives, whereas others will be neutral. The stimuli that match the the internal drives will be the basis of first-order conditioning. Hence they serve as unconditioned stimuli that otherwise neutral stimuli can become associated with. A neutral stimulus that has become associated with an unconditioned stimulus is said to have gained predictive value. It now predicts the aversive/appetitive outcome. From this basis it can now become the subject of second-order conditioning, i.e. its predictive value can propagate to preceding stimuli.

Amend the conditions of association formation

As have been accounted for in the description of temporal difference learning, Q-learning is a predictor of future reward, i.e. state-action values accumulate future reward following a specific policy from the current state and onwards. This prediction is estimated by the Q-value update equation:

$$Q(s,a) \leftarrow Q(s,a) + \alpha\big[r' + \gamma \max_{a'} Q(s',a') - Q(s,a)\big] \tag{3.1}$$

In equation 3.1 the term $\big[r' + \gamma \max_{a'} Q(s',a') - Q(s,a)\big]$ represents the temporal difference error between the current state-value estimate ($Q(s,a)$) and the new state-value estimate ($r' + \gamma \max_{a'} Q(s,a)$). When comparing the Q-value update equation to the Rescorla-Wagner equation for error correction of associative strength, several differences, but also some similarities can be noted. The Rescorla-Wagner equation is repeated for reasons of comparison with Q-learning [Rescorla & Wagner 1972]:

$$V\Delta_{CS} \leftarrow \alpha\beta\big[\lambda_{US} - \sum_{i=0}^{n} V_{CS_i}\big] \tag{3.2}$$

Differences

	Associative learning (Rescorla-Wagner model)	Reinforcement learning (Q-learning)
How learning rate is determined	Learning rate is dependent on the salience (i.e. modality strength) of the conditioned stimulus (α) as well as on the salience of the unconditioned stimulus (β). Consequently, the learning rates are dynamic and dependent on the situation (i.e. the salience of the stimuli).	The learning rate α is not dependent on the situation, and is often held constant over the whole learning period.
Changes in learning rate	In addition to being initially determined by the salience of the stimuli, the learning rate of the conditioned stimulus varies according to the absolute discrepancy of the US asymptote and the sum of associative strength of all conditioned stimuli present (see equation 2.10).	There is no change in learning rate according to the surprisingness of an outcome (reward).
Association formation	The Rescorla-Wagner model derives the change in associative strength of one of the conditioned stimuli, on the basis of the discrepancy between the US asymptote λ and the sum of associative strength of all conditioned stimuli present. Stimuli are thus competing for the associative strength available (as determined by the asymptote λ). Consequently two different situations can share stimuli and thus get a savings effect by transfer of associative strength to the new, but similar situations.	Q-learning considers states as irreducible entities, and updates the reward prediction based merely on two states (or stimuli compounds); the current one and the next. There is no transfer of reward prediction from one state to the other, even though they might be similar and share elements.

	Associative learning (Rescorla-Wagner model)	Reinforcement learning (Q-learning)
Second-order conditioning	The Rescorla-Wagner model does not account for second-order conditioning explicitly, although as described later, there is nothing to suggest that a neutral stimulus cannot take on asymptotic value and thus gain US qualities. This can happen when a previously neutral stimulus becomes a strong predictor of an US. The associative strength of the CS can then be interpreted as the asymptote λ. By this account, any stimuli can become a US.	One of the things Q-learning explains well is second-order conditioning. In the beginning of a learning period, every state is neutral. As the agent explores the environment it will eventually encounter a highly rewarding state. The reward value from this state will thereafter be propagated backwards in time to previously neutral states which now become predictors of reward. The definition of the Q-value update (equation 3.1 with bootstrapping shows that second-order conditioning is the basis of temporal difference learning.
Instrumental learning	Classical conditioning, which the Rescorla-Wagner model caters for, does not consider operant behavior, that is, the discriminatory response needed in order to reach an outcome. However, the Rescorla-Wagner equation can be used to encode other types of associations as well, including R→O- and S→(R→O)-associations. This is done by considering the respective elements, responses, stimuli, and outcomes, more generally as events [Mackintosh 1983].	Obviously, Q-values are defined by state-action pairs, and hence Q-learning encodes S→R associations (i.e. instrumental learning).

	Associative learning (Rescorla-Wagner model)	Reinforcement learning (Q-learning)
Outcome categories	Classical conditioning and the Rescorla-Wagner model do not differ in whether outcomes are aversive or appetitive. Both categories of outcomes may enter into associations with neutral stimuli. A stimulus can equally well be a predictor of an aversive outcome as an appetitive outcome. It shall be noted, however, that in the case of the Grid World, associations with appetitive outcomes will probably tend to be stronger due to the fact that aversive outcomes will be avoided, leaving less chance to grow associative strength with aversive outcomes. The Rescorla-Wagner model can thus be seen as a predictive model of the world; it merely predicts outcomes and does not say anything about the reward value of an outcome. This value is separated from the model, and is more related to instrumental learning; in order to learn correct behavior, the agent has to value the outcomes in terms of reward.	In reinforcement learning, the appetitiveness or aversiveness of an outcome is determined by the amount of reward it predicts. Q-learning is nothing more than a model of the correct behavior, and does not describe the relationships between states in the environment. S→R-associations describe habits; that is, simple reflexive actions determined by the amount of reward predicted by state-action pairs.
Policy evaluation	Associative learning is on-policy; the agent learns to associate events from its actual behavior.	Q-learning is off-policy; it always estimates the optimal policy, but behaves sub-optimally in order to explore new states.

Similarities

- Both the Q-value update rule and the Rescorla-Wagner equation base their estimation of future reward/associative value on a discrepancy between the current value and the ”correct” value (the asymptote).
- The asymptotic value λ used in the Rescorla-Wagner model can be interpreted

as future reward.

- Both models learn more when value discrepancies are large, i.e. they both learn more from surprising events than from less surprising events. This is due to the error correction term used by both models.

3.1.3 Design of model

This section carries on from the analysis section to provide a technical account of the design of the learning algorithm, and the Grid World learning simulator. This detailed account is then used to implement the model.

As described earlier, in RL the state signal is an irreducible entity. For this reason the agent cannot extract or infer which parts of the state are actually important. On the same theme as the reward-environment abstraction problem (see section 2.5), this has some unfortunate consequences for RL:

- The ability to reason over elements present in the current state signal is non-existent
- Some elements present in the state signal will often be more important than other elements
- The reward-environment abstraction problem is a direct consequence of considering states as irreducible entities

States vs. Stimuli compounds

It is proposed that a state S is redefined as consisting of a set of stimuli Sc. Several stimuli can be present at any time in any spatial location. This is known as a stimuli compound, (also known as the state signal in RL). Consequently, the Markov property (2.8) will be relaxed from the point of view of the agent. Because different stimuli, all present in the same spatial location, can be of different importance to the agent, it is the responsibility of the agent to encode its own state signal by extracting the important elements from the stimuli compound. Moreover, as can be deduced from this argument, the associative model of the environment will be more compact and general because elements can be shared across spatial locations. The stimuli compound is defined as a vector of elements, where each element of the stimuli vector represents some modality, along with a value property denoting its strength:

$$\begin{aligned} Sc &= \langle s_1, s_2, \ldots, s_n \rangle \\ s_n.modality &= \{0, 1, \ldots, m\} \\ s_n.value &= [0, max\_strength] \end{aligned}$$

Rewards vs. outcome values

In reinforcement learning the agent receives a reward upon entering a new state. Hence it does not know why the new state is good or bad; it only knows the immediate value of

entering that state. The only information available to an agent about a state in reinforcement learning is the amount of reward it predicts. Therefore the agent is completely dependent on the environment to provide the correct reward values in order to guide its behavior. The agent has no way of reasoning about the states in terms of its internal needs, because it has no internal needs other than reward maximization. As a consequence, the agent doesn't really understand the problem it is trying to solve; it only understands reward maximization. To remedy the reward-environment abstraction problem, where rewards are abstractions of the value of outcomes, the proposal is to redefine the value of an outcome as a function of the agent's aversion or excitation towards that outcome. In other words the agent provides a mapping between a stimulus and its value, depending on the agent's motivational state (AMS) at time t. This proposal contradicts Reinforcement Learning, where the value of an outcome is provided explicitly and separated from the actual outcome in the form of a reward. More formally:

$$\lambda_{US_t} = f(ams_{t-1}, US_{t-1}) \tag{3.3}$$

The definition of equation 3.3 is a matter of further analysis. However, it is clear that the agent must have some representation of what it wants, rather than the designer having to define λ explicitly. If λ was to be defined explicitly it would defeat the purpose of redefining rewards as stimuli with motivational value. In order to understand how the Stimulus Value function should be defined, its desired properties must first be established. First, it should manifest some canonical instinctive aversion or excitation towards certain stimuli. Secondly, by way of becoming associated with such stimuli, otherwise neutral stimuli must be able to take on the same properties. This is paramount if rewards are to be anticipated. It is therefore suggested that the motivational value of stimuli is determined by the agent's internal drive. A drive is a primary instinct in the agent, one that it possesses "naturally", that is, from instantiation. Because stimuli compounds have already been defined as vectors, it is logical to define drives as vectors as well. This means that a drive is essentially the agent's internal representation of a stimulus. By this suggestion it will be straightforward to compare a drive with a stimulus. The question then remains how an otherwise neutral stimulus, one that the agent has no natural excitation or aversion towards, will take on properties that will evoke a representation of a drive. A neutral stimulus is by definition similar to a non-neutral stimulus in that they both are made up of the same elements; modality and modality-strength. One possible solution would be to add the neutral stimulus to the set of drive vectors present in the agent, but this would be too memory consuming. Instead, two possible solutions are proposed:

1. A trace of the previously perceived stimuli compound. Any associative strength gained by the current stimuli compound with its successor, will be transferred to the previous stimuli compound along the behavioral path of the agent.

2. A lookahead function. Upon encounter of a neutral outcome A, the lookahead function will find the *maximum* associative value of A. This value will then be transferred to the stimuli preceding A.

The role of the similarity function and internal drives

In order to generalize between stimuli, and to compare stimuli with internal drives, a similarity function is needed. This function should return a value between 0 and 1, when comparing two stimuli, where 0 is no similarity and 1 denotes equality. The Gaussian function has been found to be a good match for the purpose. Let the modality strength of stimulus A represent the mean μ of a normal distribution ND with standard deviation σ, and let x_B and x_A represent the strength of stimulus B and A respectively. The probability of stimulus A $P(x_A)$ occurring in ND is given by:

$$P(x_A) = \frac{1}{\sigma\sqrt{2\pi}} e^{\frac{-(x_A-\mu)^2}{2\sigma^2}} \tag{3.4}$$

Now, it is possible to calculate the probability $P(x_B)$ that a second stimulus B occurs in the same normal distribution:

$$P(x_B) = \frac{1}{\sigma\sqrt{2\pi}} e^{\frac{-(x_B-\mu)^2}{2\sigma^2}} \tag{3.5}$$

This leads to the definition of the similarity function:

$$sim(A,B) = \frac{P(x_B)}{P(x_A)} I(A,B) \tag{3.6}$$

$$I(A,B) = \begin{cases} 1 & \text{if A.modality = B.modality} \\ 0 & \text{if A.modality} \neq \text{B.modality} \end{cases}$$

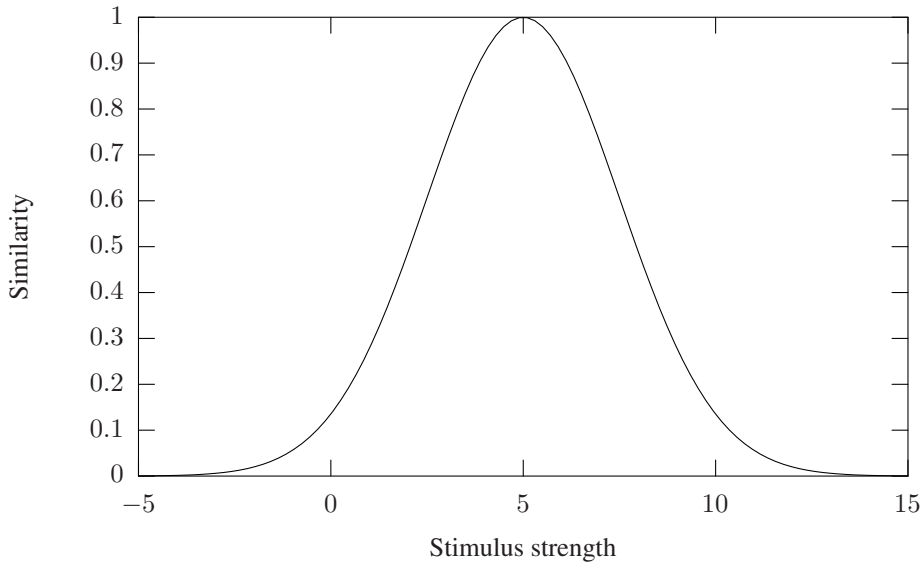
Thus the similarity function is bell-shaped, continuous, and insensitive to the sign of the difference between the value of two stimuli. This can most readily be seen from the plot in figure 3.1.3. The sensitivity of the similarity function is controlled by the standard deviation σ, where if $sigma$ is small the similarity function will allow for only small differences in modality strength, and contrarily it will be less specific for larger values of σ.

Moreover, for the purpose of deriving outcome values, the internal drives (ID) of an agent is a vector of stimuli, along with a category-indicator for each element denoting whether the stimulus is aversive or appetitive:

$$\begin{aligned} ID &= \langle s_1, s_2, \ldots, s_n \rangle \\ s_n.modality &= \{0, 1, \ldots, m\} \\ s_n.value &= [0, max\_strength] \\ s_n.category &= \{aversive, appetitive\} \\ aversive &= -1 \\ appetitive &= 1 \end{aligned}$$

Accordingly, the function for deriving the asymptotic value of an outcome (for first-order conditioning) can now be defined. As accounted for in the previous section, the motivational state of an agent is constant; i.e. it will always have a need to approach

Figure 3.1: The similarity function with a mean μ of stimulus strength 5



appetitive outcomes[2]. For first-order conditioning the asymptotic value of an outcome (US) is thus set in the following way:

$$\lambda_{US} = \max(\forall s \in ID : sim(s, US)) \tag{3.7}$$

The value of an outcome is the maximum similarity when comparing to all internal drives. For the new learning model, the asymptote λ thus takes the place of reward.

A causal model of the environment

It is now appropriate to introduce the way in which the causal associative model of the environment will form. As have been shown earlier the Rescorla-Wagner learning equation accounts for associations between Stimuli, that is, S→S-associations. By this account it is possible for the agent to derive a causal model of the environment, i.e. an event map of the environment dynamics. For example; stimulus A signals an Unconditioned Stimulus B. This kind of model is strictly a causal event map of the environment; it does not say anything about reward maximization, which will be discussed in the next section. It is proposed that the agent forms an associative memory of S→S-connections, merely signalling events without regard of whether they are aversive or appetitive. The question is then; why does the agent need an associative memory which

[2]However, the motivational state of the agent might be modulated in order to weight the value of outcomes differently. This is beyond the scope of this project.

does not account for reward maximization? In reinforcement learning, the associative strength of S→R-associations represent reward predictions. However, they do not say anything about the availability of outcomes. A causal model of the environment, in the form of S→S-associations, will inform the agent of the available outcomes in any situations, as well as their salience. This means that the causal model will predict important outcomes. The most important outcomes will attract the most associative strength. As stated above the causal model does not take into account reward maximization, that is, the sign value of an outcome (i.e. aversive/appetitive). Therefore associations in the causal model, S→S, make use of equation 3.7, when adjusting associative strength. Equation 3.7 can thus be plugged into the Rescorla-Wagner error correcting rule:

$$V\Delta_{CS} = \alpha\beta\big[\max(\forall s \in ID : sim(s, US)) - \sum_{i=0}^{n} V_{CS_i}\big] \tag{3.8}$$

The predictive strength of a stimulus (denoted V_{CS} in equation 3.8), will be stored as a (Stimulus,Stimulus)-pair (i.e. (CS,US)). These pairs form the *Associative Memory* (AM) of the agent. As will be shown later, the AM memory plays a role in the functional aspects of the learning model.

Reward maximization

As hinted to earlier the sole goal of Associative learning is not to build learning models that are able to solve the Reinforcement Learning problem, that is, to maximize returns in the long run. However, models of associative learning can be interpreted in a way that makes them suitable to solve this problem, and indeed it seems like animals are exhibiting some form of reward maximization over time. It is described in [Alonso & Mondragón 2005, Mackintosh 1983] that animals have, naturally, internal drives (see above). These drives cause reflexes upon perception of unconditional stimuli, as described by [Pavlov 1927]. The reflexes undoubtedly play a role in survival, e.g. in the search for food and avoidance of danger. Consequently the drives can be seen as a substitute for reward maximization in Reinforcement Learning. In order for the agent to approach appetitive Unconditioned Stimuli with the least amount of effort, it therefore needs some form of expectance towards outcome values, including outcome value sign (aversive/appetitive). In this way the agent can predict, not only the outcome, but also whether the outcome is aversive or appetitive. Reward maximization is manifested in behavior, and is derived from instrumental associations, i.e. involving responses. In reinforcement learning, the reward expectance is encoded in S→R-associations, or in other words habits. It is clear that this form of association does not allow the agent to anticipate the actual outcome. Consequently, with S→R-associations, the agent cannot reevaluate outcomes; e.g. if the outcome loses its value. An RL agent is dependent on rewards from the environment in order to reevaluate outcomes. For the new learning model it is therefore suggested to include the outcome in the instrumental associations, i.e. S→(R→O). This leads to two new properties. Firstly, the agent now has to value outcomes itself, and secondly it can now predict actual outcomes in addition to their reward value. Equation 3.8 is amended to include the sign of unconditioned stimuli, in

order to encode reward value in instrumental associations:

$$V\Delta_{CS,R} = \alpha\beta\big[\max(\forall s \in ID : sim(s, US)) \times s.category - \sum_{i=0}^{n} V_{CS_i}\big] \quad (3.9)$$

The instrumental associations of the learning model thus constitute an *Expectance Memory* (EM), encoded in S→(R→O)-associations. These associations are very specific, because they predict the amount of reward from a particular outcome, given a specific Response R and Stimulus CS. The EM memory is similar to state-action values in Reinforcement Learning, two important differences being that the outcome is included in the association (thereby forming state→action→state values), and the ability to reevaluate outcomes based on the internal drive vector (ID). The term $sim(s, US) \times s.category$ in equation 3.9 takes into account whether the stimulus s from the internal drives is appetitive or aversive, by multiplying the similarity function with the $s.category$, which takes a value of -1 for aversive stimuli and 1 for appetitive stimuli.

Second-order conditioning

So far it has been shown how first-order conditioning takes place, in the description of the associative memory and the expectance memory. However, it is paramount that second-order conditioning is catered for in the update equations of both of these memories. Second-order conditioning is basically propagation of associative strength to otherwise neutral stimuli, and is something that Q-learning does well. The two previous sections accounts for first-order associations with Unconditioned Stimuli. In order to implement second-order conditioning both equation 3.8 and 3.9 need to be amended. As described earlier, two methods have been suggested for propagation of associative strength; trace and lookahead. The trace method propagates associative strength of the current stimuli to the previously visited stimuli compound, and is therefore on-policy. On the other hand, the lookahead method propagates the maximum associative strength of the succeeding stimuli compound to the current stimuli compound, and is hence off-policy like Q-learning. Algorithm 4 and 5 describe the implementation of second-order conditioning for the trace method, with regards to the causal model and the reward maximization model respectively. In algorithms 6 and 7 the lookahead method is described for the causal model and the reward maximization model respectively. Both methods employ a decay parameter δ, which serves the same purpose as the discount rate parameter γ in Q-learning.

As can be seen from the pseudo code in algorithm 4, in lines 9 to 18, the associative strength of the current stimuli compound Sc towards each stimuli in the next stimuli compound NSc is accumulated in the variable AGS. A record is then kept of the maximum value of AGS in the variable MAS, which is then used to update the associative strength of the previous stimuli compound, in lines 19 to 25. The exact same procedure is repeated in algorithm 5, except that outcome category is utilized to take into account the sign of the outcome value.

The lookahead method, presented in algorithms 6 and 7, provides a more compact implementation of second-order conditioning. Like Q-learning, it updates associative strength on the basis of a maximum prediction. On line 8 of algorithm 6, where λ is

Algorithm 4 Trace method for the causal model

Initialize:
  $PSc \leftarrow$ Previous stimuli compound
  $Sc \leftarrow$ Current stimuli compound
  $NSc \leftarrow$ Next stimuli compound
  AGS (Aggregate Associative Strength) $\leftarrow 0$
  MAS (Max Associative Strength) $\leftarrow -\infty$
  $\lambda \leftarrow 0$
  ST (Similarity Threshold) $\leftarrow 0.85$
  $\delta \leftarrow 0.9$

For each ns in NSc
  $AGS \leftarrow \sum_{s \in Sc} V_{s \to ns}$
  If $AGS > MAS$ then $MAS \leftarrow AGS$
  $\lambda \leftarrow \max(\forall id \in ID : sim(id, s))$
  If $\lambda \geq ST$ then
    For each s in Sc
      $V_{s \to ns} \leftarrow V_{s \to ns} + \alpha\beta\big[\lambda - AGS\big]$
    Next
  End if
Next

For each s in Sc
  $AGS \leftarrow \sum_{ps \in PSc} V_{ps \to s}$
  For each ps in PSc
    $V_{ps \to s} \leftarrow V_{ps \to s} + \alpha\beta\big[\delta MAS - AGS\big]$
  Next
Next

Algorithm 5 Trace method for reward maximization

Initialize:
  $PSc \leftarrow$ Previous stimuli compound
  $Sc \leftarrow$ Current stimuli compound
  $NSc \leftarrow$ Next stimuli compound
  $R \leftarrow$ Response elicited to get from Sc to NSc
  $pR \leftarrow$ Response elicited to get from PSc to Sc
  AGS (Aggregate Associative Strength) $\leftarrow 0$
  MAS (Max Associative Strength) $\leftarrow -\infty$
  $\lambda \leftarrow 0$
  ST (Similarity Threshold) $\leftarrow 0.85$
  $\delta \leftarrow 0.9$

For each ns in NSc
  $AGS \leftarrow \sum_{s \in Sc} V_{s \rightarrow (R \rightarrow ns)}$
  If $AGS > MAS$ then $MAS \leftarrow AGS$
  $\lambda \leftarrow \max(\forall id \in ID : sim(id, s)) \times id.category$
  If $\lambda \geq ST$ then
    For each s in Sc
      $V_{s \rightarrow (R \rightarrow ns)} \leftarrow V_{s \rightarrow (R \rightarrow ns)} + \alpha\beta\big[\lambda - AGS\big]$
    Next
  End if
Next

For each s in Sc
  $AGS \leftarrow \sum_{ps \in PSc} V_{ps \rightarrow (pR \rightarrow s)}$
  For each ps in PSc
    $V_{ps \rightarrow (pR \rightarrow s)} \leftarrow V_{ps \rightarrow (pR \rightarrow s)} + \alpha\beta\big[\delta MAS - AGS\big]$
  Next
Next

Algorithm 6 Lookahead method for causal model

Initialize:
  $Sc \leftarrow$ Current stimuli compound
  $NSc \leftarrow$ Next stimuli compound
  AGS (Aggregate Associative Strength) $\leftarrow 0$
  $\lambda \leftarrow 0$
  $\delta \leftarrow 0.9$ (Decay parameter)

For each ns in NSc
  $AGS \leftarrow \sum_{s \in Sc} V_{s \rightarrow ns}$
  $\lambda \leftarrow \max(\forall id \in ID : sim(id, s)) + \delta \max(\forall o \in O : \sum_{ns \in NSc} V_{ns \rightarrow o})$
  For each s in Sc
    $V_{s \rightarrow ns} \leftarrow V_{s \rightarrow ns} + \alpha\beta\big[\lambda - AGS\big]$
  Next
Next

calculated, an extra term, $(\delta \max(\forall o \in O : \sum_{ns \in NSc} V_{ns \rightarrow o}))$, is added to the first-order conditioning definition. This term finds the maximum associative strength of the NSc compound over all outcomes predicted by NSc. Algorithm 7 is identical with 6, except that the sign of the outcome value is taken into account.

Modelling of responses and exploration

In its most trivial form the Grid-World merely allows for a simple set of actions, namely the set of directions relative to the current position of the agent, i.e. (North, North East, East, South East, South, South West, West, North West). Hence, there is no direct way of modulating the strength of a response; the agent either moves in one of these directions or not. However, it is possible to vary the response frequency given a certain stimulus. This can be done according to the expectance value of the signalling cue, i.e. the likelihood of receiving an appetitive outcome by performing a specific action. The associative strength of the signalling cue is then used as the policy setter. In order to understand the functional aspects of the learning model, a formal account is needed on how the associative memory and the expectance memory are utilized in order to derive the best response. The function definition in 3.10 summarizes the procedure for finding the best response in a given situation (cR is an abbreviation for choose Response, and the Sc parameter of the cR function is the currently perceived stimuli compound).

$$cR(Sc) \leftarrow (\forall r \in R, \forall o \in (Sc \rightarrow O) : \max_{r} \big(\sum_{s \in Sc} V_{s \rightarrow (r \rightarrow o)} \big)) \tag{3.10}$$

Statement 3.10 uses the expectance memory in order to select actions, and hence does not take into account the causal model of the world. Alternatively, the associative memory and the expectance memory can be combined together in order to weight the reward predicted by the expectance memory.

$$cR(Sc) \leftarrow (\forall r \in R, \forall o \in (Sc \rightarrow O) : \max_{r} \big(\sum_{s \in Sc} V_{s \rightarrow (r \rightarrow o)} * V_{s \rightarrow o} \big)) \tag{3.11}$$

Algorithm 7 Lookahead method for reward maximization

Initialize:
  $Sc \leftarrow$ Current stimuli compound
  $NSc \leftarrow$ Next stimuli compound
  $R \leftarrow$ Response elicited to get from Sc to NSc
  $pR \leftarrow$ Response elicited to get from PSc to Sc
  AGS (Aggregate Associative Strength) $\leftarrow 0$
  $\lambda \leftarrow 0$
  $\delta \leftarrow 0.9$ (Decay parameter)
For each ns in NSc
  $AGS \leftarrow \sum_{s \in Sc} V_{s \rightarrow (R \rightarrow ns)}$
  $\lambda \leftarrow \max(\forall id \in ID : sim(id, s)) \times id.category + \delta \max(\forall o \in O : \sum_{ns \in NSc} V_{ns \rightarrow (R \rightarrow o)})$
  For each s in Sc
    $V_{s \rightarrow (R \rightarrow ns)} \leftarrow V_{s \rightarrow (R \rightarrow ns)} + \alpha\beta\big[\lambda - AGS\big]$
  Next
Next

The associative memory tells the agent which outcomes are important, and is more general than the expectance memory, because it does not include responses in the association. On the other hand, the expectance memory encodes reward expectation, but is very specific. By combining the associative memory with the expectance memory, the learning model can more quickly adapt to small changes in the environment. This can be seen by realizing that the causal model is non-directional, that is, independent of the behavior of the agent. Therefore, if a stimulus B is adjacent to a stimulus A, and stimulus B is moved to a different location, (but still adjacent to stimulus A), the causal model will remain unchanged. In this case, the expectance memory will have to change in two aspects; first it has to decrease the associative strength of the A→(r_1→B)-association (i.e. the expectance before B was moved and response r_1 would lead to A), and secondly it has to increase the associative strength of the A→(r_2→B)-association (the expectance after B was moved and whereby the new response r_2 will lead to B). Basically, a new response has to be learned. By combining the two memories, the new association A→(r_2→B), will be weighted by the associative memory, when that memory remains unchanged. This weighting can in turn lead to a higher probability of the new response r_2 being selected, under certain conditions [3].

The previous paragraph describes the procedure for selecting the best response. As have been described in the literature review, it is important to maintain exploration. Therefore, a response other than the best one has to be chosen on some occasions. An ϵ-greedy exploration policy has been chosen for the new model[4]. Choosing the

[3] This idea has not been fully developed

[4] This means that the objective of introducing exploration as a specific drive has not been addressed extensively. However, an ϵ-greedy policy can be viewed as a random drive. Ultimately though, nothing new has been suggested for the problem of exploration. This is due to time-constraints.

currently best known response in a given situation, is known as the greedy policy. Under an ϵ-greedy policy the best response is selected with a probability of ϵ, whereas an explorative response is chosen with a probability of 1-ϵ.

Dynamic change of learning rates

In the Rescorla-Wagner equation (3.2), the two learning rate parameters α and β, (specific to the conditioned stimulus and the unconditioned stimuli respectively), are dependent on the salience of the corresponding stimuli. Additionally, according to theories by [Pearce & Hall 1980], the α parameter changes as the discrepancy between the associative strength of the CS with regards to the US becomes smaller [Pearce & Bouton 2001]:

$$\alpha_{A_t} =\mid \lambda - \sum_{i=0}^{n} V_i \mid_{t-1} \tag{3.12}$$

Along the lines of the theories of [Rescorla & Wagner 1972] and [Pearce & Hall 1980], two changes are therefore proposed. First, both α and β will be determined on the basis of the relative salience all stimuli present. Secondly, $alpha$ will be modulated by equation 3.12. The initial learning rate of a CS or a US is thus set according to the following equation:

$$\alpha_{CS} \leftarrow \alpha_{max} \frac{CS.strength}{\sum_{s \in Sc_1} s.strength} \tag{3.13}$$

$$\beta_{US} \leftarrow \beta_{max} \frac{US.strength}{\sum_{s \in Sc_2} s.strength} \tag{3.14}$$

α_{max} and β_{max} refer to the maximum values that α and β can take on respectively. The initialization of β can be solely determined by 3.14. On the other hand, α is determined by both equation 3.12 and 3.13. In order to combine equation 3.12 and equation 3.13 to determine α, the following is suggested:

$$\alpha_{CS} \leftarrow \big(\alpha_{max} \frac{CS.strength}{\sum_{s \in Sc_1} s.strength}\big) \times \mid \lambda_{US} - \sum_{s \in Sc} V_{s \rightarrow US} \mid \tag{3.15}$$

3.1.4 The algorithms

Carrying on from the previous design section, the two algorithms; "Trace" and "Lookahead", are now presented with pseudo code in algorithm 8 (Trace) and algorithm 9 (Lookahead). As can be seen from inspecting the pseudo code for the second-order conditioning methods (algorithms 4 (Trace causal model), 5 (Trace reward maximization), 6 (Lookahead causal model), and 7 (Lookahead reward maximization)), the two algorithms presented in algorithm 8 and algorithm 9 encapsulate first-order conditioning and second-order conditioning in a functional framework. This means that the algorithms interact with an environment, i.e. the Grid World. Thus the method for choosing responses (function 3.10 or 3.11), is utilized in order to find the best response given the currently perceived stimuli compound. This should all be clear from the

pseudo code, along with the design specification of the various components, (similarity function, how responses are chosen, stimulus representation, internal drives), given in the previous sections. The new learning model agent, which is termed Associative Reinforcement Learner (ARL), hence has two algorithms at its disposal (i.e. "Trace" and "Lookahead"). The relative merits of these two algorithms will be compared to the performance of Q-learning, in experiments of which the results are presented in the next chapter.

3.1.5 Implementation

The implementation of the new learning model (algorithms 8 "Trace" and 9 "Lookahead") is centered around a learning simulator framework which is implemented in the Java programming language[5]. It is not the purpose of this thesis to report all the technical aspects of this simulator. Instead the main features of the simulator are presented here, along with an illustration of the interaction between the ARL agent and the environment, as well as between the Q-learner agent and the environment. The learning simulator has the following features:

- The ability to define a template list of stimuli
- The ability to add stimuli from the template list to the vector of internal drives specific to the ARL agent
- The ability to add stimuli from the template list to spatial locations in the Grid World
- The Q-learner agent is dependent on rewards provided by the environment, whereas the ARL agent derives rewards on the basis of its internal drive vector. Hence the learning simulator is able to synchronize the environment reward structure provided for the Q-learner, based on reward definition of the ARL agent. This is paramount when comparing the ARL agent with the Q-learner agent.
- The ability to calculate the minimal path cost (MPC) of the Grid World (without obstacles). In the Grid World, as implemented in the simulator, there is a start location and a goal location. The minimal path cost is the sum of all paths (in number of movements) from each spatial location to the goal. Additionally, the path cost for both ARL agent and Q-learner agent can be calculated and compared to MPC.
- The ability to save/load Grid World stimuli layouts, ARL agent settings and internal drives, and Q-learner settings.
- The ability to save convergence results to file, as well as path costs.
- Live plotting of average fluctuation per episode in associative strength and Q-values

[5] see appendix D for user instructions, and the attached CD-ROM for running the learning simulator. Appendix B provides source code listings.

Algorithm 8 Trace algorithm

Initialize:
  $PSc \leftarrow$ Nothing (Previous stimuli compound)
  $Sc \leftarrow$ Nothing (Current stimuli compound)
  $NSc \leftarrow$ Nothing (Next stimuli compound)
  $r \leftarrow$ Nothing (Response elicited to get from Sc to NSc)
  $pr \leftarrow$ Nothing (Response elicited to get from PSc to Sc)
  $AGS \leftarrow 0$ (Aggregate Associative Strength)
  $AES \leftarrow 0$ (Aggregate reward Expectance Strength)
  $MAS \leftarrow -\infty$ (Max Associative Strength)
  $MAES \leftarrow -\infty$ (Max reward Expectance Strength)
  $ST \leftarrow 0.85$ (Similarity Threshold), $\delta \leftarrow 0.9$ (decay parameter)

For each episode
  $Sc \leftarrow$ Perceive current stimuli compound
  $r \leftarrow cR(Sc)$ (According to function 3.10 or 3.11)
  environment.doAction(r) (elicit response in environment)
  $NSc \leftarrow$ Perceive next stimuli compound
  For each ns in NSc
    $AGS \leftarrow \sum_{s \in Sc} V_{s \rightarrow ns}$
    $AES \leftarrow \sum_{s \in Sc} V_{s \rightarrow (R \rightarrow ns)}$
    If $AGS > MAS$ then $MAS \leftarrow AGS$
    If $AES > MAES$ then $MAES \leftarrow AES$
    $\lambda_{AM} \leftarrow \max(\forall id \in ID : sim(id, ns))$
    $\lambda_{EM} \leftarrow \max(\forall id \in ID : sim(id, ns)) \times id.category$
    If $\lambda_{AM} \geq ST$ then
      For each s in Sc
        $V_{s \rightarrow ns} \leftarrow V_{s \rightarrow ns} + \alpha\beta\big[\lambda - AGS\big]$
        $V_{s \rightarrow (R \rightarrow ns)} \leftarrow V_{s \rightarrow (R \rightarrow ns)} + \alpha\beta\big[\lambda - AES\big]$
      Next
    End if
  Next

  If $PSc \neq$ Nothing Then
    For each s in Sc
      $AGS \leftarrow \sum_{ps \in PSc} V_{ps \rightarrow s}$
      $AES \leftarrow \sum_{ps \in PSc} V_{ps \rightarrow (pR \rightarrow s)}$
      For each ps in PSc
        $V_{ps \rightarrow s} \leftarrow V_{ps \rightarrow s} + \alpha\beta\big[\delta MAS - AGS\big]$
        $V_{ps \rightarrow (pR \rightarrow s)} \leftarrow V_{ps \rightarrow (pR \rightarrow s)} + \alpha\beta\big[\delta MAES - AES\big]$
      Next
    Next
  End if
  $pr \leftarrow r$
  $PSc \leftarrow Sc$
  $Sc \leftarrow NSc$
Next

Algorithm 9 Lookahead algorithm

Initialize:
  $Sc \leftarrow$ Nothing (Current stimuli compound)
  $NSc \leftarrow$ Nothing (Next stimuli compound)
  $r \leftarrow$ Nothing (Response elicited to get from Sc to NSc)
  $AGS \leftarrow 0$ (Aggregate Associative Strength)
  $AES \leftarrow 0$ (Aggregate reward Expectance Strength)
  $\delta \leftarrow 0.9$ (decay parameter)

For each episode
  $Sc \leftarrow$ Perceive current stimuli compound
  $r \leftarrow cR(Sc)$ (According to function 3.10 or 3.11)
  environment.doAction(r) (elicit response in environment)
  $NSc \leftarrow$ Perceive next stimuli compound
  For each ns in NSc
    $AGS \leftarrow \sum_{s \in Sc} V_{s \rightarrow ns}$
    $AES \leftarrow \sum_{s \in Sc} V_{s \rightarrow (R \rightarrow ns)}$
    $\lambda_{AM} \leftarrow \max(\forall id \in ID : sim(id, ns)) + \delta \max(\forall o \in O : \sum_{ns \in NSc} V_{ns \rightarrow o})$
    $\lambda_{EM} \leftarrow \max(\forall id \in ID : sim(id, ns)) \times id.category + \delta \max(\forall o \in O : \sum_{ns \in NSc} V_{ns \rightarrow (R \rightarrow o)})$
    For each s in Sc
      $V_{s \rightarrow ns} \leftarrow V_{s \rightarrow ns} + \alpha\beta\big[\lambda - AGS\big]$
      $V_{s \rightarrow (R \rightarrow ns)} \leftarrow V_{s \rightarrow (R \rightarrow ns)} + \alpha\beta\big[\lambda - AES\big]$
    Next
  Next
Next

Figure 3.2: The interaction of the ARL agent and the Q-learner agent, with the Grid world environment

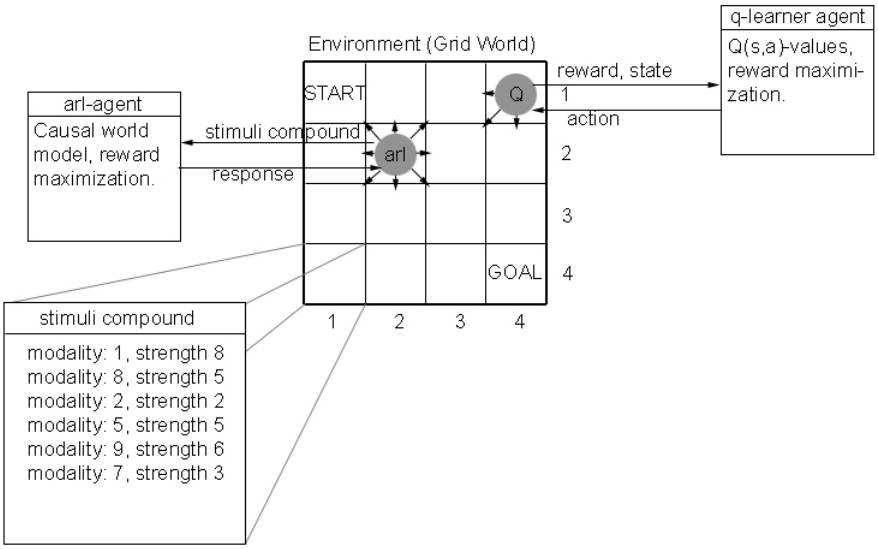


Figure 3.2 shows the interaction of the ARL agent and the Q-learner agent, with the Grid world environment. Additionally, a stimuli compound is illustrated. As can be seen from the figure the illustration shows a 4x4 squares Grid World, with the grey circles representing the spatial locations of the ARL agent and the Q-learner agent. The arrows going out from each of the grey circles denote the possible actions/responses available to the agents (i.e. the set of movements relative to the spatial locations; North, North East, *etc.*). Each aspect of figure 3.2 will be described in the following subsections. This is important in order to depict the functionality of the learning simulator.

Grid world

The Grid World is a generic environment consisting of nxn squares (also known as spatial locations). It has one start location, and one goal location, each of which can be set to any spatial location. A learning agent interacting with the Grid world will begin a learning episode in the start location and upon reaching the goal location, the learning episode is over, and the agent is automatically moved back to the start location. Each spatial location holds a vector of stimuli, i.e. a stimuli compound, that the learning agent can perceive in some way upon reaching the spatial location in question. It is possible to add/remove stimuli to/from any spatial location, through functionality in the learning simulator.

ARL agent

Upon entering a spatial location in the Grid world environment, the ARL agent perceives a stimuli compound. According to algorithm 8 (Trace) or algorithm 9, it then updates its associative memory and its expectance memory. The agent then communicates a response to the Grid world environment, which updates its record of the position of the ARL agent. Internally, the ARL agent maintains two lookup tables; one for the associative memory and one for the expectance memory. The associative memory lookup table stores (Stimulus,Stimulus)-pairs, whereas the expectance memory lookup table stores (Stimulus,response,Stimulus)-pairs. Coarsely coded arrays are used for both memories. Two stimuli can have the same modality, but different strength values. Because the strength value is continuous by definition, and computer arrays are discrete, the strength value is divided into n discrete slots. Hence the associative memory array is indexed firstly by modality, and secondly by modality strength, where the closest modality strength slot value of the array is used to index the modality strength of a specific stimulus. For (Stimulus,Stimulus)-pairs, i.e. the associative memory (AM), the array is indexed in the following way:

$$am[s_1.modality][s_2.modality][(int)s_1.strength][(int)s_2.strength],$$

where s_1 refers to the stimulus preceding stimulus s_2, and (int) refers to type casting in Java[6]. Similarly the expectance memory (EM) is indexed in the following way:

$$em[s_1.modality][(int)s_1.strength][response][s_2.modality][(int)s_2.strength],$$

where $response$ is an integer value representing any of the 8 possible responses available to the agent in the Grid World environment.

In reinforcement learning, the environment designer has control over the rewards given to the agent in a direct way. This means that responses leading to no movement (e.g. trying to walk of the edge of the Grid world) can be punished or given 0 reward value. With the ARL agent this is not possible, because it derives its own reward values. The solution for punishing responses which lead to no movement is to prevent a stimulus in becoming associated with itself. In the implementation of the ARL agent, the value update rule is therefore cancelled if the update involves an association between a stimulus and itself. This prevents the agent from trying to walk of the edge of the Grid world.

The dynamic change and initialization of learning rate parameters α and β, described in section 3.1.3, has not been implemented for either the Trace algorithm or the Lookahead algorithm. This is due to time constraints. Hence, both learning rates, α and β, of the ARL agent are held constant during learning.

Q-learner

As have been described earlier, the Q-learner does not perceive stimuli compounds in the same way as the ARL agent. Instead of forming associations with single stimuli,

[6]The strength of a modality is a real number. By casting the strength value to an integer (i.e. (int)), it can be used as an array index.

the Q-learner agent forms associations with the stimuli compound. It thus perceives the stimuli compound as a unit. The learning simulator provides the Q-learner with a configural state description for each stimuli compound. This is implemented in the following way: Given n possible modalities, each of which can take on a strength value from the set $\{0, 1, \ldots, m\}$ (due to integer type casting as with ARL), the filter function returns a string array with n items, where the item index represents modality, and the item value represents the strength of that value. The string array is then used as a unitary state description, which is utilized as a lookup key in the Q-value table of the Q-learner. Upon entering a new spatial location, the Q-learner agent receives a reward r and a state description s from the environment. In response the Q-learner agent performs an action a, which it communicates to the environment, which in turn updates its record of the Q-learner agent position. For specific details regarding the Q-learning algorithm, see algorithm 3.

3.2 Testing: experimental design

In order to verify the new learning algorithms, Lookahead and Trace, available to the ARL agent, and compare these algorithms to the Q-learning algorithm, the following performance criteria have been specified:

- Whether or not the algorithm converges to the optimal policy
- Speed of convergence (either to optimal or sub-optimal policy)
- Generalization: Whether or not the sharing of stimuli across spatial locations in the Grid world will increase the speed of convergence.

3.2.1 Convergence

Convergence is measured by recording the average absolute fluctuation in associative strength (for both associative memory and expectance memory) and Q-values per episode. This is done by accumulating the absolute differences $diff_{abs}$ between the respective values of the associative memory, expectance memory and the Q-values, and their values after an update has been carried out. Additionally the number of steps to reach the goal, $episode_{length}$, is recorded. The average absolute fluctuation (AAF), is thus given by $diff_{abs}/episode_{length}$. For Q-learning, there is only one Q-value update per episode step, but for ARL there can be several updates of the associative memory and the expectance memory. This is because of the consideration of states as stimuli compounds, where each stimulus of the compound enters into separate associations from the others, as is clear from the definition of algorithm 8 and algorithm 9. For ARL, it is therefore necessary to record the number of updates per episode step $numerrors_{episodestep}$ as well. The average absolute fluctuation per episode for ARL is therefore given by $(diff_{abs}/numerrors_{episodestep})/episode_{length}$. AAF is recorded for both the associative memory and the expectance memory. The definition of convergence, according to [Russell & Norvig 2003], states that it is a function which

Algorithm 10 Algorithm to find the cost of a given policy

```
1: function policyCost(policy)
2:   policycost ← 0
3:   For each l in (Grid world locations)
4:     nl ← l
5:     do
6:       nl ← move(policy(nl.best_response))
7:       policycost ← policycost + 1
8:     while(nl ≠ goal)
9:   Next
10:  return policycost
11: End function
```

returns a contraction of its arguments. For Q-learning and ARL this function is represented by the respective value updates of the algorithms. As the algorithms converge to the optimal policy the error of the associative memory, expectance memory, and the Q-values will decrease (if there is convergence), and hence the average absolute fluctuation will decrease as well. In this case the minimum point of convergence is 0, but due to static learning rates, convergence will occur in the mean of the step-size parameters α (for Q-learning and ARL) and β (for ARL).

3.2.2 Optimality

The optimal policy is the shortest path from any spatial location to the goal. By measuring the number of steps, for a given policy, from each spatial location to the goal, the policy cost is found. A Grid world policy states which response to elicit for each spatial location, that is, for each stimuli compound. In order to have a basis of comparison with the policies found by the ARL agent and the Q-learner agent, the optimal policy is found objectively in the following way:

$$\forall l \in GridWorldlocations : l.best\_response \leftarrow$$
$$\forall al \in adjacent(l) : arg_{direction}(min(\sqrt{(al.x - goal.x)^2 + (al.y - goal.y)^2}))$$

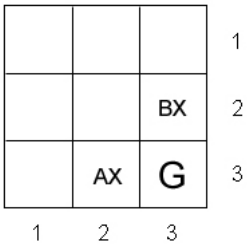
The cost of the policy is then calculated according to the pseudo code in algorithm 10.

3.2.3 Generalization

Generalization in the Grid World means how the consideration of states as stimuli compounds can help the transfer of learning between similar situations. When a stimulus X, at two different locations signals the same outcome, there is said to be a sharing of associations between the two locations. This can be most readily seen from figure 3.3.

In figure 3.3 stimulus X enters into an association with the outcome stimulus G. This association is strengthened at two locations. Additionally, both stimulus A and B enter into an association with G separately. In this situation it is said that stimulus X is generalized from location (2,3) to location (3,2) and vice versa. There is thus a sharing

Figure 3.3: Generalization in the Grid World, by means of sharing of stimuli across compounds



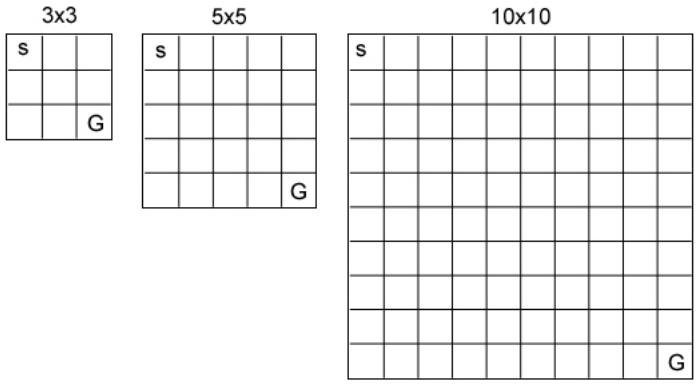
of stimulus X between the stimuli compound at location (2,3) and the compound at location (3,2). An algorithm which manages to gain a savings effect from this type of sharing is said to be able to generalize. This generalization and sharing's effect should be manifested in faster convergence to the optimal policy if the algorithm is successful in using the redundant association to its benefit. Generalization also means to change a hypothical situation A on some accounts, which leads to the similar situation B. An algorithm which has been trained in situation A, and which thereafter is able to use this experience in situation B to find the optimal policy more rapidly than without this similarity, is said to be able to generalize.

3.2.4 The experiments

On the basis of the criteria presented in the previous sections; i.e. convergence, optimality, and generalization, a number of experiments have been designed. Firstly the experiments aim to test for convergence to the optimal policy, and secondly for generalization. For testing of convergence three instances of the Grid world has been employed, namely grid sizes of 3x3, 5x5 and 10x10. In each of these instances the agent starts an episode in the upper left corner, and ends the episode in the lower right corner (the goal). Thus the aim of the agent is to find the optimal policy, i.e. the least number of steps from each location in the Grid world to the goal. For the purpose of merely testing convergence to the optimal policy, the three instances of grid sizes contain one stimulus per location, where all stimuli are neutral, except the stimulus at the goal location, which is appetitive. Hence, for the ARL agent the stimulus at the goal location has an outcome value of 1, because it is specified as appetitive in the ARL agent's list of internal drives. Similarly, for the Q-learner agent, the reward received upon entering the goal location is 1, and 0 for all other locations. The two agents, ARL and Q-learner, thus begin their learning with the same preconditions. Figure 3.4 shows the three instances of the Grid world used to test convergence.

Two algorithms are available to the ARL agent, algorithm 8 (Trace) and algorithm 9 (Lookahead). Both of these algorithms can either combine the associative memory with the expectance memory in order to find the best response (as in function 3.11),

Figure 3.4: Grid sizes for baseline convergence experiments



or use the expectance memory by itself (as in function 3.11). For the ARL agent there are therefore 4 combinations for each instance of the Grid world, giving a total of 12 combinations. The Q-learner agent has one set of options per experiment. In the following table, the different combinations of the ARL agent are summarized for each experiment:

Experiment	Combine memories (EM & AM)	Algorithm
3x3	yes	Trace
3x3	no	Trace
3x3	yes	Lookahead
3x3	no	Lookahead
5x5	yes	Trace
5x5	no	Trace
5x5	yes	Lookahead
5x5	no	Lookahead
10x10	yes	Trace
10x10	no	Trace
10x10	yes	Lookahead
10x10	no	Lookahead

- The following algorithm parameters are set for the ARL agent:
  - $\sigma = 0.2$ (The sensitivity of the similarity function)
  - $\alpha = 0.1$ (Learning rate specific to the the CS)
  - $\beta = 0.1$ (Learner rate specific to the US)
  - $\epsilon = 0.8$ (Choose the greedy response in 80% of the cases)
  - $ST = 0.85$ (Used for the trace algorithm. Update first-order conditioning if similarity to US is greater than.)

- $\delta = 0.9$ (Reward discounting)

- The following algorithm parameters are set for the Q-learner agent:
  - $\alpha = 0.01$ (Learning rate. Has been set to $\alpha * \beta$ from ARL parameters.)
  - $\gamma = 0.9$ (Reward discounting)
  - $\epsilon = 0.8$ (Choose the greedy response in 80% of the cases)

Generalization

The purpose of redefining states as stimuli compounds, (i.e. no longer as irreducible entitities), is to achieve generalization. When stimuli are shared across stimuli compounds (in two different, but similar situations), and when these shared stimuli signal the same outcomes, an algorithm capable of generalizing should be able to exploit this redundancy in the environment to gain an increase in convergence speed. The ARL agent is able to take this redundancy into account by definition. What has been learned during a learning period for one Grid World layout, should be able to transfer to the same Grid world layout after some changes have been made to the layout. Four experiments have therefore been designed to test whether the ARL agent will converge faster in a slightly changed environment, than in an environment which has been changed more. Each of these experiments have been divided into two phases; phase 1 and phase 2. In phase 1 the agent is trained with an initial Grid world layout, and in phase 2 this initial Grid world layout is changed. The 'G' in the lower right corner of each grid world layout represents the goal stimulus and is the same for all layouts, for all experiments and phases. Phase 1 has the same stimuli as phase 2 for all locations, unless otherwise stated. E.g. in 3.5, 'AX' in phase 1 refers to a stimuli compound containing the stimulus 'A' and 'X'. All four experiments aim to test whether phase 2 will converge faster to the optimal policy through generalization with phase 1. Two groups are employed for each experiment. In group 1 there is supposed to be generalization from phase 1 to phase 2 due to an environment change which leaves an aspect of the environment layout intact, but changes another. Group 2, on the other hand, changes the environment, but leaves no aspect of phase 1 similar in phase 2. If convergence is faster in phase 2 of group 1, than in phase 2 of group 2, it can be seen as an indicator of generalization.

The experiment in 3.5 aims to test whether the signalling of the goal G by stimulus X in phase 1, will transfer to phase 2, where stimulus A has been replaced with stimulus B in the stimuli compound of X. The stimuli compound AX is discriminatory with regards to the stimuli compound BX, but obviously they share the stimulus X. In group 2, a different stimulus, Y, is added to the same position as stimulus B, whereas stimulus X, from phase 2, is removed from the environment, so that there is no common association with phase 1 (i.e. AZ→G share no stimulus with BW→G). The point of the experiment is then to see whether there is any difference in convergence speed in phase 2 of the two groups.

In figure 3.7 the shared stimulus (X) does not signal the goal directly. The experiment aims to test whether there is any generalization between the two phases, when the

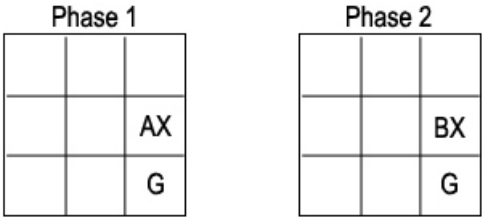


Figure 3.5: Experiment 1, group 1

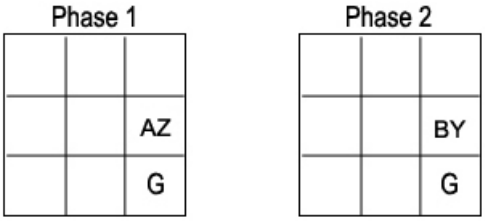


Figure 3.6: Experiment 1, group 2

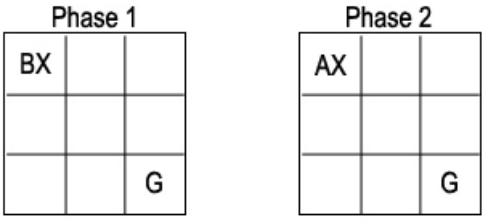


Figure 3.7: Experiment 2, group 1

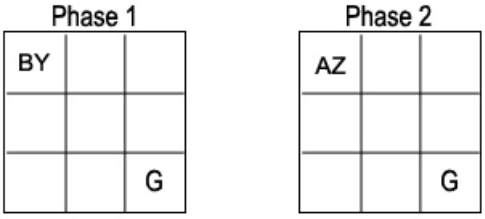


Figure 3.8: Experiment 2, group 2

shared stimulus signals another outcome than the goal. Apart from the spatial locations, experiment 2 share the same characteristics as experiment 1.

The experiment in figure 3.9 aims to test whether placing two stimuli (X and Y) in the same location for both phases, will increase the speed of convergence to the optimal

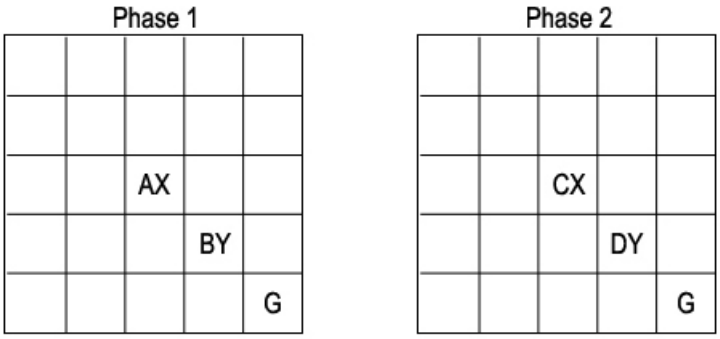


Figure 3.9: Experiment 3, group 1

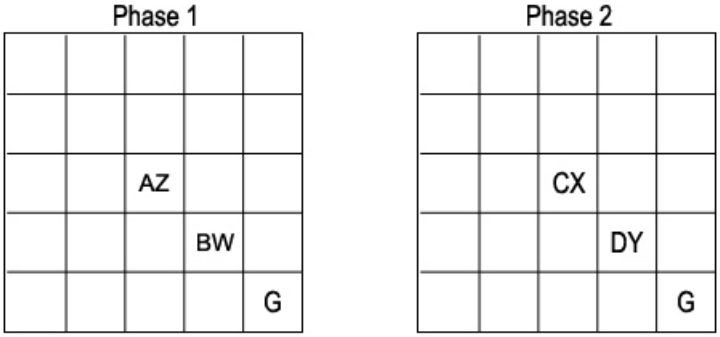


Figure 3.10: Experiment 3, group 2

policy. Thus stimulus X and Y are held in the same locations for both phases in group 1. In group 2, X and Y are replaced by Z and W for phase 1. It is thus predicted that phase 2 of group 1 should generalize with phase 1 of group 1, and therefore not having to relearn a policy for the changed stimuli (A and B are changed with C and D), but use the remaining stimuli (X and Y). For group 2, the opposite is supposed to happen. Because both compounds are changed (AZ and BW are changed to CX and DY), phase 2 of group 2 should take longer time to converge to the optimal policy than phase 2 of group 1. If this is the case, then it can be seen as an indicator that the algorithm is able to generalize.

Experiment 4 in figure 3.11 is similar to experiment 3, but has a more complex environment and a larger Grid world configuration (10x10). Instead of changing two stimuli compounds, three are changed, although in the same way as with experiment 3.

Experiment 1, 2, 3, and 4 will be tested on the ARL lookahead algorithm with combined memories. Additionally, the Q-learning algorithm will be tested on these experiments as well, to see whether it fails to generalize due to using configural state representations. The same algorithm parameters as in the convergence experiments are used for the generalization experiment.

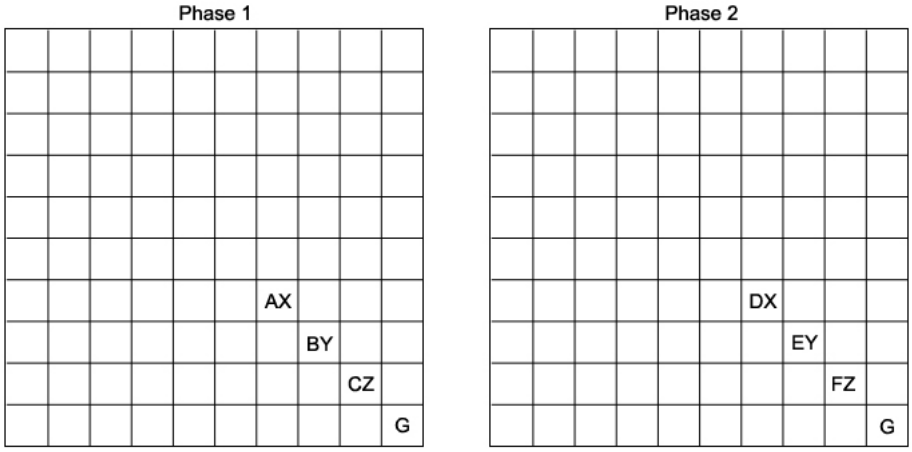


Figure 3.11: Experiment 4, group 1

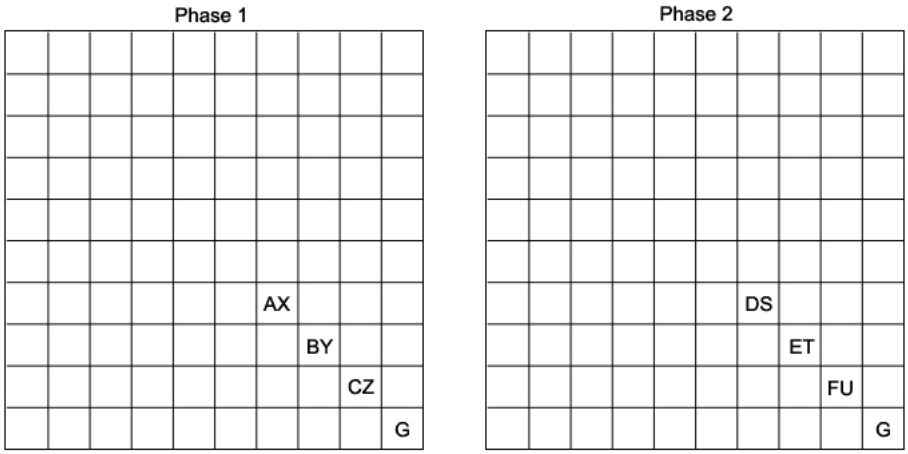
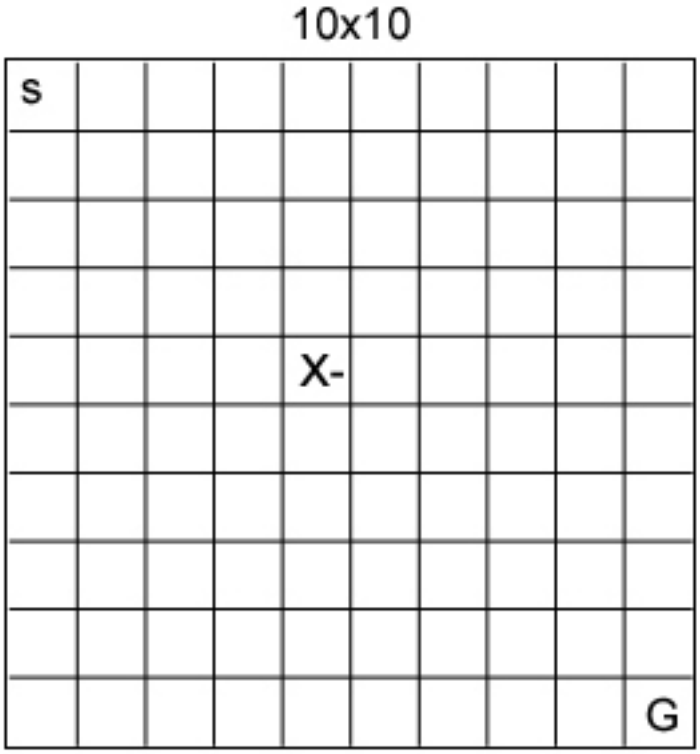


Figure 3.12: Experiment 4, group 2

Avoiding aversive stimuli

Finally, a simple experiment has been designed involving a 10x10 grid world layout with one aversive stimulus, and as usual, one appetitive stimulus. The point is to test whether the ARL agent is able to avoid the aversive stimulus, but still approach the appetitive stimulus. Refer to figure 3.13 for a specification of the experiment. The stimulus denoted X- is aversive, S refers to the start location, and G is the appetitive stimulus in the goal location. Hence, the aim of the agent is to avoid the aversive stimulus, i.e. move around it. The optimal policy should then be sub-optimal with regards to the standard 10x10 grid world convergence experiment, and therefore the policy cost should be slightly higher. This experiment will be tested on the ARL lookahead

Figure 3.13: 10x10 Grid world with aversive stimulus (X-)



algorithm (expectance memory only) and the Q-learner, with algorithm parameters being the same, except for learning rates, as for the convergence experiments. For ARL lookahead the learning rates are: $\alpha = 0.5, \beta = 0.5$. For Q-learning the learning rate is: $\alpha = 0.25$. Again, as for the convergence experiments, the testing criteria are convergence, and optimality of policy.

Chapter 4

Results

In this chapter the results from the experiments set out in the Methods chapter are presented. An experiment to test convergence is first presented, and thereafter a generalization experiment. Finally, the results from a simple experiment to test whether the agent is able to avoid aversive stimuli in the environment, are presented. The algorithms share the same preconditions for all experiments (i.e. learning rate, reward structure in the environment, and reward discounting). The differences between the two algorithms can be found in the way in which the state signal is represented, and it is therefore expected that the convergence experiments (and avoidance of aversive stimuli) will show no noteworthy differences between Q-learning and ARL (lookahead), because the state signal is equal for the two agents by the definition of stimuli in the environment. The ARL lookahead algorithm and Q-learning are functionally equivalent; they both facilitate second-order conditioning on the basis of the best successive prediction independent of the behavioral policy. ARL trace, on the other hand, caters for second-order conditioning by using the best successive prediction from the behavioral path of the agent. The difference between the ARL agent (lookahead) and the Q-learner agent is representational. For ARL trace the difference is functional as well.

The real difference between the two agents is expected to be seen in the generalization experiment, where it is tested whether the agent is able to use learned behavior from one situation in a similar situation[1].

4.1 Convergence

First, and most importantly, the algorithms must be able to converge to the optimal policy. This is a baseline result, and a minimum requirement. The purpose of the convergence experiments is to show that the algorithms find the optimal policy and that the prediction error decreases as the number of learning episodes increases. For Q-learning the prediction error is given by the term $\alpha\big[r' + \gamma \max_{a'} Q(s', a') - Q(s, a)\big]$. Quite similarly to Q-learning the ARL algorithms, trace and lookahead, use the [Rescorla & Wagner 1972] prediction error term $\alpha\beta\big[\lambda_{US} - \sum_{i=0}^{n} V_{CS_i}\big]$. Hence, both Q-learning

[1] For results data and experiment definitions, refer to appendix C

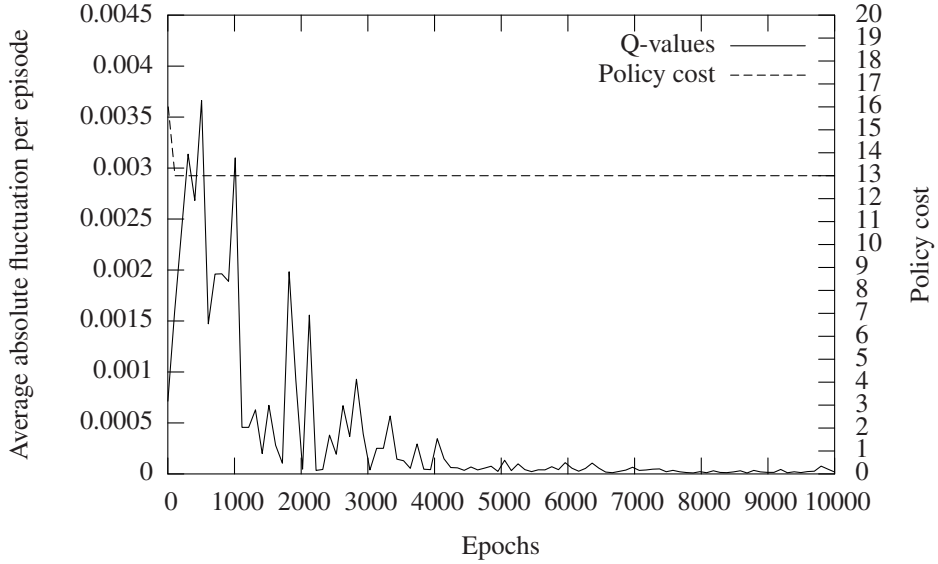
and ARL use an error correction rule. The error given by these rules is averaged per episode, in order to produce the convergence graphs, which plot the average absolute fluctuation in Q-values, associative values, and expectance value. Secondly, the policy cost is plotted per episode. Generalization is not attempted tested in the convergence experiments. Thus, there is only one stimulus per spatial location. No statistical test is made for the convergence experiments, i.e. each algorithm is run once for each Grid world configuration (3x3, 5x5, 10x10). Because the algorithms are only run once per configuration, there is no basis for statistical analysis[2]. Convergence is therefore only measured qualitatively by means of the graph plot. If the average absolute fluctuation moves towards 0 on the plot as the number of epochs increases, this is seen as an indication of convergence. In the following pages graph plots are presented for the three Grid world layouts (3x3, 5x5, 10x10), for each combination of the ARL agent (trace and lookahed with or without combined memories), and for Q-learning. Each graph plot figure is captioned to show which algorithm and combination is being plotted. Below each graph the policy is illustrated after 10000 epochs (i.e. each experiment is run 10000 epochs for each algorithm combination). This policy figure shows the optimal response in each spatial location. As described earlier the upper left corner is the start location, and the lower right corner is the goal location. For a square Grid world, i.e. where there are an equal number of vertical squares and horizontal squares, the optimal policy cost can be calculated objectively either by algorithm 10 (which can also calculate the cost for rectangular Grid World shapes), or by equation 4.1. Table 4.1 presents a summary of the results for the convergence experiments.

$$policyCost = \sum_{n=2}^{m}(n+n-1)(n-1) \tag{4.1}$$

$m =$ number of horizontal or vertical squares

[2]A formal explanation will be given in the discussion chapter on why the algorithms converge or not under the conditions set in the convergence experiments, and given the parameters of the algorithms.

Figure 4.1: Q-learning 3x3



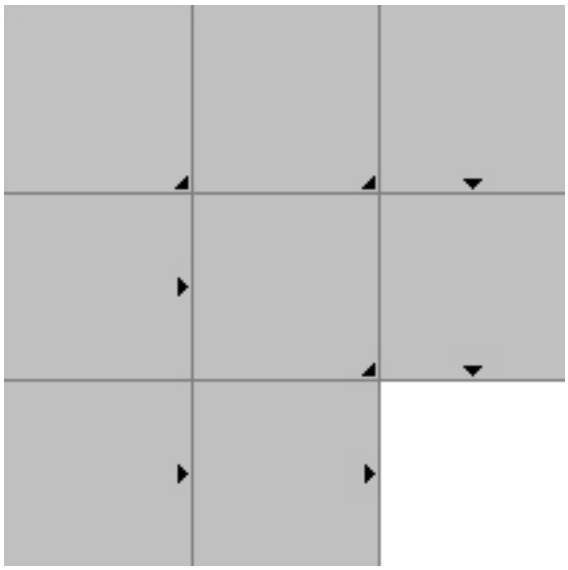
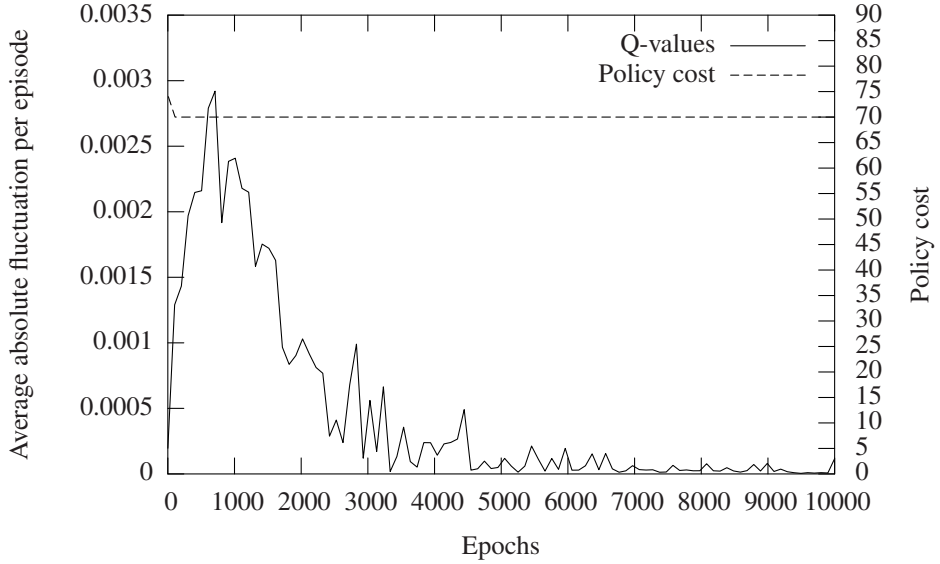


Figure 4.2: Policy found (cost: 13)

Figure 4.3: Q-learning 5x5



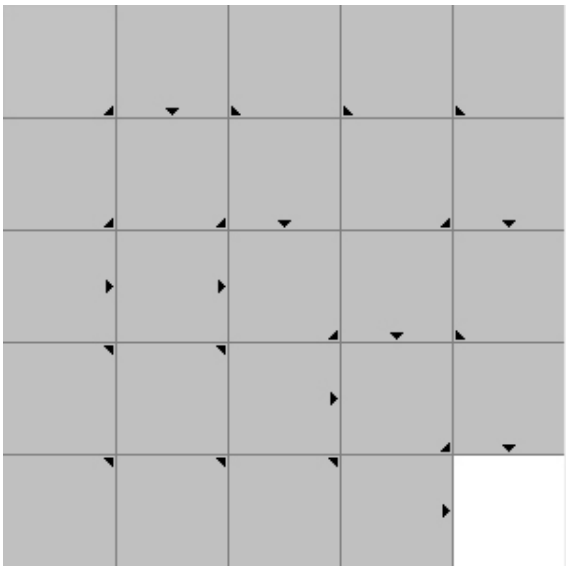
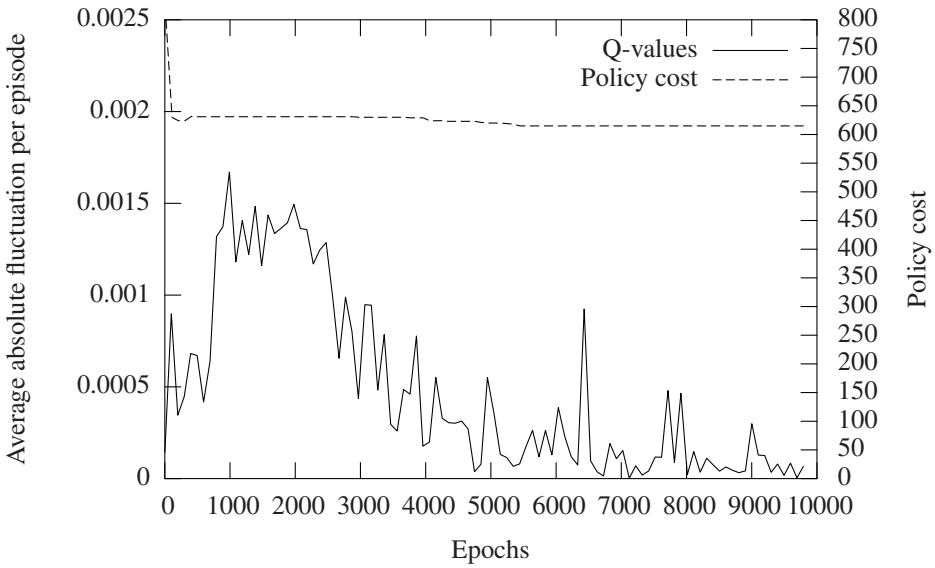


Figure 4.4: Policy found (cost: 70)

Figure 4.5: Q-learning 10x10



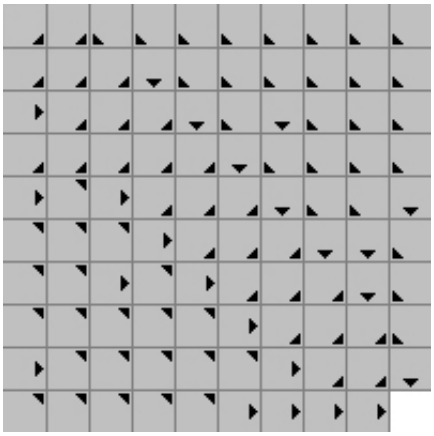
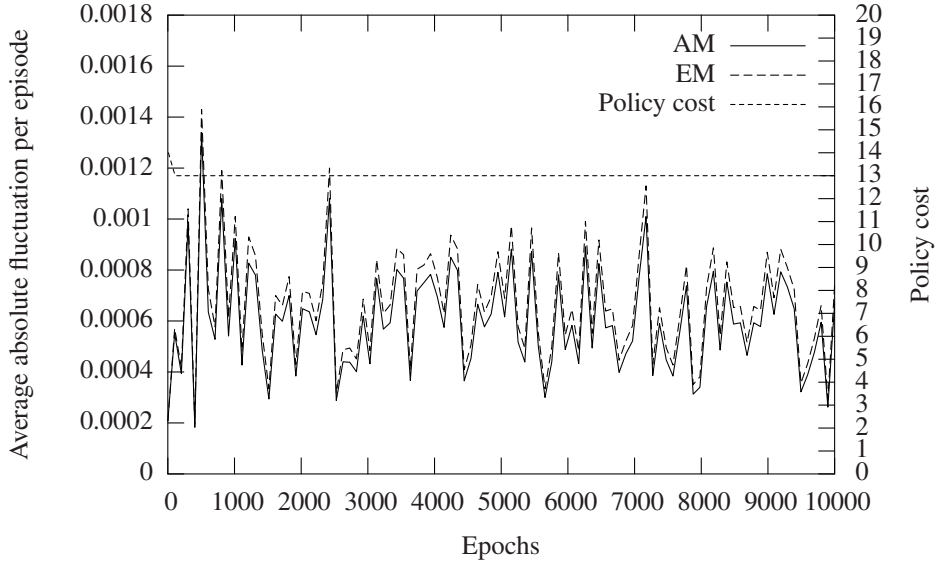


Figure 4.6: Policy found (cost: 615)

Figure 4.7: Trace 3x3 combine memories



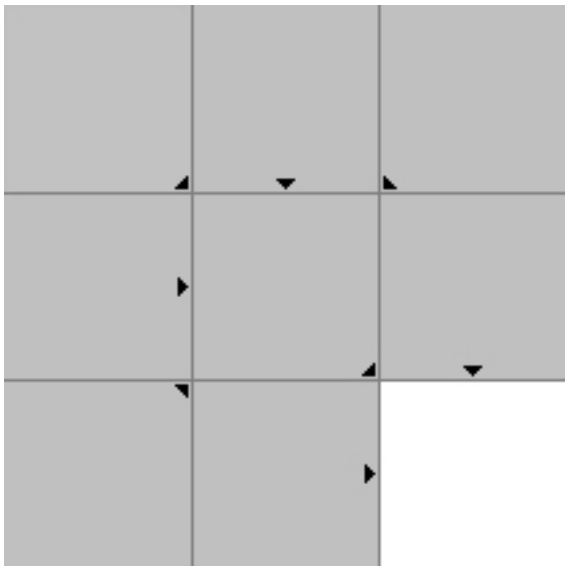
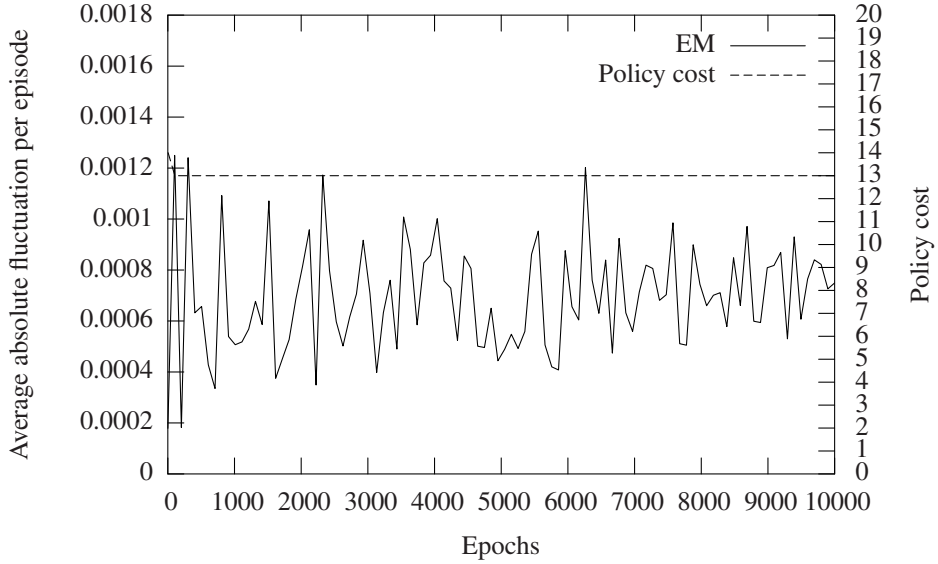


Figure 4.8: Policy found (cost: 13)

Figure 4.9: Trace 3x3 expectance memory only



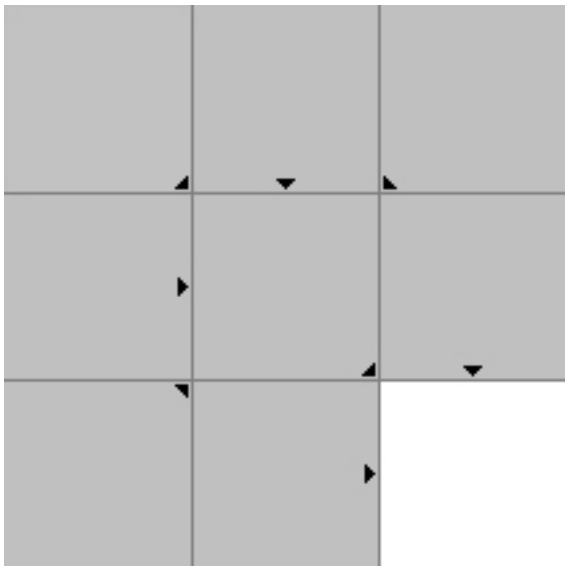
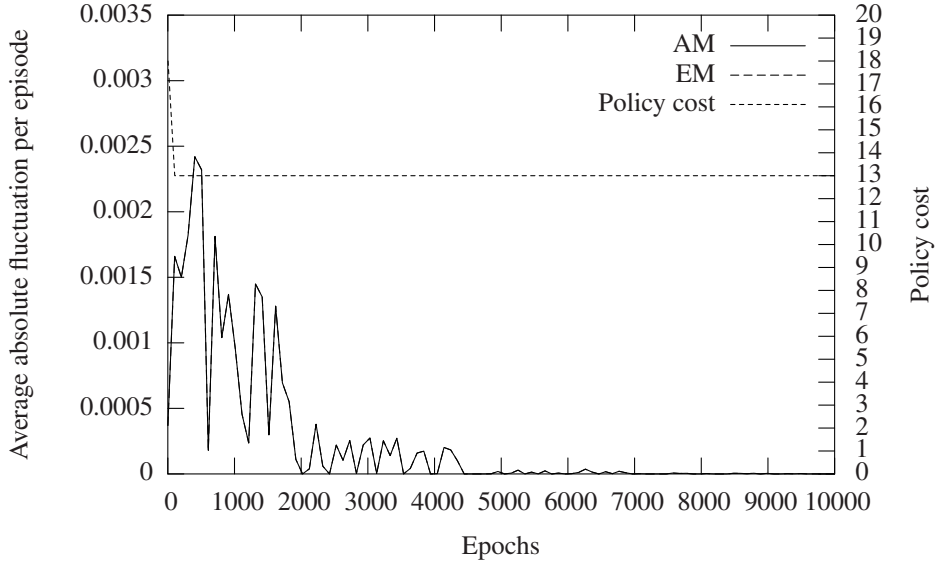


Figure 4.10: Policy found (cost: 13)

Figure 4.11: Lookahead 3x3 combine memories



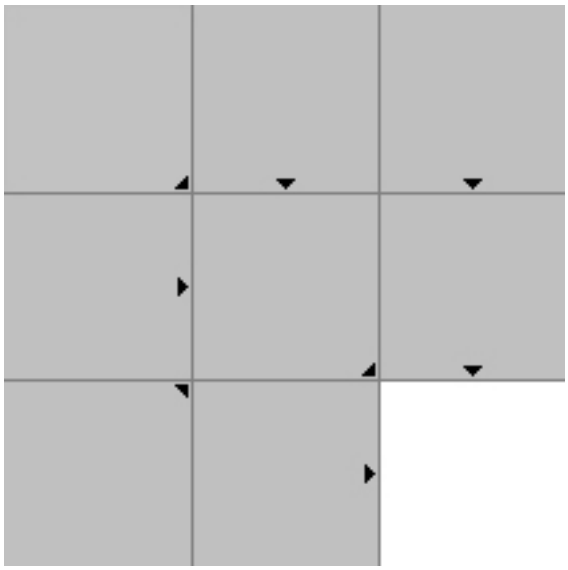
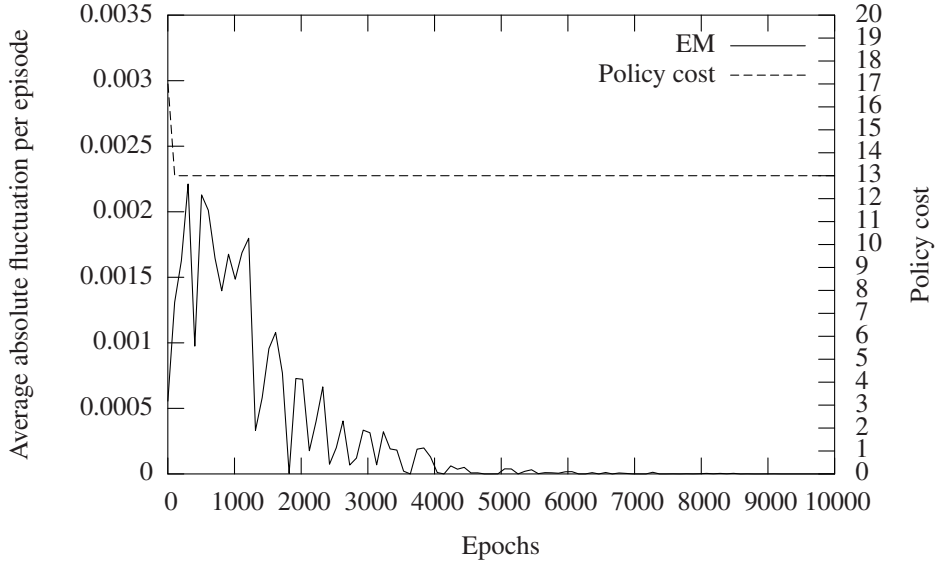


Figure 4.12: Policy found (cost: 13)

Figure 4.13: Lookahead 3x3 expectance memory only



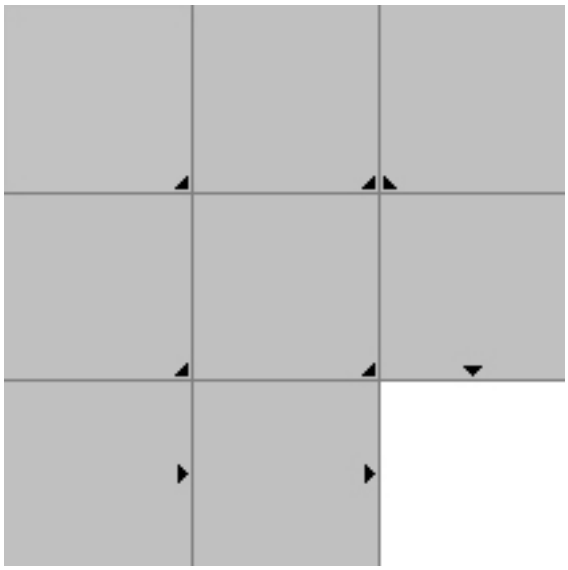
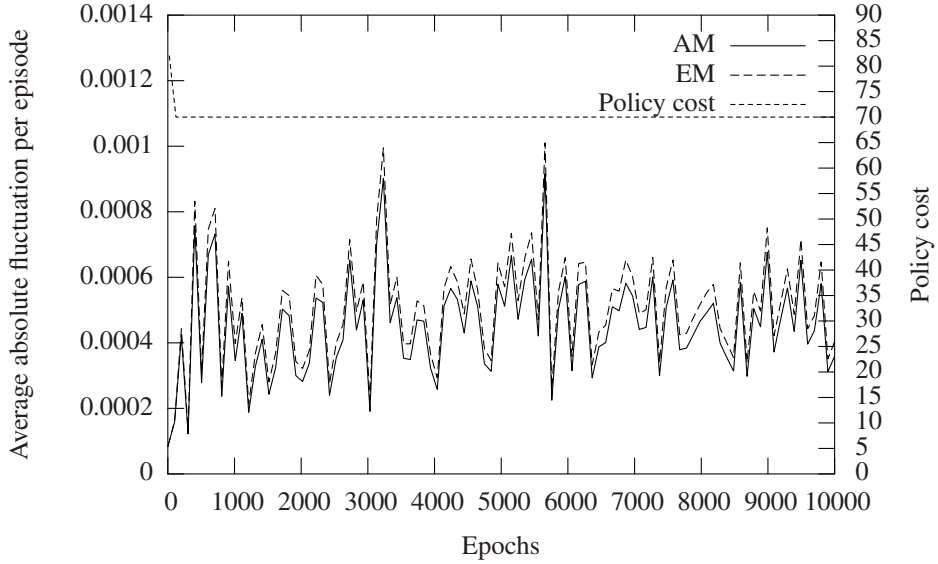


Figure 4.14: Policy found (cost: 13)

Figure 4.15: Trace 5x5 combine memories



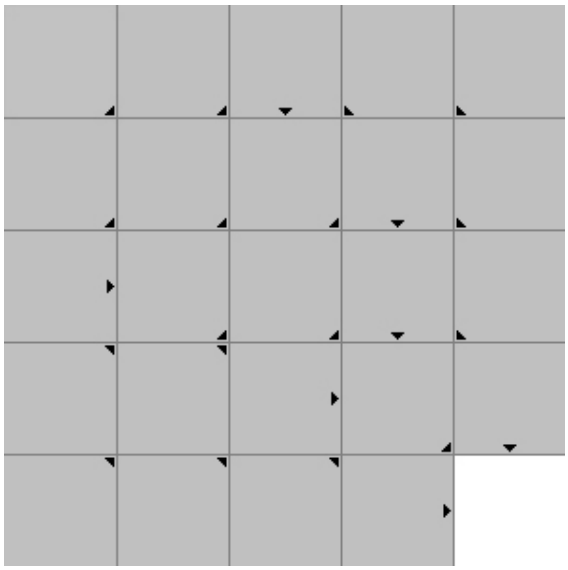
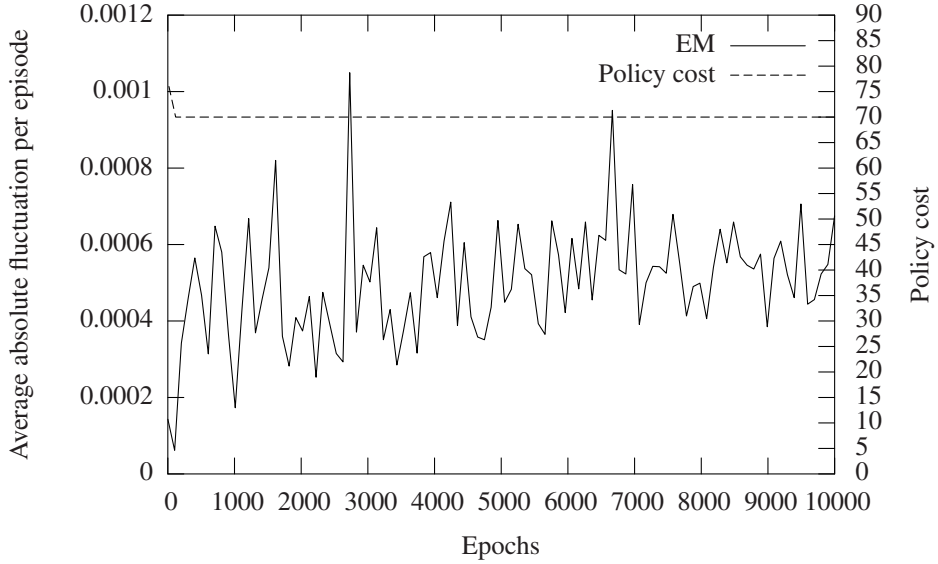


Figure 4.16: Policy found (cost: 70)

Figure 4.17: Trace 5x5 expectance memory only



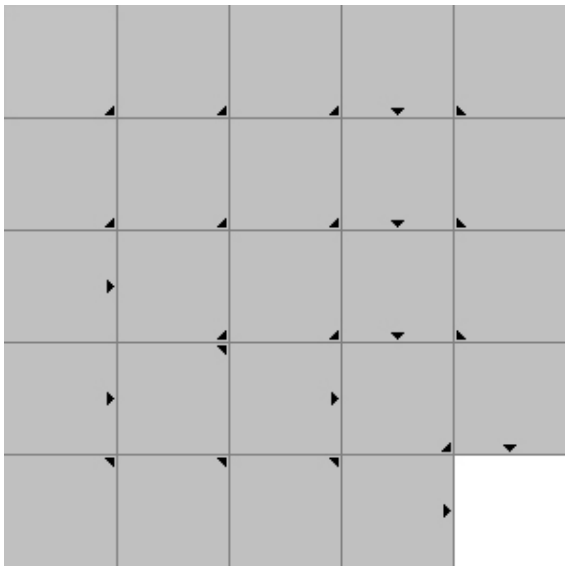
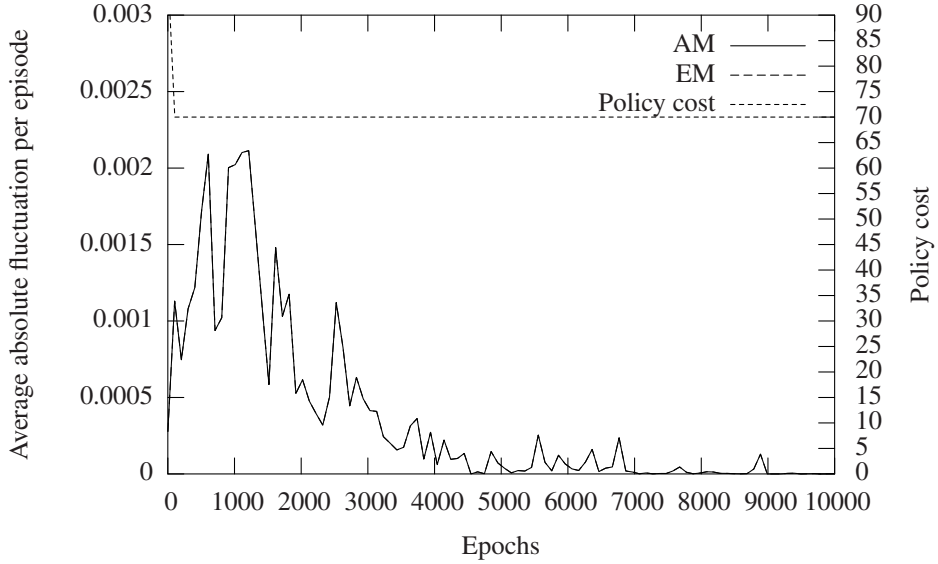


Figure 4.18: Policy found (cost: 70)

Figure 4.19: Lookahead 5x5 combine memories



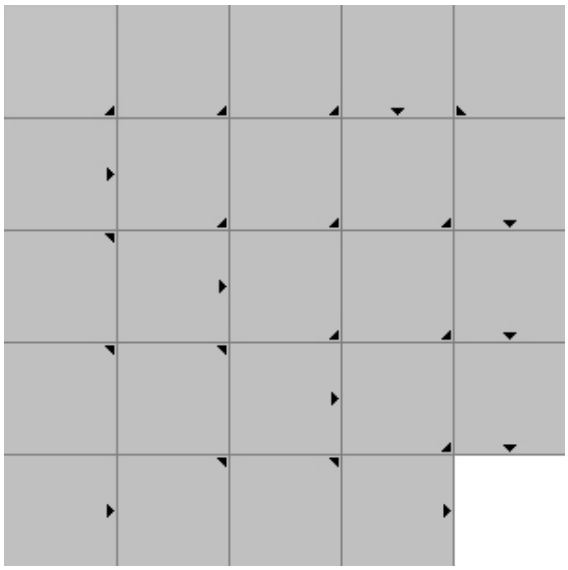
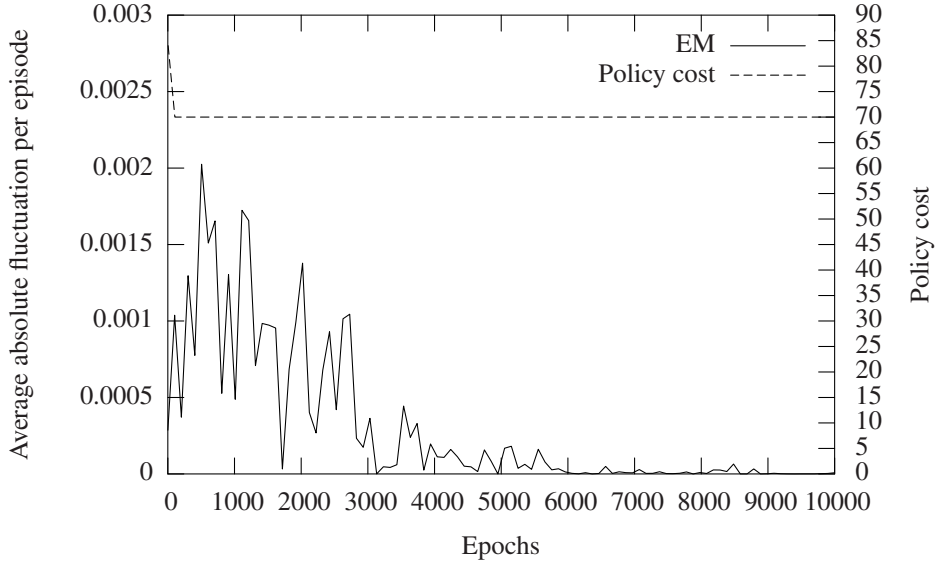


Figure 4.20: Policy found (cost: 70)

Figure 4.21: Lookahead 5x5 expectance memory only



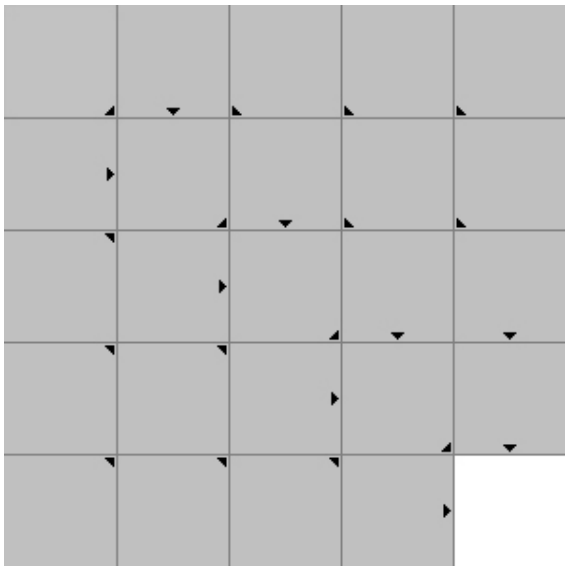
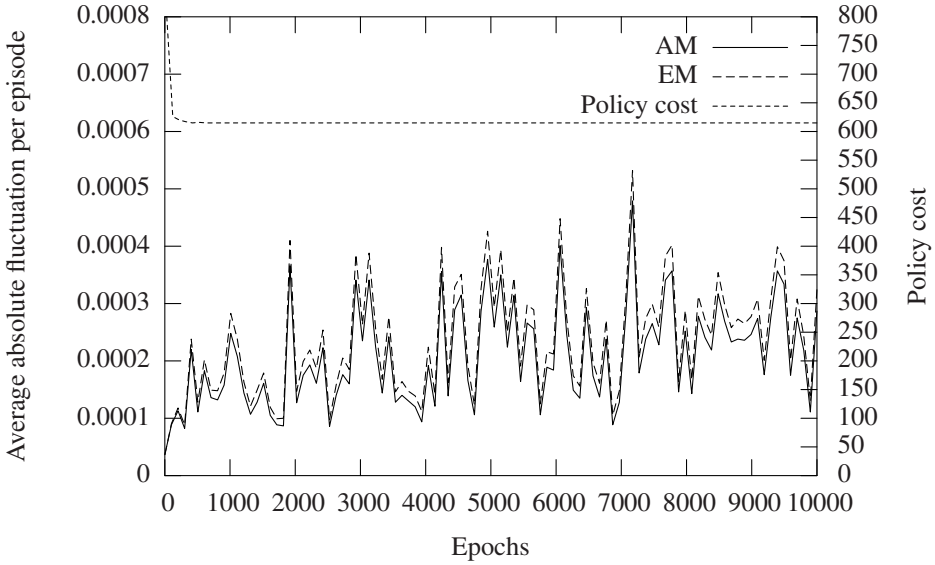


Figure 4.22: Policy found (cost: 70)

Figure 4.23: Trace 10x10 combine memories



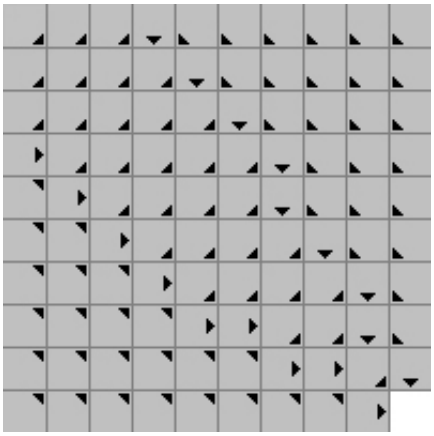
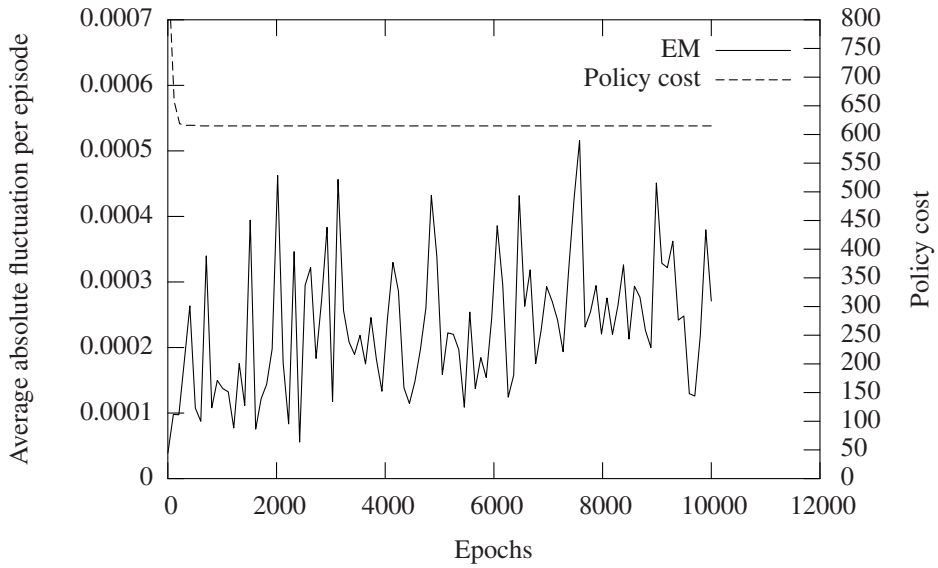


Figure 4.24: Policy found (cost: 615)

Figure 4.25: Trace 10x10 expectance memory only



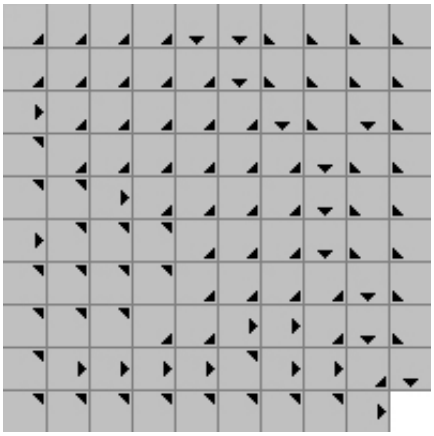
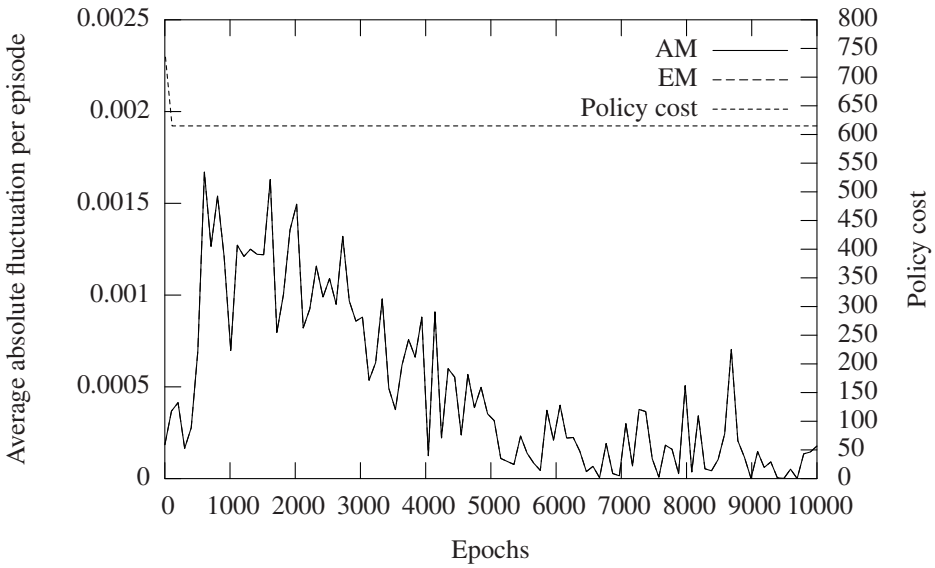


Figure 4.26: Policy found (cost: 615)

Figure 4.27: Lookahead 10x10 combine memories



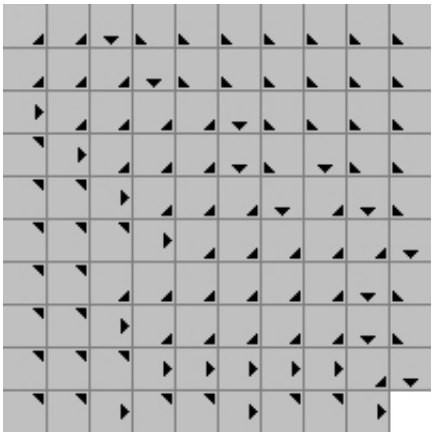
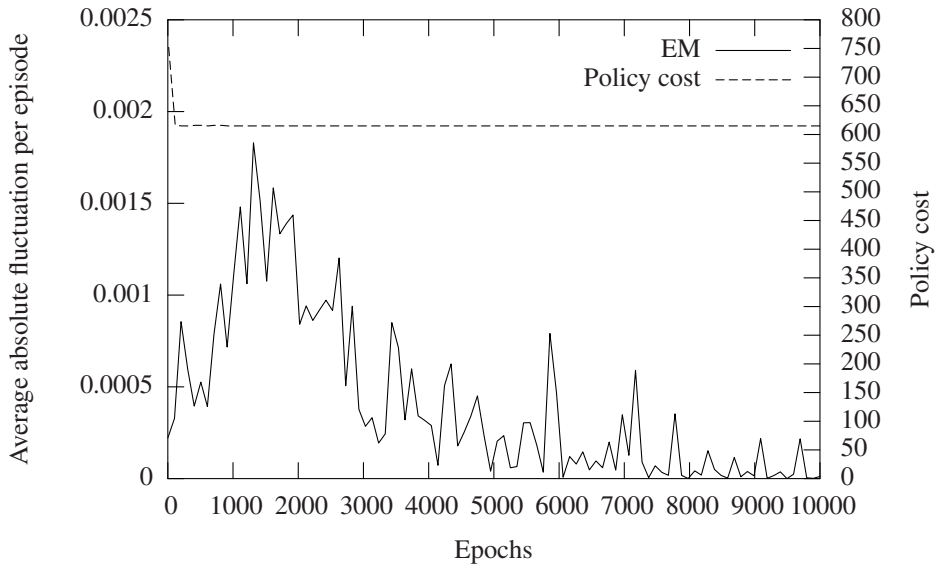


Figure 4.28: Policy found (cost: 615)

Figure 4.29: Lookahead 10x10 expectance memory only



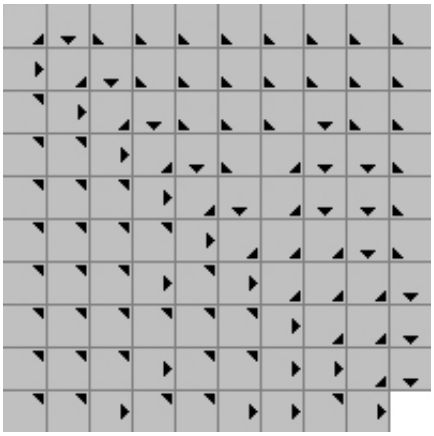


Figure 4.30: Policy found (cost: 615)

Table 4.1: Summary of results for convergence experiments. (MPC = Maximum Policy Cost for learning period, OFPE = Optimal policy found after n epochs)

Problem	Algorithm	Convergence	MPC	OPFE
3x3	Trace CM	no	15	16
3x3	Trace EM	no	14	6
3x3	Lookahead CM	yes	14	22
3x3	Lookahead EM	yes	13	15
3x3	Q-learning	yes	13	11
5x5	Trace CM	no	74	97
5x5	Trace EM	no	72	41
5x5	Lookahead CM	yes	72	73
5x5	Lookahead EM	yes	71	13
5x5	Q-learning	yes	70	12
10x10	Trace CM	no	692	420
10x10	Trace EM	no	672	596
10x10	Lookahead CM	yes	616	55
10x10	Lookahead EM	yes	626	843
10x10	Q-learning	yes	619	5352

4.2 Generalization

Section 3.2.4 describes the generalization experiments. Results from these experiments are presented here. All in all 32 experiments were carried out, of which each were run 8 times. Each experiment run involved training for 9000 epochs. To recap the experiment description from 3.2.4, for each experiment; in phase 2 of group 1, a small change change is made in the environment with regards to phase 1. This change involves manipulating one or several stimuli compounds (containing two stimuli), by changing one stimulus and leaving one stimulus intact from phase 1. For example, if stimuli compound AX signals an outcome G in phase 1, then stimulus X is changed to Y in phase 2, yielding the stimulus compound AY. Now, because each stimulus of the compound enters into an association with stimulus G, the association A→G is left intact in phase 2 of group 1, whereas the association X→G is removed and replaced with a new association Y→G, which has to be learned. For group 2, both stimuli in the compound AX are changed from phase 1 to phase 2, yielding the stimuli compound BY in phase 2. Thus, in phase 2 of group 2, two new associations (B→G and Y→G), have to be learned. Each of the 4 experiments follow this general description, but differ with regards to how many stimuli compounds are changed in the environment from phase 1 to phase 2, as well as the location of the changed compounds. In experiment 1, one compound signalling the goal is changed. Experiment 2 is similar to experiment 1, but the changed compound is a cue, not for the goal, but for second-order conditioned stimuli. In experiment 3 two stimuli compounds are changed, where one of them signals the goal, whereas the other signals the first compound. Experiment 4 changes three compounds lined up as a chain signalling the goal. For details regarding the Grid world

layouts, see figure 3.5(group 1) and 3.6(group 2) for experiment 1, figure 3.7(group 1) and 3.8(group 2) for experiment 2, figure 3.9(group 1) and 3.10(group 2) for experiment 3, and figure 3.11(group 1) and 3.12(group 2) for experiment 4. Table 4.2 presents the 32 experiment instances. It is important to note that the agent's associative memories (expectance memory (EM) and associative memory (AM) for ARL lookahead, and Q-values for Q-learning), were NOT reset from phase 1 to phase 2. This should be obvious when realizing that the purpose of the experiment is to test whether what has been learned in phase 1 can be utilized in phase 2.

Each experiment instance was run 8 times with the goal of gathering raw data to perform a statistical analysis in the form of a t-test. The raw data contains the following for each algorithm:

- ARL looakahead: Average absolute fluctuation per episode in the associative memory, Average absolute fluctuation per episode in the expectance memory, and policy cost after each episode
- Q-learning: Average absolute fluctuation per episode in the Q-values, and policy cost after each episode

The statistic used to perform the statistical analysis was the number of epochs[3] before the optimal policy was found and converged to[4] in phase 2 of each group. This statistic is referred to as Optimal Policy Refound (OPR). For each of the involved Grid world sizes, the optimal policy costs can be found by equation 4.1:

- 3x3 optimal policy cost: 13
- 5x5 optimal policy cost: 70
- 10x10 optimal policy cost: 615

4.2.1 Hypotheses testing

In order to test whether the two algorithms, ARL lookahead and Q-learning, find the optimal policy faster in phase 2 of group 1 than in group 2 or not (by the test statistic OPR), a null hypothesis H_0 and an alternative hypothesis H_a have been stated:

H_0: there is no difference in mean OPR for phase 2 of group 1 and group 2($\overline{x}_1 = \overline{x}_2$)

H_a: there is a difference in OPR for phase 2 of group 1 and group 2($\overline{x}_1 \neq \overline{x}_2$)

The independent variable is the difference in stimuli compounds between phase 1 and phase 2 of the two groups, and the dependent variable is the test statistic OPR. As described earlier each experiment group have been run eight times. Hence, there are 8 samples of Phase 2 for each group of each experiment. Using the OPR test statistic, an independent t-test performed in order to test the hypotheses. Regarding the assumptions: The two groups are independent because the agent's associative memories are

[3] In this thesis, episode and epoch refer to the same thing.

[4] Converged to the optimal policy means that the policy cost was constant for the remaining epochs of the experiment

Table 4.2: The generalization experiments. (CM = combine memories)

Experiment	Algorithm	Group	Phase
1	ARL lookahead (CM)	Group 1	Phase 1
1	ARL lookahead (CM)	Group 1	Phase 2
1	ARL lookahead (CM)	Group 2	Phase 1
1	ARL lookahead (CM)	Group 2	Phase 2
1	Q-learning	Group 1	Phase 1
1	Q-learning	Group 1	Phase 2
1	Q-learning	Group 2	Phase 1
1	Q-learning	Group 2	Phase 2
2	ARL lookahead (CM)	Group 1	Phase 1
2	ARL lookahead (CM)	Group 1	Phase 2
2	ARL lookahead (CM)	Group 2	Phase 1
2	ARL lookahead (CM)	Group 2	Phase 2
2	Q-learning	Group 1	Phase 1
2	Q-learning	Group 1	Phase 2
2	Q-learning	Group 2	Phase 1
2	Q-learning	Group 2	Phase 2
3	ARL lookahead (CM)	Group 1	Phase 1
3	ARL lookahead (CM)	Group 1	Phase 2
3	ARL lookahead (CM)	Group 2	Phase 1
3	ARL lookahead (CM)	Group 2	Phase 2
3	Q-learning	Group 1	Phase 1
3	Q-learning	Group 1	Phase 2
3	Q-learning	Group 2	Phase 1
3	Q-learning	Group 2	Phase 2
4	ARL lookahead (CM)	Group 1	Phase 1
4	ARL lookahead (CM)	Group 1	Phase 2
4	ARL lookahead (CM)	Group 2	Phase 1
4	ARL lookahead (CM)	Group 2	Phase 2
4	Q-learning	Group 1	Phase 1
4	Q-learning	Group 1	Phase 2
4	Q-learning	Group 2	Phase 1
4	Q-learning	Group 2	Phase 2

reset for each group (i.e. phase 1 of each group is *tabula rasa* learning). For the two other assumptions of the independent t-test; whether the samples are normally distributed and the homogeneity of variance, an inspection of the data samples is needed.

ARL lookahead has a sample mean OPR of 0 and a variance of 0 in phase 2 of group 1 for all eight samples in experiments 1, 2, and 3, whereas for phase 2 of group 2 the mean OPR is 848, 217.625, and 825.375, and a variance of 124792, 32463.69643, and 43929.41071, for experiments 1,2 and 3 respectively. Hence, for group 1 the data is not normally distributed, and the variances are different from those of group 2, for ARL lookahead in experiments 1, 2, and 3. There is therefore no basis for a t-test, because the assumptions are violated. On the other hand, just by looking at the data samples, it is obvious that the difference is big between groups 1 and 2. For all samples from group 1 for experiments 1, 2, and 3, there is no change in policy cost due to the change in the environment, whereas the change in policy cost is quite large for group 2. It is therefore concluded, for ARL lookahead in experiment instances 1, 2, and 3, that the difference between the two groups is so obvious that there is no need for statistical analysis. However, due to the lack of a t-test for these instances H_0 can neither be rejected or retained. In the next chapter a formal analysis will be given on why the ARL lookahead algorithm incurs no change in policy cost for group 1 of experiments 1, 2, and 3.

The situation is different for experiment 4 and the ARL lookahead algorithm. Table 4.6 shows a sample mean OPR of 496.375 for group 1 and 1059.75 for group 2. The respective variances for group 1 and 2 are 15107.69643, 134965.0714. It is assumed that the data is normally distributed for all eight samples in both groups. In order to check whether the heterogeneity of variance is significant at the .05 level, an F Max test is performed. In table 4.6 the f value is reported to be 0.009841503. The degrees of freedom for the numerator is $(n_1\text{-}1) = 7$, and $(n_2\text{-}1) = 7$ for the denominator. According to the f distribution [Coolidge 2000, Appendix E] this gives a critical value of 3.79 at the .05 level of significance, which the f value does not exceed, so the heterogeneity of variance is not significant. A one-tailed t-test is therefore performed (it is expected that the difference between group 2 and group 1 is positive). Table 4.6 presents a t-value of 4.397315147. At $(n_1+n_2\text{-}2)=14$ degrees of freedom, this gives a critical value of 1.761 at the .05 level of significance ([Coolidge 2000, Appendix B]). The t-value well exceeds the critical value at the .05 level, and also at the .01 level (critical value: 2.624), as well as at the .001 level (critical value: 3.787). It is therefore concluded that the alternative hypothesis H_a is not rejected for the ARL lookahead algorithm for experiment 4.

For Q-learning, a two-tailed t-test has been performed for all four experiments, because it is expected that the difference between group 1 and 2 can be either positive or negative. As can be seen from tables 4.7, 4.8, 4.9, and 4.10, none of the f-values exceed the critical value at the .05 level of significance (critical value: 3.79, as reported in the previous paragraph), which means that the heterogeneity of variance is not significant. As before, it is assumed that the samples from group 1 and 2 are normally distributed. The t-values for experiment 1, 2, 3, and 4, are 1.574557199, 1.288501538, -0.455236673, and -1.168917156 respectively. According to the t distribution [Coolidge 2000, Appendix B] the critical value for the .05 level of significance (at $(n_1+n_2\text{-}2)=14$ degrees of freedom), is ± 2.145. None of the t-values exceed or go

below this critical value. It is therefore concluded that the null hypothesis H_0 cannot be rejected for Q-learning for neither of the four generalization experiments.

For purposes of visualization and comparison, the average convergence from phase 2 of group 1 and 2, is plotted in the same graph per algorithm and per experiment. Additionally, the same is done for policy cost. This means that for ARL lookahead, the average convergence, (the average of the average absolute fluctuation in associative memory and expectance memory per episode, for all 8 samples), is plotted in the same graph for phase 2 for groups 1 and 2, (in different graphs for the EM and AM memories). The same is done for the Q-values. If the policy is circular at any epoch or never reaches the goal, the policy cost is reported as negative or positive infinity (as implemented by the Java programming language) by the learning simulator. These values are considered as outliers and therefore irrelevant, and have been removed from the results data which was used to produce the plots. Therefore, a sudden drop in policy value below optimal on the plot, is an indication that the policy was circular or did not lead to the goal[5].

[5]This does not affect the test statistic.

Table 4.3: Experiment 1, ARL

Sample	OPR Group 1 (Phase 2)	OPR Group 2 (Phase 2)
1	0	910
2	0	933
3	0	4
4	0	843
5	0	1108
6	0	943
7	0	1103
8	0	940
$\overline{x}$	0	848
S	0	353.2591117
S^2	0	124792
MAX	0	1108
MIN	0	4

Table 4.4: Experiment 2, ARL

Sample	OPR Group 1 (Phase 2)	OPR Group 2 (Phase 2)
1	0	310
2	0	0
3	0	455
4	0	312
5	0	271
6	0	16
7	0	14
8	0	363
$\overline{x}$	0	217.625
S	0	180.1768476
S^2	0	32463.69643
MAX	0	455
MIN	0	0

Table 4.5: Experiment 3, ARL

Sample	OPR Group 1 (Phase 2)	OPR Group 2 (Phase 2)
1	0	784
2	0	841
3	0	1087
4	0	482
5	0	751
6	0	921
7	0	644
8	0	1093
$\overline{x}$	0	825.375
S	0	209.5934415
S^2	0	43929.41071
MAX	0	1093
MIN	0	482

Table 4.6: Experiment 4, ARL

Sample	OPR Group 1 (Phase 2)	OPR Group 2 (Phase 2)
1	455	1379
2	772	1787
3	564	768
4	389	715
5	470	1066
6	428	736
7	475	993
8	418	1034
$\overline{x}$	496.375	1059.75
S	122.9133696	367.3759266
S^2	15107.69643	134965.0714
MAX	772	1787
MIN	389	715
sum of sample squares	2076859	9929316
square of sum of samples	15768841	71876484
n	8	8
F Max test	0.009841503	
t	4.397315147	

Table 4.7: Experiment 1, Q-learning

Sample	OPR Group 1 (Phase 2)	OPR Group 2 (Phase 2)
1	2	446
2	946	527
3	1196	3
4	0	6
5	19	22
6	737	172
7	794	1006
8	1201	59
$\overline{x}$	611.875	280.125
S	527.3349268	357.4742498
S^2	278082.125	127787.8393
MAX	1201	1006
MIN	0	3
sum of sample squares	4941703	1522275
square of sum of samples	23961025	5022081
n	8	8
F Max test	0.326598388	
t	1.574557199	

Table 4.8: Experiment 2, Q-learning

Sample	OPR Group 1 (Phase 2)	OPR Group 2 (Phase 2)
1	386	391
2	493	0
3	14	425
4	404	4
5	449	12
6	18	4
7	497	433
8	405	391
$\overline{x}$	333.25	207.5
S	199.9898212	216.9930874
S^2	39995.92857	47086
MAX	497	433
MIN	14	0
sum of sample squares	1168416	674052
square of sum of samples	7107556	2755600
n	8	8
F Max test	0.83504932	
t	1.288501538	

Table 4.9: Experiment 3, Q-learning

Sample	OPR Group 1 (Phase 2)	OPR Group 2 (Phase 2)
1	5218	4626
2	3308	4113
3	4085	4967
4	4095	3039
5	3979	4556
6	4037	3690
7	4114	4479
8	4384	4747
$\overline{x}$	4152.5	4277.125
S	528.4546203	637.131392
S^2	279264.2857	405936.4107
MAX	5218	4967
MIN	3308	3039
sum of sample squares	139900900	149191941
square of sum of samples	1103568400	1170803089
n	8	8
F Max test	0.633950422	
t	-0.455236673	

Table 4.10: Experiment 4, Q-learning

Sample	OPR Group 1 (Phase 2)	OPR Group 2 (Phase 2)
1	4797	4085
2	3806	4029
3	4201	5958
4	4638	6876
5	5452	5211
6	5632	6617
7	3838	3708
8	4433	4646
$\overline{x}$	4599.625	5141.25
S	679.216973	1225.408591
S^2	461335.6964	1501626.214
MAX	5632	6876
MIN	3806	3708
sum of sample squares	172481751	221970996
square of sum of samples	1354019209	1691676900
n	8	8
F Max test	0.142222382	
t	-1.168917156	

4.3 Avoidance of aversive stimuli

As described in the previous chapter, the experiment testing the avoidance of aversive stimuli involves a 10x10 grid world with one aversive stimulus and one appetitive stimulus. As in the convergence experiments, only one learning period has been carried out for the two algorithms being tested, ARL lookahead and Q-learning. There is therefore no basis for statistical analysis. The results are presented and interpreted qualitatively in the form of a plot illustrating the average absolute fluctuation in the expectance memory, and Q-values, as well as the policy cost for each episode. Each algorithm is run for a learning period of 10000 epochs. Their respective plots show the progress in convergence and policy cost as usual. For the convergence of ARL lookahead (expectance memory only), refer to figure 4.43. Figure 4.44 illustrates the convergence of Q-learning. It can be seen from the plots that both algorithms converge to a policy cost of 619, in contrast with the optimal policy cost of 615 for the 10x10 grid world convergence experiment. This is because both the Q-learner and the ARL agent learns to avoid the aversive stimulus[6].

[6] As for the previous experiments, a formal analysis of the situation will be given in the discussion chapter.

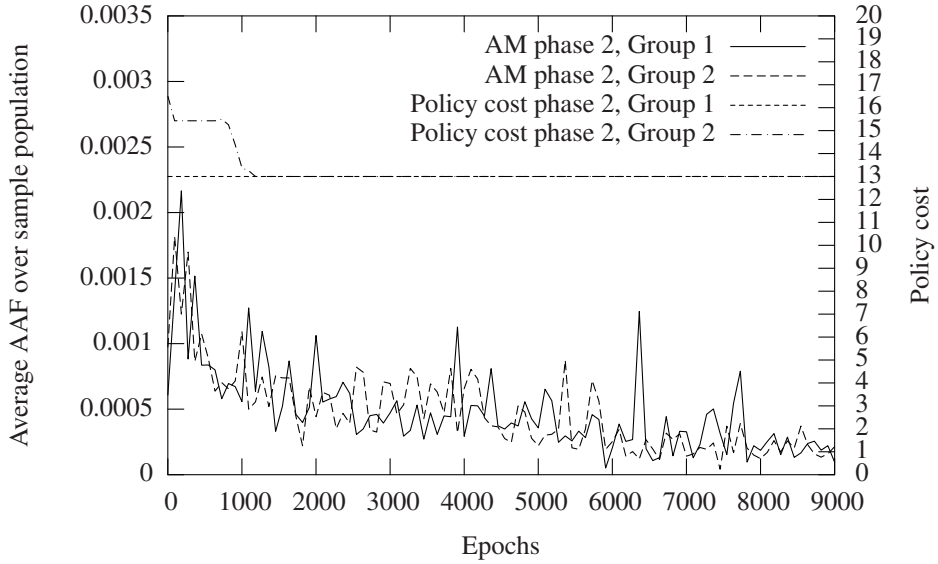


Figure 4.31: Experiment 1 (ARL lookahead): Associative memory average convergence, and average policy cost, over 8 samples

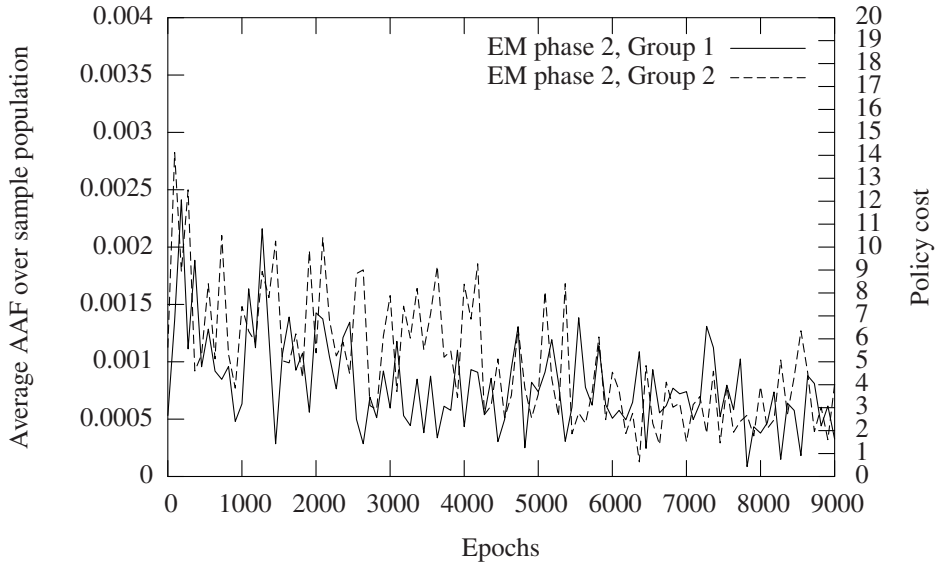


Figure 4.32: Experiment 1 (ARL lookahead): Expectance memory average convergence, and average policy cost, over 8 samples

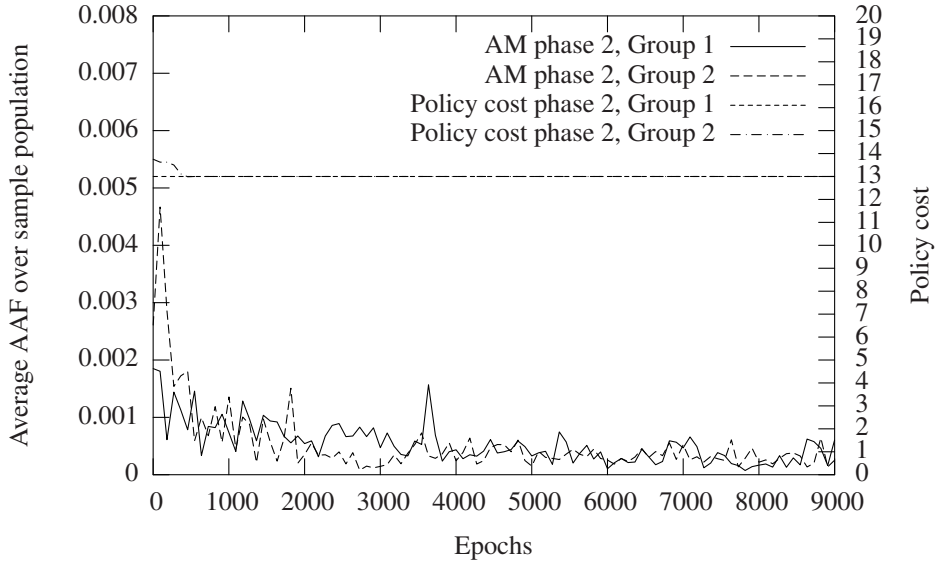


Figure 4.33: Experiment 2 (ARL lookahead): Associative memory average convergence, and average policy cost, over 8 samples

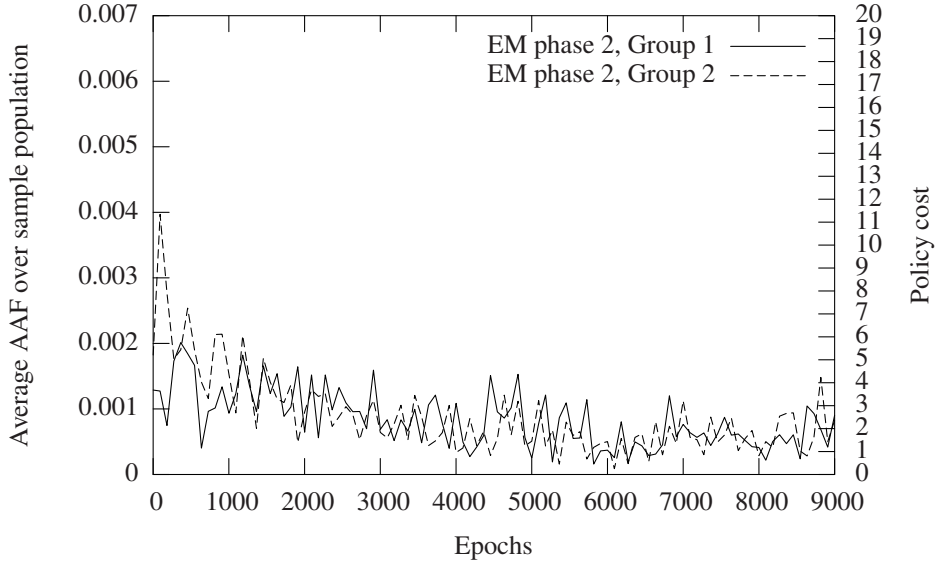


Figure 4.34: Experiment 2 (ARL lookahead): Expectance memory average convergence, and average policy cost, over 8 samples

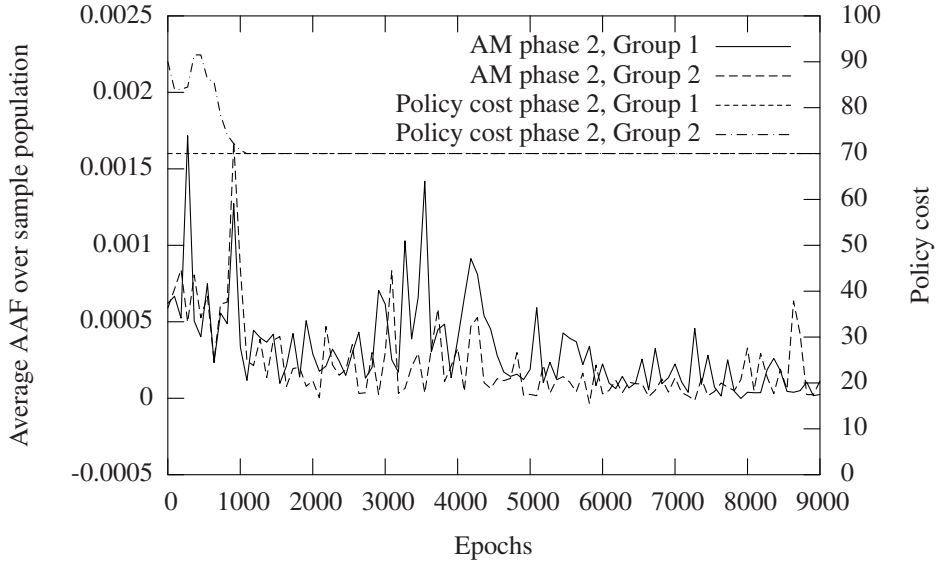


Figure 4.35: Experiment 3 (ARL lookahead): Associative memory average convergence, and average policy cost, over 8 samples

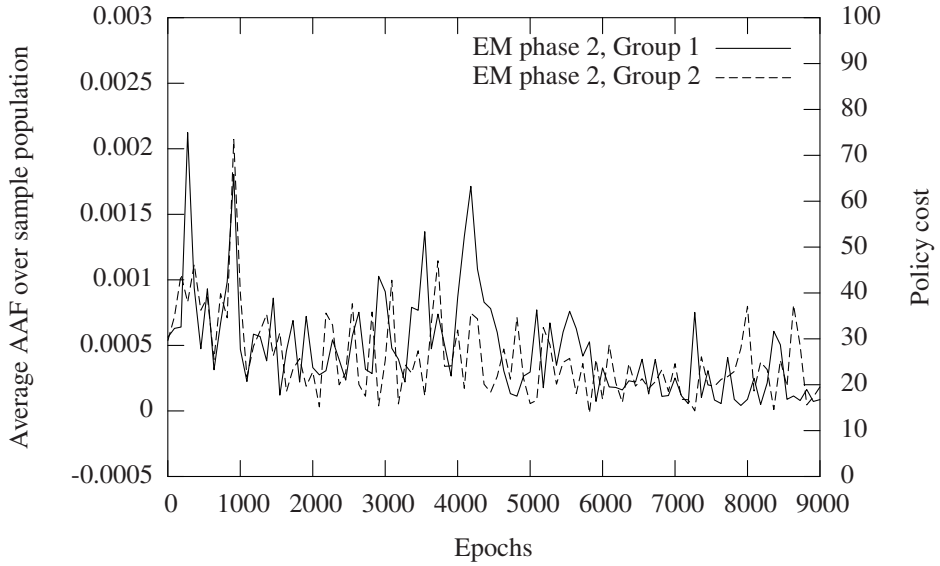


Figure 4.36: Experiment 3 (ARL lookahead): Expectance memory average convergence, and average policy cost, over 8 samples

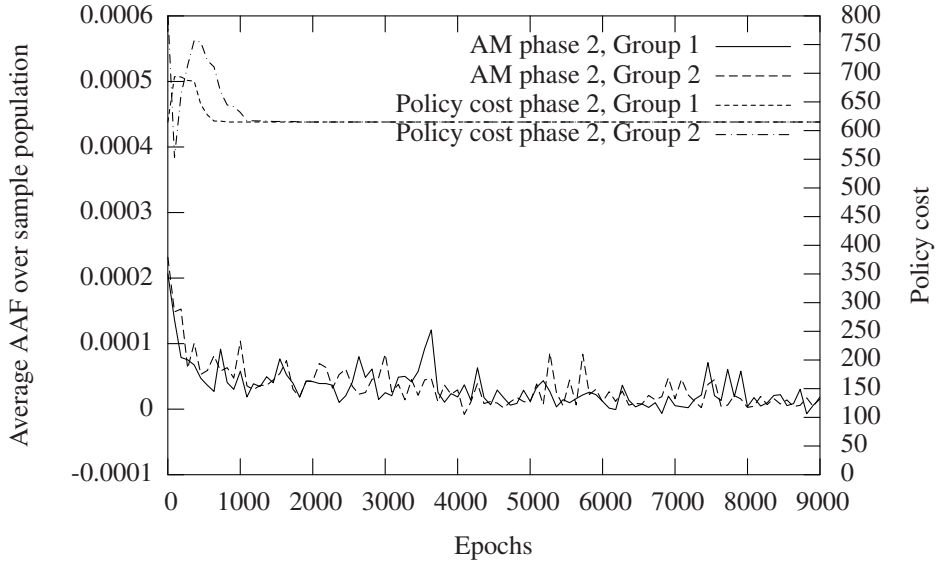


Figure 4.37: Experiment 4 (ARL lookahead): Associative memory average convergence, and average policy cost, over 8 samples

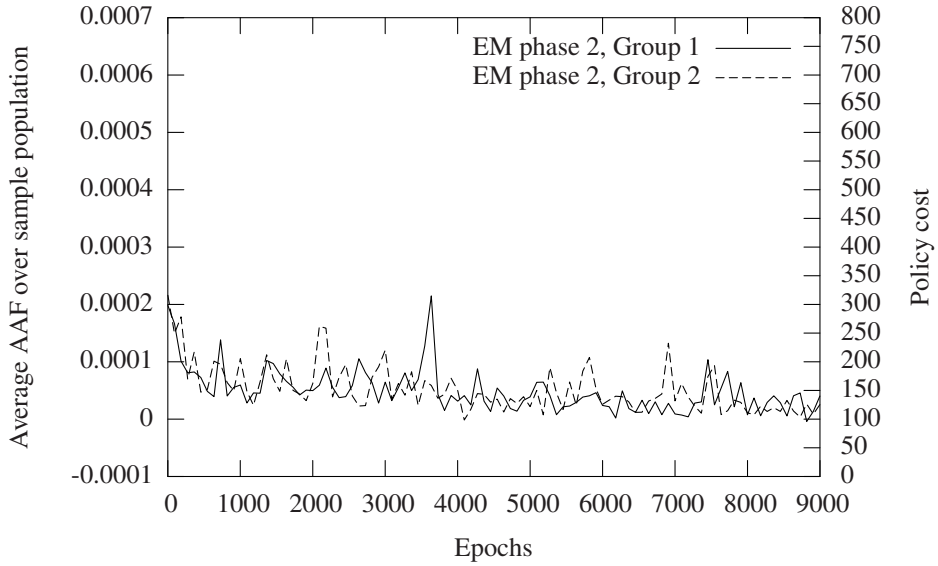


Figure 4.38: Experiment 4 (ARL lookahead): Expectance memory average convergence, and average policy cost, over 8 samples

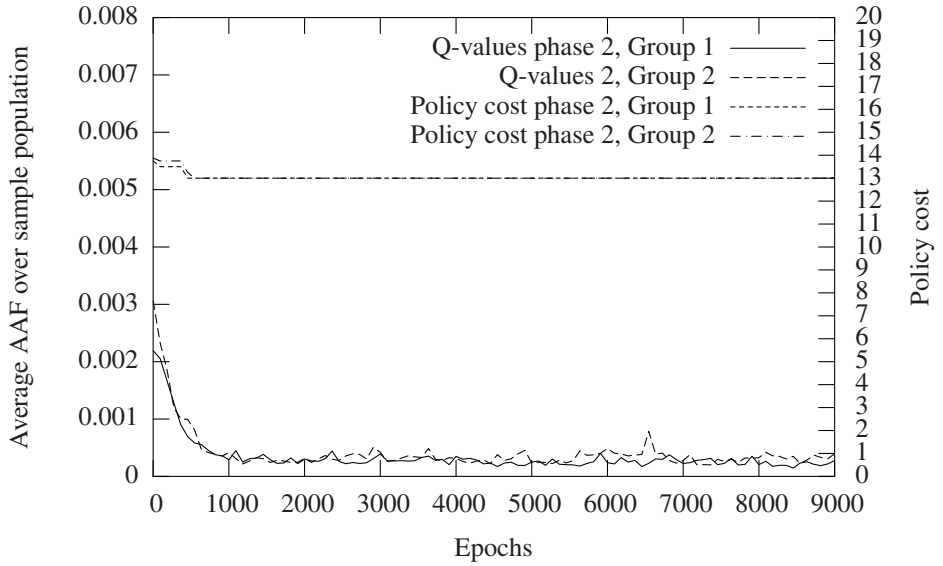


Figure 4.39: Experiment 1 (Q-learning): Associative memory average convergence, and average policy cost, over 8 samples

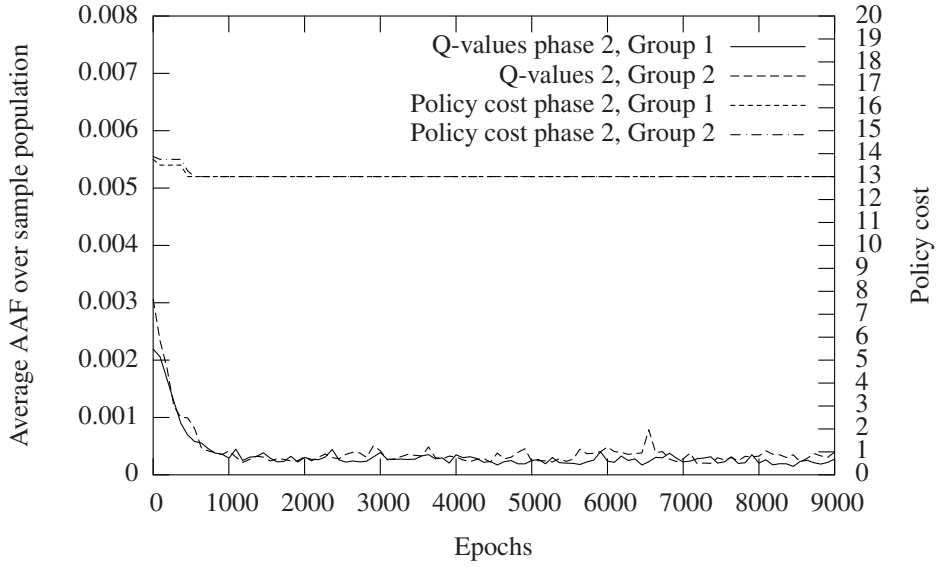


Figure 4.40: Experiment 2 (Q-learning): Associative memory average convergence, and average policy cost, over 8 samples

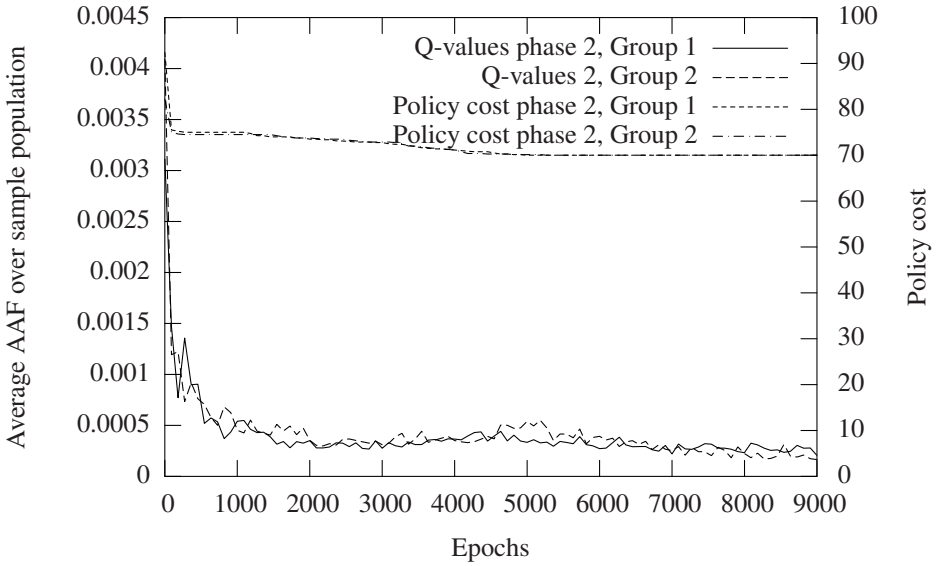


Figure 4.41: Experiment 3 (Q-learning): Associative memory average convergence, and average policy cost, over 8 samples

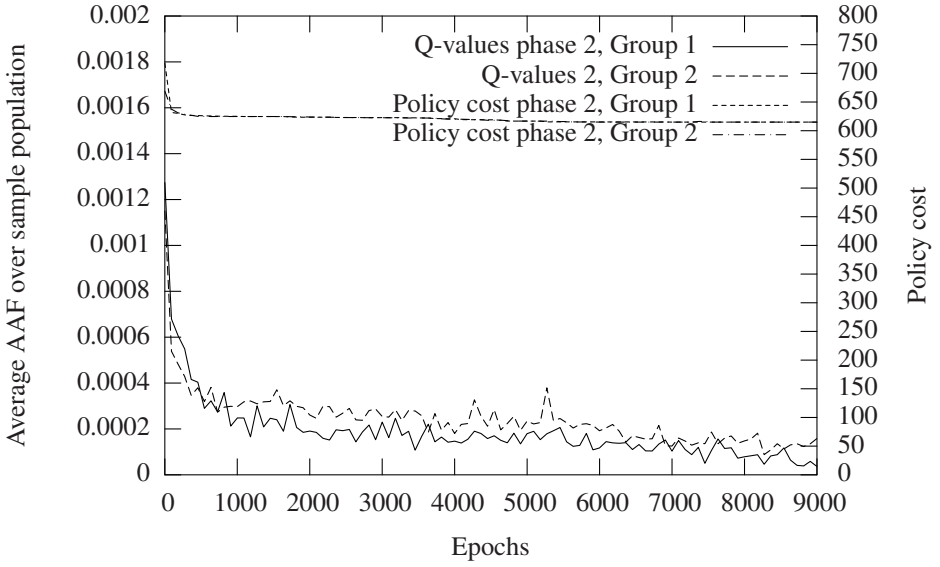


Figure 4.42: Experiment 4 (Q-learning): Associative memory average convergence, and average policy cost, over 8 samples

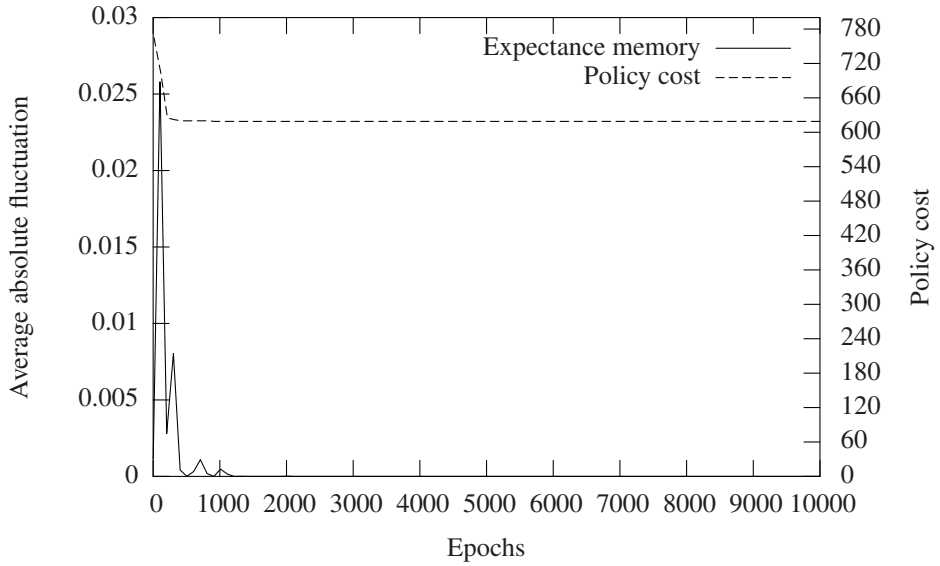


Figure 4.43: Avoidance of aversive stimuli experiment (ARL lookahead): Convergence of memory and policy cost

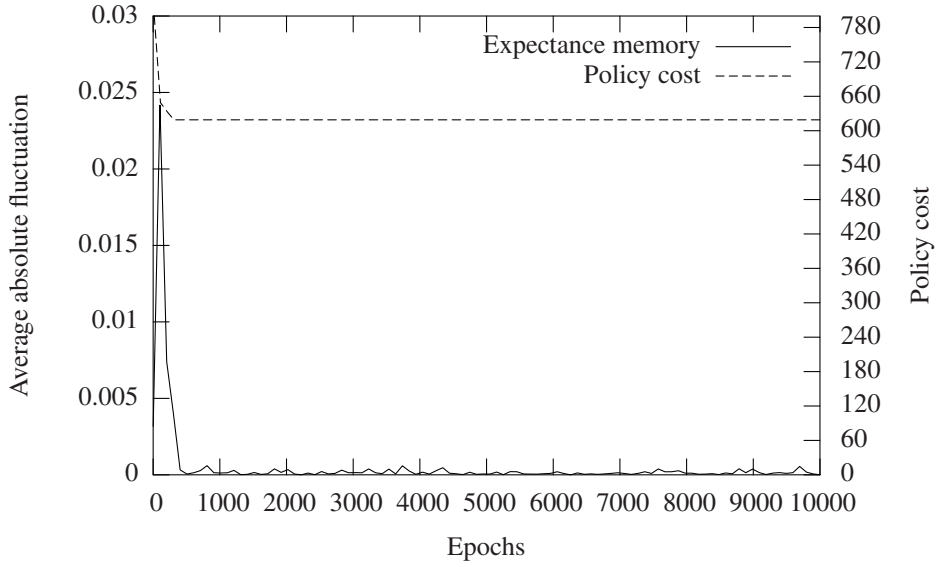


Figure 4.44: Avoidance of aversive stimuli experiment (Q-learning): Convergence of memory and policy cost

Chapter 5

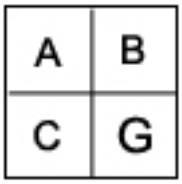
Discussion

5.1 Convergence

5.1.1 ARL Trace

For all experiments which are testing convergence, it is evident from the plots that the trace algorithm (8) does not converge. Neither the associative memory nor the expectance memory converges. However, the algorithm still finds the optimal policy in all cases. As can be seen from figure 4.7, 4.9, 4.15, 4.17, 4.23 and 4.25, the average absolute fluctuation increases as the number of epochs increases. The reason for this increase can be realized by analyzing the Trace algorithm. For first-order conditioning (i.e. lines 17-30 in algorithm 8), the algorithm only takes into account the immediate value of the outcome, and not any second-order strength. This is why the similarity threshold is needed on line 24. For this reason the Trace algorithm is unable to learn that an outcome has lost its value; it can only learn the value of an outcome once, and cannot reevaluate it below the level of the similarity threshold. A more serious problem with the Trace algorithm, however, is manifested in the way second-order conditioning is facilitated (lines 32-39 in algorithm 8). Second-order conditioning in Trace is based on the maximum associative strength propagated from first-order conditioning, and hence the algorithm fails to take into account the value of the outcome itself. For this reason the algorithm does not converge, but rather the opposite; the error increases as the number of epochs increase. An example will show why the associative memory and the expectance memory does not converge under the control of ARL Trace. Consider the Grid world in figure 5.1. The start state is the spatial location of stimulus A, and the goal state is the spatial location of stimulus G, which is appetitive. All other stimuli are initially neutral. Given two episodes (i.e. behavioral paths of the ARL agent), A - B - G (E_1), and A - B - C - G (E_2), the algorithm can be analyzed to see why it does not converge. In E_1 the association B→G is updated by first-order conditioning (lines 24-29 of algorithm 8). Additionally, stimulus B has the maximum associative strength with regards to stimulus G. This maximum strength, stored in the variable MAS for associative memory and $MAES$ for expectance memory (lines 20-21 of algorithm 8), is then transferred to the association A→B (second-order conditioning)

Figure 5.1: Sample 2x2 Grid world for ARL trace functionality demonstration



of episode E_1 (lines 35-38 of algorithm 8). So far, everything is fine. The core of the convergence problem for ARL Trace can be realized when analyzing what happens in episode E_2. In E_2, the association B→C represents the maximum strength for the first-order conditioning part of the trace algorithm (again lines 20-21 of algorithm 8). Because the agent is at the start of its learning period, stimulus B has not yet gained much associative strength with regards to stimulus C; in fact it will have gained as much strength as stimulus A has with regards to stimulus B from E_1. The point is, however, that in E_2, the associative strength of stimulus A with regards to stimulus B will be updated with less associative strength than in E_1, and hence the MAS and $MAES$ variable will be different from what they were in E_1 for association A→B. This is because the behavioral path of the agent is longer in E_2 than in E_1, and association A→B will gain second-order strength not from B→G, but from B→C. Because the associations are updated according to the behavioral path of the agent, the second-order associations will never reach a stable asymptotic level under continued exploration, due to the way in which second-order conditioning is facilitated in ARL trace.

The question is then: Why does the algorithm still find the optimal policy? To answer this question, it is necessary to look at the algorithm parameters. For the convergence experiments, the compound learning rate (i.e. $\alpha \times \beta$) is 0.01, and the exploration parameter ϵ is 0.8. This means that the greedy response is chosen in 80% of the cases. Additionally, the effects of any second-order conditioning update is minimized due to the small learning rate. Because the greedy response is chosen most of the time, the algorithm will mostly follow the currently known shortest behavioral path. When the algorithm occasionally explores, the second-order associations will in some cases be ”wrongly” updated (i.e. in the direction of a sub-optimal policy), but this error will be minimized by the small compound learning rate. Because the algorithm never stops exploring, the policy will never stabilize and the second-order association will continue to fluctuate according to the explorative behavior of the agent. In sum, under the given conditions, ARL trace succeeds in improving the policy by continued exploration (Policy improvement), but fails in converging the association values of the policy to their true value (Policy evaluation). The optimal policy value is affected by exploration, but the effect is minimized by the small learning rate. Thus, the agent is able to follow the optimal policy even though its true value is never found.

5.1.2 ARL lookahead

In contrast with ARL trace, the ARL lookahead algorithm converges for the experiments in the previous chapter. This is evident from the plots in figures 4.27, 4.29, 4.19, 4.21, 4.12, and 4.14. A natural question is then, why? Under certain conditions, which hold for the convergence experiments, it can be shown by example that ARL lookahead will converge. Consider table 5.1. The columns represent the number of steps from the goal, and the rows represent time. Given a static poliy π, of 4 steps length; step 4 of 5.1 represent stimulus D, step 3 stimulus C, step 2 stimulus B, step 1 stimulus A, and finally step 0 represent the goal stimulus G. There is thus a static behavioral path D - C - B - A - G, where stimulus D is at the start location, and G is at the goal location. Stimulus G is initially appetitive, whereas the other stimuli are initially neutral. At each step, there is one possible response, namely to move to the next step. Upon reaching the goal step (0, stimulus G), the agent is taken back to the start step (4, stimulus D). The asymptotic value of moving from stimulus A to the goal stimulus G, here called λ_G, is 1, and is given by:

$$\lambda_G = \max(\forall s \in ID : sim(s, US)) \times s.category$$

Furthermore, the value of moving from stimulus B to stimulus A, here called λ_A, is given by:

$$\lambda_A = \delta\lambda_G$$

A general equation for the value of the next step asymptote of the example in table 5.1 (for second-order conditioning), can then be derived. Let n be the number of steps from the goal. For $n > 1$, the equation is then:

$$\lambda_n = \delta^{n-1}\lambda_G \tag{5.1}$$

Second-order conditioning is thus given by the exponentially decreasing decay δ multiplied by the first-order asymptote λ_G. The ARL lookahead algorithm incorporates first-order and second-order conditioning in one equation to derive the asymptote, and thus effectively does the same thing as equation 5.1:

$$\lambda \leftarrow \max(\forall id \in ID : sim(id, ns)) + \delta \max(\forall o \in O : \sum_{ns \in NSc} V_{ns \rightarrow o}) \tag{5.2}$$

where s: current stimulus, ns: next stimulus,

NSc: next compound, ID: vector of internal drives

For equation 5.1, the term $\max(\forall id \in ID : sim(id, ns))$ will be zero when the next stimulus in the behavioral path is neutral, whereas the term $\delta \max(\forall o \in O : \sum_{ns \in NSc} V_{ns \rightarrow o})$ will be zero when the next stimulus is in the goal location. Thus for all neutral stimuli, except those signalling the goal immediately, equation 5.2 is equivalent to equation 5.1. Starting at time 1 and step 4 of table 5.1, equation 5.2 is then applied iteratively at each step towards the goal. We can then see that the goal asymptote λ_G propagates backwards, exponentially by the δ parameter, towards the start step 4 as time increases. For the example in table 5.1, the parameters of equation 5.2 rates

were set to $\alpha = 0.9$, $\beta = 0.9$, and $\delta = 0.9$. At time 21, the policy π has converged to its true value.

Regarding the necessary conditions of the ARL lookahead algorithm, the following can be noted. The decay parameter δ must be in the range $\langle 0, 1 \rangle$. If the decay parameter is 0, the agent will not learn any associatons except for those involving the goal. Contrarily, if the decay parameter is 1, the associative strength of all associations will eventually converge to the asymptote of the appetitive stimulus, that is, 1. A necessary assumption is also continued exploration; all states must be explored an infinite number of times in order to find the true policy value. ARL lookahead converges to the optimal policy for two reasons. First, it always estimates the optimal policy (due to the max lookahead term of equation 5.2). Secondly, once the policy has converged to its true value, the shortest path will always have the maximum associative strength (due to the exponential decay parameter). This can be seen from table 5.1; as the number of steps to the goal decreases, the associative strength increases according to equation 5.1. Finally, ARL lookahead is, as are other reinforcement learning algorithms, dependent on the Markov assumption (see section 2.1.1). If the environment does not provide a unique sequence of stimuli for all possible path combinations, the policy value will not converge for the episodic case (i.e. as in the example from table 5.1 and the experiments described in the previous chapter). Figure 5.2 illustrates the time-derivative convergence of the example in table 5.1. Refer to figure 5.3 for an example of a Grid World policy converged to its true value, as found by ARL lookahead. The lower right square is the goal location, and the upper left square is the start location. The number in each square, except the squares immediately adjacent to the goal, represents the decay exponent of the best response association for that square's stimulus as indicated by the black arrow, i.e. the association between the stimulus in that square with the response and the outcome stimulus (the square which the agent ends up in by eliciting the best response). Hence, the predictions of the 5x5 Grid world policy (fig. 5.3) equals those of table 5.1, for a decay parameter δ of 0.9.

5.1.3 Q-learning

For the baseline experiments (those testing convergence), the Q-learning algorithm is functionally equivalent to ARL lookahead (or vice versa). Thus it converges for all experiments given the same preconditions. In fact, the error correction rule used in ARL lookahead is equivalent to the temporal-difference error correction rule used by Q-learning, under the given conditions of the convergence experiments. These conditions are; only one stimulus per spatial location; and that reward is only given upon reaching the goal state, which is what makes the two error correction rules identical for the convergence experiments. Consider the Q-value update:

$$Q(s,a) + \alpha\big[r + \gamma max_{a'} Q(s',a') - Q(s,a)\big] \tag{5.3}$$

Under the given conditions, r is equivalent to λ_G or the term $\max(\forall id \in ID : sim(id, ns))$ from equation 5.2, which means that it is zero for all states except those

Table 5.1: Example of stepwise convergence of the ARL lookahead algorithm

Steps from goal					
4	3	2	1	0	Time
0	0	0	0	1	1
0	0	0	0.81	1	2
0	0	0.59049	0.9639	1	3
0	0.43046721	0.8148762	0.993141	1	4
0.313810596	0.67583352	0.878826267	0.99869679	1	5
0.552306649	0.769072717	0.895026951	0.99975239	1	6
0.665592274	0.798598463	0.898874613	0.999952954	1	7
0.708640812	0.807013301	0.89975188	0.999991061	1	8
0.722954451	0.809251648	0.899946341	0.999998302	1	9
0.727305797	0.809818696	0.899988567	0.999999677	1	10
0.72854593	0.809957217	0.899997592	0.999999939	1	11
0.728882538	0.809990116	0.899999498	0.999999988	1	12
0.728970477	0.809997756	0.899999896	0.999999998	1	13
0.728992755	0.809999498	0.899999979	1	1	14
0.728998257	0.809999889	0.899999996	1	1	15
0.728999588	0.809999976	0.899999999	1	1	16
0.728999904	0.809999995	0.9	1	1	17
0.728999978	0.809999999	0.9	1	1	18
0.728999995	0.81	0.9	1	1	19
0.728999999	0.81	0.9	1	1	20
0.729	0.81	0.9	1	1	21
0.729	0.81	0.9	1	1	22

Figure 5.2: Time derivative convergence of the ARL lookahead algorithm: from table 5.1

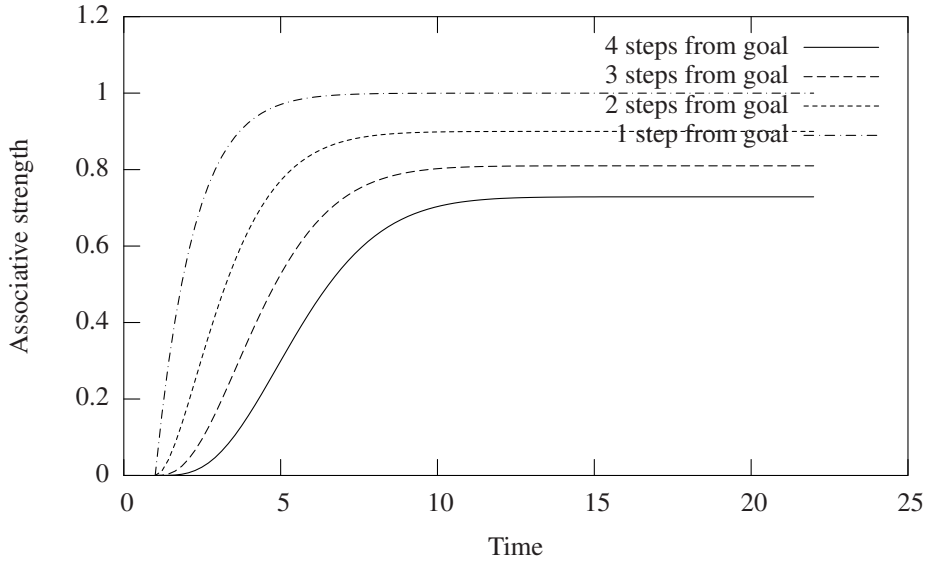
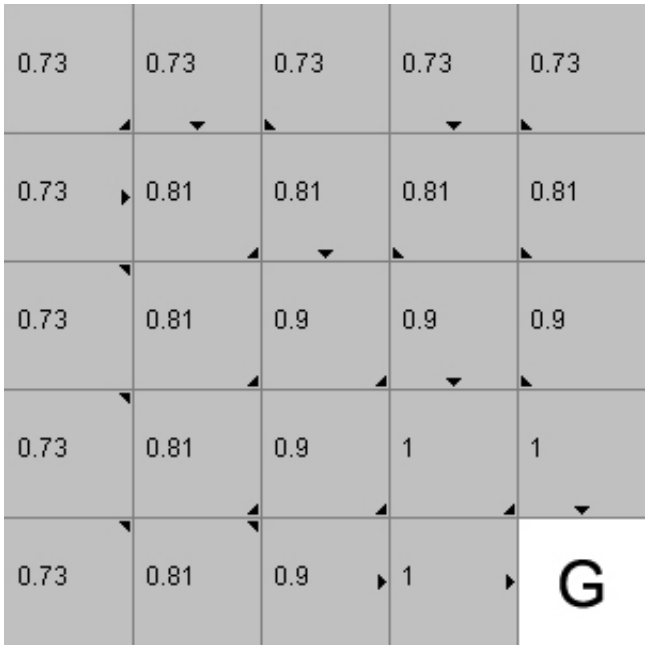


Figure 5.3: A 5x5 Grid World converged to the optimal policy by ARL lookahead



immediataly adjacent to the goal. Secondly, the term $\gamma max_{a'} Q(s', a')$ is equivalent to $\delta \max(\forall o \in O : \sum_{ns \in NSc} V_{ns \to o})$ from equation 5.2, so that it is zero for the states immediately adjacent to the goal. For the baseline convergence experiments the difference between the two algorithms, is hence that ARL lookahead derives the value of the outcome on its own by means of internal drives, whereas Q-learning is provided with a reward from the environment.

5.2 Generalization

5.2.1 ARL lookahead

According to the statistical analysis of the generalization experiments it was found that, for experiment 4, the ARL lookahead algorithm found the optimal policy faster in phase 2 of group 1 than in phase 2 of group 2. For experiments 1, 2, and 3, the policy cost did not change at all from phase 1 to phase 2 of group 1, whereas it increased and took some time to decrease again to the optimal policy cost for phase 2 of group 2. However, because the variances of the two groups for experiments 1, 2, and 3, for the test statistic, were significantly different, no t-test could be performed. Although the raw data showed no change in policy cost in phase 2 of group 1 for the first three experiment, this does not necessarily mean that the policy remained unchanged. Rather, the policy can change without incurring an increase in policy cost (i.e. there is more than one optimal policy). Both this, and why the algorithm found an optimal policy faster in phase 2 of group 1 for experiment 4, can be shown by analyzing the situation.

Consider a stimuli compound AX which signals the appetitive stimulus G. The asymptote λ_G of the stimulus G is 1. According to the [Rescorla & Wagner 1972] equation, which is incorporated in the ARL lookahead algorithm, stimulus A and X will enter into separate associations with stimulus G. More importantly, however, is the way in which stimulus A and X gains associative strength with regards to the asymptote λ_G. The strength of each association is updated on the basis of the discrepancy between λ_G and the aggregate associative strength of the compound, i.e. $\big[\lambda_G - (V_{A \to G} + V_{X \to G})\big]$. Assuming that stimulus A and X are equally salient, they will have the same learning rate α. The learning rate specific to stimulus G, β, is also considered constant. Hence, what will happen is that both associatons, A→G and X→G, will gain an equal amount of associative strength when the asymptote is reached. This can be seen from the following equations:

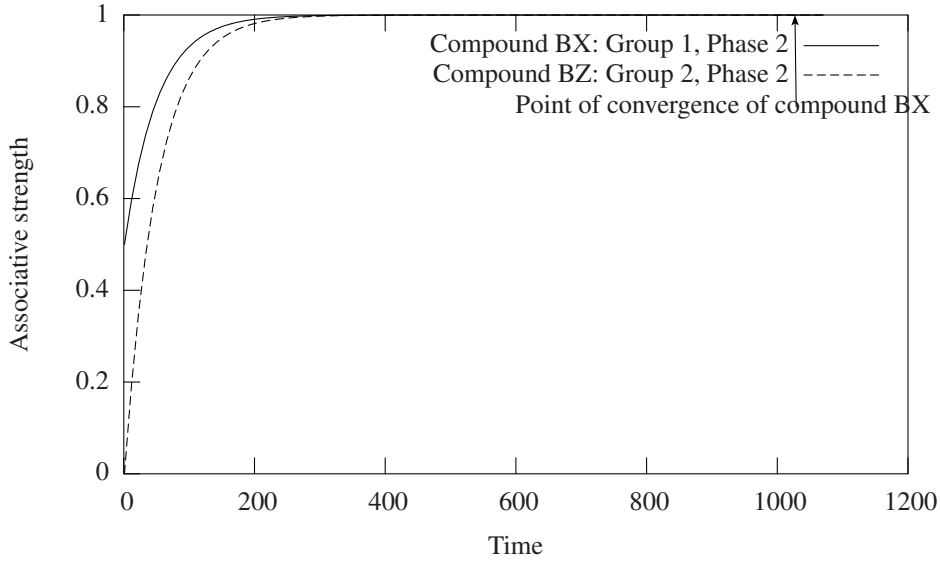
$$V_{A \to G,t} \leftarrow V_{A \to G,t} + \alpha\beta\big[\lambda_G - (V_{A \to G,t-1} + V_{X \to G,t-1})\big]$$
$$V_{X \to G,t} \leftarrow V_{X \to G,t} + \alpha\beta\big[\lambda_G - (V_{A \to G,t-1} + V_{X \to G,t-1})\big]$$

Both associations will thus eventually reach an associative strength of $\lambda_G/2$ when t increases infinitely. This is exactly what happens for the stimuli compounds in phase 1 of groups 1 and 2 for the generalization experiments. With this insight it is possible to analyze the generalization experiments. Now, consider what happens when stimulus A is replaced with stimulus B, yielding the new compound BX (as is done from phase 1 to phase 2 in group 1 for generalization experiments 1 and 2, see figures 3.5 and 3.7). Stimulus B will then have no associative strength with regards to G, whereas stimulus A

still will have an associative strength of 0.5, giving a compound associative strength for BX of 0.5. There are two effects of this for phase 2 of group 1. First, it will undoubtedly affect the policy, but not necessarily the policy cost. Secondly, as learning continues towards the asymptote λ in phase 2, stimulus X will gain more associative strength than stimulus B, because stimulus X already has an associative strength of 0.5. In associative learning theory, this is known as blocking, (or in this case partial blocking). This is a direct effect of the Rescorla-Wagner equation [Rescorla & Wagner 1972]. Because the associative error $\lambda_G - V_X$ is smaller for stimulus X than the associative error $\lambda_G - V_B$ for stimulus B, both stimuli will gain as much associative strength when the asymptotic level is reached (an increase of 0.25), but because stimulus B already had an associative strength of 0.5, there is only room for another 0.5 units of associative strength which is divided equally between stimulus X and B. This can be seen from table 5.2, where the asymptotic level is reached at time step 30. Next consider what happens for phase 2 of group 2. Here, the stimuli compound AX is changed to BY, and hence both stimulus B and Y will have an initial associative strength of 0 at the beginning of learning in phase 2. Table 5.3 illustrates the number of time steps before the asymptotic level is reached, which is 31. Both table 5.2 and 5.3 used a compound learning rate of 0.25 (0.5×0.5). For such a large learning rate the difference in asymptotic level of convergence is only 1 time step, between phase 2 of groups 1 and 2, according to table 5.2 and 5.3. However, as is indicated in the plot in figure 5.4, where the compound learning rate was set to 0.01 (α=0.1, β=0.1), the effects on convergence increases as the learning rate decreases: The associative strength of compound BX of Phase 2 of Group 1 converges to the asymptote at time step 1027, whereas the associative strength of compound BY of phase 2 of group 2 converges at time step 1062. More importantly, however, is the difference in associative strength between the compounds at the beginning of the learning periods of phase 2 of group 1 and 2. This difference, as will be shown below, is what really affects the policy value and implicitly the policy cost for the generalization experiments, under the given conditions and algorithm parameters.

By inspecting the policy costs, at the beginning of the learning period, for the ARL lookahead algorithm, it is evident what is happening in phase 2 of group 1 and group 2. Consider the policy cost of ARL lookahead, at the end of the learning period of phase 1 of group 1, for generalization experiment 1 as given in figure 5.5. Here, the policy has converged to its true value, and the optimal policy is stable. Moving on to the beginning of the learning period of phase 2 of group 1 (figure 5.6), it can be seen that the associative strength of the modified compound (in square (1,2)) has decreased to half the value of the initial compound of phase 1 of group 1, i.e. 0.5. This is because one stimulus was swapped with a new stimulus for phase 2. Now, by looking at figure 5.6, it can be seen that this change in compound associative strength with regards to the goal in the lower right corner, does not affect the policy cost. The path (1,3) - (2,2) - (1,1) is equal in length to the path (1,3) - (1,2) - (1,1). Hence there is no change in policy cost for phase 2 of group 1 in experiment 1. Furthermore, the associative strength of the changed compound towards any outcome in phase 2 of group 2 is 0 at the beginning of the learning period, because both stimuli are changed from phase 1 to phase 2. The associative memory does not yet know which response is best for the new compound. If the agent then by pure chance takes the path sub-optimal path (1,3) - (1,2) - (2,2), the new compound will update its associative strength towards the

Figure 5.4: Example: convergence of compound BX and BY of phase 2 of groups 1 and 2



stimulus in square (2,2). Because the agent has seen nothing else, this association will be the maximum of square (1,2). It is thereby obvious that the policy is sub-optimal at this stage, because the agent will choose the path (1,2) - (2,2) - (1,1), which is one step longer than the optimal path; (1,2) - (1,1). For phase 2 of group 1, this can not happen because the remaining stimulus still signals the goal location. Although not a full analysis of what happens when the stimuli compound is completely changed, this gives an idea of why the policy cost changes.

By applying the same type of analysis to generalization experiment 2, it can be realized why the policy cost changes in phase 2 of group 2, but remains unchanged in phase 2 of group 1. For experiment 2 the stimuli compound in square (3,3) is changed from phase 1 to phase 2. By referring to figure 5.9, it is seen that the remaining stimulus in phase 2 of group 1 holds half of the associative strength of the initial compound from phase 1, i.e. 0.45, which still signals the shortest path towards the goal in the lower right corner; (3,3) - (2,2) - (1,1). Therefore the policy cost remains unchanged in phase 2 of group 1. On the other hand, for the same reasons as in experiment 1, the policy cost changes in phase 2 of group 2. Again the compound associative strength of the two new stimuli is 0 at the beginning of the learning period of , that is, the agent does not yet know which is the best response in square (3,3). If the agent then moves from square (3,3) to square (2,3) in figure 5.9, the policy will be sub-optimal.

In generalization experiment 3, not one, but two stimuli compounds are changed from phase 1 to phase 2. According to the results statistics the policy cost does not

Table 5.2: Learning for compound BX in phase 2 of group 1

V_X	V_B	Time
0.5	0	1
0.625	0.125	2
0.6875	0.1875	3
0.71875	0.21875	4
0.734375	0.234375	5
0.7421875	0.2421875	6
0.74609375	0.24609375	7
0.748046875	0.248046875	8
0.749023438	0.249023438	9
0.749511719	0.249511719	10
0.749755859	0.249755859	11
0.74987793	0.24987793	12
0.749938965	0.249938965	13
0.749969482	0.249969482	14
0.749984741	0.249984741	15
0.749992371	0.249992371	16
0.749996185	0.249996185	17
0.749998093	0.249998093	18
0.749999046	0.249999046	19
0.749999523	0.249999523	20
0.749999762	0.249999762	21
0.749999881	0.249999881	22
0.74999994	0.24999994	23
0.74999997	0.24999997	24
0.749999985	0.249999985	25
0.749999993	0.249999993	26
0.749999996	0.249999996	27
0.749999998	0.249999998	28
0.749999999	0.249999999	29
0.75	0.25	30

Table 5.3: Learning for compound BY in phase 2 of group 1

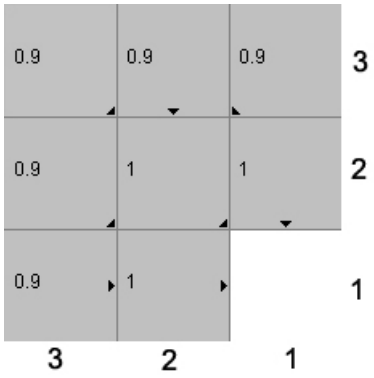
V_X	V_B	Time
0	0	1
0.25	0.25	2
0.375	0.375	3
0.4375	0.4375	4
0.46875	0.46875	5
0.484375	0.484375	6
0.4921875	0.4921875	7
0.49609375	0.49609375	8
0.498046875	0.498046875	9
0.499023438	0.499023438	10
0.499511719	0.499511719	11
0.499755859	0.499755859	12
0.49987793	0.49987793	13
0.499938965	0.499938965	14
0.499969482	0.499969482	15
0.499984741	0.499984741	16
0.499992371	0.499992371	17
0.499996185	0.499996185	18
0.499998093	0.499998093	19
0.499999046	0.499999046	20
0.499999523	0.499999523	21
0.499999762	0.499999762	22
0.499999881	0.499999881	23
0.49999994	0.49999994	24
0.49999997	0.49999997	25
0.499999985	0.499999985	26
0.499999993	0.499999993	27
0.499999996	0.499999996	28
0.499999998	0.499999998	29
0.499999999	0.499999999	30
0.5	0.5	31

change for phase 2 of group 1, but does for phase 2 of group 2. In figure 5.12 the compound associative strength of the two changed compounds, at squares (3,3) and (2,2), have half the associative strength of the initial value in figure 5.8; 0.45 and 0.5. For phase 2 of group 2 (fig. 5.10, the compound associative strength is zero for both the changed compounds. As usual one stimulus is changed for each compound from phase 1 to phase 2 in group 1, whereas both stimuli in each compound are changed from phase 1 to phase 2 in group 2. As pointed out earlier, even though the policy cost does not change, the policy can still change to an alternative optimal policy. This has most likely happened in phase 2 of group 1 for experiment 3. However, the remaining stimulus in each compound for phase 2 in group, still signals the optimal path towards the goal. This is why the agent will not be lead along a sub-optimal path by the remaining stimuli of the changed compounds. Still, because the changed compound of phase 2 of group 1 has not yet regained its true value, the agent will take an alternative path around the change compounds until the true value has been reinstated. The alternative path in phase 2 of group 1 does not incur an extra cost under the given circumstances. For phase 2 of group 2, the policy cost will increase for the same reasons as in experiment 1 and 2, but because more is changed and the environment is more complex, the effect on the policy cost is greater.

Experiment 4 is a more interesting case because the policy cost changes in phase 2 of both group 1 and 2. According to the statistical analysis, however, the optimal policy is more quickly refound in phase 2 of group 1 than in phase 2 of group 2. Consider figure 5.14 for the initial policy value after learning is completed in phase 1 of group 1, and figure 5.15 for the policy value at the beginning of the learning period in phase 2 of group 1. In experiment 4, the compounds at squares (4,4), (3,3), and (2,2) are changed from phase 1 to phase 2. Thus, the respective compound values are half of the values in figure 5.14, i.e 0.4, 0.45, and 0.5, for phase 2 of group 1. For phase 2 of group 2, figure 5.16, the compound associative strength is 0 for all three compounds. For an insight into why the policy cost changes in phase 2 of group 1, consider the following path taken randomly by the agent; (5,5) - (4,5) - (3,4) - (2,3) - (1,2) - (1,1). This path is sub-optimal by one cost unit, but because the original optimal path (5,5) - (4,4) - (3,3) - (2,2) - (1,1), (which goes along the changed stimuli compounds), has lost half of its associative strength, the agent will continue to estimate the value of the sub-optimal path until the optimal path has regained its true value. It can therefore be concluded that, when one stimulus is changed for each of the three compounds, the policy temporarily loses its true value, but the remaining stimulus of the changed compounds continues to maintain the optimal path although at an initially lower associative value. Therefore the optimal path will temporarily be avoided until it has regained its true value. On the other hand, when the compounds are changed altogether as in phase 2 of group 2, there is nothing left in the changed compounds to signal the best response, and consequently the agent will in many cases choose a sub-optimal policy due to necessary exploration. The error in policy value will then propagate to other spatial locations, and it will take longer time to refind the optimal policy in phase 2 of group 2.

To summarize the generalization experiments for the ARL lookahead algorithm: In phase 2 of Group 1 the policy will change (either to a sub-optimal policy or to an alternative optimal policy), but the remaining stimuli in the changed compounds still

Figure 5.5: Generalization experiment 1 (ARL lookahead):Policy value @ end of learning period for phase 1 of group 1



signal the optimal path, and have a higher level of compound associative strength than the changed stimuli compounds in phase 2 of group 2. Therefore, due to remaining but weaker associations, the optimal policy will more quickly be reinstated in phase 2 of group 1 than in phase 2 of group 2 (which has no remaining associations in the changed compounds). Thus the OPR statistic will be lower in phase 2 of group 1, than in phase 2 of group 2. This applies to all 4 generalization experiments, under the given conditions and algorithm parameters.

5.2.2 Q-learning

As described earlier, Q-learning considers states (stimuli compounds) as irrediculble entities. For the implementation of Q-learning in this thesis, it thus maintains a configural approach to stimuli compounds, or in other words, each compound is unique no matter if it shares stimuli with other compounds. As long as the compounds differ on some account they are considered different and unique. For this reason, there is an equal amount of difference between phase 1 and 2 of both group 1 and 2. This can be seen from figures 5.18 and 5.19, where the initial policy value at the beginning of the learning periods for phase 2 of group 1 and 2 are illustrated. Hence, for the illustrated experiment (number 4), where the compounds in squares (4,4), (3,3), and (2,2) are changed from phase 1 in group 1 (figure 5.17) and phase 1 in group 2 respectively, the policy value will be 0 for the changed compounds in phase 2. Therefore, the Q-learner agent will suffer from the exact same problem in phase 2 of both group 1 and 2, as the ARL lookahead algorithm did in phase 2 of group 2 for all four generalization experiments. The same analysis therefore applies to the Q-learnier agent for each generalization experiment. In other words, it has to learn as much in phase 2 of both group 1 and 2 as ARL lookahead has to learn in phase 2 of group 2. It can be concluded that the statistical analysis of the previous chapter correctly indicates that there is no difference between groups 1 and 2 for Q-learning, and thus that it is not able to generalize

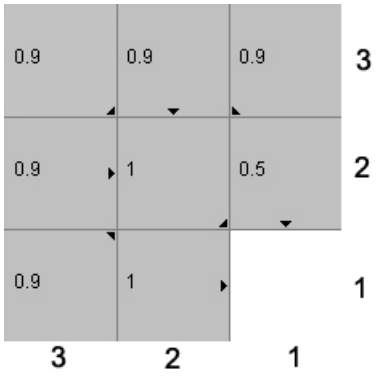


Figure 5.6: Generalization experiment 1 (ARL lookahead):Policy value @ beginning of learning period for phase 2 of group 1

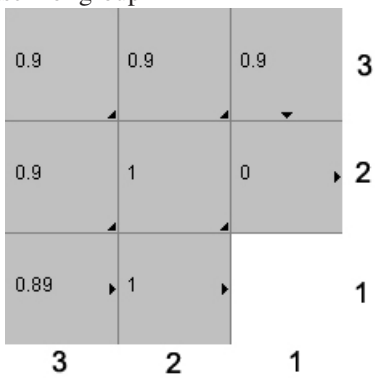
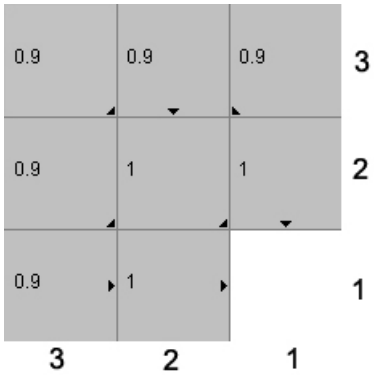


Figure 5.7: Generalization experiment 1 (ARL lookahead):Policy value @ beginning of learning period for phase 2 of group 2

Figure 5.8: Generalization experiment 2 (ARL lookahead):Policy value @ end of learning period for phase 1 of group 1



under the given experiment conditions and algorithm parameters.

5.3 Avoidance of aversive stimuli

As its name implies the point of the aversive stimuli avoidance experiment was to show that the agent took an alternative path around the aversive stimulus. This is exactly what happens. Consider the policy illustrations in figure 5.20 and 5.21. Square (6,6) contains the aversive stimulus. The raised arrows of each figure show the alternative path found by the agent. This alternative path adds an extra 4 policy costs units to the optimal cost of 615, resulting in a policy cost of 619. Upon reaching square (6,6) the Q-learner agent will receive a reward of -1, whereas the ARL agent will perceive the aversive stimulus, compare it to its list of internal drives, and derive a value of -1. All associations involving the aversive stimulus will therefore be negative for the expectance memory of the ARL agent. Similarly, for the Q-learning agent, the state-action values signalling the aversive stimulus will also be negative. Square (6,6) will therefore be avoided by both Q-learning and the ARL lookahead algorithm. Notice also the policy value in square (10,10), which normally (withouth any aversive stimulus at square (6,6)) would be 0.43 (δ^8; 9 steps from the goal), but is now 0.39 (δ^9: 10 steps from the goal). Consequently, by adding the aversive stimulus at square (6,6) the policy cost will increase, and the policy value will decrease for the associations along the affected paths.

5.4 Generalizability of results

No claim is made about the generalizability of the above results in domains other than the Gridworld environment. The Gridworld environment is an abstract ”toy problem environment”, and for this project the problem facing the agent has been one of finding

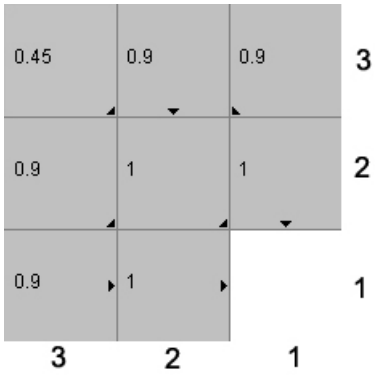


Figure 5.9: Generalization experiment 1 (ARL lookahead):Policy value @ beginning of learning period for phase 2 of group 1

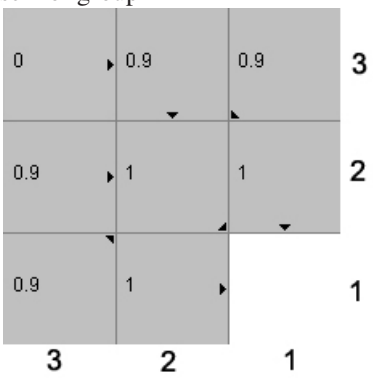
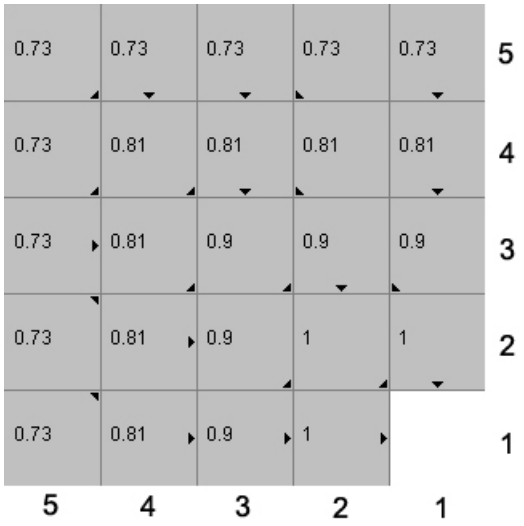


Figure 5.10: Generalization experiment 2 (ARL lookahead):Policy value @ beginning of learning period for phase 2 of group 2

Figure 5.11: Generalization experiment 3 (ARL lookahead):Policy value @ end of learning period for phase 1 of group 1



the shortest path from one spatial location to another by means of associating stimuli from one spatial location to another. Also, for such a problem, it has been shown how the agent can avoid spatial locations with low utility (i.e. aversive stimuli), and generalize one set of stimuli in a Gridworld environment, to another set of stimuli in a Gridworld environment, when these two environment share most elements but differ on a few. As a theoretical framework, the Gridworld is very popular in the field of Reinforcement learning (see [Sutton & Barto 1998]). This is because, by definition, all states are unique (when 2d coordinates are used as state description), and the state space is small and manageable so that lookup tables can be utilized. Hence, the Markov property is easily maintained. In this project the state description is not automatically unique, because stimuli compounds are utilized for state description. However, for all experiments each spatial location have been defined with a unique stimuli compound. Therefore, the Grid world, as defined in the experiments of this project, meet the assumptions of the original simple Gridworld definition which uses 2d coordinates for state description. The Gridworld environment is popular because many real problems, such as board games, can be abstracted in terms of shortest path solutions (i.e. where reward expectance is highest). It can be questioned, however, whether any given problem can be stated and solved in terms of reward maximization. Ignoring this question, real world problems with complex environments have sophisticated state descriptions (just consider the number of stimuli for a board configuration in the game of chess, or merely describing what a stimulus is for this game). In real problems, there is always an intricate correlation between elements of the state description, and any decision (the choice of response) would necessarily need to involve some thought process or

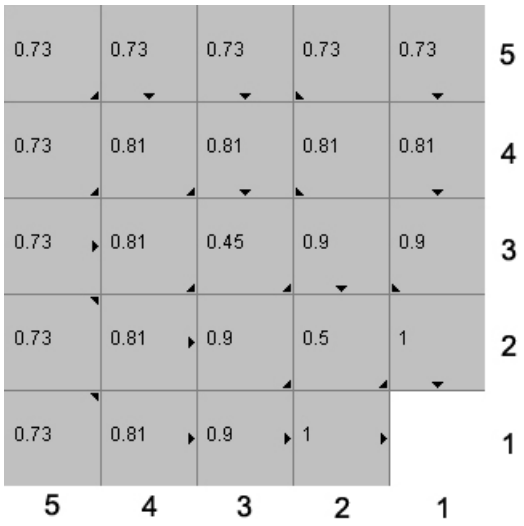


Figure 5.12: Generalization experiment 3 (ARL lookahead):Policy value @ beginning of learning period for phase 2 of group 1

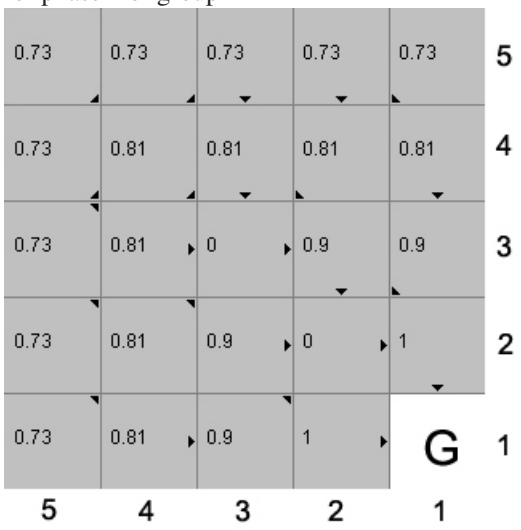
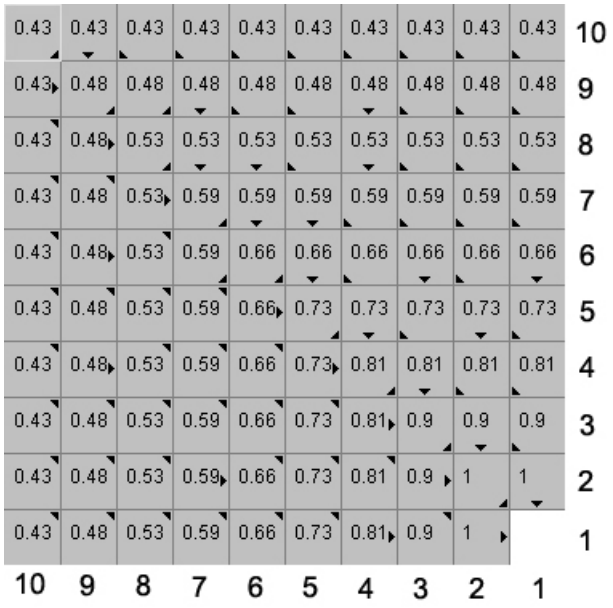


Figure 5.13: Generalization experiment 3 (ARL lookahead):Policy value @ beginning of learning period for phase 2 of group 2

Figure 5.14: Generalization experiment 4 (ARL lookahead):Policy value @ end of learning period for phase 1 of group 1



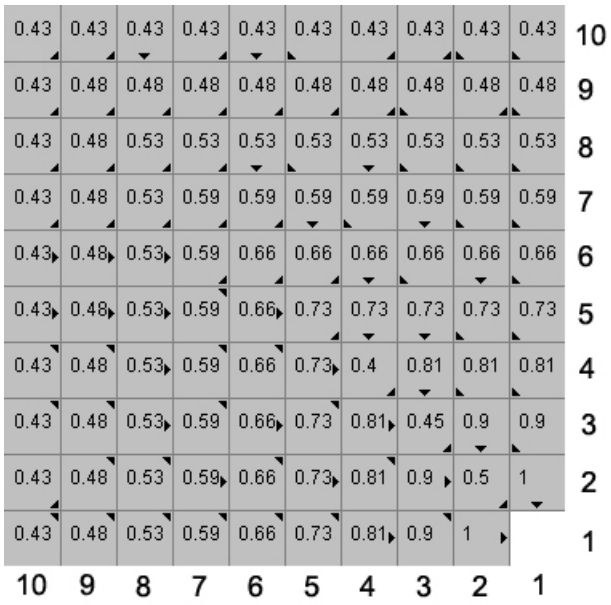


Figure 5.15: Generalization experiment 4 (ARL lookahead):Policy value @ beginning of learning period for phase 2 of group 1

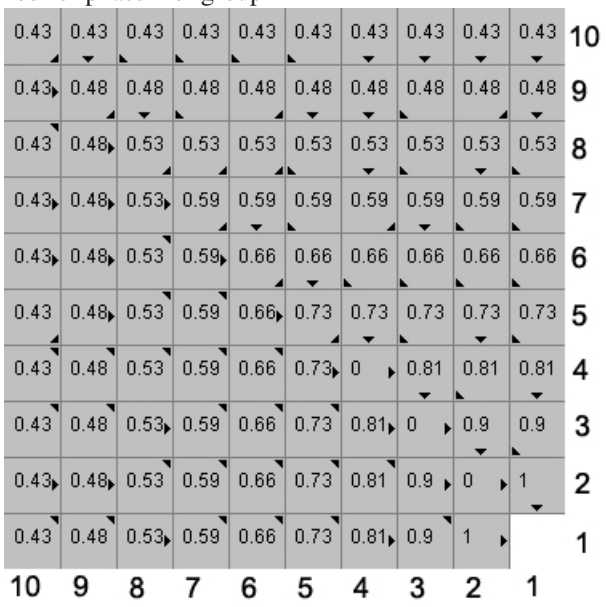


Figure 5.16: Generalization experiment 4 (ARL lookahead):Policy value @ beginning of learning period for phase 2 of group 2

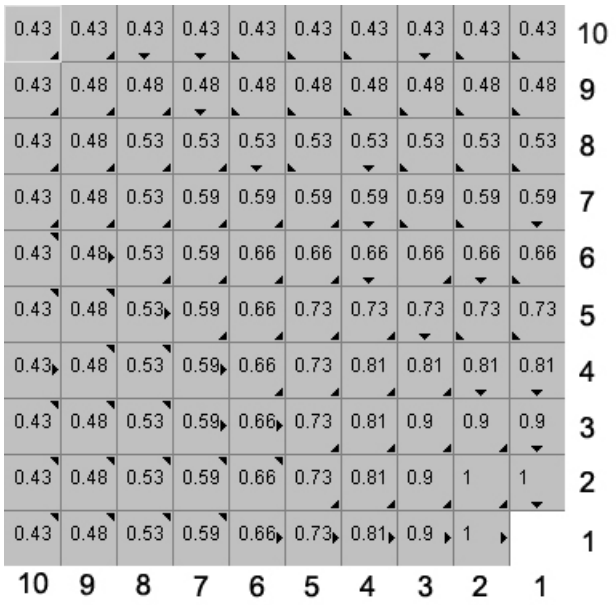


Figure 5.17: Generalization experiment 4 (Q-learning):Policy value @ end of learning period for phase 1 of group 1

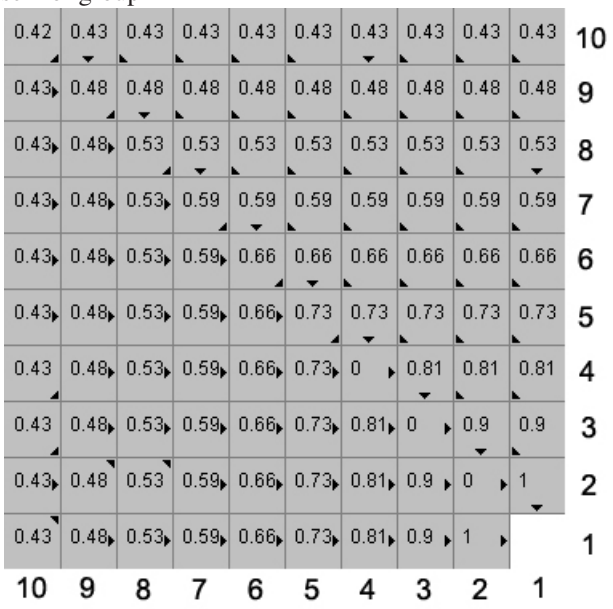
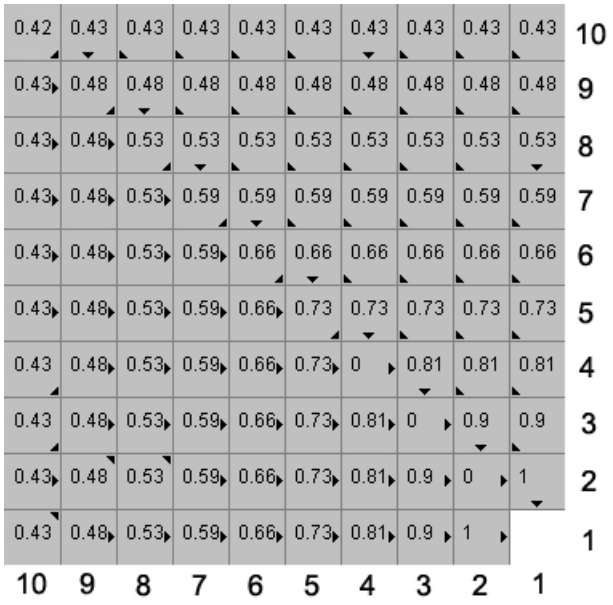


Figure 5.18: Generalization experiment 4 (Q-learning):Policy value @ beginning of learning period for phase 2 of group 1

Figure 5.19: Generalization experiment 4 (Q-learning):Policy value @ beginning of learning period for phase 2 of group 2



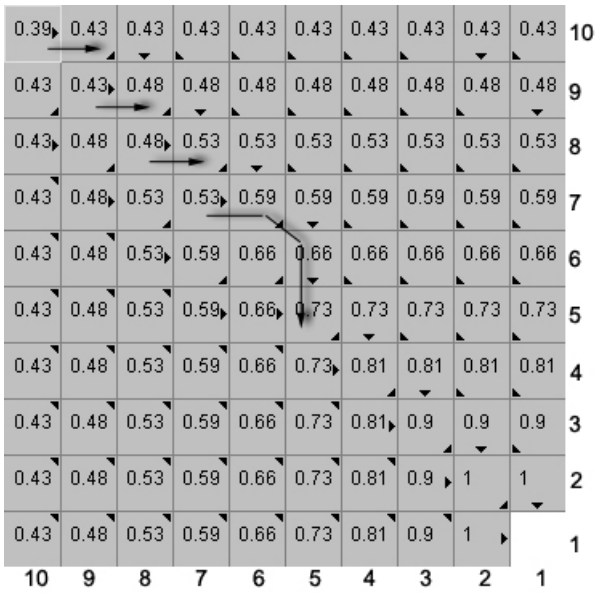


Figure 5.20: Avoidance of aversive stimuli experiment (ARL lookahead): Optimal policy cost @ end of learning period

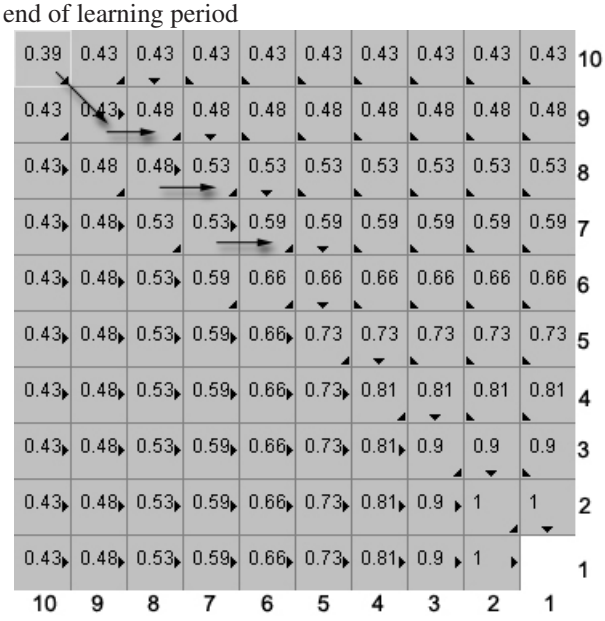


Figure 5.21: Avoidance of aversive stimuli experiment (Q-learning): Optimal policy cost @ end of learning period

model of the environment dynamics to understand this correlation. The Gridworld environment elegantly avoids this kind of complexity, because the problem is only one of changing position from one location to another with a minimum number of movements. Hence the agent can follow the shortest path according to reflexive responses which react most strongly to the maximum reward expectance. There is no thought process involved, and the environment model is no more complex than a causal map showing which state immediately follows another state. To apply the ARL lookahead algorithm to any other problem than the Gridworld environment, the concept of stimulus and stimuli compounds would need to be redefined, and a significant amount of engineering would most likely be needed. On the other hand, the same can be said for Q-learning. It is not an all-purpose problem solver which can be applied blindly without analysis of the state description and environment model. However, given a stimulus definition and an environment which meet the conditions and assumptions of the experiments in this project, it is probable that the ARL lookahead algorithm can have some use and show generalization for other problems too.

5.5 A note on combining the expectance memory and the associative memory

For static environments, as in the convergence experiments, combining the expectance memory and the associative memory has the effect of squaring the associative strength, because the associative memory and the expectance memory will gain the same amount of strength. For example, the association A→G from the associative memory might have a strength of 0.9, whereby the association A→(R→G) from the expectance memory will have the same strength. The combined value will then be 0.81, due to the choose response function:

$$cR(Sc) \leftarrow (\forall r \in R, \forall o \in (Sc \rightarrow O) : \max_{r} \big(\sum_{s \in Sc} V_{s \rightarrow (r \rightarrow o)} * V_{s \rightarrow o} \big)) \qquad (5.4)$$

This does not affect the policy value or the convergence to the optimal policy. The only effect is on the choice of responses, but because the combination of memories for the choose response function effectively results in squaring the associative strength, it is not affected either. Therefore the combination of memories is of no use in simple static environments as employed for the convergence experiments and the avoidance of aversive stimuli experiment. On the other hand, although not analyzed above, the generalization experiments involved an ARL lookahead algorithm using combined memories. The analysis of generalization assumed an ARL lookahead algorithm using the expectance memory only, because it is thought that the combination of expectance memory and associative memory would have no effect on the ”dynamic” environments of the generalization experiments. This might not be the case though. The only way to find out if the combination of memories does affect the generalization experiments, would be to run them again using only the expectance memory. If it is then shown that there is a difference between using combined memories and expectance memory only, the above analysis might be incomplete. It must be stressed that the generalization experiments were run with combined memories, and that this might be the reason why

the ARL lookahead algorithm refound the optimal policy faster in phase 2 of group 1 than in phase 2 of group 2. Testing this has been beyond the time-limits of this project, but the combination of memories might have had an unforeseen effect that the author was unaware of.

5.6 Extinction

Although not tested in any of the experiments, the ARL agent will by definition be able to reevaluate outcomes, and thereby determine if they have lost their value. This is due to the definition of internal drives, which the ARL agent uses to derive outcome values. If a drive corresponding to a specific outcome ceases to exist, the outcome will be worth 0 (for first-order conditioning), and consequently the associative strength of stimuli that have been previously associated with the outcome will decrease in strength.

5.7 The XOR problem revisited

It shall be noted that with the elemental approach of association formations that has been employed in the ARL agent's associative memory and expectance memory in this thesis, the XOR problem cannot be solved. Actually, the problem has been avoided by all experiments (i.e. all experiments employ environments with unique stimuli compounds across all spatial locations). However, as proposed by [Rescorla & Wagner 1972], the XOR problem can be remedied by adding a separate and unique configural unit for the stimuli compound as a whole, i.e. as in Q-learning. This unit will then gain as much strength as the sum of elemental strength in combination with any given compound. For compounds where one stimulus is discriminatory towards an aversive outcome, but the other stimuli of the compound signal the outcome as appetitive, an additional configural unit can be applied in order cancel the associative strength of the stimuli which falsely signal the outcome as appetitive. It is not straightforward, however, for the algorithm to "know" when to apply learning to the configural unit, and when to apply learning to the elemental components.

5.8 Objectives

The objectives of this project were defined to modify the Q-learning framework, and thereby aiming at improving the algorithm. It has been showed above that, under the conditions of the experiments, the Q-learning algorithm and the ARL lookahead are functionally equivalent. This is explained in section 5.1.3. The difference betweeen the ARL lookahead and Q-learning is purely representational. As described in section 5.2, this is what makes the ARL lookahead algorithm capable of generalization. Reconsider the objectives of this project:

1. To incorporate psychological bias in the architecture of the agent that will guide its learning process. Internal drives will be introduced, that will make the agent approach appetitive stimuli and avoid aversive stimuli.

2. To endow the agent with the ability to form other types of association other than S-R associations. Associative theory envisages three types of association:
    - Stimulus-Response (S-R) associations.
    - Stimulus-Stimulus (S-S) associations.
    - Response-Outcome (R-O) associations.
3. To take into account associative theory's conception of event representation.
4. To redefine outcomes as comprising sensorial and motivational elements.
5. To take into account the fundamental conditions of association formation proposed by associative theory.

Each objective, as described in the above list, has been met, and together provide a sufficient framework for generalization. Objective 1 is necessary for the agent to be able to derive outcome values. In order to be able to derive outcome values in the first place, the agent must be able to anticipate the outcome. Hence, the outcome must be incorporated in each association. Therefore, objective 2 is necessary in order to derive outcome values. Objective 3 has redefined the state signal as a compound of stimuli, each of which can enter into separate associations with the outcome. This is actually the core of the generalization framework, and is related with objective 4. Because the agent now derives outcome values on its own, on the basis of sensorial and motivational elements of stimuli compounds and the provision of internal drives, a burden is removed from the environment designer of defining rewards. Consider the complexity of having to define separate rewards for each stimulus of a compound. Thus, because the agent itself derives outcome values, it will analyze the stimuli compound in order to differentiate what is aversive/appetitive in the compound, and thereby form separate associations for each stimulus of the compound. Finally, objective 5 is a necessary modification to the error correction rule, which allows each stimulus of a compound to enter into separate associations; the [Rescorla & Wagner 1972] equation. It is because of these separate associations that the ARL agent is able to generalize when one stimulus of a compound is changed and one stimulus is not, from one situation to another. The agent will then be able to use the remaining association to signal the optimal response. It can therefore be concluded that the implemented objectives provide a sufficient framework for generalization.

Chapter 6

Evaluation and Conclusions

6.1 Conclusion

The aim of this project was to improve Q-learning, by developing a synthesis between associative learning theory from Psychology and reinforcement learning. This aim was manifested in the objectives of this project. To see whether this have been achieved, the motivation behind the project needs to be revisited. Four problems were identified with the Q-learning framework:

1. **Exploration-exploitation equilibrium:** The agent has to explore the environment in order to find the true value of a policy, which is necessary for policy improvement. Hence, to be certain that the optimal policy has been found, its true value has to be known. The problem is that the agent does not know when to stop exploring, because it does not know if all states have been visited.

2. **Temporal discounting:** The agent needs to maintain a balance between immediate greediness and long-term reward maximization. This is done by providing a discount factor γ. Consequently, the problem is according to [Alonso & Mondragón 2005], that a small discount factor may result in sub-optimal policies because the algorithm may converge prematurely to a sub-optimal goal. On the other hand, a large discount factor will slow down learning, because the agent has to visit more states to find the optimal policy.

3. **Generalization:** Because the Q-learning algorithm considers states as irreducible entities, it cannot retain reward prediction from one state to another even though they share common features. Consequently, even the slightest change in the environment will cause a complete loss of policy value for the affected states. The affected q-values will then need to be relearned, something which affects other unchanged states as well as the error propagates.

4. **Large sized problems:** Relying upon irreducible state descriptions and correct reward prediction (i.e. policy value), the Q-learning algorithm cannot successfully reuse q-values from a small learning environment to a larger environment.

When the problem increases in size, the Q-learner has to relearn the policy value. Additionally, similar states may reoccur from time to time in larger problems. This is something that Q-learning cannot exploit. Therefore, for large sized problems, learning will take even longer time.

The above problems can now be discussed in relation with the objectives and with regards to the research questions, to conclude the findings of this project:

1. **Exploration-exploitation equilibrium:** By considering exploration as a drive, S-S associations will form, allowing the agent to form a model of the environment, which in turn will help to discard unsuccessful exploratory policies.

2. **Temporal discounting:** Factors other than the immediate temporal contiguity between events or actions and their outcomes are integrated in the learning structure, and modulate the absolute value of the reinforcers.

3. **Generalization:** Associative theory treats stimuli as compounds of elements, each of which has an associative strength. It follows that two stimuli compounds which share some of these elements thus share some of their associative strength. Generalization follows directly from this analysis.

4. **Large sized problems:** All the factors included in the structure of learning will reduce the processing required. Based on stimulus generalization, new and larger environments will share elements and relationships with smaller ones, facilitating learning and reducing computational complexity.

The problems of exploration-exploitation equilibrium and temporal discounting are closely related. Because the idea of rewards have been abandoned in the proposed learning framework of this project, the temporal reward discounting has a slightly different function than when rewards are considered. The ARL learning framework does not choose responses on the basis of reward maximization, but reward (appetitive) expectance; the greedy response is the one signalling a stimulus (or stimuli compound) with the maximum appetitive reward expectance, for any situation. As shown in the previous chapter, the reward expectance decreases exponentially with the decay parameter δ, which is equivalent to the reward discount parameter γ of Q-learning, the further away from the goal the agent is. Hence, because there are no rewards, the reward expectance is dependent on this exponential decrease in value in order to choose the best response. For a decay parameter of 1 all associations would eventually reach the asymptotic value of the appetitive goal stimulus. Therefore, the actual value of the decay parameter does not matter as long as it is in the range $\langle 0, 1 \rangle$[1]. According to research question 1, it was assumed that by allowing the agent to form S→S-associations (the associative memory), and implementing exploration as a drive, the agent would form a causal model of the environment which would help avoid unsuccessful exploratory policies. The objective of introducing drives has been met, but exploration has not been implemented as a specific drive. The ARL learning framework uses an ϵ-greedy exploration policy like Q-learning. Furthermore, the agent does learn a causal

[1]This preposition can only be claimed to be true for the experiments of this project.

model of the environment, the associative memory, but at the same pace as the behavioral reward expectance model (the expectance memory; S→(R→O)-associations). The S→S-associations therefore evolve in the exact same way as the expectance memory, and hence does not provide any additional information to the agent which could reduce its uncertainty regarding unsuccessful exploratory responses. Finally, because the optimal policy still assumes the provision of its true value, for the ARL learning framework, the problem of exploration-exploitation equilibrium remains. Regarding research question 2, it has been shown that the ARL learning framework relies on an exponential decrease in reward expectance, as do Q-learning, to facilitate second-order conditioning. Therefore, it can be said that the factor included in the learning structure, the decay parameter δ, as a modification to the [Rescorla & Wagner 1972] equation, does modulate the absolute value of the final reinforcer (the goal stimulus) beyond the level of first-order conditioning. This, however, results in an error correction rule which is functionally equivalent to the Q-value error correction rule (under the given experiment conditions). To conclude, the problem of reward discounting also remains.

The contribution of this project is manifested in the way the ARL learning framework facilitates generalization. By considering states as stimuli compounds (i.e. reducible entitities), and modifying the error correction rule to allow for separate associations to form between each stimulus element of a compound and the elements of another compound, a savings effect can be achieved. When the environment is only slightly changed, as in the generalization experiments, from one learning phase to the next, the agent is able to utilize the associations of the remaining stimuli in the changed compounds, so that the policy remains, but loses some of its value (half its value for the modified compounds). Because the policy remains intact, but slightly devalued, the ARL agent can see that there is only a slight change in the environment, as opposed to Q-learning which regards the slight change as complete loss of policy value for the affected states (stimuli compounds). Consequently, the ARL algorithm only has to correct the error of the slightly devalued policy, whereas the Q-learning algorithm has to relearn the policy and its value for the affected states. Conclusively, research question 3 correctly assumes that by considering states as reducible entitities to allow for sharing of associative strength between similar states, generalization can be achieved.

Regarding large sized problems, it was shown that ARL lookahead was able to generalize for Grid world configurations of size 3x3, 5x5, and 10x10. However, the savings effect decreased as the environment complexity increased. For the 10x10 Grid world experiment the slight change in the environment did reduce the optimality of the policy for a longer time than for the generalization experiments involving smaller grid world layouts. Additionally, because none of the experiments have employed redundancy in the environment (i.e. all compounds have been unique), it is not clear whether the repetition of associations across spatial locations would have resulted in an increase in convergence speed. Most likely, this would have violated the Markov assumption, so that the algorithm would have never converged. It has not been tested whether learning can be transferred from a small environment to a larger environment along the lines of the generelization experiments. A complete answer can therefore not be given to research question 4. The only conclusion to be drawn is that the ARL learning framework is able to generalize even as the complexity of the environment increases and more is changed (as in generalization experiment 4).

Finally, it shall be stressed that there is no point in using the ARL learning framework when states are irreducible entities by definition, as in the convergence experiments (i.e. only one stimulus per spatial location). For such cases the added complexity of the ARL framework only results in increased algorithm running time. Because, as have been shown in the previous chapter, Q-learning and ARL lookahead are equivalent for these cases, Q-learning should be the preferred algorithm as it does the exact same thing, but faster due to its simple definition and provision of rewards from the environment.

6.2 Evaluation

The project has been undertaken following a general development approach; requirements gathering, analysis, design, implementation and test (experiments). Requirements were gathered in parallel with the literature review. During this period, regular meetings were held with the project supervisor and an expert on animal learning, Dr. Esther Mondragón, at University College London. The purpose of these meetings was to suggest reading material for the literature review, and also explain concepts that were poorly understood. The first portion of the project work therefore centered around the student reading about theories from animal learning to see how these could be combined with reinforcement learning, and fit with the objectives of the project. It shall be stressed that animal learning theories and reinforcement learning, have different aims as to the usage of their respective models. Whereas the animal learning community provide descriptive and predictive accounts of animal learning and behavior, reinforcement learning aims to develop prescriptive computational model that can solve the reinforcement learning problem (reward maximization). The latter is therefore much more detailed in terms of algorithmic specifications and detailed mathematical notation. Animal learning theories, on the other hand, provide more general descriptions and models that fit statistical data from observations of animal learning and behavior. Not necessarily because of this, but animal learning theories are not very concerned with reward maximization and process control in the way reinforcement learning is. It has therefore been quite problematic to understand how the Q-learning framework could be combined with animal learning theories. The solution, however, was to maintain the notion of reward maximization (actually changed slightly to reward expectance), whereby the Q-learning algorithm could be retained as a functional framework.

A great deal of time has been spent trying to understand the implications of generalization. During the course of this confusion, it was thought that generalization meant transfer of associative strength to similar stimuli. This is why the similarity function was defined. However, as the algorithm design began to take shape, it became evident that transfer of associative strength to similar stimuli is pointless for the purposes of this project. Generalization simply means that two stimuli compounds have common stimulus elements.

The experimental design and testing phase of the project carried with it some confusion. Particularly the generalization experiments, which have been designed in the general case by the animal learning expert, lead to some misunderstandings for the student. It was believed that the different phases of these experiments involved more

changes to the grid world than was actually the case. Therefore, some time was spent and lost on running erronous experiments which did not prove anything either way.

It is worth noting what could have been done differently or extended given more time:

- More specific objectives. This would have allowed more time for experimental design and statistical analysis. However, as the project implied a fairly new angle of research the objectives needed to be quite open, in order not to bias the results.
- Experimental design patterns from Psychology could have been employed. Psychology have a long tradition of defining experiments which produce results prompt to statistical analysis.
- Given more time, the actual experiments should have been run for many more repetitions, to minimize the likelihood of statistical errors.
- More rigorous analysis of results, together with formal mathematical proof of convergence and generalization.

6.3 Future work

Although a sound basis for future work has been proposed and implemented during the course of this project, this thesis has only scratched the surface of what is possible, in terms of possible syntheses between reinforcement learning and associative learning from Psychology. In the future it would be interesting to investigate the following:

- **Dynamic learning rates:** According to associative learning theory the learning rates of the [Rescorla & Wagner 1972] equation are not constant. Section 3.1.3 discusses this briefly, and proposes one possible but very incomplete design. This proposal has not been implemented in the learning framework. It is believed that dynamic learning rates can have a positive effect on the existing solution for generalization. When a policy loses half its value, but is still correct, the algorithm should notice that the policy should regain its value more rapidly than is the case now. The component responsible for this would most likely need some record of policy cost, in order to differentiate a regular policy value error, and one due to generalization.
- **Configural and elemental associations:** In order to solve the XOR problem, the current elemental associative needs to be extended with additional configural units. It is problematic, however, to know when the configural unit applies and when the compound elemental associative strength applies.
- **Sparse associative memory instead of dense lookup-table:** The current implementation uses a dense lookup table to store associative strength. Therefore it requires as much memory as the number of possible associations. An association which does not yet exist simply has a value of 0. This kind of memory is very fast for error correction, but slow for response selection (the algorithm needs to loop through many possible associations to find the one with maximum reward

expectance). A sparse memory, which only stores values for associations that actually exist in the environment, is much more efficient during response selection, which is believed to be the main speed bottleneck.

- **Combining the causal model and the instrumental memory:** An investigation into the effects of combining the associative memory and the expectance memory for generalization is needed. Also, the causal model should be taken advantage of in order to signal unsuccessful exploratory policies. For this to work, however, the causal model (associative memory) would need to gain policy value faster than the instrumental expectance memory.

- **New experiments:** Many new experiments are needed. Specifically the algorithm needs to be tested in environments involving extinction, obstacles, different environment complexity, more than one appetitive stimulus. It is likely that animal learning phenomena can yield new and better insights into solutions to the problems of exploration-exploitation equilibrium, reward discounting, generalization, than those which have been found during this project.

Bibliography

Alonso, E. & Mondragón, E. [2004], Agency, *in* M. Nickles, M. Rovatsos & G. Weiss, eds, 'Agents and Computational Autonomy: Potential, Risks and Solutions', Springer–Verlag, LCNS 2969, pp. 1–6.

Alonso, E. & Mondragón, E. [2005], Associative learning for reinforcement learning: Where animal learning and machine learning meet, *in* 'Proceedings of the Fifth Symposium on Adaptive Agents and Multi-Agent Systems (AAMAS–05)', LIP6, Paris.

Arnauld, S. [1662], *La logique, ou l'art de penser*, Chez Charles Savreux, au pied de la Tour de Nostre Dame, Paris.

Bellman, R. [1957*a*], *Dynamic Programming*, Princeton University Press.

Bellman, R. [1957*b*], 'A markov decision process', *Journal of Mathematical Mechanics* **6**, 679 – 684.

Coolidge, F. L. [2000], *Statistics: A Gentle Introduction*, SAGE Publications.

Dayan, P. & Abbot, L. F. [2001], *Theoretical Neuroscience: Computational and Mathematical Modeling of Neural Systems*, The MIT Press, Cambridge, Massachusetts.

Gabriel, M. & Moore, J. W., eds [1990], *Learning and Computational Neuroscience: Foundations of Adaptive Networks*, MIT Press.

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

Kehoe, E. J., Schreurs, B. G. & Graham, P. [1987], 'Temporal primacy overrides prior training in serial compound conditioning of the rabbit's nictitating membrane response', *Animal Learning and Behavior* **15**, 455–464.

Lubow, R. & Moore, A. [1959], 'Latent inhibition: the effect of nonreinforced preexposure to the conditional stimulus', *J. Comp. Physiol. Psychol.* **54**, 415–419.

Mackintosh, N. [1975], 'A theory of attention: variations in the associability of stimuli with reinforcement', *Psychol. Rev.* **82**, 276–298.

Mackintosh, N. [1983], *Conditioning and associative learning*, Oxford University Press, Oxford.

Mazur, J. E. [1986], *Learning and Behavior*, Prentice Hall, New Jersey.

Mazur, J. E. [1998], *Learning and behavior*, 4 edn, Prentice-Hall.

Negnevitsky, M. [2002], *Artificial Intelligence: A guide to intelligent systems*, 1 edn, Addison-Wesley.

Pavlov, I. [1927], *Conditioned reflexes*, Routledge and Kegan Paul.

Pearce, J. & Hall, G. [1980], 'A model for pavlovian learning: variations in the effectiveness of conditioned but not unconditioned stimuli', *Psychol. Rev.* **87**, 532–552.

Pearce, J. M. [1997], *Animal Learning and Cognition: An Introduction*, Psychology Press, Erlbaum.

Pearce, J. M. & Bouton, M. E. [2001], 'Theories of associative learning in animals', *Annual Review of Psychology* **52**, 111–139.

Rescorla, R. A. [1992], 'Hierarchical associative relations in pavlovian conditioning and instrumental training', *Current Directions in Psychological Science* **1**(2), 66–70.

Rescorla, R. A. & Wagner, A. R. [1972], A theory of pavlovian conditioning: variations in the effectiveness of reinforcement and nonreinforcement, *in* A. H. Black & W. F. Prokasy, eds, 'Classical Conditioning II', Appleton, New York, chapter 3, pp. 64–99.

Russell, S. & Norvig, P. [2003], *Artificial Intelligence: A Modern Approach*, 2nd edn, Prentice-Hall, Englewood Cliffs, NJ.

Saksida, L., Raymond, S. & Touretzky, D. S. [1997], 'Shaping robot behavior using principles from instrumental conditioning', *Robotics and Autonomous Systems* **22**, 231–249.

Samuel, A. L. [1967], 'Some studies in machine learning using the game of checkers. II—recent progress', *IBM Journal of Research and Development* **11**(6), 601–617.

Singh, S. & Bertsekas, D. [1997], Reinforcement learning for dynamic channel allocation in cellular telephone systems, *in* M. C. Mozer, M. I. Jordan & T. Petsche, eds, 'Advances in Neural Information Processing Systems', Vol. 9, The MIT Press, p. 974.

Singh, S. P. [1992], 'Transfer of learning by composing solutions of elemental sequential tasks', *Machine Learning* **8**, 323–339.

Sutton, R. S. [1988], 'Learning to predict by the methods of temporal differences', *Machine Learning* **3**, 9.

Sutton, R. S. & Barto, A. G. [1990], *A Time-Derivative Theory of Pavlovian Conditioning*, MIT Press, pp. 497–537.

Sutton, R. S. & Barto, A. G. [1998], *Reinforcement Learning: An introduction*, The MIT Press, Cambridge, Massachusets.

Tesauro, G. [1995], 'Temporal Difference Learning and TD-Gammon', *Communications of the ACM* **38**(3), 58–68.

Thorndike, E. [1911], *Animal intelligence: Experimental studies*, Macmillan, New York.

Touretzky, D. S., Daw, N. & Tira-Thompson, E. [2002], Combining configural and td learning on a robot, *in* 'Proceedings of the IEEE Second International Conference on Development and Learning', pp. 47 – 52.

Wagner, A. R. [1981], Sop: a model of automatic memory processing in animal behavior, *in* N. Spear & R. Miller, eds, 'Information Processing in Animals: Memory Mechanisms', Erlbaum, Hillsdale, NJ, pp. 5–47.

Watkins, C. J. [1989], Learning from Delayed Rewards, PhD thesis, Psychology Department, Cambridge University, Cambridge, United Kingdom.

Weisstein, E. W. [2005], 'Chess', From Mathworld–A Wolfram Web Resource. Retrieved: August 1, 2005, from http://mathworld.wolfram.com/Chess.html.

Appendix A

Project Definition for MSc in Artificial Intelligence

Name: Niclas Kjäll-Ohlsson
E-mail address: bv339@city.ac.uk
Contact Phone number: +447967932683
Project Title: Associative Reinforcement Learning
Supervisor: Dr. Eduardo Alonso

Dr. Eduardo Alonso	Date
Niclas Kjäll-Ohlsson	Date

A.1 Aim

The aim of this project is to develop an improved Q-learning algorithm by incorporating principles from modern associative models of learning from psychology and cognition. In order to answer to this aim the following main objectives have been identified:

1. To integrate reinforcement learning models from machine learning with psychological theories of Pavlovian and instrumental learning.
2. To design, build and implement a new Q-learning algorithm accordingly.
3. To evaluate the algorithm in different experimental settings against well-established reinforcement learning techniques.

A.2 Background

[Sutton & Barto 1998] define the reinforcement learning problem as consisting of an agent situated in an environment, where the agent performs actions in specific states with the aim of reaching some goal. The environment presents a reward and a new state to the agent after the agent has performed an action. Rewards can be any real number. The goal of reinforcement learning is to maximize the reward signal over time. Several techniques have been used to solve the reinforcement learning problem: Dynamic programming, Monte Carlo methods, and temporal-difference learning [Sutton & Barto 1998]. These techniques differ mainly in two dimensions; whether they work with a model of the dynamics of the environment; and whether or not they bootstrap, that is, compute state-action value estimates based on successor estimates. The project will focus on an instance of temporal difference learning called Q-learning. Despite their popularity, several difficulties have prohibited the application of the techniques to large problems [Alonso & Mondragón 2005]:

1. Finding the right balance between exploration and exploitation. How to search for the optimal policy.
2. Temporal discounting: Modelling long-term goals versus short-term goals through use of reward discounting.
3. Generalisation: Learning is dependent on the reward structure, and so therefore the learning has to start over when presented with a new problem.
4. Large sized problems: Convergence is prohibited by large state spaces, due to the requirement of repetitive visits to all states.

According to [Alonso & Mondragón 2004] reinforcement learning models are based on outdated principles, specifically Thorndikes Law of Effect [Thorndike 1911]. This law states that an association between a stimuli (state) and a response (action) are strengthened when a reinforcing outcome (reward) succeeds. It has been established experimentally that rewards are not essential for learning, and so the reinforcement learning models are incomplete or even wrong. Also it has been called for, in [Kaelbling

et al. 1996], that new bias should be incorporated into the reinforcement learning model.

A.3 Objectives

It is hypothesized in [Alonso & Mondragón 2005] that existing reinforcement learning models and algorithms can be improved upon by incorporating the associative principles currently used to explain trial and error learning in animals. This hypothesis will be made testable, and the aim will possibly be met, by implementing the following sub-objectives [Alonso & Mondragón 2005]:

1. To incorporate psychological bias in the architecture of the agents that will guide their learning. Internal drives will be introduced, that will make the agent approach appetitive stimuli and avoid aversive stimuli.

2. To endow agents with the ability to form types of association other than S-R associations. Associative theory envisages three types of association:

   - Stimulus-Response (S-R) associations
   - Stimulus-Stimulus (S-S) association.
   - Response-Outcome (R-O) associations

3. To take into account associative theorys conception of event (state) representation.

4. To redefine outcomes as comprising sensorial and motivational elements.

5. To take into account the fundamental conditions of association formation proposed by associative theory.

It is contended that achieving these sub-objectives will help solve the problems described in A.2. The research questions are thus specified in light of the identified problems with reinforcement learning techniques:

1. Exploration-exploitation equilibrium: By considering exploration as a drive, S-S associations will form, allowing the agent to form a model of the environment, which in turn will help to discard unsuccessful exploratory policies.

2. Temporal discounting: Factors other than the immediate temporal contiguity between events or actions and their outcomes are integrated in the learning structure, and modulate the absolute value of the reinforcers.

3. Generalization: Associative theory treats stimuli as compounds of elements, each of which has an associative strength. It follows that two stimuli which share some of these elements thus share some of their associative strength. Generalization follows directly from this analysis.

4. Large sized problems: All the factors included in the structure of learning will reduce the processing required. Based on stimulus generalization, new and larger environments will share elements and relationships with smaller ones, facilitating learning and reducing computational complexity.

A.4 Tools

Grid-world will be used as the testing environment for the algorithm. The Grid-world consists of a grid of squares through which a robot may navigate its way to a goal state while avoiding obstacles on its way. Grid-world presents many advantages when designing and testing reinforcement learning techniques:

1. It is standard. The aim of the project is not to develop a new environment.
2. It has been extensively used for the evaluation of various reinforcement learning techniques, thereby facilitating comparison with existing algorithms.
3. It is simple, and can be changed in many ways to represent different experimental conditions.
4. It can be easily extended so as to add the new elements and types of association described in the previous section.

A.5 Method

A.5.1 Evaluation of the algorithm

The resulting algorithm will be tested and evaluated against different well-established reinforcement learning techniques in scenarios of various degrees of complexity. Measuring learning performance will be carried out according to the following parameters:

1. Eventual convergence to optimality (that is, provable guarantee of asymptotic convergence to optimal behavior).
2. Speed of convergence to optimality and level of performance after a given time.
3. Regret, the difference between the expected total reward gained by following a learning algorithm and the expected total reward one could gain by playing for the maximum expected reward from the start.
4. Generalization. Assessment will be carried out on how experience with a limited subset of the state space can be usefully generalized to produce a good approximation over a much larger subset.

A.5.2 Methodology

The development of the learning model and algorithm will be managed by the Unified Software Development Process, a methodology that emphasizes a recursive development process in which, after every stage, a new recursion of the system life cycle is performed. This will, in turn, be used to refine assumptions and reduce the risk that the analysis and design stages have been carried out incorrectly. The project will in-

Figure A.1: The Unified Software Development Process

volve collaboration with psychologists from the University College London and the University of York. This interaction will persist throughout each stage of the project. A reinforcement learning model based on animal-learning theories will be built in consultation with psychologists; following such a model, an algorithm will be produced along with an experimental scenario to test the model. The resulting algorithm will be thoroughly tested in several settings using the Grid-world. The parameters to be used in each case will make explicit reference to existing reinforcement learning algorithms so that comparisons are relevant and the results prompt to experimental analysis.

A.6 Project beneficiaries

The main beneficiaries will be systems engineers and computer scientists with interests in developing adaptive software. In particular, it will benefit Artificial intelligence and machine learning researchers. Due to the interdisciplinary nature of this project, the results will also benefit neuroscientists and psychologists. First, it will provide a tool for behavioral researchers to apply to associative and instrumental models to solve computational problems. Second, improved models of reinforcement learning may, in turn, be used to solve optimization problems.

A.7 Project feasibility

A.7.1 Student ability

- The student has documented knowledge on AI techniques, both from undergraduate and postgraduate studies. He has previously implemented several reinforcement learning algorithms, including Q-learning, Sarsa, Dyna-Q, Q-learning with Function Approximation (FA), in an environment with a large state space (Minesweeper), and is therefore aware of the limitations and problems with reinforcement learning techniques.
- Knowledge of associative learning models from psychology is not required. The project will be undertaken in collaboration (although limited) with psychologists

from University College London and the University of York, and hence expertise will be provided where necessary.

A.7.2 Project risks

- Lack of commitment from collaborators: A previous project have been undertaken successfully by a student in the department, in collaboration with the psychologists, and so there is evidence of past enthusiasm and willingness to provide expertise. Also it is in the interest of the psychologists to see practical implementations of their theories.
- Not understanding theory: As stated earlier the development of the model will progress iteratively. If parts of the theory are not well understood, it will be necessary to revisit and revise earlier stages of the process so as to make sure the requirements and analysis have been specified unambiguously.
- Flawed experimental design: Testing and analysis of results compose an important aspect of the project. Given that the objectives have been met, it is paramount to perform extensive testing of the algorithm so that the research questions can be answered and objectives confirmed. As described below a considerable amount of time will be spent on design of testing scenarios and procedures.

A.8 Work plan

This project has several deliverables, the most important one being the Project Report document. The main deliverables resulting from the project work are related to the new reinforcement learning model and algorithm:

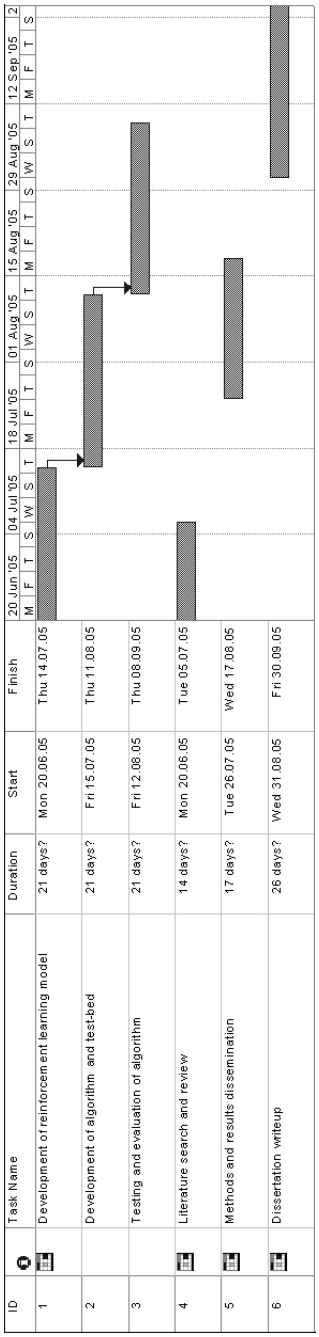
Work-package I (Weeks 1-3). To underpin the research with a sound theoretical framework. This will take into account both associative and instrumental learning and reinforcement learning.
Deliverable I.1. Reinforcement learning model.

Work-package II (Weeks 3-7). Development of the algorithm and the basic infrastructure components of a test-bed. This test-bed will be used to evaluate the model from Work-package I.
Deliverable II.1. Reinforcement learning algorithm.
Deliverable II.2. Demonstration scenarios and evaluation procedures.

Work-package III (Weeks 7-10). Testing and evaluation of the algorithm will then proceed following the procedures devised during work-package II.
Deliverable III.1. Results from testing.
Deliverable III.2. Evaluation of results.

Project Report Document (PRD) (Weeks 1-12). The project report shall introduce the reader to the research, explain the current state of the topic, report the work being done, present results and evaluations of the work. It will be written during the course of the project work.
Deliverable PRD.1 The literature survey will be undertaken during work package I.
Deliverable PRD.2 Methods and results will be documented during work packages II and III.
Deliverable PRD.3 Discussion, evaluation and conclusion; to be written at end of project after actual project work is finalized.

Figure A.2: Work schedule



Appendix B

Source code

Listing B.1: Java code listing for ARL Agent

```
/*
 * An implementation of the ARL Trace and ARL lookahead algorithms
 * Niclas Kjall-Ohlsson, copyright 2005
 */
public class ARLAgent implements Agent {

    private int id;

    private double StdDev = 0.1; // similarity function sensitivity

    private double max_alpha = 0.5; // CS specific learning rate
    private double max_beta = 0.5; // US specific learning rate

    private double epsilon = 0.8; // exploration parameter

    private double similarity_threshold = 0.85; // ARL trace specific parameter

    private double decay = 0.9; // Decay of first-order asymptote to facilitate second-order conditioning

    public java.util.Vector unconditioned_stimuli; // internal drives

    private Simulator env; // the grid world environment handle

    private java.util.Vector stimuli; // currently perceived stimuli compound

    private int previous_response; // previous response: ARL trace specific
    private java.util.Vector previous_stimuli; // previous stimuli compound: ARL trace specific

    // Dense Associative memory
    // Stimulus-Stimulus association
    // Indices: [CS.modality][US.modality][CS.strength][US.strength]
    private double associative_memory[][][][];

    // Response stimulus values. NOT IN USE!!!!!
    private double expectance_memory[][][]; // obsolete

    // Dense Expectance memory
    // Stimulus-(Response-Outcome)-associations
    // Indices: [CS.modality][CS.strength][Response][US.modality][US.strength]
    private double hierarchical_memory[][][][][];

    public double avg_error = 0; // aggregate absolute error of associative memory per episode
    public double avg_error_expectance = 0; // aggregate absolute error of expectance memory per episode
    public long episode_step = 0; //

    public boolean learning = true; // update memories or not

    // if true: then the associative memory and the expectance memory are
    // combined during response selection
    public boolean combineMemories = false;

    // 0: ARL lookahead
    // 1: ARL trace
    public int version = 0;

    // epoch counter
    private long epochs = -1;
```

```
    private double max_outcome_value; // for visualization purposes

    /** Creates a new instance of ARLAgent */
    public ARLAgent(Simulator env, int id) {
        this.unconditioned_stimuli = new java.util.Vector();
        this.id = id;
        this.env = env;
        initAgentMemory();
    }

    public void resetInternalDrives() {
        this.unconditioned_stimuli = new java.util.Vector();
    }

    public void initAgentMemory() {
        // reset associative memory
        associative_memory = new double[env.getNumModalities()][env.getNumModalities()]
                                        [env.getNumStrengths()][env.getNumStrengths()];
        expectance_memory = new double[env.getNumActions()][env.getNumModalities()]
                                      [env.getNumStrengths()]; // obsolete
        //reset expectance memory
        hierarchical_memory = new double[env.getNumModalities()][env.getNumStrengths()][env.getNumActions()]
                                        [env.getNumModalities()][env.getNumStrengths()];
        epochs = -1;
    }

    public int getID() {
        return id;
    }

    public void setStdDev(double var) {
        this.StdDev = var;
    }

    public double getStdDev() {
        return this.StdDev;
    }

    public double getMaxAlpha() {
        return max_alpha;
    }

    public double getMaxBeta() {
        return max_beta;
    }

    public double getEpsilon() {
        return epsilon;
    }

    public double getSimilarityThreshold() {
        return this.similarity_threshold;
    }

    public double getDecay() {
        return decay;
    }

    public void setMaxAlpha(double alpha) {
        this.max_alpha = alpha;
    }

    public void setMaxBeta(double beta) {
        this.max_beta = beta;
    }

    public void setEpsilon(double epsilon) {
        this.epsilon = epsilon;
    }

    public void setSimilarityThreshold(double simt) {
        this.similarity_threshold = simt;
    }

    public void setDecay(double decay) {
        this.decay = decay;
    }

    public void setLearning(boolean learning) {
        this.learning = learning;
    }

    public boolean getCombineMemories() {
        return this.combineMemories;
    }

    public void setCombineMemories(boolean cm) {
        this.combineMemories = cm;
    }

    public int getVersion() {
```

```
    return version;
}

public void setVersion(int version) {
    this.version = version;
}

// initialize agent episode
public void initEpisode() {
    stimuli = env.getStimuliCompound(id);
    previous_response = -1;
    previous_stimuli = null;
    episode_step = 0;
    avg_error = 0;
    avg_error_expectance = 0;
    epochs++;
}


/*
 * This method is called by the environment
 * for each step of the episode. This is the functional part of the
 * ARL agent.
 */
public void doStep() {
    if(version == 0) doStep_lookahead();
    else if(version == 1) doStep_trace();
}

/*
 * Perform one episode step and update associative memory and
 * expectance memory according to the ARL lookahead algorithm
 */
public void doStep_lookahead() {
    int response;
    java.util.Vector nextStimuli;

    stimuli = env.getStimuliCompound(id); // perceive current stimuli compound
    response = chooseResponse(stimuli, true); // select response according to epsilon-greedy exploration policy
    // perform response in environment
    env.doAction(response, id);
    // observe next stimuli compound
    nextStimuli = env.getStimuliCompound(id);

    int num_errors = 0;
    int num_errors_expectance = 0;

    double assoc_strength = 0, error = 0, sim_us = 0;
    double old_value = 0;
    double alpha = 0.0, beta = 0.0;
    Stimulus st1, st2;
    double old_val;
    double second_order_assoc, second_order_expectance;

    // retrieve maximum associative strength for next stimuli compound
    second_order_assoc = this.getSecondaryAssocStrengthComp(nextStimuli);
    // refine associative model (S-S) for first-order conditioning
    for(int i=0;i<nextStimuli.size();i++) {
        assoc_strength = 0;
        st2 = (Stimulus)nextStimuli.get(i);
        beta = max_beta;
        // accumulate total associative strength of compound
        for(int j=0;j<stimuli.size();j++) {
            st1 = (Stimulus)stimuli.get(j);
            assoc_strength += associative_memory[st1.getModality()][st2.getModality()]
                                                [st1.getStrength()][st2.getStrength()];
        }

        sim_us = compStimUS(st2, false);
        // lambda = similarity to US + secondary associative strength of preceding stimulus
        error = (sim_us+decay*second_order_assoc - assoc_strength);

        // update the associative strength of each stimulus according to the Rescorla-Wagner rule
        for(int j=0;j<stimuli.size();j++) {
            st1 = (Stimulus)stimuli.get(j);
            alpha = max_alpha;
            old_val = associative_memory[st1.getModality()][st2.getModality()]
                                        [st1.getStrength()][st2.getStrength()];
            associative_memory[st1.getModality()][st2.getModality()]
                              [st1.getStrength()][st2.getStrength()]
                    += alpha*beta*error*(stimuli.equals(nextStimuli) ? 0 : 1);
            avg_error += java.lang.Math.abs(old_val-associative_memory[st1.getModality()][st2.getModality()]
                                                                      [st1.getStrength()][st2.getStrength()]);
            num_errors++;
        }
    }


    // second-order reward expectance for expectance memory
    second_order_expectance = this.getSecondaryExpectanceComp(nextStimuli);
```

```
        // update the expectance memory
        for(int i=0;i<nextStimuli.size();i++) {
            st2 = (Stimulus)nextStimuli.get(i);
            // first-order and second-order conditioning in one
            sim_us = compStimUS(st2, true)+decay*second_order_expectance;

            alpha=max_alpha;
            beta=max_beta;
            assoc_strength = 0;

            // accumulate reward expectance for compound
            for(int h=0;h<stimuli.size();h++) {
                st1 = (Stimulus)stimuli.get(h);
                assoc_strength += hierarchical_memory[st1.getModality()][st1.getStrength()][response]
                                                     [st2.getModality()][st2.getStrength()];
            }

            // update expectance memory
            for(int h=0;h<stimuli.size();h++) {
                st1 = (Stimulus)stimuli.get(h);
                error = sim_us-assoc_strength; // Rescorla-Wagner error

                num_errors_expectance++;
                old_val = hierarchical_memory[st1.getModality()][st1.getStrength()][response]
                                             [st2.getModality()][st2.getStrength()];
                hierarchical_memory[st1.getModality()][st1.getStrength()][response]
                                   [st2.getModality()][st2.getStrength()]
                        += alpha*beta*error*(stimuli.equals(nextStimuli) ? 0 : 1); // Modified Rescorla-Wagner
                                                                                   // error correction rule
                avg_error_expectance += java.lang.Math.abs(old_val - hierarchical_memory[st1.getModality()]
                                                                                        [st1.getStrength()][response]
                                                                                        [st2.getModality()][st2.getStrength()]);
            }
        }

        if(num_errors > 1) avg_error /= num_errors;
        if(num_errors_expectance > 1) avg_error_expectance /= num_errors_expectance;
        episode_step++;
    }

    /*
     * Perform one episode step and update associative memory and
     * expectance memory according to the ARL trace algorithm
     */
    public void doStep_trace() {
        int response;
        java.util.Vector nextStimuli;

        stimuli = env.getStimuliCompound(id); // perceive current stimuli compound
        response = chooseResponse(stimuli, true); // choose response according to epsilon-greedy policy
        env.doAction(response, id); // perform response in the environment
        nextStimuli = env.getStimuliCompound(id); // observe next stimuli compound

        // number of updates in associations
        // for associative memory and expectance memory
        int num_errors = 0, num_errors_expectance = 0;


        double assoc_strength = 0, secondary_assoc_strength = 0, error = 0, sim_us = 0;
        double max_assoc_strength = Double.MIN_VALUE, old_value = 0;
        double alpha = 0.0, beta = 0.0;
        double old_val = 0;
        Stimulus st1, st2; // CS, US

        // refine associative model (S-S) for first-order conditioning
        // Each stimulus of the current compound enters into an association with
        // each stimulus of the next compound
        for(int i=0;i<nextStimuli.size();i++) {
            st2 = (Stimulus)nextStimuli.get(i);
            beta = max_beta;

            // accumulate total associative strength of compound
            for(int j=0;j<stimuli.size();j++) {
                st1 = (Stimulus)stimuli.get(i);
                assoc_strength += associative_memory[st1.getModality()][st2.getModality()]
                                                    [st1.getStrength()][st2.getStrength()];
            }

            if(assoc_strength > max_assoc_strength) max_assoc_strength = assoc_strength;

            sim_us = compStimUS(st2, false); // retrieve first-order asymptote
            if(sim_us > similarity_threshold) { // only update if simalarity to any internal drive
                                                // is greater than similarity threshold
                error = (sim_us - assoc_strength); // the discrepancy between the
                                                   // compound associative strength and the asymptote
                // update the associative strength of each stimulus according to the Rescorla-Wagner rule
                for(int j=0;j<stimuli.size();j++) {
                    st1 = (Stimulus)stimuli.get(i);
                    alpha = max_alpha;
                    old_val = associative_memory[st1.getModality()][st2.getModality()]
                                                [st1.getStrength()][st2.getStrength()];
```

```
                associative_memory[st1.getModality()][st2.getModality()]
                                [st1.getStrength()][st2.getStrength()]
                        += alpha*beta*error*weightSimilar(st1,st2); // Rescorla-Wagner error update
                                                                    // for first-order conditioning
                // accumulate error
                avg_error += java.lang.Math.abs(old_val-associative_memory[st1.getModality()]
                            [st2.getModality()][st1.getStrength()][st2.getStrength()]);
                num_errors++; // number of associations updated
                // for purposes of second-order conditioning
                if(associative_memory[st1.getModality()][st2.getModality()][st1.getStrength()]
                                                        [st2.getStrength()] > max_assoc_strength)
                    max_assoc_strength = associative_memory[st1.getModality()][st2.getModality()]
                                                           [st1.getStrength()][st2.getStrength()];
            }
        }
    }

    if(max_assoc_strength == Double.MIN_VALUE) max_assoc_strength = 0; // no change.
                                                                       // no second-order conditioning

    assoc_strength = 0;
    if(previous_stimuli != null) {
        // refine associative model (S-S) for second-order conditioning
        for(int i=0;i<stimuli.size();i++) {
            st2 = (Stimulus)stimuli.get(i);
            beta = max_beta;
            // accumulate total associative strength of compound
            // and modality strength
            for(int j=0;j<previous_stimuli.size();j++) {
                st1 = (Stimulus)previous_stimuli.get(j);
                assoc_strength += (Double.isNaN(associative_memory[st1.getModality()][st2.getModality()]
                                                [st1.getStrength()][st2.getStrength()]) ? 0 :
                                                associative_memory[st1.getModality()][st2.getModality()]
                                                [st1.getStrength()][st2.getStrength()]);
            }

            // the discrepancy of the total associative strength of the compound
            // and the maximum associative strength of the successive compound
            // from first-order conditioning
            error = (max_assoc_strength - assoc_strength);
            for(int j=0;j<previous_stimuli.size();j++) {
                st1 = (Stimulus)previous_stimuli.get(j);
                alpha = max_alpha;
                old_val = associative_memory[st1.getModality()][st2.getModality()]
                                            [st1.getStrength()][st2.getStrength()];
                // Rescorla-Wagner error correction for second-order conditioning. Includes decay parameter
                associative_memory[st1.getModality()][st2.getModality()]
                                [st1.getStrength()][st2.getStrength()]
                        += alpha*beta*decay*error*weightSimilar(st1,st2)
                           *(stimuli.equals(previous_stimuli) ? 0 : 1);
                // accumulate error for episode
                avg_error += java.lang.Math.abs(old_val-associative_memory[st1.getModality()]
                                            [st2.getModality()][st1.getStrength()][st2.getStrength()]);
                // number of associations updated
                num_errors++;
            }
        }
    }

    double  h_aggr_expectance = 0, h_error = 0, max_h_aggr_expectance = Double.MIN_VALUE;

    // update policy for first-order conditioning
    // for the expectance memory
    for(int i=0;i<nextStimuli.size();i++) {
        st2 = (Stimulus)nextStimuli.get(i);
        sim_us = compStimUS(st2, true); // retrieve first-order asymptote.
                                        // take into account whether aversive/appetitive
        beta = max_beta;
        // accumulate the aggregate expectance for the current response and stimuli compund
        // towards each stimulus of the next stimuli compound
        for(int h=0;h<stimuli.size();h++) {
            st1 = (Stimulus)stimuli.get(h);
            h_aggr_expectance += hierarchical_memory[st1.getModality()][st1.getStrength()]
                        [response][st2.getModality()][st2.getStrength()];
        }
        if(sim_us > similarity_threshold) { // same as above
            for(int h=0;h<stimuli.size();h++) {
                st1 = (Stimulus)stimuli.get(h);
                h_error = sim_us-h_aggr_expectance; // First-order error
                alpha = max_alpha;
                old_val = hierarchical_memory[st1.getModality()][st1.getStrength()]
                            [response][st2.getModality()][st2.getStrength()];
                hierarchical_memory[st1.getModality()][st1.getStrength()]
                            [response][st2.getModality()][st2.getStrength()]
                        += alpha*beta*h_error*weightSimilar(st1,st2); // Rescorla-Wagner for
                                                                      // first-order error correction
                num_errors_expectance++;
                avg_error_expectance
                        += java.lang.Math.abs(old_val - hierarchical_memory[st1.getModality()]
                        [st1.getStrength()][response][st2.getModality()][st2.getStrength()]);
            }
```

```
            }
            if(h_aggr_expectance > max_h_aggr_expectance) max_h_aggr_expectance = h_aggr_expectance;
            h_aggr_expectance = 0;
        }

        // update policy for second-order conditioning
        // for the expectance memory
        if(previous_stimuli != null) { // only update if there was a previous compound in this episode
            if(max_h_aggr_expectance == Double.MIN_VALUE) max_h_aggr_expectance = 0; // no change
            for(int i=0;i<stimuli.size();i++) {
                st2 = (Stimulus)stimuli.get(i);
                h_aggr_expectance = 0;
                beta = max_beta;

                // accumulate reward expectance
                // for the previous compound and previous
                // towards each stimulus of the current compund|
                for(int h=0;h<previous_stimuli.size();h++) {
                    st1 = (Stimulus)previous_stimuli.get(h);
                    h_aggr_expectance += hierarchical_memory[st1.getModality()][st1.getStrength()]
                            [previous_response][st2.getModality()][st2.getStrength()];
                }

                //
                for(int h=0;h<previous_stimuli.size();h++) {
                    st1 = (Stimulus)previous_stimuli.get(h);
                    h_error = max_h_aggr_expectance-h_aggr_expectance;
                    alpha = max_alpha;
                    old_val = hierarchical_memory[st1.getModality()][st1.getStrength()]
                            [previous_response][st2.getModality()][st2.getStrength()];
                    hierarchical_memory[st1.getModality()][st1.getStrength()]
                            [previous_response][st2.getModality()][st2.getStrength()]
                            += alpha*beta*h_error*weightSimilar(st1,st2)
                            *(stimuli.equals(previous_stimuli) ? 0 : 1);  // Rescorla-Wagner error. Cancel
                                                                          // if current compund equals
                                                                          // previous compound
                    num_errors_expectance++;
                    avg_error_expectance
                            += java.lang.Math.abs(old_val - hierarchical_memory[st1.getModality()]
                            [st1.getStrength()][previous_response][st2.getModality()][st2.getStrength()]);
                }
            }
        }

        previous_response = response;
        previous_stimuli = stimuli;
        stimuli = nextStimuli;
        if(num_errors > 1) avg_error /= num_errors;
        if(num_errors_expectance > 1) avg_error_expectance /= num_errors_expectance;
        episode_step++;
    }

    private double weightSimilar(Stimulus s1, Stimulus s2) {
        return (s1.getModality() == s2.getModality() && s1.getStrength() == s2.getStrength()
                ? 0.0 : 1);
    }

    public double getAvgErrorEpisode() {
        return (double)(avg_error/episode_step);
    }

    public double getAvgErrorExpectanceEpisode() {
        return (double)(avg_error_expectance/episode_step);
    }

    public long getEpochs() {
        return epochs;
    }

    public int getBestResponse(java.util.Vector compound) {
        return chooseResponse(compound, false);
    }

    /*
     * Choose response according to associative value and expectance value combined
     * or according to expectance memory only.
     * Finally, selects responses according to a epsilon-greedy policy
     */
    private int chooseResponse(java.util.Vector compound, boolean random) {
        double max_outcome_val = Double.MIN_VALUE,
               outcome_val = Double.MIN_VALUE;
        int response = 0;
        double r_o[][][] = new double[env.getNumActions()]
                [env.getNumModalities()][env.getNumStrengths()];

        for(int i=0;i<compound.size();i++) {
            Stimulus st1 = (Stimulus)compound.get(i);
            for(int m2=0;m2<env.getNumModalities();m2++) {
                for(int s2=0;s2<env.getNumStrengths();s2++) {
                    for(int r=0;r<env.getNumActions();r++) {
                        r_o[r][m2][s2] += hierarchical_memory[st1.getModality()]
```

```
                                        [st1.getStrength()][r][m2][s2]*
                                        (combineMemories ? associative_memory[st1.getModality()]
                                        [m2][st1.getStrength()][s2] : 1);
                        if(r_o[r][m2][s2] > max_outcome_val) {
                            max_outcome_val = r_o[r][m2][s2];
                            response = r;
                        }
                    }
                }
            }
        }

        max_outcome_value = (max_outcome_val == Double.MIN_VALUE ? 0 : max_outcome_val);

        java.util.Random r = new java.util.Random();

        if(random) {
            if(r.nextDouble() > (1.0 - epsilon)) { return (int)(r.nextDouble()*env.getNumActions()); }
            else { return response; }
        } else return response;
    }

    // for vizualization
    public double getMaxOutcomeValue() {
        return max_outcome_value;
    }

    /*
     * ARL Lookahead specific:
     * Retrieves the maximum associative strength of the given compound
     * towards any associated outcome
     */
    private double getSecondaryAssocStrengthComp(java.util.Vector compound) {
        double max_assoc = Double.MIN_VALUE;
        double aggr = 0;

        for(int m2=0;m2<env.getNumModalities();m2++) {
            for(int s2=0;s2<env.getNumStrengths();s2++) {
                aggr = 0;
                for(int i=0;i<compound.size();i++) {
                    Stimulus st1 = (Stimulus)compound.get(i);
                    aggr += associative_memory[st1.getModality()][m2][st1.getStrength()][s2];
                }
                if(aggr > max_assoc) max_assoc = aggr;
            }
        }

        return (max_assoc == Double.MIN_VALUE ? 0 : max_assoc);
    }

    /*
     * ARL Lookahead specific:
     * Retrieves the maximum expectance value of the given compound
     * towards any associated outcome for any response
     */
    private double getSecondaryExpectanceComp(java.util.Vector compound) {
        double max_outcome_val = Double.MIN_VALUE;
        double aggr = 0;

        for(int m2=0;m2<env.getNumModalities();m2++) {
            for(int s2=0;s2<env.getNumStrengths();s2++) {
                for(int r=0;r<env.getNumActions();r++) {
                    aggr = 0;
                    for(int i=0;i<compound.size();i++) {
                        Stimulus st1 = (Stimulus)compound.get(i);
                        aggr += hierarchical_memory[st1.getModality()][st1.getStrength()][r][m2][s2];
                    }
                    if(aggr > max_outcome_val) max_outcome_val = aggr;
                }
            }
        }

        return (max_outcome_val == Double.MIN_VALUE ? 0 : max_outcome_val);
    }

    /*
     * Compares the given stimulus to the list of internal drives
     * using the similarity function (gaussianSim) and returns the maximum similarity
     * If parameter expectance is true, then takes into account sign (aversive/appetitive)
     */
    private double compStimUS(Stimulus st, boolean expectance) {
        double max_sim = 0;
        int index = 0;
        for(int i=0;i<unconditioned_stimuli.size();i++) {
            Stimulus us = (Stimulus)((Object[])unconditioned_stimuli.get(i))[0];
            if(us.getModality() == st.getModality()) {
                double gs = gaussianSim(st.getStrength(), us.getStrength());
                if(gs > max_sim) {
                    max_sim = gs;
                    index = i;
                }
```

```
            }
        }

        return max_sim*(expectance ? ((Integer)((Object[])unconditioned_stimuli.get(index))[1]).intValue() : 1);
    }
    /*
     * The similiraty function as described in the report
     */
    private double gaussianSim(double value, double mean) {
        double ex = (1/(java.lang.Math.sqrt(2*java.lang.Math.PI)*StdDev)),
               nom = -(java.lang.Math.pow(value - mean,2.0)/(2*java.lang.Math.pow(StdDev,2.0))),
               den = -(java.lang.Math.pow(0,2.0)/(2*java.lang.Math.pow(StdDev,2.0)));
        return (ex*java.lang.Math.exp(nom))/(ex*java.lang.Math.exp(den));
    }
}
```

Listing B.2: Java code listing for Q-learner agent

```
/*
 * Implementation of the Q-learning algorithm
 * Niclas Kjall-Ohlsson, copyright 2005
 */
public class RLAgent implements Agent {

    private int id; // Agent id number

    // The grid world environment
    private Simulator env;

    // Q-table for state-action values
    private java.util.HashMap Q_table;

    // current state
    private String state;

    private double epsilon = 0.91; // exploratory control parameter
    private double alpha = 0.01; // learning rate
    private double gamma = 0.9; // discount factor

    // the current best state-action value, given the current state
    private double max_q = 0.0;

    private boolean learning = true; // update q-values or not

    public double qfluct_aggr; // aggregate absolute error per episode

    public long episode_step = 0; // number of steps at end of episode

    private long epochs = -1; // epoch counter

    private double best_action_value; // current value of the best response

    /** Creates a new instance of RLAgent */
    public RLAgent(Simulator env, int id) {
        Q_table = new java.util.HashMap(env.getSizeX()*env.getSizeY());
        this.id = id;
        this.env = env;
        this.initAgentMemory();
    }

    public double getQFluctEpisode() {
        return (double)(qfluct_aggr/episode_step);
    }

    public long getEpochs() {
        return epochs;
    }

    public double getBestActionValue() {
        return best_action_value;
    }

    /*
     * The Gridworld calls this method to retrieve the best q-value for a given state
     */
    public int getBestAction(String state) {
        int bestAction = 0;
        if(Q_table.containsKey(state)) {
            double q_values[] = (double[])Q_table.get(state),
                   bestActionValue = Double.MIN_VALUE;
            for(int i=0;i<q_values.length;i++) {
                if(q_values[i] > bestActionValue) {
                    bestAction = i;
                    bestActionValue = q_values[i];
                    best_action_value = bestActionValue;
                }
            }
        } else {
```

```
            best_action_value = 0;
        }
        return bestAction;
    }

    public int getID() {
        return id;
    }

    public double getAlpha() {
        return alpha;
    }

    public double getGamma() {
        return gamma;
    }

    public double getEpsilon() {
        return epsilon;
    }

    public void setAlpha(double alpha) {
        this.alpha = alpha;
    }

    public void setGamma(double gamma) {
        this.gamma = gamma;
    }

    public void setEpsilon(double epsilon) {
        this.epsilon = epsilon;
    }

    public void setLearning(boolean learning) {
        this.learning = learning;
    }

    // initialize the associative memory
    public void initAgentMemory() {
        Q_table = new java.util.HashMap(env.getSizeX()*env.getSizeY());
        qfluct_aggr = 0.0;
        epochs = -1;
    }

    // initialize episode
    public void initEpisode() {
        qfluct_aggr = 0.0;
        episode_step = 0;
        state = env.getLocationRepresentation(id);
        epochs++;
    }

    /*
     * This method is called by the environment
     * for each step of the episode. This is the functional part of the
     * Q-learning algorithm.
     *
     */
    public void doStep() {
        int action = this.chooseAction(state); // choose action according to epsilon-greedy exploration policy

        double reward = env.doAction(action,id); // perform the action and receive reward

        if(learning) {
            String next_state = env.getLocationRepresentation(id); // perceive the next state
            int next_action = this.chooseAction(next_state); // set the global MAX_Q value of the next state

            double oldval = lookupQValue(state, action); // get the q-value of the current state-action pair

            setQValue(state, action, (oldval + alpha*(reward + gamma*max_q - oldval))); // update the q-value table

            double newval = lookupQValue(state, action);

            qfluct_aggr += java.lang.Math.abs(newval-oldval); // accumulate absolute error

            state = next_state;
        }

        episode_step++;
    }

    private void setQValue(String state, int action, double value) {
        double q_values[] = (double[])Q_table.get(state);
        q_values[action] = value;
        Q_table.put(state, q_values);
    }

    private double lookupQValue(String state, int action) {
        double q_values[] = (double[])Q_table.get(state);
        return q_values[action];
    }
```

```
    public void printQValues(String state) {
        System.out.println(state + " " + Q_table.size() + ": ");
        if(Q_table.containsKey(state)) {
            double q_values[] = (double[])Q_table.get(state);
            for(int i=0;i<q_values.length;i++) {
                System.out.print(q_values[i] + " ");
            }
            System.out.println();
        }
    }

    /*
     * choose response according to epsilon-greedy exploration policy
     */
    private int chooseAction(String state) {
        int bestAction = 0;
        max_q = 0;
        if(!Q_table.containsKey(state)) {
            double q_values[] = new double[env.getNumActions()];
            for(int i=0;i<q_values.length;i++) {
                q_values[i] = 0;//java.lang.Math.random();
            }
            Q_table.put(state, q_values);
        }
        double q_values[] = (double[])Q_table.get(state);

        java.util.Random r = new java.util.Random();

        for(int i=0;i<q_values.length;i++) {
            if(q_values[i] > max_q) {
                max_q = q_values[i];
                bestAction = i;
            }
        }

        if(learning) {
            if(r.nextDouble() > (1.0 - epsilon)) { return (int)(r.nextDouble()*env.getNumActions()); }
            else { /*System.out.println("Best action "+bestAction);*/ return bestAction; }
        } else {
            //System.out.println("Best action "+bestAction);
            return bestAction;
        }
    }

}
```

Table B.1: CD-ROM source code index for GUI components (replace d: with the drive letter of your CD-ROM)

File	Description	file path
Stimulus.java	Wrapper class for stimulus properties	`d:\ARL\code\ARL\src\Stimulus.java`
Simulator.java	Encapsulates all GUI components of the learning simulator	`d:\ARL\code\ARL\src\Simulator.java`
Chart.java	Provides live plotting	`d:\ARL\code\ARL\src\Chart.java`
Agent.java	Generic agent interface for interaction between the simulator and agents	`d:\ARL\code\ARL\src\Agent.java`

Appendix C

Result data and experiment definitions

The tables below provide an index of files on the CD-ROM for:

- the Grid world configuration files for the different experiments described in the report
- the results data files for the different experiments described in the report
- the Excel sheets with prepared results data files

Replace ”d:” with the letter of your CD-ROM drive.

Table C.1: Grid world configuration and algorithm parameter files (Experiment definitions). Load these into the learning simulator to reproduce the experiments

Experiment	Group	Phase	file path
Convergence 3x3	n/a	n/a	d:\ARL\exp\convergence\small.lrn
Convergence 5x5	n/a	n/a	d:\ARL\exp\convergence\medium.lrn
Convergence 10x10	n/a	n/a	d:\ARL\exp\convergence\large.lrn
Generalization 1	1	1	d:\ARL\exp\genex\genex1_group1_phase1.lrn
Generalization 1	1	2	d:\ARL\exp\genex\genex1_group1_phase2.lrn
Generalization 1	2	1	d:\ARL\exp\genex\genex1_group2_phase1.lrn
Generalization 1	2	2	d:\ARL\exp\genex\genex1_group2_phase2.lrn
Generalization 2	1	1	d:\ARL\exp\genex\genex2_group1_phase1.lrn
Generalization 2	1	2	d:\ARL\exp\genex\genex2_group1_phase2.lrn
Generalization 2	2	1	d:\ARL\exp\genex\genex2_group2_phase1.lrn
Generalization 2	2	2	d:\ARL\exp\genex\genex2_group2_phase2.lrn
Generalization 3	1	1	d:\ARL\exp\genex\genex3_group1_phase1.lrn
Generalization 3	1	2	d:\ARL\exp\genex\genex3_group1_phase2.lrn
Generalization 3	2	1	d:\ARL\exp\genex\genex3_group2_phase1.lrn
Generalization 3	2	2	d:\ARL\exp\genex\genex3_group2_phase2.lrn
Generalization 4	1	1	d:\ARL\exp\genex\genex4_group1_phase1.lrn
Generalization 4	1	2	d:\ARL\exp\genex\genex4_group1_phase2.lrn
Generalization 4	2	1	d:\ARL\exp\genex\genex4_group2_phase1.lrn
Generalization 4	2	2	d:\ARL\exp\genex\genex4_group2_phase2.lrn
Aversive avoidance	n/a	n/a	d:\ARL\exp\aversive_avoidance\large_obs.lrn

Table C.2: Raw results data files index 1.

Experiment	Algorithm	Group	Phase	file path (n*: replace with number from 1 to 8)
Convergence 3x3	ARL Trace CM	n/a	n/a	d:\ARL\exp\convergence\trace_results\3x3trace_cm.res
Convergence 5x5	ARL Trace CM	n/a	n/a	d:\ARL\exp\convergence\trace_results\5x5trace_cm.res
Convergence 10x10	ARL Trace CM	n/a	n/a	d:\ARL\exp\convergence\trace_results\10x10trace_cm.res
Convergence 3x3	ARL Trace IM	n/a	n/a	d:\ARL\exp\convergence\trace_results\3x3trace_im.res
Convergence 5x5	ARL Trace IM	n/a	n/a	d:\ARL\exp\convergence\trace_results\5x5trace_im.res
Convergence 10x10	ARL Trace IM	n/a	n/a	d:\ARL\exp\convergence\trace_results\10x10trace_im.res
Convergence 3x3	ARL Lookahead CM	n/a	n/a	d:\ARL\exp\convergence\lookahead_results\3x3lookahead_cm.res
Convergence 5x5	ARL Lookahead CM	n/a	n/a	d:\ARL\exp\convergence\lookahead_results\5x5lookahead_cm.res
Convergence 10x10	ARL Lookahead CM	n/a	n/a	d:\ARL\exp\convergence\lookahead_results\10x10lookahead_cm.res
Convergence 3x3	ARL Lookahead IM	n/a	n/a	d:\ARL\exp\convergence\lookahead_results\3x3lookahead_im.res
Convergence 5x5	ARL Lookahead IM	n/a	n/a	d:\ARL\exp\convergence\lookahead_results\5x5lookahead_im.res
Convergence 10x10	ARL Lookahead IM	n/a	n/a	d:\ARL\exp\convergence\lookahead_results\10x10lookahead_im.res
Convergence 3x3	Q-learning	n/a	n/a	d:\ARL\exp\convergence\qlearning_results\3x3qlearning.res
Convergence 5x5	Q-learning	n/a	n/a	d:\ARL\exp\convergence\qlearning_results\5x5qlearning.res
Convergence 10x10	Q-learning	n/a	n/a	d:\ARL\exp\convergence\qlearning_results\10x10qlearning.res
Generalization 1	ARL Lookahead CM	1	1	d:\ARL\exp\genex\results\n*\genex1_group1_phase1.res
Generalization 1	ARL Lookahead CM	1	2	d:\ARL\exp\genex\results\n*\genex1_group1_phase2.res
Generalization 1	ARL Lookahead CM	2	1	d:\ARL\exp\genex\results\n*\genex1_group2_phase1.res
Generalization 1	ARL Lookahead CM	2	2	d:\ARL\exp\genex\results\n*\genex1_group2_phase2.res
Generalization 2	ARL Lookahead CM	1	1	d:\ARL\exp\genex\results\n*\genex2_group1_phase1.res
Generalization 2	ARL Lookahead CM	1	2	d:\ARL\exp\genex\results\n*\genex2_group1_phase2.res
Generalization 2	ARL Lookahead CM	2	1	d:\ARL\exp\genex\results\n*\genex2_group2_phase1.res
Generalization 2	ARL Lookahead CM	2	2	d:\ARL\exp\genex\results\n*\genex2_group2_phase2.res

Table C.3: Raw results data files index 2.

Experiment	Algorithm	Group	Phase	file path (n*: replace with number from 1 to 8)
Generalization 3	ARL Lookahead CM	1	1	d:\ARL\exp\genex\results\n*\genex3_group1_phase1.res
Generalization 3	ARL Lookahead CM	1	2	d:\ARL\exp\genex\results\n*\genex3_group1_phase2.res
Generalization 3	ARL Lookahead CM	2	1	d:\ARL\exp\genex\results\n*\genex3_group2_phase1.res
Generalization 3	ARL Lookahead CM	2	2	d:\ARL\exp\genex\results\n*\genex3_group2_phase2.res
Generalization 4	ARL Lookahead CM	1	1	d:\ARL\exp\genex\results\n*\genex4_group1_phase1.res
Generalization 4	ARL Lookahead CM	1	2	d:\ARL\exp\genex\results\n*\genex4_group1_phase2.res
Generalization 4	ARL Lookahead CM	2	1	d:\ARL\exp\genex\results\n*\genex4_group2_phase1.res
Generalization 4	ARL Lookahead CM	2	2	d:\ARL\exp\genex\results\n*\genex4_group2_phase2.res
Generalization 1	Q-learning	1	1	d:\ARL\exp\genex\results\n*\genex1_group1_phase1_Qlearning.res
Generalization 1	Q-learning	1	2	d:\ARL\exp\genex\results\n*\genex1_group1_phase2_Qlearning.res
Generalization 1	Q-learning	2	1	d:\ARL\exp\genex\results\n*\genex1_group2_phase1_Qlearning.res
Generalization 1	Q-learning	2	2	d:\ARL\exp\genex\results\n*\genex1_group2_phase2_Qlearning.res
Generalization 2	Q-learning	1	1	d:\ARL\exp\genex\results\n*\genex2_group1_phase1_Qlearning.res
Generalization 2	Q-learning	1	2	d:\ARL\exp\genex\results\n*\genex2_group1_phase2_Qlearning.res
Generalization 2	Q-learning	2	1	d:\ARL\exp\genex\results\n*\genex2_group2_phase1_Qlearning.res
Generalization 2	Q-learning	2	2	d:\ARL\exp\genex\results\n*\genex2_group2_phase2_Qlearning.res
Generalization 3	Q-learning	1	1	d:\ARL\exp\genex\results\n*\genex3_group1_phase1_Qlearning.res
Generalization 3	Q-learning	1	2	d:\ARL\exp\genex\results\n*\genex3_group1_phase2_Qlearning.res
Generalization 3	Q-learning	2	1	d:\ARL\exp\genex\results\n*\genex3_group2_phase1_Qlearning.res
Generalization 3	Q-learning	2	2	d:\ARL\exp\genex\results\n*\genex3_group2_phase2_Qlearning.res
Generalization 4	Q-learning	1	1	d:\ARL\exp\genex\results\n*\genex4_group1_phase1_Qlearning.res
Generalization 4	Q-learning	1	2	d:\ARL\exp\genex\results\n*\genex4_group1_phase2_Qlearning.res
Generalization 4	Q-learning	2	1	d:\ARL\exp\genex\results\n*\genex4_group2_phase1_Qlearning.res
Generalization 4	Q-learning	2	2	d:\ARL\exp\genex\results\n*\genex4_group2_phase2_Qlearning.res
Aversive avoidance	ARL Lookahead IM	n/a	n/a	d:\ARL\exp\aversive_avoidance\results\ava10x10ARL.res
Aversive avoidance	Q-learning	n/a	n/a	d:\ARL\experiments\aversiveavoidance10x10qlearning.res

Table C.4: Prepared excel sheets from raw data.

Experiment	Algorithm	file path
Convergence	n/a	d:\ARL\exp\convergence\convergence_results.xls
Generalization	ARL lookahead CM	d:\ARL\exp\genex\results\excel\genex_results.xls
Generalization	Q-learning	d:\ARL\exp\genex\results\excel\genex_results_qlearning.xls
Aversive avoidance	n/a	d:\ARL\exp\aversive_avoidance\results\results.xls

Appendix D

User manual for learning simulator

D.1 Starting the simulator

1. In order to run the learning simulator, type the following at the command line (replace d: with the letter of your CD-ROM):

   ```
   java -jar d:\ARL\code\ARL\dist\ARL.jar Simulator
   ```

2. At the initial screen of the learning simulator in figure D.1, do one of the following:

   (a) Select File->New. This will open a dialog in which a new Grid world configuration can be defined (figure D.2). In this dialog the textfield for "Number of horizontal grids:" represents the number of squares of the Grid world along the x axis; enter a positive integer. The textfield for "Number of vertical grids:" represents the number of squares of the Grid world along the y axis: enter a positive integer. The textfield for "Size of grids:" represents the visual size of each square of the Grid world on the screen; enter a positive integer. At the bottom of the dialog, there are two checkboxes. The "Reset RL agent memory"-checkbox will reset the associative memory of the Q-learner agent upon pressing the OK button. The "Reset ARL agent memory"-checkbox will reset the associative memory and the expectance memory of the ARL agent upon pressing the OK button. Pressing the OK button will load a new Grid world configuration (for further instructions see section D.2). Pressing the Cancel button will result in the dialog being closed and no action taken.

   (b) Select File->Load. This will open a dialog in which an existing grid world configuration and algorithm parameters can be loaded from file. These files have the extension ".lrn". Select a ".lrn" file from a given location at the attached CD-ROM disc. The "Reset RL agent memory"-checkbox will reset

the associative memory of the Q-learner agent upon pressing the Open button. The "Reset ARL agent memory"-checkbox will reset the associative memory and the expectance memory of the ARL agent upon pressing the Open button. Pressing the Open button will load the selected Grid world configuration file (for further instructions see section D.2). Pressing the Cancel button will result in the dialog being closed and no action taken.

(c) Select File->Exit. This will close the learning simulator.

D.2 Running the learning simulator

You now have an instance of the learning simulator running with either a new Grid world configuration or an existing configuration loaded from file. The learning simulator GUI as illustrated in figure D.4 should now be visible on the screen. The learning simulator GUI has 7 panels, each of which are described in the following list:

- **Gridworld:**
  - The orange square denotes the start location
  - The white square denotes the goal location
  - The yellow square denotes the current location of the Q-learner agent
  - The pink square denotes the current location of the ARL agent
  - Black squares denote obstacles
  - A selected square has a yellow edge
- **RLAgent**
  - The Run button will start learning for the Q-learning agent, if the Learning checkbox is selected.
  - The "Reset agent" button will reset the Q-learning agent's associative memory and the epochs counter, and close any open results file.
  - The Epochs label shows the current epoch of the agent's learning period. To the right of the epochs label, the policy cost is reported
  - Textboxes Alpha, Gamma, Epsilon refer to learning rate, reward discounting, and exploration rate
  - When the VP checkbox (View Policy) is selected, the policy value and policy cost will be displayed in the Gridworld panel
- **ARLAgent**
  - The Run button will start learning for the ARL agent
  - The "Reset agent" button will reset the ARL agent's associative memory and the epochs counter, and close any open results file.
  - The Epochs label shows the current epoch of the agent's learning period. To the right of the epochs label, the policy cost is reported

- StDev (sensitivity of similarity function), Max Alpha (learning rate), Max beta (learning rate), epsilon (exploration), Sim. threshold (ARL trace specific), Decay (reward discounting)
- When the VP checkbox (View Policy) is selected, the policy value and policy cost will be displayed in the Gridworld panel
- Select radio button "Trace" for ARL trace, and "Lookahead" for ARL lookahead.
- To combine the associative memory and the expectance memory, select the "Combine memories" checkbox.
- The table at the bottom of the ARLAgent panel lists the internal drives of the ARL agent.
- The "Remove US" button deletes the selected stimulus from the "internal drives"-table.

- **Spatial location**
  - The "Selected location" label displays the coordinates of the currently selected square in the Gridworld panel, or nothing when no square is selected.
  - The Obstacle checkbox adds/removes an obstacle to the currently selected square.
  - The Start checkbox defines/undefines the currently selected square as the start location.
  - The Goal checkbox defines/undefines the currently selected square as the goal location.
  - The Reward label shows the reward value of the currently selected square.
  - The "Sync reward" button will synchronize the reward structure of the ARL agent and the Q-learner agent.
  - In the "Template list"-table, the currently defined stimuli are shown.
  - The "Stimuli list"-table displays the stimuli compound at the currently selected location.
  - The "Delete selected" button deletes the selected stimulus from the "Stimuli list"-table.
  - The "Delete" button deletes the selected stimulus from the "Template list"-table.
  - The "Add" button adds a new stimulus to the template list.

- **Expectance convergence graph:** Live plotting of Average absolute fluctuation in expectance memory per epoch
- **Associative memory convergence graph:** Live plotting of Average absolute fluctuation in associative memory per epoch
- **Q-value convergence graph:** Live plotting of Average absolute fluctuation in q-values per epoch

D.2.1 Running an experiment

1. Select File->Results. This will open the results dialog (figure D.5).
2. Click the "..."-button to the right of the "ARL path" textbox, to open a results data file for the ARL agent.
3. Click the "..."-button to the right of the "RL path" textbox, to open a results data file for the Q-learner agent.
4. The "Report policy cost @ every"-spinner to the right of the "ARL path" textbox defines the frequency at which the policy cost is reported to the ARL agent results data file and to the ARLAgent panel.
5. The "Report policy cost @ every"-spinner to the right of the "RL path" textbox defines the frequency at which the policy cost is reported to the Q-learner agent results data file and to the RLAgent panel.
6. Do either:
   (a) Click the OK button to open the results data files. A confirmation dialog will appear; click OK to confirm or Cancel to return to the results dialog. When OK is clicked, any previously open results data files will be closed, and the newly specified will be opened. The control will return to the main screen (figure D.4).
   (b) Click the Cancel button. The results dialog will disappear and no action will be taken.
7. Click the Run button in either the ARLAgent panel or the RLAgent panel, and let the agent run for as many epochs as you wish.
8. When finished, first click the "Pause" button, and then do either:
   (a) Click the "Reset agent" in either or both the ARLAgent panel and the RLAgent panel to close the results data files, and reset the agent's memory.
   (b) To open an existing or define a new grid world environment without resetting the agent's memory (e.g. as in phase 2 of the generalization experiments) goto step 2b of enumeration list D.1, and follow the instructions. Then go to step 1 of this list.

D.2.2 Defining a new stimulus

1. Click the Add button of the "Spatial location" panel. A new stimulus will appear at the end of the "Template list"-table.
2. Select the desired modality and strength for the stimulus from the dropdown boxes in the "Template list"-table.

D.2.3 Adding a stimulus to the Gridworld

1. Select a square in the Gridworld panel
2. In the ”Add to” column of the ”Template list”-table of the ”Spatial location” panel, click the ”Lo...”-button for the desired stimulus
3. The stimulus will appear in the ”Stimuli list”-table for the selected location of the Gridworld panel

D.2.4 Deleting a stimulus from the Gridworld

1. Select a square in the Gridworld panel
2. Select the stimulus that you wish to delete from the ”Stimuli list”-table of the ”Spatial location” panel
3. Click the ”Delete selected” button in the ”Spatial location” panel

D.2.5 Adding/deleting a stimulus from the list of internal drives of the ARL agent

1. In the ”Add as” column of the ”Template list”-table of the ”Spatial location” panel click the US button for the desired stimulus.
2. The stimulus will appear in the ”Internal drives”-table of the ARLAgent panel
3. In the Category column of the ”Internal drives”-table of the ARLAgent panel select whether the added stimulus shall be aversive or appetitive in the provided dropdown box.
4. Make sure the added stimulus is out of focus for the change in category to take effect (i.e. click anywhere else in the ARLAgent panel).

D.2.6 Defining obstacles, start location, and goal location

- To define an obstacle, do:
  1. Select a square in the Gridworld panel
  2. Check/uncheck the Obstacle checkbox of the ”Spatial location” panel
- To define the start location, do:
  1. Select a square in the Gridworld panel
  2. Check/uncheck the Start checkbox of the ”Spatial location” panel
- To define the goal location, do:
  1. Select a square in the Gridworld panel
  2. Check/uncheck the Goal checkbox of the ”Spatial location” panel

D.2.7 Synchronizing the environment reward structure of the ARL agent and the Q-learner agent

Click the ”Sync reward” button of the ”Spatial location” panel.

D.2.8 Selecting a spatial location

Click inside a square of the Gridworld panel to select it.

D.2.9 Adding/removing obstacles

- Click and drag inside the Gridworld panel to define obstacles, and release mouse button to stop.
- Alt-Click and drag to remove obstacles, and release mouse button to stop removing obstacles.
- Select a square in the Gridworld panel. In the Spatial location panel, check-/uncheck the Obstacle checkbox.

D.2.10 Showing/hiding live plots

In the view menu (figure D.7), check/uncheck the desired checkbox to show/hide one of live plots. (”ARL (AM) Graph” = Associative memory convergence graph, ”ARL (EM) Graph” = Expectance convergence graph, ”RL Graph” = Q-value convergence graph).

D.2.11 Saving a Grid world configuration and algorithm parameters

1. Select File->Save. A save dialog will appear (figure D.6).
2. Choose a file or type in a file name at a location of your choice.
3. Do either:
   (a) Click the Save button to save the grid world configuration and algorithm parameters
   (b) Click the Cancel button to take no action and close the Save dialog. The control will return to the main screen (figure D.4).

Figure D.1: The initial screen of the learning simulator

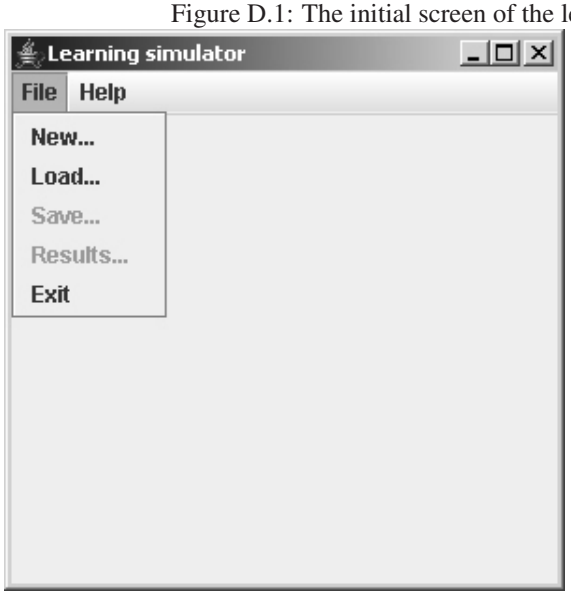


Figure D.2: Dialog box to define new grid world environment

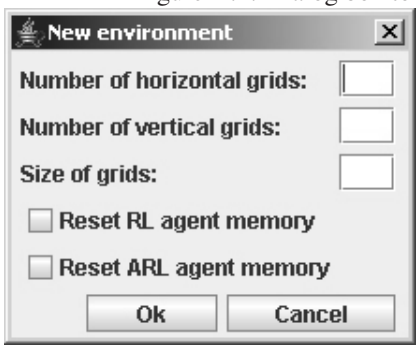


Figure D.3: Dialog box to load an existing grid world environment configuration, and algorithm parameters

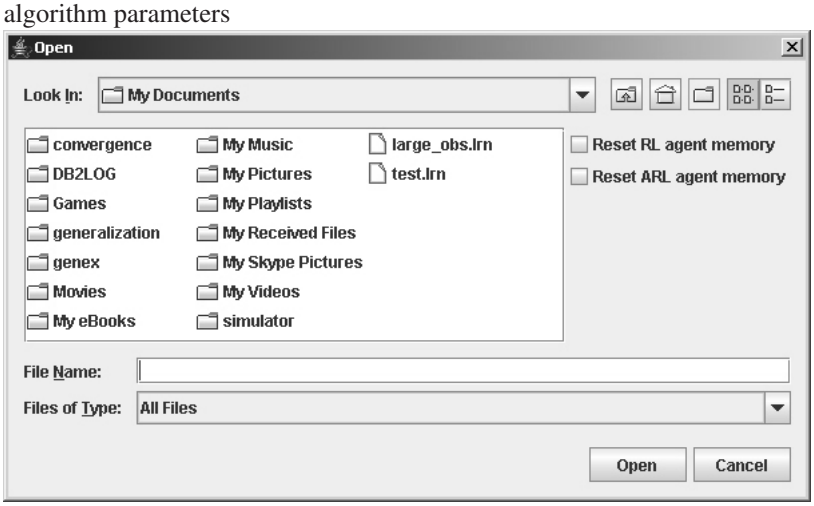


Figure D.4: The learning simulator with a loaded grid world configuration, and algorithm parameters

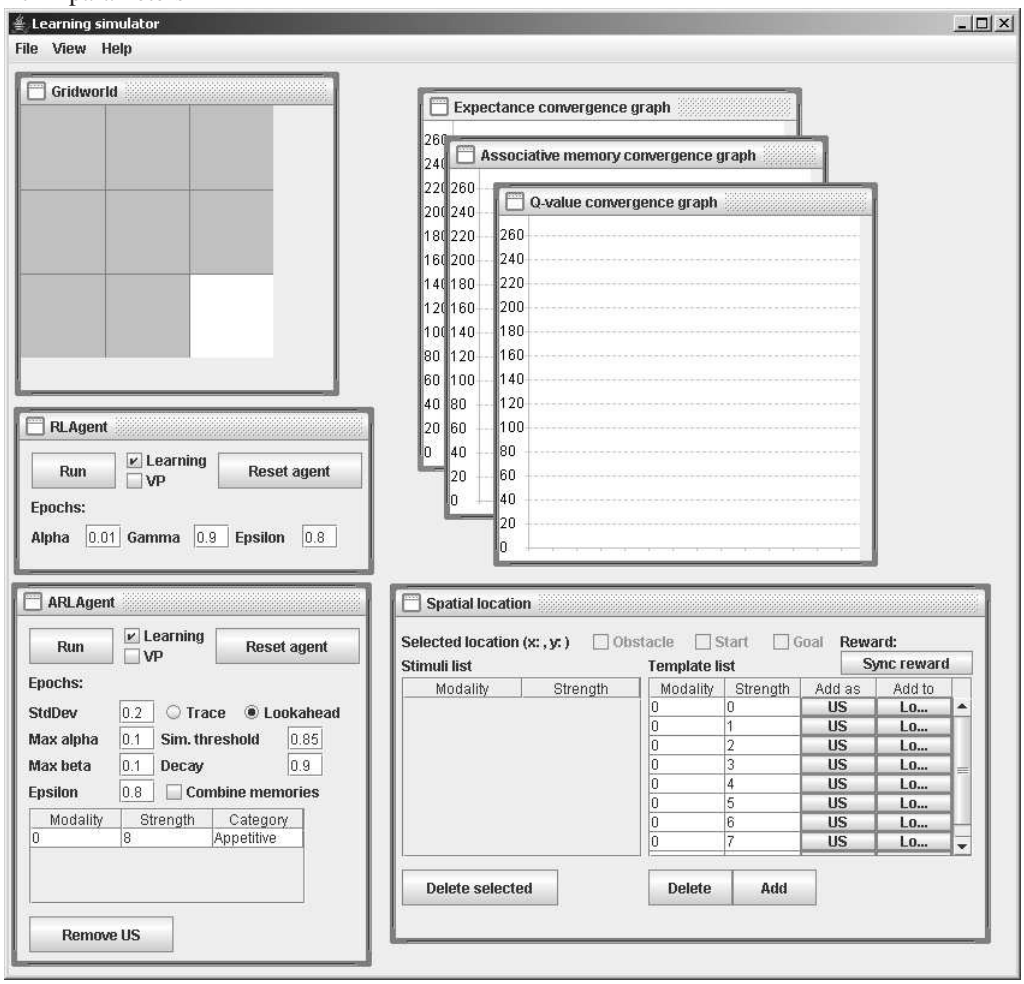


Figure D.5: Dialog to define results data files for each agent, and results reporting parameters

Figure D.6: Dialog to save a grid world configuration, and algorithm parameters

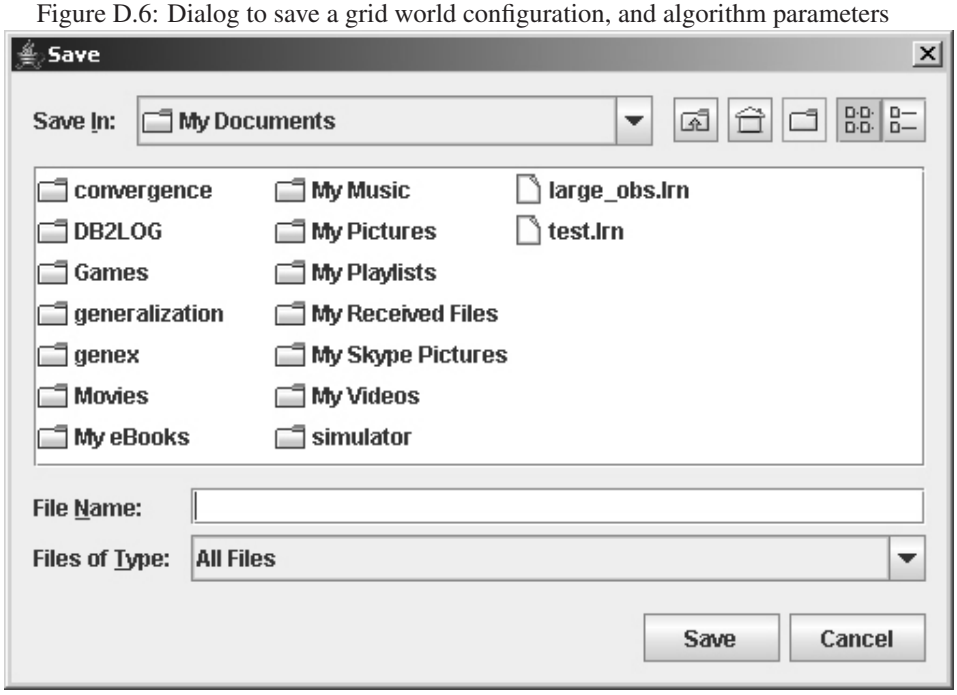


Figure D.7: The view menu