Action Selection methods using Reinforcement Learning

Mark Humphrys

Trinity Hall

A dissertation submitted for the degree of Doctor of Philosophy
in the University of Cambridge

Technical Report June 1997

Abstract

The Action Selection problem is the problem of run-time choice between conflicting and heterogenous goals, a central problem in the simulation of whole creatures (as opposed to the solution of isolated uninterrupted tasks). This thesis argues that Reinforcement Learning has been overlooked in the solution of the Action Selection problem. Considering a decentralised model of mind, with internal tension and competition between selfish behaviors, this thesis introduces an algorithm called "W-learning", whereby different parts of the mind modify their behavior based on whether or not they are succeeding in getting the body to execute their actions. This thesis sets W-learning in context among the different ways of exploiting Reinforcement Learning numbers for the purposes of Action Selection. It is a 'Minimize the Worst Unhappiness' strategy. The different methods are tested and their strengths and weaknesses analysed in an artificial world.

Declaration

I hereby declare that this dissertation is the result of my own work and, unless explicitly stated in the text, contains nothing which is an outcome of work done in collaboration. No part of this dissertation has already been or is currently being submitted for any degree, diploma or other qualification at any other university.

Contact Address

Mark Humphrys
University of Cambridge, Computer Laboratory
New Museums Site, Cambridge CB2 3QG, England
`Mark.Humphrys@cl.cam.ac.uk`
`http://www.cl.cam.ac.uk/~mh10006/`

When I move, the old web address will point to the new location.

Dedication

This dissertation is dedicated to my mother and father,
to whom I owe everything.

In loving memory of my grandmother.

Contents

Chapter 1

The Action Selection problem

By Action Selection we do not mean the low-level problem of choice of action in pursuit of a single coherent goal. Rather we mean the higher-level problem of choice between conflicting and heterogenous goals. These goals are pursued in parallel. They *may* sometimes combine to achieve larger-scale goals, but in general they simply interfere with each other. They may not have any terminating conditions.

These two problems have often been confused in the literature - this thesis shall argue in favour of carefully separating them. All systems solve the first problem in some form. The second one has been given far less thought.

Action Selection is emerging as a central problem in the simulation of whole creatures (as opposed to the solution of isolated uninterrupted tasks). It is clear that many interesting systems possess multiple parallel and conflicting goals, among which attention must endlessly switch. Animals are the prime example of such systems.

Typically, the action selection models proposed in ethology are not detailed enough to specify an algorithmic implementation (see [Tyrrell, 1993] for a survey, and for many difficulties in translating the conceptual models into computational ones). The action selection models that do lend themselves to algorithmic implementation (e.g. see [Brooks, 1991, Rosenblatt, 1995, Blumberg, 1994, Sahota, 1994, Aylett, 1995]) then typically require a considerable design effort. In the literature, one sees formulas taking weighted sums of various quantities in an attempt to estimate the utility of actions. There is much hand-coding and tuning of parameters (e.g. see [Tyrrell, 1993, §9]) until the designer is satisfied that the formulas deliver utility estimates that are fair.

In fact, there may be a way that these utility values can come for free. Learning methods that automatically assign values to actions are common in the field of Reinforcement Learning (RL), or learning from rewards. Reinforcement Learning propagates numeric rewards into behavior patterns. The rewards may be objective, external value judgements, or just internally generated numbers. This thesis compares a number of different methods of further propagating these numbers to solve the action selection problem.

The low-level problem of pursuing a single goal can be solved by straightforward RL, which assumes such a single goal. For the high-level problem of choice between conflicting goals we try various methods exploiting the low-level RL numbers. The methods range from centralised and cooperative to decentralised and selfish.

Instead of presenting yet another domain-specific program, this thesis will be searching for general principles across domains. It will be trying to categorize all the different ways in which the action selection of behaviors can be organized automatically.

1.1 Terminology

Terminology in this area is a problem. And one I'm not necessarily going to solve here. These are the crucial terms I use and their explanation:

agent - the mind or part of the mind. A piece of semi-autonomous software. The usage is from Minsky's Society of Mind [Minsky, 1986], in which multiple agents may inhabit the same body. Agents have more autonomy than traditional *modules* or *procedures*. They are autonomous actors in their own right, not waiting on some higher level to call them. They are not necessarily structured in any hierarchy and are not even necessarily cooperative. In the model I introduce in §5, an agent will control the entire body on its own if allowed. It is generally frustrated by the presence of other agents. Baum also uses *agent* in his Economy of Mind [Baum, 1996].

Behavior (e.g. [Sahota, 1994, Mataric, 1994]) is probably the term in most common use, but is often used for modules that are not autonomous actors. For somewhat-autonomous, somewhat-competing modules within a single body, [Brooks, 1986] uses *layer* (though [Brooks, 1994] also uses *process*), [Blumberg, 1994] uses *activity* (as does the ethologist McFarland), [Tyrrell, 1993] uses *system* and *subsystem* (after the ethologist Baerends) and [Selfridge and Neisser, 1960] use *demon*. Other names, from psychology, include *simpleton* and *homunculus* (of the non-intelligent kind).

The word *agent* is still not altogether satisfactory since it is being claimed by other fields. Currently multi-agent learning implies multibodies (e.g. [Mataric, 1994, Tan, 1993, Grefenstette, 1992]). Symbolic AI agents imply *communication*, where agents negotiate among themselves (this is not altogether against the meaning of the term here). And in the long term, *agent* may mean Internet alter egos acting as an 'agency' on behalf of the user.

creature - the body. The animat [Wilson, 1990] or artificial animal. In the House Robot problem of §4, the house robot. In the Ant World problem of §7, the ant.

The usage is from Brooks [Brooks, 1991]. The creature may be inhabited by a single monolithic agent (a unitary mind) or a family of agents (a decentralised mind).

1.2 Notation

Throughout this thesis, the notation:

$$D \mapsto d$$

means we adjust the estimate D *once* in the direction of the sample d. For example, if we store D explicitly we may update:

$$D := (1-\alpha)D + \alpha d$$

where α is the *learning rate*. See the appendix §A to understand how sampling with a learning rate works. Alternatively, if we are using a neural network to return an estimate D, we may backpropagate the error $D - d$.

The notation:

$$D \to E(d)$$

means that over many such updates the estimate D converges to the expected value of the samples d.

1.3 A brief guide to this dissertation

This dissertation can be divided into the following parts:

- **Chapters 1-13** - the Action Selection problem, introduction of Reinforcement Learning and our mathematical framework, the test problem, test of current strategies, invention and test of more decentralised strategies, followed by: **Chapter 14** - summary of the four schools of decentralised Action Selection methods and their empirical results.
- **Chapter 15** - survey of a wide variety of related work, translating it into our mathematical framework and comparing it to our methods, followed by: **Chapter 16** - conclusions of this thesis, what we have learnt.
- **Chapter 17** - survey of possible further work, followed by: **Chapter 18** - how these models might scale up into a general decentralised, nested, non-hierarchical model of mind.

The reader familiar with RL may wish to skip Chapters 2 and 3. There are some references in this thesis to other publications of mine,[1] but everything necessary should be explained here. This dissertation is a self-contained document.

First we introduce Reinforcement Learning.

[1] All my publications are at:
`http://www.cl.cam.ac.uk/~mh10006/publications.html`

Chapter 2

Reinforcement Learning

A Reinforcement Learning (RL) agent senses a world, takes actions in it, and receives numeric rewards and punishments from some reward function based on the consequences of the actions it takes. By trial-and-error, the agent learns to take the actions which maximize its rewards. For a broad survey of the field see [Kaelbling et al., 1996].

2.1 Q-learning

Watkins [Watkins, 1989] introduced the method of reinforcement learning called *Q-learning*. The agent exists within a world that can be modelled as a Markov Decision Process (MDP). It observes discrete states of the world x ($\in X$, a finite set) and can execute discrete actions a ($\in A$, a finite set). Each discrete time step, the agent observes state x, takes action a, observes new state y, and receives immediate reward r. Transitions are probabilistic, that is, y and r are drawn from stationary probability distributions $P_{xa}(y)$ and $P_{xa}(r)$, where $P_{xa}(y)$ is the probability that taking action a in state x will lead to state y and $P_{xa}(r)$ is the probability that taking action a in state x will generate reward r. We have $\sum_y P_{xa}(y) = 1$ and $\sum_r P_{xa}(r) = 1$.

Note that having a reward every time step is actually no restriction. The classic *delayed reinforcement* problem is just a special case with, say, most rewards zero, and only a small number of infrequent rewards non-zero.

The transition probability can be viewed as a conditional probability. Where x_t, a_t are the values of x, a at time t, the Markov property is expressed as:

$$\begin{aligned} &P(x_{t+1} = y | x_t = x, a_t = a, \quad x_{t-1} = s_{t-1}, a_{t-1} = b_{t-1}, \quad \dots, \quad x_0 = s_0, a_0 = b_0) \\ &\quad = P(x_{t+1} = y | x_t = x,\ a_t = a) \end{aligned}$$

and the stationary property is expressed as:

$$P(x_{t+1} = y | x_t = x, a_t = a) = P_{xa}(y) \quad \forall t$$

2.1.1 Special case - Deterministic worlds

The simple *deterministic* world is a special case with all transition probabilities equal to 1 or 0. For any pair x, a, there will be a unique state y_{xa} and a unique reward r_{xa} such that:

$$P_{xa}(y) = \begin{cases} 1 & \text{if } y = y_{xa} \\ 0 & \text{otherwise} \end{cases}$$
$$P_{xa}(r) = \begin{cases} 1 & \text{if } r = r_{xa} \\ 0 & \text{otherwise} \end{cases}$$

2.1.2 Special case - Different action sets

The set of actions available may differ from state to state. Depending on what we mean by this, this may also possibly be represented within the model, as the special case of an action that when executed (in this particular state, not in general) does nothing:

$$P_{xa}(x) = 1$$

for all actions a in the 'unavailable' set for x.

Later in this thesis, we will be considering how agents react when the actions of *other agents* are taken, actions that this agent does not recognise. Here the action *is* recognised by the agent, just in some states it becomes unavailable.

In the multi-agent case, an agent could assume that an unrecognised action has the effect of 'do nothing' but it might be unwise to assume without observing.

2.1.3 Notes on expected reward

When we take action a in state x, the reward we expect to receive is:

$$E(r) = \sum_r r P_{xa}(r)$$

Typically, the reward function is not phrased in terms of what action we took but rather what new state we arrived at, that is, r is a function of the transition x to y. The general idea is that we *can't* reward the right a because we don't know what it is - that's why we're learning. If we knew the correct a we could use ordinary *supervised* learning.

Writing $r = r(x, y)$, the probability of a *particular* reward r is:

$$P_{xa}(r) = \sum_{\{y \mid r(x,y)=r\}} P_{xa}(y)$$

and the expected reward becomes:

$$E(r) = \sum_{y} r(x, y) P_{xa}(y)$$

That is, normally we never think in terms of $P_{xa}(r)$ - we only think in terms of $P_{xa}(y)$. Kaelbling [Kaelbling, 1993] defines a *globally consistent* world as one in which, for a given x, a, $E(r)$ is constant. This is equivalent to requiring that $P_{xa}(r)$ is stationary (hence, with the typical type of reward function, $P_{xa}(y)$ stationary).

In some models, rewards are associated with states, rather than with transitions, that is, $r = r(y)$. The agent is not just rewarded for arriving at state y - it is also rewarded continually for remaining in state y. This is obviously just a special case of $r = r(x, y)$ with:

$$r(x, y) = r(y) \quad \forall x$$

and hence:

$$E(r) = \sum_{y} r(y) P_{xa}(y)$$

Rewards r are bounded by $r_{\min}, r_{\max}$, where $r_{\min} < r_{\max}$ ($r_{\min} = r_{\max}$ would be a system where the reward was the same no matter what action was taken. The agent would always behave randomly and would be of no use or interest). Hence for a given x, a, $r_{\min} \leq E(r) \leq r_{\max}$.

2.1.4 The task

The agent acts according to a policy π which tells it what action to take in each state x. A policy that specifies a unique action to be performed $a = \pi(x)$ is called a *deterministic* policy - as opposed to a *stochastic* policy, where an action a is chosen from a distribution P_x^{π} with probability $P_x^{\pi}(a)$.

A policy that has no concept of time is called a *stationary* or *memoryless* policy - as opposed to a non-stationary policy (e.g.'do actions a and b alternately'). A non-stationary policy requires the agent to possess memory. Note that stochastic does *not* imply non-stationary.

Following a stationary deterministic policy π, at time t, the agent observes state x_t, takes action $a_t = \pi(x_t)$, observes new state x_{t+1}, and receives reward r_t with expected value:

$$E(r_t) = \sum_r r P_{x_t a_t}(r)$$

The agent is interested not just in immediate rewards, but in the *total discounted reward.* In this measure, rewards received n steps into the future are worth less than rewards received now, by a factor of γ^n where $0 \leq \gamma < 1$:

$$R = r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} + \cdots$$

The *discounting factor* γ defines how much expected future rewards affect decisions now. Genuine immediate reinforcement learning is the special case $\gamma = 0$, where we only try to maximize *immediate* reward. Low γ means pay little attention to the future. High γ means that potential future rewards have a major influence on decisions now - we may be willing to trade short-term loss for long-term gain. Note that for $\gamma < 1$, the value of future rewards always eventually becomes negligible. The expected total discounted reward if we follow policy π forever, starting from x_t, is:

$$\begin{aligned}
V^\pi(x_t) = E(R) \quad &= E(r_t) + \gamma E(r_{t+1}) + \gamma^2 E(r_{t+2}) + \cdots \\
&= E(r_t) + \gamma \left[E(r_{t+1}) + \gamma E(r_{t+2}) + \gamma^2 E(r_{t+3}) + \cdots \right] \\
&= E(r_t) + \gamma V^\pi(x_{t+1}) \\
&= \textstyle\sum_r r P_{x_t a_t}(r) + \gamma \sum_y V^\pi(y) P_{x_t a_t}(y)
\end{aligned}$$

$V^\pi(x)$ is called the *value* of state x under policy π. The problem for the agent is to find an optimal policy - one that maximizes the total discounted expected reward (there may be more than one policy that does this). Dynamic Programming (DP) theory [Ross, 1983] assures us of the existence of an optimal policy for an MDP, also (perhaps surprisingly) that there is a stationary and deterministic optimal policy. In an MDP one does not need memory to behave optimally, the reason being essentially that the current state contains all information about what is going to happen next. A stationary deterministic optimal policy π^* satisfies, for all x:

$$V^{\pi^*}(x) = \max_{b \in A} \left[\sum_r r P_{xb}(r) + \gamma \sum_y V^{\pi^*}(y) P_{xb}(y) \right]$$

For any state x, there is a unique value $V^*(x)$, which is the best that an agent can do from x. Optimal policies π^* may be non-unique, but the value V^* is unique. All optimal policies π^* will have:

$$V^*(x) = V^{\pi^*}(x)$$

2.1.5 The strategy

The strategy that the Q-learning agent adopts is to build up *Quality-values* (Q-values) $Q(x,a)$ for each pair (x,a). If the transition probabilities $P_{xa}(y)$, $P_{xa}(r)$ are explicitly known, Dynamic Programming finds an optimal policy by starting with random $V(x)$ and random $Q(x,a)$ and repeating forever (or at least until the policy is considered good enough):

for all x
  for all a
    $Q(x,a) := \sum_r r P_{xa}(r) + \gamma \sum_y V(y) P_{xa}(y)$
  $V(x) := \max_{a \in A} Q(x,a)$

This systematic iterative DP process is obviously an *off-line* process. The agent must cease interacting with the world while it runs through this loop until a satisfactory policy is found.

The problem for the Q-learning agent is to find an optimal policy when $P_{xa}(y)$, $P_{xa}(r)$ are initially unknown (i.e. we are assuming when modelling the world that *some* such transition probabilities exist). The agent must *interact* with the world to learn these probabilities. Hence we cannot try out states and actions systematically as in Dynamic Programming. We may not know how to *return* to x to try out a different action (x,b) - we just have to wait until x happens to present itself again. We will experience states and actions rather haphazardly.

Fortunately, we can still learn from this. In the DP process above, the updates of $V(x)$ can in fact occur in any order, provided that each x will be visited infinitely often over an infinite run. So we can interleave updates with acting in the world, having a modest amount of computation per timestep. In Q-learning we cannot update directly from the transition probabilities - we can only update from individual experiences. In 1-step Q-learning, after each experience, we observe state y, receive reward r, and update:

$$Q(x,a) \mapsto (r + \gamma \max_{b \in A} Q(y,b))$$

(remembering the notation of §1.2). For example, in the discrete case, where we store each $Q(x,a)$ explicitly in lookup tables, we update:

$$Q(x,a) := (1-\alpha)Q(x,a) + \alpha(r + \gamma \max_{b \in A} Q(y,b))$$

Since the rewards are bounded, it follows that the Q-values are bounded (Theorem B.2). Note that we are updating from the current *estimate* of $Q(y,b)$ - this too is constantly changing.

2.1.6 How Q-learning works

Figure 2.1 illustrates how Q-learning deals with delayed reinforcement. Let $V(x) = \max_{b \in A} Q(x,b)$ be the *estimated* value of x during Q-learning. $V(x) = Q(x,a)$, for some a which leads to some state y. $Q(x,a)$ will be a combination of immediate reward r_{xa} plus the estimated value $V(y)$. $V(y)$ itself will be equal to some $Q(y,b)$, where b leads to z. $Q(y,b)$ is a combination of r_{yb} plus $V(z)$, and so on.

Imagine that $r_{xa} = r_{yb} = 0$, but from state z the agent can take an action c that yields a high reward $r_{zc} = r$ (Watkins used the example of an animal arriving at a waterhole). Then $Q(z,c)$ will be scored high, and so $V(z)$ increases. $Q(y,b)$ does not immediately increase, but *next time* it is updated, the increased $V(z)$ is factored in, discounted by γ. $Q(y,b)$ and hence $V(y)$ increase - y is now valuable because from there you can get to z. $Q(x,a)$ does not immediately increase, but next time *it* is updated, the increased $V(y)$ is factored in, discounted by γ (hence $V(z)$ is effectively factored in, discounted by γ^2). $Q(x,a)$ increases because it led to y, and from there you can get to z, and from there you can get a high reward. In this way the rewards are slowly propagated back through time.

In this way, the agent can learn *chains* of actions. It looks like it can see into the future - though in reality it lives only in the present (it cannot even predict what the next state will be). Using the language of ethology [Tyrrell, 1993], this is how credit is propagated from a *consummatory* action back through the chain of *appetitive* actions that preceded it.

Q-learning solves the temporal credit assignment problem without having to keep a memory of the last few chains of states. The price you pay is that it demands repeated visits. Weir [Weir, 1984] argues that the notion of

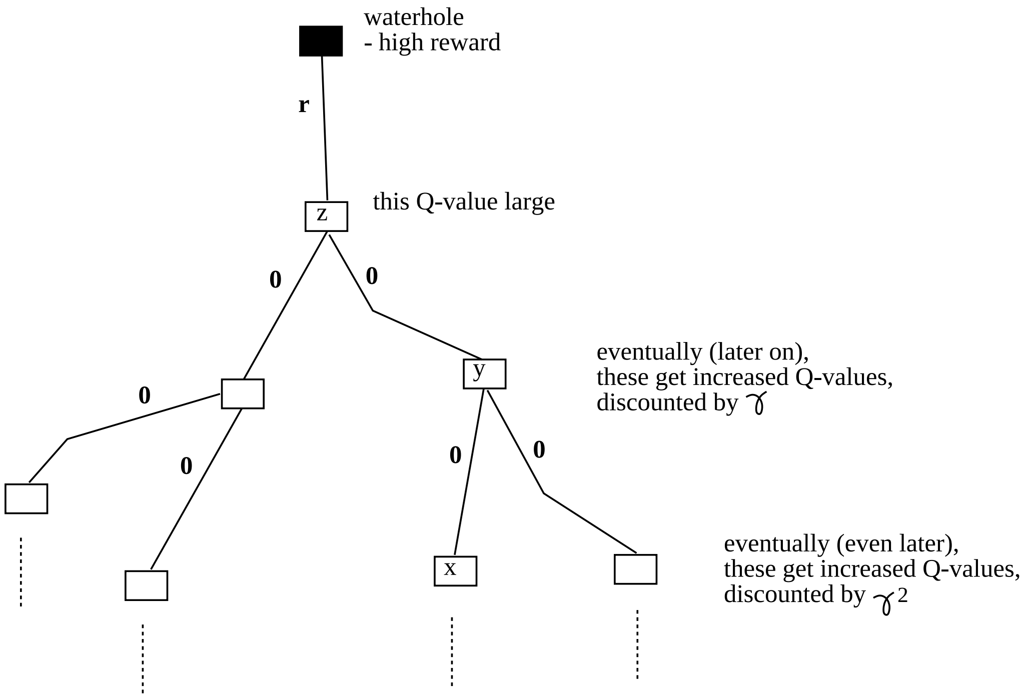


Figure 2.1: How Q-learning works (adapted from a talk by Watkins).

state needs to be expanded into a *trajectory* or chain of states through the statespace, but this example shows that such a complex model is not needed for the purposes of time-based behavior.

2.1.7 Notes on building a world model

As Q-learning progresses, the agent experiences the world's transitions. It does not normally build up an explicit map of $P_{xa}(r)$ of the form $x, a, r \rightarrow [0, 1]$. Instead it sums the rewards to build up the probably more useful mapping $x, a \rightarrow E(r)$. In fact this exact mapping itself is not stored, but is only one component of the Q-values, which express more than just immediate reward. If $\gamma = 0$ the Q-values express the exact map $x, a \rightarrow E(r)$.

The agent also experiences the state transitions and can if it chooses build up a *world model*, that is, estimates of $P_{xa}(y)$. This would be the map $x, a, y \rightarrow [0, 1]$, whose implementation would demand a much larger statespace. Rewards can be coalesced of course but states can't. There is no such thing as $E(y)$ (except for the special case of a deterministic world $x, a \rightarrow y$).

Moore [Moore, 1990] applies machine learning to robot control using a state-space approach, also beginning without knowledge of the world's transitions. His robot does not receive reinforcement, but rather learns $P_{xa}(y)$ in situations where the designer can specify what the desired next y should be.

Moore works with a statespace too big for lookup tables. He stores (x, a, y) triples (memories) in a nearest-neighbour generalisation. With probabilistic transitions, the agent may experience (x, a, y) at some times and (x, a, z) at others. There may then be a conflict between finding the needed action: x, y predicts a, and predicting what it will do: x, a predicts not y. In fact Moore shows that this conflict can arise even in a deterministic world after limited experience (where the exemplars you use for prediction are not exact but are the nearest neighbours only).

Sutton's DYNA architecture [Sutton, 1990, Sutton, 1990a] uses reinforcement learning and builds a model in a deterministic world with a statespace small enough to use lookup tables. It just fills in the map $x, a \rightarrow y$.

The point that Q-learning illustrates though is that learning an explicit world model is not *necessary* for the central purpose of learning what actions to take. Q-learning is *model-free* ([Watkins, 1989, §7] calls this *primitive learning*). We ignore the question of what x, a actually lead to, and the problem of storage of that large and complex mapping, and just concentrate on scoring how *good* x, a is.

The agent can perform adaptively in a world without understanding it. All it tries to do is sort out good actions to perform from bad ones. A good introduction to this approach is [Clocksin and Moore, 1989], which uses a state-space approach to control a robot arm. Instead of trying to explicitly solve the real-time inverse dynamics equations for the arm, Clocksin and Moore view this as some generally unknown nonlinear I/O mapping and concentrate on filling in the statespace by trial and error until it is effective. They argue against constructing a mathematical model of the world in advance and hoping that the world will broadly conform to it (with perhaps some parameter adjustment during interaction). But here we could learn the entire world model by interaction. What would be the advantage of doing this?

For a single problem in a given world, a model is of limited use. As [Kaelbling, 1993] points out, in the time it takes to learn a model, the agent can learn a good policy anyway. The real advantage of a model is to avoid having to re-learn everything when either the world or problem change:

- Changing $P_{xa}(y)$ - when the dynamics of the world changes.

  Sutton shows in [Sutton, 1990a] how a model can be exploited to deal with a *changing world*. His agent keeps running over the world model in its head, updating values based on its estimates for the transition probabilities, as in Dynamic Programming. As it senses that the world has changed it will start to do a backup in its head. For example, in the waterhole scenario above, when it senses (by interaction with the world) that state z no longer leads to the waterhole, it will start to do a backup in its head, without having to wait to revisit y and x in real life.

- Changing $P_{xa}(r)$ - when we want to give the agent new goals in the same world.

  A standard usage of reinforcement learning is to have some known desired state y, at which rewards are given, leaving open what states the agent might pass through on the way to y. The agent learns by rewards propagating back from y (as in §2.1.6). The problem with this is we then can't give the agent a new explicit goal z and say 'Go there'. We have to *re-train* it to go to z.

  In Sutton's model, when the agent suddenly (in real life) starts getting rewards at state z it will immediately update its model and the rewards will rapidly propagate back in its mental updates, quicker than they would if the states had to be revisited in real life. Kaelbling's Hierarchical Distance to Goal (HDG) algorithm [Kaelbling, 1993a] addresses the issue of giving the agent new explicit z goals at run-time.

2.2 Discrete Q-learning

In the discrete case, we store each $Q(x, a)$ explicitly, and update:

$$Q(x, a) := (1 - \alpha)Q(x, a) + \alpha(r + \gamma \max_{b \in A} Q(y, b))$$

for some learning rate α which controls how much weight we give to the reward just experienced, as opposed to the old Q-estimate. We typically start with $\alpha = 1$, i.e. give full weight to the new experience. As α decreases, the Q-value is building up an average of all experiences, and the odd new unusual experience won't disturb the established Q-value much. As time goes to infinity, $\alpha \to 0$, which would mean no learning at all, with the Q-value fixed.

Remember that because in the short term we have inaccurate $Q(y, b)$ estimates, and because (more importantly) in the long term we have probabilistic transitions anyway, the return from the same (x, a) will vary. See §A.1 to understand how a decreasing α is generally used to take a converging expected value of a distribution.

We use a different $\alpha = \alpha(x, a)$ for each pair (x, a), depending on how often we have visited state x and tried action a there. When we are in a new area of state-space (i.e. we have poor knowledge), the learning rate is high. We will have taken very few samples, so the average of them will be highly influenced by the current sample. When we are in a well-visited area of state-space, the learning rate is low. We have taken lots of samples, so the single current sample can't make much difference to the average of all of them (the Q-value).

2.2.1 Convergence

[Watkins and Dayan, 1992] proved that the discrete case of Q-learning will converge to an optimal policy under the following conditions. The learning rate α, where $0 \leq \alpha \leq 1$, should take decreasing (with each update) successive values $\alpha_1, \alpha_2, \alpha_3 \ldots$, such that $\sum_{i=1}^{\infty} \alpha_i = \infty$ and $\sum_{i=1}^{\infty} \alpha_i^2 < \infty$. The typical scheme (and the one used in this work) is, where $n(x, a) = 1, 2, 3, \ldots$ is the number of times $Q(x, a)$ has been visited:

$$\begin{aligned} \alpha(x, a) &= \tfrac{1}{n(x,a)} \\ &= 1, \tfrac{1}{2}, \tfrac{1}{3}, \ldots \end{aligned}$$

If each pair (x, a) is visited an infinite number of times, then [Watkins and Dayan, 1992] shows that for lookup tables Q-learning converges to a unique set of values $Q(x, a) = Q^*(x, a)$ which define a stationary deterministic optimal policy. Q-learning is asynchronous and sampled - each $Q(x, a)$ is updated one at a time, and the control policy may visit them in any order, so long as it visits them an infinite number of times. Watkins [Watkins, 1989] describes his algorithm as "incremental dynamic programming by a Monte Carlo method".

After convergence, the agent then will maximize its total discounted expected reward if it always takes the action with the highest Q^*-value. That is, the optimal policy π^* is defined by $\pi^*(x) = a^*(x)$ where:

$$Q^*(x, a^*(x)) = \max_{b \in A} Q^*(x, b)$$

Then:

$$\begin{aligned} V^*(x) &= Q^*(x, a^*(x)) \\ &= \max_{b \in A} Q^*(x, b) \end{aligned}$$

a^* may be non-unique, but the value Q^* is unique.

Initial Q-values can be random. $(1-\alpha)$ typically starts at 0, wiping out the initial Q-value in the first update (Theorem A.1). Although the $\gamma \max_{b \in A} Q(y,b)$ term may factor in an inaccurate initial Q-value from elsewhere, these Q-values will themselves be wiped out by their own first update, and poor initial samples have no effect if in the long run samples are accurate (Theorem A.2). See [Watkins and Dayan, 1992] for the proof that initial values are irrelevant.

Note that $\alpha_i = \frac{1}{i^p}$ will satisfy the conditions for any $\frac{1}{2} < p \leq 1$. The typical α goes from 1 down to 0, but in fact if the conditions hold, then for any t, $\sum_{i=t}^{\infty} \alpha_i = \infty$ and $\sum_{i=t}^{\infty} \alpha_i^2 < \infty$, so α may start anywhere along the sequence. That is, α may take successive values $\alpha_t, \alpha_{t+1}, \alpha_{t+2} \ldots$ (see Theorem A.2).

2.2.2 Discussion

Q-learning is an attractive method of learning because of the simplicity of the computational demands per timestep, and also because of this proof of convergence to a global optimum, avoiding all local optima. One catch though is that the world has to be a Markov Decision Process. As we shall see, even interesting artificial worlds are liable to break the MDP model.

Another catch is that convergence is only guaranteed when using lookup tables, while the statespace may be so large as to make discrete lookup tables, with one cell for every combination of state and action, require impractically large amounts of memory. In large statespaces one will want to use some form of generalisation, but then the convergence proof no longer holds.

Finally, even if the world can be approximated by an MDP, and our generalisation can reasonably approximate lookup tables, the policy that Q-learning finds may be surprising. The optimal solution means *maximising the rewards*, which may or may not actually solve the problem the designer of the rewards had in mind. The agent may find a policy that maximizes the rewards in unexpected and unwelcome ways (see [Humphrys, 1996, §4.1] for an example). Still, the promise of RL is that designing reward functions will be easier than designing behavior. We will return to the theme of designing reward functions in §4.3.1 and §4.4.3.

Also note that Q-learning's infinite number of (x,a) visits can only be approximated. This is not necessarily a worry because the important thing is not the behavior and Q-values at infinity, but rather that in finite time it quickly gets down to focusing on one or two actions in each state. We usually find that a *greedy* policy (execute the action with the highest Q-value) is an optimal policy long before the Q-values have converged to their final values.

2.2.3 The control policy

In real life, since we cannot visit each (x, a) an infinite number of times, and then exploit our knowledge, we can only approximate Q-learning. We could do a large finite amount of random exploration, then exploit our knowledge. In fact this is the simple method I use on the smaller Q-learning statespaces in this work (§4.4.1).

On the larger-statespace Q-learning problems, random exploration takes too long to focus on the best actions, and also in a generalisation (§4.3.2) it will cause a huge amount of noise, so instead I use a method that interleaves exploration and exploitation. The idea is to start with high exploration and decrease it to nothing as time goes on, so that after a while we are only exploring (x, a)'s that have worked out at least moderately well before.

The specific control policy used is a standard one in the field and originally comes from [Watkins, 1989] and [Sutton, 1990]. The agent tries out actions probabilistically based on their Q-values using a Boltzmann or *soft max* distribution. Given a state x, it tries out action a with probability:

$$p_x(a) = \frac{e^{\frac{Q(x,a)}{T}}}{\sum_{b \in A} e^{\frac{Q(x,b)}{T}}}$$

Note that $e^{\frac{Q(x,a)}{T}} > 0$ whether the Q-value is positive or negative, and that $\sum_a p_x(a) = 1$. The *temperature* T controls the amount of exploration (the probability of executing actions other than the one with the highest Q-value). If T is high, or if Q-values are all the same, this will pick a random action. If T is low and Q-values are different, it will tend to pick the action with the highest Q-value.

At the start, Q is assumed to be totally inaccurate, so T is high (high exploration), and actions all have a roughly equal chance of being executed. T decreases as time goes on, and it becomes more and more likely to pick among the actions with the higher Q-values, until finally, as we assume Q is converging to Q^*, T approaches zero (pure exploitation) and we tend to only pick the action with the highest Q-value:

$$p_x(a) = \begin{cases} 1 & \text{if } Q(x,a) = \max_{b \in A} Q(x,b) \\ 0 & \text{otherwise} \end{cases}$$

That is, the agent acts according to a stochastic policy which as time goes on becomes closer and closer to a deterministic policy.

In fact, while I use simple random exploration (no Boltzmann) to learn the Q-values on the small statespaces (§4.4.1), when it came to *testing* the result instead of using a deterministic strict highest Q I used a low temperature Boltzmann. Strict determinism can lead to a creature with brittle behavior. If there are a number of more-or-less equally good actions with very little difference in their Q-values, we should probably rotate around them a little rather than pick the same one every time.

2.2.4 Special case - Absorbing states

An absorbing state x is one for which, $\forall a$:

$$\begin{array}{l} P_{xa}(x) = 1 \\ P_{xa}(y) = 0 \quad \text{if} \quad y \neq x \end{array}$$

Once in x, the agent can't leave. The problem with this is that it will stop us visiting all (x, a) an infinite number of times. Learning stops for all other states once the absorbing state is entered, unless there is some sort of artificial reset of the experiment.

In a real environment, where x contains information from the senses, it is hard to see how there could be such thing as an absorbing state anyway, since the external world will be constantly changing irrespective of what the agent does. It is hard to see how there could be a situation in which it could be relied on to supply the same sensory data forever.

No life-like artificial world would have absorbing states either (the one I use later certainly does not).

Chapter 3

Multi-Module Reinforcement Learning

In general, Reinforcement Learning work has concentrated on problems with a single goal. As the complexity of problems scales up, both the size of the statespace and the complexity of the reward function increase. We will clearly be interested in methods of breaking problems up into subproblems which can work with smaller statespaces and simpler reward functions, and then having some method of combining the subproblems to solve the main task.

Most of the work in RL either designs the decomposition by hand [Moore, 1990], or deals with problems where the sub-tasks have termination conditions and combine sequentially to solve the main problem [Singh, 1992, Tham and Prager, 1994].

The Action Selection problem essentially concerns subtasks acting in parallel, and interrupting each other rather than running to completion. Typically, each subtask can only ever be *partially* satisfied [Maes, 1989].

3.1 Hierarchical Q-learning

Lin has devised a form of multi-module RL suitable for such problems, and this will be the second method tested below.

Lin [Lin, 1993] suggests breaking up a complex problem into sub-problems, having a collection of Q-learning agents $A_1, \ldots, A_n$ learn the sub-problems, and then have a single controlling Q-learning agent which learns $Q(x, i)$, where i is which agent to choose in state x. This is clearly an easier function to learn than $Q(x, a)$, since the sub-agents have already learnt sensible actions. When the creature observes state x, each agent A_i suggests an action

a_i. The switch chooses a winner k and executes a_k.

Lin concentrates on problems where subtasks combine to solve a global task, but one may equally apply the architecture to problems where the subagents simply compete and interfere with each other, that is, to classic action selection problems.

Chapter 4

The House Robot problem

We will test the action selection methods in use in the hypothetical world of a 'house robot'. The house robot is given a range of multiple parallel and conflicting goals and must partially satisfy them all as best as it can. We will test Q-learning and Hierarchical Q-learning in this world, and then introduce a number of new methods, implementing and testing each one as it is introduced.

Consider what kind of useful systems might have multiple parallel, partially-satisfied, non-terminating goals. Inspired by a familiar such system, the common household dog, I asked the question: What could an autonomous mobile robot do in the home?

Consider that the main fears of any household are (a) fire, (b) burglary and (c) intruders/attackers. These all tend to happen because there is only one or no people at home or the family is asleep. At least, none of these things would happen if there were enough alert adults wandering round all the time.

So in the absence of enough alert adults, how about an alert child's toy? Even if about all a small mobile robot could do was cover ground and look at things, it might still be useful. In this hypothetical scenario, the robot would be a wandering security camera, transmitting pictures of what it saw to some remote mainframe. Imagine it as a furry, big-eyed, child's toy, with no weapons except curiosity. The intruder breaks into the house and the toy slowly wanders up to him and looks at him, and that's it. There's no point in him attacking it since his picture already exists somewhere remote. If the continuous signal from the robot is lost, some human will come and investigate. The house robot could also double as a mobile wandering smoke alarm, and default perhaps to a vacuum cleaner when nothing was happening.

Microworlds have (very often justifiably) a bad press, so I must state from the outset that I am not interested in realism but in producing a *hard* *action selection problem*, and also one that is different from the normal ones encountered say in animal behavior. A more detailed defence of this problem follows shortly (§4.1).

In the artificial grid-world of Figure 4.1, the positions of entrances and internal walls are randomised on each run. Humans are constantly making random crossings from one entrance to another. The creature (the house robot) should avoid getting in the way of family, but should follow strangers. It must go up close first to identify the human as family or stranger. Dirt trails after all humans. The creature picks up dirt and must occasionally return to some base to re-charge and empty its bag. Fire starts at random and then grows by a random walk. The creature puts out the fire on a square by moving onto it. The world is not a torus - the creature is blocked by inner walls and outer walls and also can't leave through the entrances.[1]

The creature can only detect objects in a small radius around it and can't see through walls - with the exception of smoke, which can be detected even if a wall is in the way. Each time step, the creature senses state $x = (d, i, p, w, h, c, f, w_f)$ where:

- d is the direction (but not distance) of the nearest visible dirt, and takes values 0-7 (the primary and secondary compass directions, see Figure 4.2), 8 (when dirt is on the same square) and 9 (no dirt visible within a small radius).

- i is whether the vacuum bag is full and needs emptying, and takes values 0 and 1.

- p (0-9) is the direction of the plug.

- w (0-9) is the direction of the nearest visible wall.

- h (0-9) is the direction of the nearest visible human.

- c is the classification of the human, taking values 0 (no current classification), 1 (known member of family) and 2 (stranger).

[1]**Problem Details -** Every 3 timesteps, the human drops a piece of dirt in its vicinity, up to a maximum amount of dirt on the grid of 5. What tends to happen is the amount of dirt usually stays up at 5, and every time the creature picks up a piece, the human will soon drop another. The creature can only carry 10 pieces of dirt before it has to return to base to empty its bag. The world is a 15 x 15 grid with walls all around the edge (broken by 3 entrances), and two randomly-placed internal walls. The topology is randomised every 700 timesteps. The probability of a fire starting is $\frac{1}{70}$ per timestep. Every 10 timesteps, an existing fire grows by a random walk of 1 square.

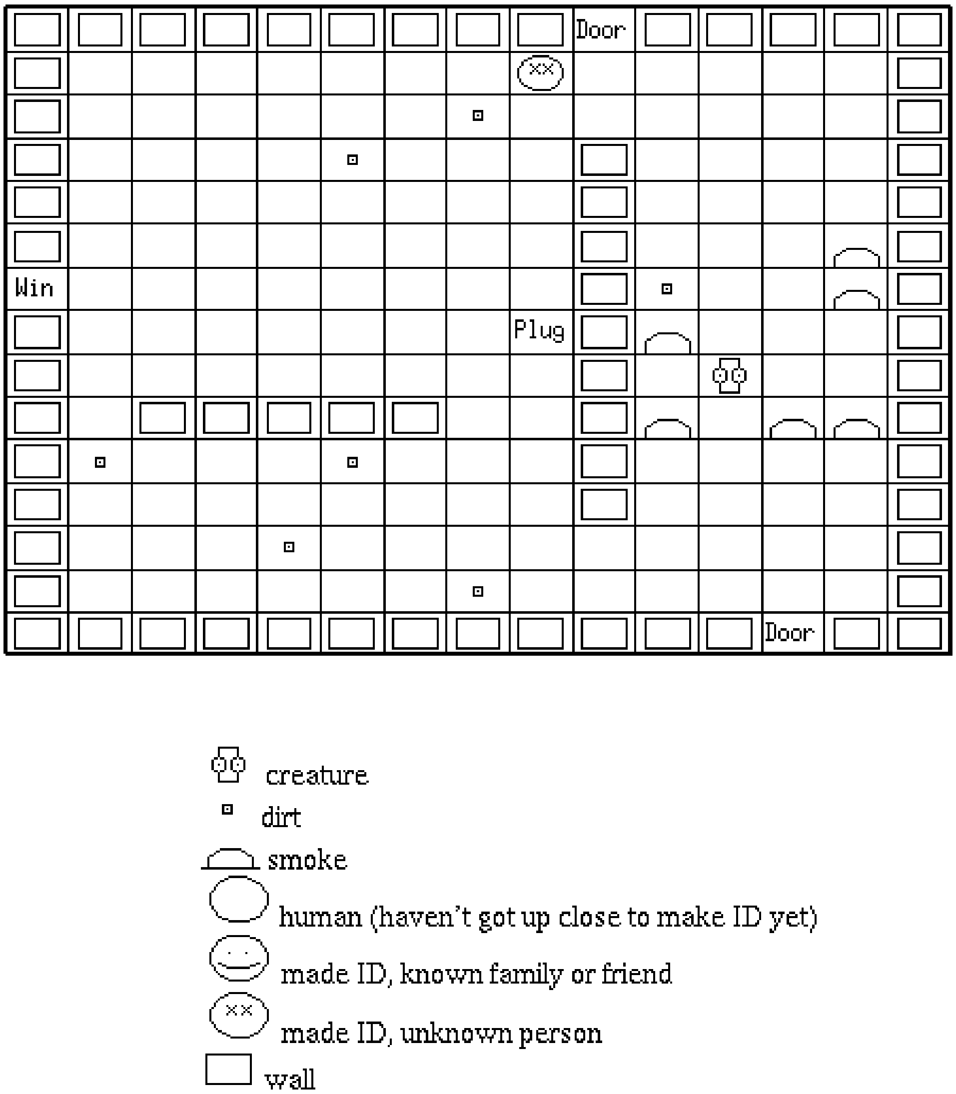


Figure 4.1: The House Robot problem. Here, the building is on fire, dirt is scattered everywhere, and a stranger is in the house.

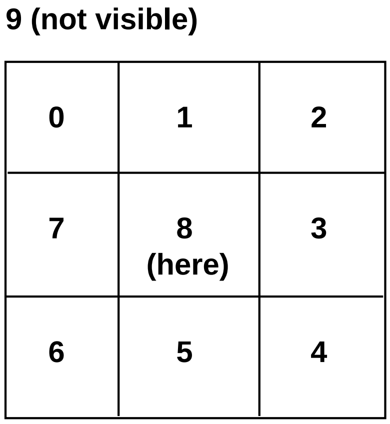


Figure 4.2: The creature senses the relative direction of things within a small radius around it.

- f (0-9) is the direction of the nearest visible smoke.
- w_f is whether the smoke is being detected through a wall, and takes values 0 and 1.

The creature takes actions a, which take values 0-7 (move in that direction) and 8 (stay still).

The fact that its senses are limited, and that there are other, unpredictably behaving entities in the world, means that we must give up on being able to consistently *predict* the next state of the world. For example, AI planning techniques that assume a static world would be of limited use. The creature must be tightly coupled to the world by continuous observation and reaction.

Given its limited sensory information, the creature needs to develop a reactive strategy to put out fire, clean up dirt, watch strangers, and regularly return to base to re-charge. When we specify precisely (§4.3.1) what we want, we find that the optimum is not any strict hierarchy of goals. Rather some interleaving of goals is necessary, with different goals partially satisfied on the way to solving other goals. Such goal-interleaving programs are difficult to write and make good candidates for learning.

4.1 Simulations and Artificial Worlds

This artificial world was not constructed to be a *simulation* of a robot environment. There is no attempt to simulate realistic robot sensors. There is no explicit simulated noise, though we do have limited and confusing sensory information. All I am interested in is setting up a hard action selection problem. And it is hard, as we shall see.

Tyrrell [Tyrrell, 1993, §1.1] defends the value of such microworld experiments in relation to the Action Selection problem at least. At the moment, it is difficult enough to set up a robotics experiments with a single goal. It is still harder to set up multiple conflicting goals and pose decent action selection problems to them.

We can see how far this is from actual robotics. It is unrealistic to think that the robot could reliably *classify* objects in the world as 'dirt', 'wall' etc, based on its senses. In fact, it seems to imply that the job of perception is to translate sensory data into symbolic tags. In fact, I make no such claim. I am merely trying to construct a hard action selection problem in a way that we can think about it easily, understand its difficulty, and suggest possible strategies.

Similarly, the *actions* here (of moving in a particular direction) may actually need to be implemented using lower-level mechanisms. Again, the object is to concentrate on the higher-level problems of the overall strategy.

Since the methods we will test take as input any vector of numbers, and can produce any vector as output, they can be reapplied afterwards to other problems with more realistic sensory and motor data. RL has been applied e.g. by [Mataric, 1994] to multiple, real autonomous mobile robots. I make no stronger assumptions about the world than RL does, so if RL can be applied to real robots then so can my work.

Also this work is not just about robotic or animal-like behavior problems. We are addressing the general issue of a system with multiple conflicting parallel goals. The model could be applied, for example, to such problems as lift scheduling, intelligent building control or Internet agents (see [Charpillet et al., 1996]).

4.1.1 Separation of world from user interface

Much time is often spent on the visual user interface of such artificial worlds. But the only important thing is the actual problem to be solved.

In my implementation, I rigorously separate the essential part of the program, the equations to be solved to represent interaction with the world, from the unessential part, any visual representation showing on-screen what is happening. As a result, I can do long learning runs in hidden, overnight batch jobs, being just number-crunching with no user interface at all.

Both the artificial world and the learning algorithms here were implemented in C++.

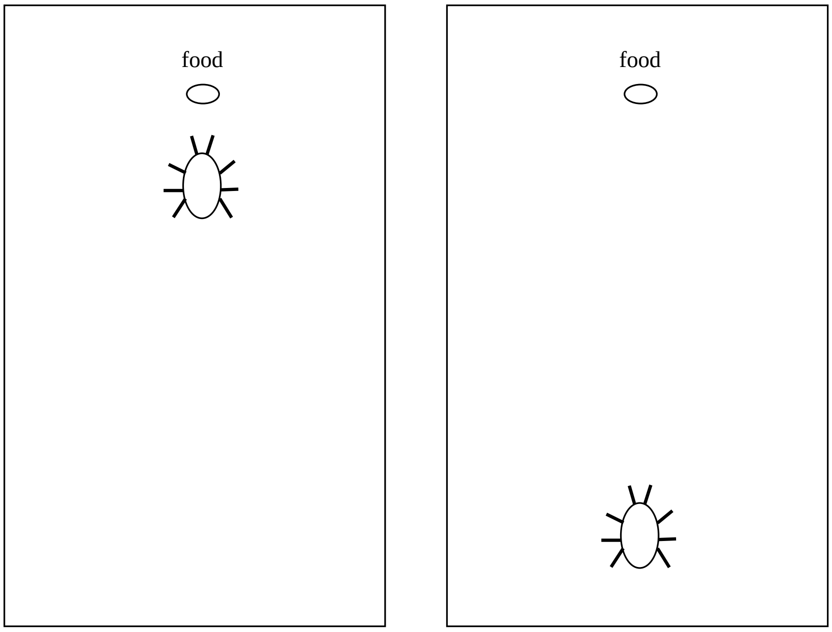


Figure 4.3: Probabilistic transitions $P_{xa}(y)$.

4.2 Notes on MDP

Note that sensing direction but not distance means that features of the world that are deterministic can actually appear probabilistic to the agent. This is illustrated in Figure 4.3. When in the situation on the left, the agent will experience: $x =$ (food is North), $a =$ (move North), $y =$ (got food). When in the situation on the right, the agent will experience: $x =$ (food is North), $a =$ (move North), $y =$ (food is North).

In fact, as shown in Figure 4.4, the world is not exactly an MDP. (x, North) leads to state y, and $(x, \text{NorthEast})$ also leads to state y. But the probability of (y, East) leading to an immediate reward depends on the agent's policy. If it tends to take action $(x, \text{NorthEast})$, the value of y will be higher than if it tends to take action (x, North). But it will not necessarily *learn* to take action $(x, \text{NorthEast})$ as a result - that would just be normal learning, where it can tell the states apart. (x, North) is just as likely a policy because (x, North) is *also* rewarded when the value of y increases.

$P_{xa}(r)$ is non-stationary. In fact, the artificial world is a *Partially-Observable MDP* [Singh et al., 1994]. But it is very much a non-malicious one, and can be reasonably approximated as a probabilistic MDP - this will be proved shortly by the success of RL techniques when compared against hand-coded solutions in this world.

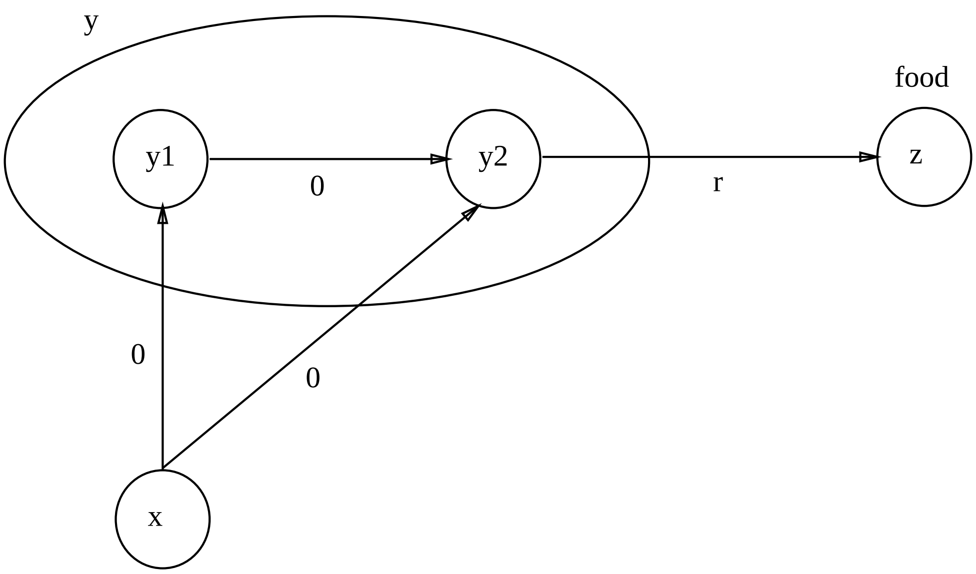


Figure 4.4: Non-stationary probabilistic transitions $P_{xa}(r)$. The states' positions in the diagram correspond to their geographical positions. x is the state 'food is EastNorthEast'. y is the state 'food is East'. The agent sees both $y1$ and $y2$ as the same state y. The quantities on the arrows are the rewards for that transition.

4.3 Test of Q-learning

The first method we apply in this world is a single monolithic Q-learning agent learning from rewards.

4.3.1 Designing a Global reward function

Reinforcement Learning is attractive because it propagates rewards into behavior, and presumably reward functions (value judgements) are easier to design than behavior itself. Even so, designing the global reward function here is not easy (see [Humphrys, 1996, §4.1] for an example of accidentally designing one in which the optimum solution was to jump in and out of the plug non-stop). Later (§8.1.3) we will ask if we can avoid having to design this explicitly:

```
reward for single step from x to y
 points := 0
 once-off type scores:
  if (got in way of family) subtract 1 point
  if (picked up dirt)            add 1 point
  if (put out fire)              add 5 points
 continuous-type scores:
  if (arrived at plug)             add 0.1 points
  if (stranger exists unseen) subtract 0.1 points
  if (fire exists)            subtract 0.1 points
  if (fire is large)          subtract 0.5 points
 return points
```

Note that our global reward function, in looking at things from the house's point of view, actually refers to information that may not be in the state x, e.g. 'fire exists' causes punishment even if the creature cannot see the fire. From the creature's point of view, this is a stochastic global reward function, punishing it at random sometimes for no apparent reason. Such a stochastic reward function is not disallowed under our Markov framework provided that $P_{xa}(r)$ is stationary.

4.3.2 Neural network implementation

The number of possible states x is 1.2 million, and with 9 possible actions we have a state-action space of size 10.8 million. To hold each $Q(x, a)$ explicitly as a floating point number, assuming 4 byte floats, would therefore require 40 M of memory, which on my machine anyway was becoming impractical both in terms of allocating memory and the time taken to traverse it. On many machines, 40 M is not impractical, but we are *approaching* a limit. Just add two more objects to the world to be sensed and we need 4000 M memory.

In any case, perhaps an even more serious problem is that we have to *visit* all 10.8 million state-action pairs a few times to build up their values if data from one is not going to be generalized to others. So for reasons of both memory and time, instead of using lookup tables we need to use some sort of generalisation - here, multi-layer neural networks.[2]

Following Lin [Lin, 1992], because we have a small finite number of actions we can reduce interference by breaking the state-action space up into one network per action a. We have 9 separate nets acting as function approximators. Each takes a vector input x and produces a floating point output $Q_a(x) = Q(x, a)$. Each net is responsible for a different action a. This also allows us to easily calculate the $\max_{b \in A} Q(y, b)$ term - just enumerate the actions and get a value from each network.[3]

We also note that, as in [Rummery and Niranjan, 1994], although we have a large statespace, each element of the input vector x takes only a small number of discrete values. So instead say of one input unit for (d) taking values 0-9, we can have 10 input units taking values 0 or 1 (a single unit will be set to 1, all the others set to 0). This makes it easier for the network to identify and separate the inputs. Employing this strategy, we represent all possible inputs x in 57 input units which are all binary 0 or 1. Also like [Rummery and Niranjan, 1994], we found that a small number of hidden units (10 here) gave the best performance. To summarise, we have 9 nets. Each has 57 input units, 10 hidden units, and a single output unit.

As [Tesauro, 1992] notes, learning here is not like ordinary supervised learning where we learn from (Input,Output) exemplars. Here we're not presenting constant $(x, Q^*(x))$ exemplars to the network but instead we are learning from *estimates* of Q^*:

$$Q_a(x) \mapsto (r + \gamma \max_{b \in A} Q_b(y))$$

where the $Q_b(y)$ value is an estimate, and may come from a different

[2]The same would hold if our input x was a vector of *real numbers*. In this case we could also use neural networks but they would be of a different architecture to the ones described here. Note also that in this case we effectively *can't* see the same state twice.

[3]If we had a large action space, the actions would need to be generalised too, e.g. we could have a single neural net with vector input x, a and a single floating point output $Q(x, a)$. The problem then is to calculate the $\max_{b \in A} Q(y, b)$ term - we can't enumerate the actions. See [Kaelbling et al., 1996, §6.2] for a survey of generalising action spaces.

network to the $Q_a(x)$ estimate. We are telling the network to update in a certain direction. Next time round we may tell it to update in a different direction. That is, we need to repeatedly apply the update as the estimate $Q_b(y)$ improves. The network needs a *lot* of updates to learn, but that doesn't mean we need a lot of different interactions with the world. Instead we repeatedly apply the same set of examples many times. Also, we don't want to replay an experience multiple times *immediately*. Instead we save a number of experiences and mix them up in the replay, each time factoring in new $Q(y)$'s.

Our strategy roughly follows [Lin, 1992]. We do 100 trials. In each trial we interact with the world 1400 times, remember our experiences, and then *replay* the experiences 30 times, each time factoring in more accurate $Q(y)$ estimates. Like Lin, we use *backward replay* as more efficient (update $Q(y)$ before updating the $Q(x)$ that led to it).

[Lin, 1992] points out doing one huge trial and then replaying it is a bad idea because the actions will all be random. With lookup tables, we can just experience state-action pairs randomly (§2.2.1), but with neural networks, replaying lots of random actions is a bad idea because the update affects the values of other entries. The few good actions will be drowned by the noise of the vast majority of bad actions. A better strategy is to continually update the control policy to choose better actions over time. Doing lots of short trials, replaying after each one, allows our control policy to improve for each trial.

As Lin points out [Lin, 1993], experience replay is actually like building a model and running over it in our head, as in Sutton's DYNA architecture (§2.1.7). Too few real-world interactions and too much replay would lead to *overfitting*, where the network learns that state $x = (2, 0, 7, 3, 4, 1, 7, 1)$ and action $a = (2)$ lead to the picking up dirt reward, when it should learn that state $x = (2, 0, *, *, *, *, *, *)$ and action $a = (2)$ lead to that reward.

Throughout this thesis, for lookup tables we use learning rate $\alpha = 1, \frac{1}{2}, \frac{1}{3}, \ldots$, and start with the tables randomised. For neural networks we use backpropagation learning rate 0.5, and start with all weights small random positive or negative. For the Q-learning throughout we use discounting factor $\gamma = 0.6$.

Adjusting the amount of replay, and the architecture of the network, the most successful monolithic Q-learner, tested over 20000 steps (involving 30 different randomised houses) scored an average of 6.285 points per 100 steps. The coding into 57 input units was the only way to get any decent performance.

4.3.3 Hand-coded programs

As it turns out, 6.285 points per 100 steps is not an optimal policy. Writing a strict hierarchical program to solve the problem, with attention devoted to humans only when there was no fire, and attention devoted to dirt only when there was no fire or humans, could achieve a score of 8.612. When state x is observed, the hand-coded program (the best of a range of such programs) generates action a as follows:

```
if (smoke visible)
{
 if (wall in way)
  stochastic move towards smoke
 else
  move towards smoke
}
else if (human visible)
{
 if (classification = family)
  move in opposite direction to human
 else
  move towards human
}
else if (full)
{
 if (plug visible)
  move towards plug
 else
  random move other than towards wall
}
else
{
 if (dirt visible)
  move towards dirt
 else
  random move other than towards wall
}
```

All strategies, hand-coded or learnt, must deal with the fact that there is no memory. Note that the hand-coded program implements a *stochastic* policy.

So why did Q-learning not find an optimal policy? The answer is because we are not using lookup tables, and we do not have time anyway to experience each full state. If the world was an MDP with lookup tables and we had infinite time, then yes, we couldn't beat ordinary Q-learning (§2.2.1).

But because these conditions don't hold, we can go on and explore better action selection methods than Q-learning. Note that we don't actually know how good the optimum policy will be. As we shall see, we will be able to do a lot better than this, but will still not know at the end of the thesis if we have found the best possible solution.

Clearly, it is difficult to learn such a complex single mapping. We will now look at ways in which the learning problem may be broken up. First we test Hierarchical Q-learning.

4.4 Test of Hierarchical Q-learning

To implement Hierarchical Q-learning in the House Robot problem, we set the following 5 agents to build up personal $Q_i(x,a)$ values. They learn by reference to personal reward functions r_i. The switch then learns $Q(x,i)$ from the global reward function as before.

A_d	senses: (d,i)	
	reward: if (picked up dirt)	1 else 0
A_p	senses: (p)	
	reward: if (arrived at plug)	1 else 0
A_s	senses: (h,c)	
	reward: if (ID=stranger and visible)	1 else 0
A_m	senses: (h,c)	
	reward: if (ID=family and here)	0 else 1
A_f	senses: (f,w_f)	
	reward: if (put out fire)	1 else 0

4.4.1 Q-learning with subspaces

Each agent need only sense the subspace of (d,i,p,w,h,c,f,w_f) that is relevant to its reward function. To be precise, agent A_i need only work with the *minimum* collection of senses required to make $P_{xa}(y)$, $P^i_{xa}(r)$ stationary.[4] There is no advantage to having the agents sense the full space for learning Q. If it is given extra unnecessary senses, it will just learn the same Q-values repeated.

For example, the plug-seeking agent A_p has a reward function $r(x,y) = r(p_t, p_{t+1})$ that only references the location of the plug, so it need only sense (p). Any extra senses will not affect the policy it learns - it will suggest

[4]That is, if the full-statespace transitions were stationary to begin with (recall §4.2).

the same action independent of the values of d, i, w, h, c, f, w_f. It builds up $Q_p((p), a)$ values where:

$$Q_p((p), a) \equiv Q_p((d, i, p, w, h, c, f, w_f), a) \;\; \forall d, i, w, h, c, f, w_f$$

When the creature interacts with the world, each agent translates what is happening into its own subspace. For example (using the notation `x,a -> y`):

```
creature sees (5,0,8,3,1,1,1,0),(5) -> (9,1,1,3,9,1,1,0)

      Ad sees (5,0),(5) -> (9,1)
      Ap sees   (8),(5) -> (1)
      As sees (1,1),(5) -> (9,1)
      Am sees (1,1),(5) -> (9,1)
      Af sees (1,0),(5) -> (1,0)
```

As well as requiring less memory, this builds up the Q-values much quicker. Here, all of the subspaces are small enough to use lookup tables.

The switch still sees the full state x, and its $Q(x, i)$ mapping needs to be implemented as a generalisation. As before in §4.3.2, we reduce interference by breaking the state-action space up into one network per 'action' i. The switch is implemented as 5 neural networks. Each takes a vector input x and produces a floating point output $Q_i(x) = Q(x, i)$. Each net is responsible for a different 'action' i.

We try to keep the number of agents low, since a large number of agents will require a large (x, i) statespace. As before, we go through a number of trials, the switch replaying its experiences after each one.

Here the agents share the same suite of actions a = 0-8, but in general we may be interested in breaking up the action space also.

4.4.2 Learning Q together

The agents can all learn their Q-values together in parallel. The agent A_k that suggested the executed action a_k can update:

$$Q_k(x, a_k) := (1 - \alpha) Q_k(x, a_k) + \alpha (r_k + \gamma \max_{b \in A} Q_k(y, b))$$

If the agents share the same suite of actions (which we don't assume in general) then in fact all other agents can learn at same time. We can update for *all* i:

$$Q_i(x, a_k) := (1-\alpha)Q_i(x, a_k) + \alpha(r_i + \gamma \max_{b \in A} Q_i(y, b))$$

using their different reward functions r_i. We can do this because Q-learning is asynchronous and sampled (we can learn from the single transition, no matter what came before and no matter what will come after).[5]

We must ensure, when learning together, that all agents experience a large number of visits to each of their (x, a). An agent shouldn't miss out on some (x, a) just because other agents are winning. I simply use random winners during this collective Q-learning phase.

4.4.3 Designing Local reward functions

If an agent is of the form:

A_i reward: if (good event) r else s

where $r > s$, then it is irrelevant what r and s actually are, the agent will still learn the same policy (Theorem C.1). So in particular setting $r = 1$ and $s = 0$ here doesn't reduce our options. If we replace 1 above by any number > 0, the agent still learns the same pattern of behavior. It still sends the same preferred action a_i to the switch.

This does not hold for reward functions with 3 or more rewards (see §D.1), where relative difference between rewards will lead to different policies. We mentioned in §2.2.2 that it can be difficult to write a reward function so that maximising the rewards solves your problem. For 3-reward (or more) reward functions such as our global reward function (§4.3.1), experiment quickly shows that it is *not* simply a matter of $r_{\max}$ for all good things and $r_{\min}$ for all bad things.

It can be difficult to predict what behavior maximising the rewards will lead to. For example, because it looks at future rewards, an agent may voluntarily suffer a punishment now in order to gain a high reward later, so simply coding a punishment for a certain transition may not be enough to ensure the agent avoids that transition. Reward functions are the "black art" of Reinforcement Learning, the place where design comes in. RL papers often list unintuitive and apparently arbitrary reward schemes which one realises may be the result of a lengthy process of trial-and-error.

[5]Note that in [Humphrys, 1995, §3] I assumed that agents share the same suite of actions.

To summarise, much attention has been given to breaking up the statespace of large problems, but the reward functions do not scale well either. Multi-reward functions like our global reward function are hard to design. It is much easier to design specialised 2- or 3-reward local functions (these 2-reward functions could not be easier to design: 1 for the good thing and 0 otherwise will do). Hierarchical Q-learning has not got rid of the global reward function, but we shall be attempting to do that later.

With the same replay strategy as before, and the same number of test runs, the Hierarchical Q-learning system scored 13.641 points per 100 steps, a considerable improvement on the single Q-learner. This is also considerably better than the hand-coded program - now we see that the optimum is not any strict hierarchy of goals. Rather some interleaving of goals is necessary.

Again though, Hierarchical Q-learning did not find an optimal policy. We are going to introduce methods which will perform even better. It did not find an optimal $Q^*(x, i)$ policy because again, we are not using lookup tables, and do not have time anyway to experience each full state.

Chapter 5

An abstract decentralised model of mind

Looking at exactly what $Q(x, i)$ are learnt in the previous method, the switch learns things like - if dirt is visible A_d wins because it expects to make a good contribution to the global reward - otherwise if the plug is visible A_p wins because it expects to make a (smaller) contribution. But the agents could have told us this themselves with reference only to their personal rewards. In other words, the agents could have organised themselves like this in the absence of a global reward.

In this chapter I consider how agents might organise themselves sensibly in the absence of a global reward. Watkins in his PhD [Watkins, 1989] was interested in learning methods that might plausibly take place within an animal, involving a small number of simple calculations per timestep, and so on. Similarly, I am looking for more biologically-plausible action selection, something that could plausibly self-organize in the development of a juvenile, that would not require an all-knowing homunculus. Like Watkins, I will propose methods deriving from this motivation rather than trying to copy anything seen in nature.

The starting point for this exploration of decentralised minds is Rodney Brooks' contrast in [Brooks, 1986] between the traditional, vertical AI architecture (Figure 5.1) where representations come together in a central 'reasoning' area, and his horizontal subsumption architecture (Figure 5.2) where there are multiple paths from sensing to acting, and representations used by one path may not be shared by another.

Brooks' method is to build full working robots at each stage. He builds in layers: layer 1 is a simple complete working system, layers 1-2 together form a complete, more sophisticated system, layers 1-3 together form a complete, even more sophisticated system, and so on. Lower layers do not depend on the existence of higher layers, which may be removed without problem. Higher layers may interfere with the data flow in layers below them - they may 'use' the layers below them for their own purposes. To be precise, each layer continuously generates output unless *inhibited* by a higher layer. *Subsumption* is where a higher layer inhibits output by replacing it with its own signal. If a higher layer doesn't interfere, lower layers just run as if it wasn't there.

Brooks' model develops some interesting ideas:

- The concept of default behavior. e.g. the 'Avoid All Things' layer 1 takes control of the robot by default whenever the 'Look For Food' layer 2 is idle. Whenever a higher layer is idle, there is always someone lower down willing to take over.

- Multiple parallel goals. There are multiple candidates competing to be given control of the robot, e.g. control could be given to layer 1, which has its own purposes, or to layer 5, which has different purposes (and may use layers 1-4 to achieve them). Which to give control to may not be an easy decision - we may find it hard to rank them in a strict hierarchy. Multiple parallel goals are seen everywhere in nature, e.g. the conflict between feeding and vigilance in any animal with predators.

- The concept of multiple independent channels connecting sensing to action. Brooks breaks with the traditional, vertical AI architecture of having a 'perception' subsystem, whose job it is to deliver a representation of the world to some central symbolic module where all the 'real' intelligence resides. Instead, he has multiple sensing-to-action channels, working in parallel, each processing sensory information in different ways for its own purposes, some crude, some sophisticated. Each layer may live in an entirely different sensory world.

The logical development of this kind of decentralisation is the model in Figure 5.3. Here, the layers (which we now call agents) have become peers, not ranked in any hierarchy. Each can function in the absence of *all* the others. Each agent is connected directly from senses to action and will drive the body in a pattern of behavior if left alone in it. Each is frustrated by the presence of other agents, who are trying to use the body to implement *their* plans. Brooks' hierarchy would be a special case of this where higher-level agents happen to always win when they compete against lower-level ones.

This is actually the model that Lin's Hierarchical Q-learning used. Each agent must send its actions to a switch which will either obey or refuse it. In Hierarchical Q-learning, the switch is complex and what is sent is simple

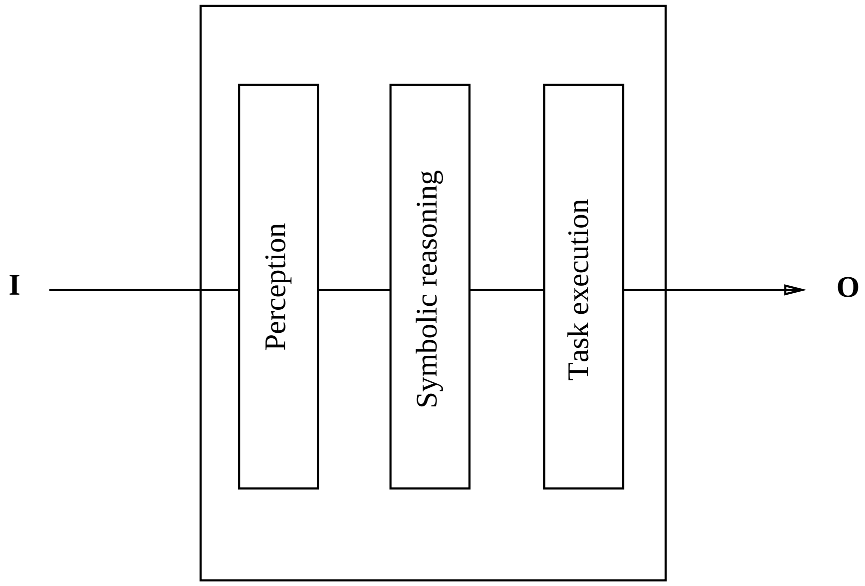


Figure 5.1: The traditional, vertical AI architecture. The central intelligence operates at a symbolic level.

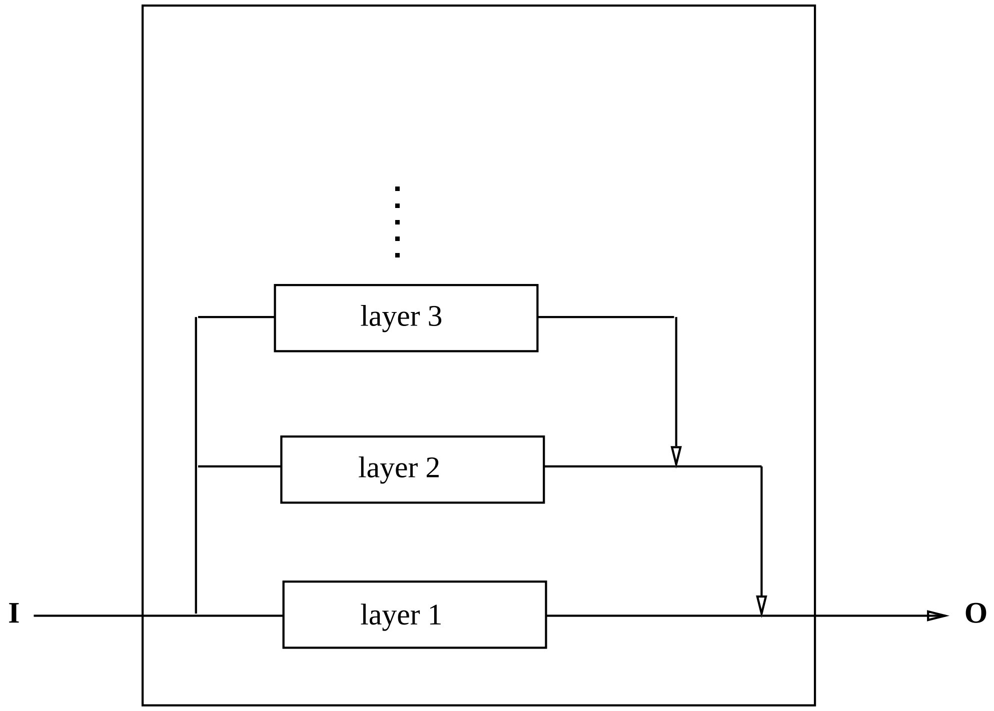


Figure 5.2: Brooks' horizontal subsumption architecture.

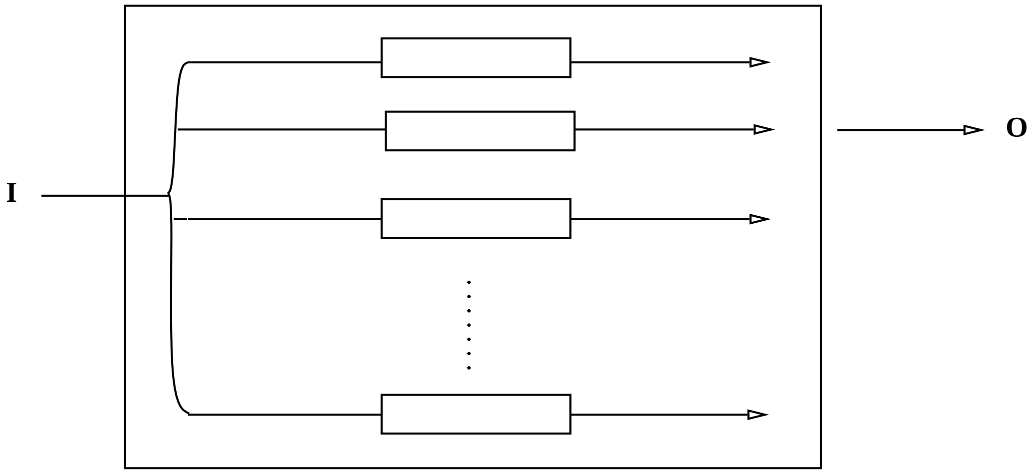


Figure 5.3: Competition among peer agents in a horizontal architecture. Each agent suggests an action, but only one action is executed. Which agent is obeyed changes dynamically.

(a constant action $a_i(x)$). Now we ask if we can make the switch simple, and what is sent more complex. We ask: Rather than having a smart switch organise the agents, Can the agents organise themselves off a dumb switch?

5.1 The weight W

To answer this, I tried to look at it from an agent's point of view. Agents can't know about the global system, or the mission of the creature as a whole. Each must live in a local world. Here agents have no explicit knowledge of the existence of any other agents. Each, if you like, believes itself to be the entire nervous system.[1]

The basic model is that when the creature observes state x, the agents suggest their actions with numeric strengths or *Weights* $W_i(x)$ (I call these W-values). The switch in Figure 5.3 becomes a simple gate letting through the highest W-value. To be precise, the creature contains agents $A_1, \ldots, A_n$. In state x, each agent A_i suggests an action a_i. The switch finds the winner k such that:

[1]Perhaps there is some evolutionary plausibility in this. Consider that the next step after getting a simple sensor-to-effector link working is not a hierarchy or a co-operative architecture but rather a mutation where, by accident, two links are built, and each of course tries to work the body as if it were alone.

$$W_k(x) = \max_{i \in 1,\ldots,n} W_i(x)$$

and the creature executes a_k. We call A_k the *leader* in the competition for state x at the moment, or the *owner* of x at the moment. The next time x is visited there might be a different winner.

This model will work if we can find some automatic way of resolving competition so that the 'right' agent wins. The basic idea is that an agent will always have an action to suggest (being a complete sensing-and-acting machine), but it will 'care' some times more than others. When no predators are in sight, the 'Avoid Predator' agent will continue to generate perhaps random actions, but it will make no difference to its purposes whether these actions are actually executed or not. When a predator does come into sight however, the 'Avoid Predator' agent must be listened to, and given priority over the default, background agent, 'Wander Around Eating Food'.

For example, in Figure 5.4, when the creature is not carrying food, the 'Food' agent tends to be obeyed. When the creature is carrying food, the 'Hide' agent tends to be obeyed. The result is the adaptive food-foraging creature of Figure 5.5.

Note that the 'Hide' agent goes on suggesting the same action with the same W in all situations. It just wants to hide all the time - it has no idea why sometimes it is obeyed, and other times it isn't.

We can draw a map of the state-space, showing for each state x, which agent succeeds in getting its action executed. Clearly, a creature in which one agent achieves total victory (wins all states) is not very interesting. It will be no surprise what the creature's behavior will be then - it will be the behavior of the agent alone. Rather, interesting creatures are ones where the state-space is fragmented among different agents (Figure 5.6).

In fact, with the agents we will be using, creatures entirely under the control of one agent will be quite unadaptive. Only a collection of agents will be adaptive, not any one on its own (see §7.3 later).

One of the advantages of Brooks' model is that layer 2, for example, does not need to re-implement its own version of the wandering skill implemented in layer 1 - it can just allow layer 1 be executed when it wants to use it. Similarly here, even though agents have their own entire plan for driving the creature (for every x they can suggest an a), they might still come to depend on each other. An agent can *use* another agent by learning to cede control of appropriate areas of state-space (which the other agent will take over).

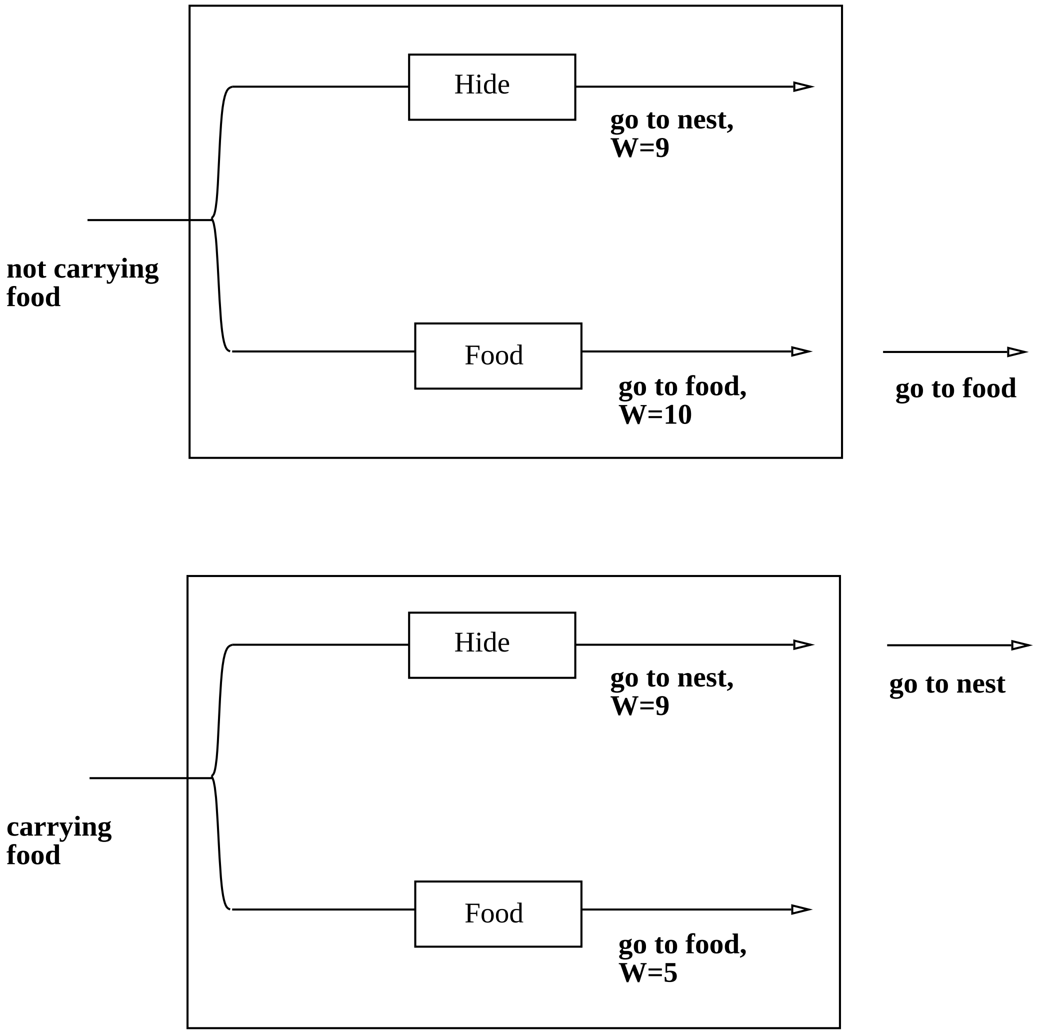


Figure 5.4: Competition is decided on the basis of W-values. The action with the highest W-value is executed by the creature.

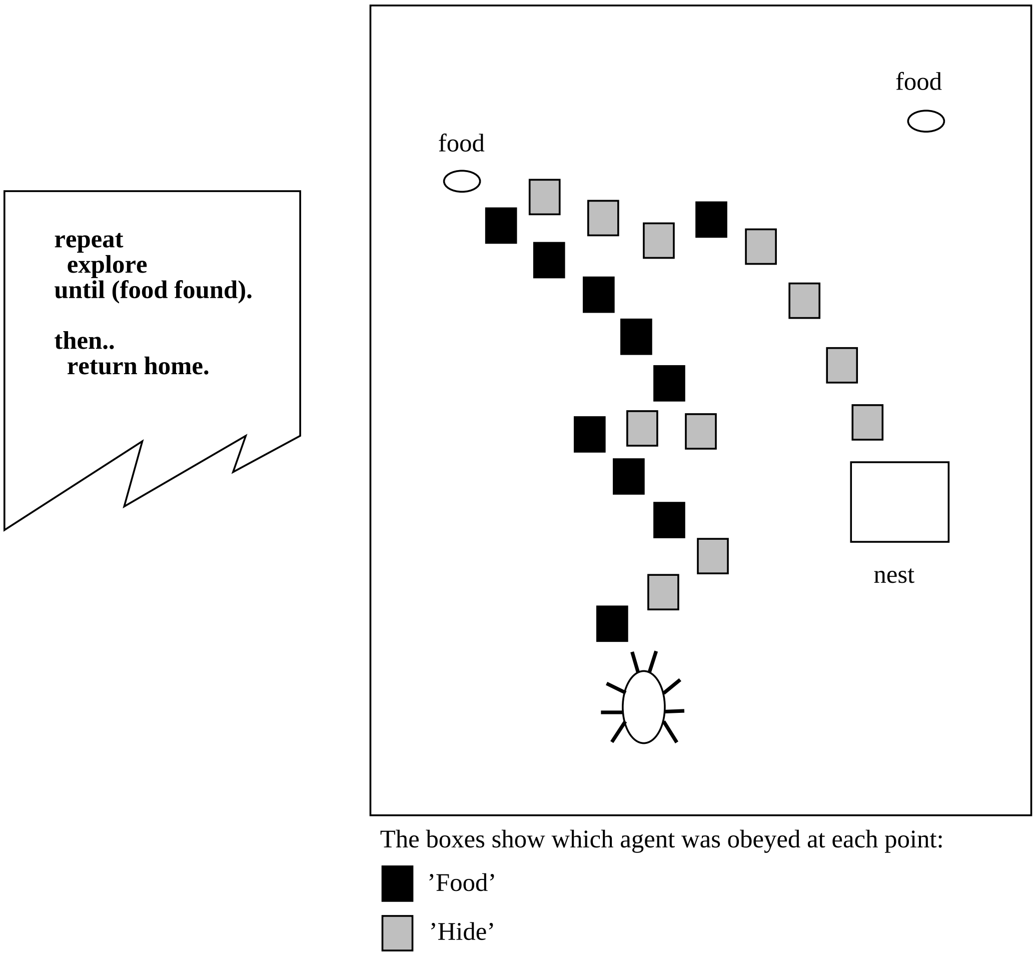


Figure 5.5: Internal conflict, adaptive external behavior: the conflict between the agents 'Hide' and 'Food' results in an adaptive food-forager. On the left is the kind of global, serial, control program one would normally associate with such behavior. On the right, the two competing agents produce the same effect without global control.

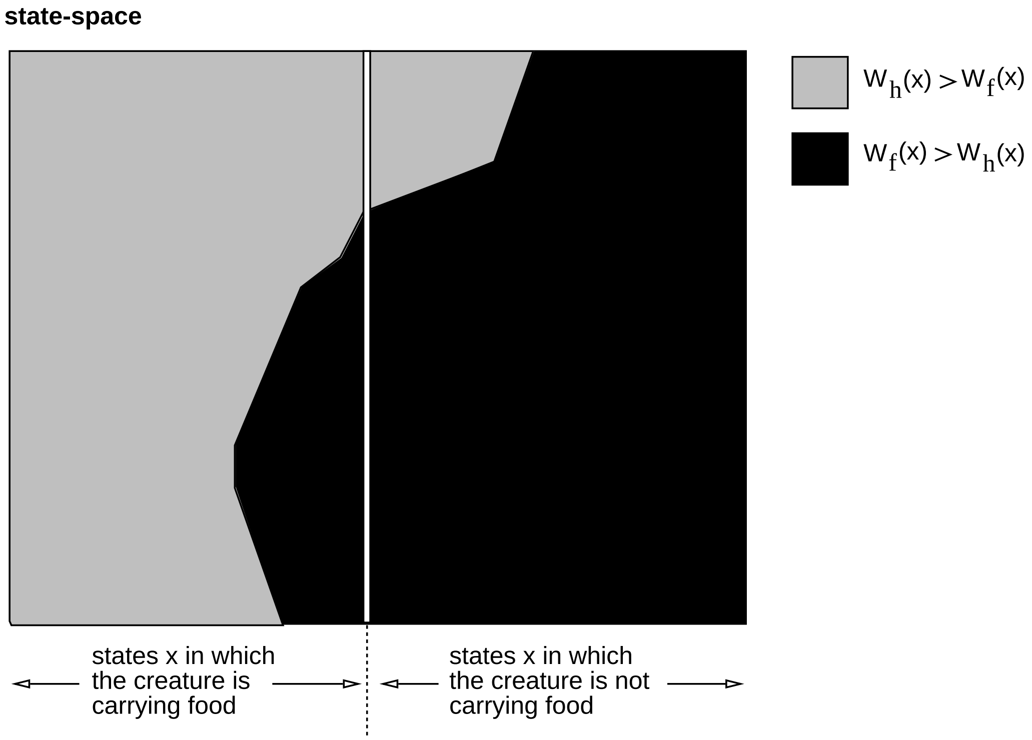


Figure 5.6: We expect competition to result in fragmentation of the state-space among the different agents. In each state x, the 'Hide' agent suggests some action with weight $W_h(x)$, and the 'Food' agent suggests an action with weight $W_f(x)$. The grey area is the area where $W_h(x) > W_f(x)$. The 'Hide' agent wins all the states in this area. The black area shows the states that the 'Food' agent wins.

5.2 W = Drive Strength

Our abstract decentralised model of mind is a form of the *drives* model commonly used in ethology (dating back to Hull's work in the 1940s), where the 'drive strength' or 'importance variable' [Tyrrell, 1993] is equivalent to the W-value, and the highest one wins (the exact action of that agent is executed). Tyrrell's equation [Tyrrell, 1993, §8.1]:

```
drive_strength
  = stimulus_strength
  = f(internal, external and indeterminate stimuli)
```

is equivalent to saying that:

$$W = W(x)$$

It is drive strength *relative to the other drives* that matters rather than absolute drive strength.

The big question of course is where do these drive strengths or W-values come from. Do they have to all be designed by hand? Or can they be somehow learned? Ethologists have lots of ideas for what the weights should represent, but can't come up with actual algorithms that are guaranteed to produce these kind of weights automatically. They tend to just leave it as a problem for evolution, or in our case a designer. See [Tyrrell, 1993, §4.2,§9.3.1] for the difficult design problems in actually implementing a drives model. Tyrrell has the problem of designing sensible drive strengths, relative drive strengths, and further has to design the appropriate action for each agent to execute if it wins.

It should be clear by now that we are going to use Reinforcement Learning to automatically generate these numbers. RL methods, in contrast to many forms of machine learning, build up value functions for actions. That is, an agent not only knows 'what' it wants to do, it also knows 'how much' it wants to do it. Traditionally, the latter are used to produce the former and are then ignored, since the agent is assumed to act alone. But the latter numbers contain useful information - they tell us how much the agent will suffer if its action is not executed (perhaps not much). They tell us which actions the agent can compromise on and which it cannot.

Systems that only know 'What I want to do' find it difficult to compromise. Reinforcement Learning systems, that know 'How much I want to do it', are all set up for negotiating compromises. In fact, 'how much' it wants to take an action is about *all* the simple Q-learning agent knows - it may not even be able to explain *why* it wants to take it.

Our Reinforcement Learning model is also useful because given *any* state x at any time a state-space agent can suggest an action a - as opposed to programmed agents which will demand preconditions and also demand control for some time until they terminate. State-space agents are completely interruptable. In the terminology of [Watkins, 1989, §9], the Action Selection methods I propose in this thesis are *supervisory* methods, where there is a continuous stream of commands interrupting each other, rather than *delegatory* methods, where a top level sends a command and then is blocked waiting for some lower level loop to terminate. Watkins argues that the latter methods characterise machines (or at least, machines as traditionally conceived) while the former characterise what animals actually do.

Recently there has been an emphasis on autonomous agents as *dynamical systems* (e.g. [Steels, 1994]), emphasising the continuous interaction between the agent and its world such that the two cannot be meaningfully separated. It is not absolutely clear what the defining features of a dynamical system are (over and above just any self-modifying agent), but complete interruptability seems to be one part of it.

To formalise, each agent in our system will be a Q-learning agent, with its own set of Q-values $Q_i(x, a)$ and more importantly, with its own reward function. Each agent A_i will receive rewards r_i from a personal distribution $P^i_{xa}(r)$. The distribution $P_{xa}(y)$ is a property of the world - it is common across all agents. Each agent A_i also maintains personal W-values $W_i(x)$. Given a state x, agent A_i suggests an action a_i according to some control policy (§2.2.3), which is most likely, as time goes on, to be such that:

$$Q_i(x, a_i) = \max_{b \in A} Q_i(x, b)$$

The creature works as follows. Each time step:

observe x
for (all agents):
  get suggested action a_i with strength $W_i(x)$
find $W_k(x) = \max_{i \in 1,\ldots,n} W_i(x)$
execute a_k
observe y
for (all agents):
  get reward r_i
  possibly update Q_i
  possibly update W_i

Note that the transition will generate a different reward r_i for each agent. For updating Q, we use normal Q-learning. For updating W, we want somehow to make use of the numerical Q-values. There are a number of possibilities.

5.3 Static W=Q

A *static* measure of W is one where the agent promotes its action with the same strength no matter what (if any) its competition. For example:

$$W_i(x) = Q_i(x, a_i)$$

This simple method will be the fourth method we test (§9).

5.4 Static W = importance

A more sophisticated static W would represent the difference between taking the action and taking other actions, i.e. how *important* this state is for the agent. If the agent does not win the state, some other agent will, and some other action will be taken. In an unimportant state, we may not mind if we are not obeyed. The concept of importance is illustrated in Figure 5.7.

For example, W could be the difference between a_i and the worst possible action:

$$W_i(x) = Q_i(x, a_i) - \min_{b \in A} Q_i(x, b)$$

Or W could be a Boltzmann function:

$$W_i(x) = \frac{e^{\frac{Q_i(x,a_i)}{T}}}{\sum_{b \in A} e^{\frac{Q_i(x,b)}{T}}}$$

The idea of this kind of *scaling* is that what might be a high Q-value in one agent might not be a high Q-value to another.

5.4.1 Static W=z

A scaled measure of W could be how many standard deviations a_i's Q-value is from the mean Q-value. That is, W is the z-score. To be precise, the mean would be:

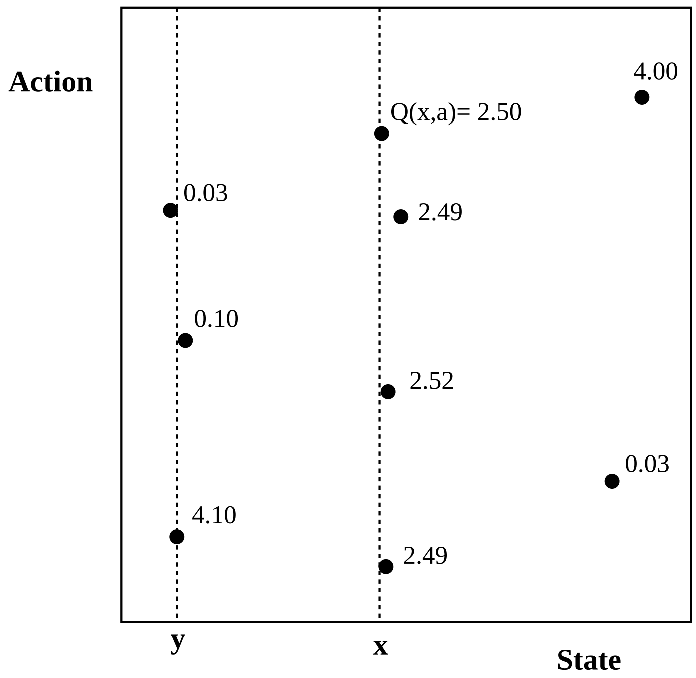


Figure 5.7: The concept of *importance*. State x is a relatively unimportant state for the agent (no matter what action is taken, the discounted reward will be much the same). State y is a relatively important state (the action taken matters considerably to the discounted reward).

$$M = \frac{1}{|A|} \sum_{b \in A} Q_i(x, b)$$

the standard deviation would be:

$$\sigma^2 = \frac{1}{|A|} \sum_{b \in A} (Q_i(x, b) - M)^2$$

and this scheme is:

$$W_i(x) = \begin{cases} 0 & \text{if the Q-values are all the same} \\ \frac{Q_i(x, a_i) - M}{\sigma} & \text{otherwise} \end{cases}$$

If the Q-values are all the same, $\sigma = 0$ and $Q_i(x, a_i) - M = 0$. Otherwise $\sigma > 0$, and remember that a_i is the *best* action, so $Q_i(x, a_i) - M > 0$.

But even this measure may still be a rather crude form of competition. Consider where there are just two actions. Either their Q-values are the same ($W_i(x) = 0$) or else the mean is halfway between them and the larger one is precisely 1 standard deviation above the mean. So to compete in a state where we care what happens, we have $W_i(x) = 1$ always, independent of the values of or the gap between the Q-values.

5.5 Dynamic (learnt) W-values

The real problem with a static measure of W is that it fails to take into account what the other agents are doing. If agent A_i is not obeyed, the actions chosen will not be random - they will be actions desirable to other agents. It will depend on the particular collection what these actions are, but they may overlap in places with its own suggested actions. If another agent happens to be promoting the same action as A_i, then A_i does not need to be obeyed. Or more subtly, the other agent might be suggesting an action which is almost-perfect for A_i, while if A_i's exact action succeeded, it would be disastrous for the other agent, which would fight it all the way.

We have two types of states that A_i need not compete for:

Type 1 - A state which is relatively unimportant to it. It doesn't matter much to A_i's discounted reward what action is taken here. In particular, it doesn't matter if some other agent A_k takes an action instead of it.

Type 2 - A state in which it does matter to A_i what action is taken, but where some agent A_k just happens to be suggesting an action which is good for A_i. This may or may not be the action A_i would itself have suggested.

With dynamic W-values, the agents *observe* what happened when they were not obeyed, and then *modify* their $W_i(x)$ values based on how bad it was, so that next time round in state x there may be a different winner.

Imagine W as the physical strength agents transmit their signals with, each transmission using up a certain amount of energy. It would be more efficient for agents to have low W if possible, to only spend energy on the states that it has to fight for. The first naive idea was if an agent is suffering a loss it raises its W-value to try to get obeyed. But this will result in an "arms race" and all W's gradually going to infinity. The second naive idea then was let the agent that *is* being obeyed decrease its W-value. But then its W-value will head back down until it is not obeyed any more. There will be no resolution of competition. W-values will go up and down forever.

Clearly, we want W to head to some stable value. And not just the same value, such as having all unobeyed agents $W \to 1$. We want the agents to not fight equally for all states. We need something akin to an *economy*, where agents have finite spending power and must choose what to spend it on. We will have a resource (statespace), agents competing for ownership of the resource, caring more about some parts than others, and with a limited ability to get their way.

5.5.1 Dynamic W = D - F

What we really need to express in W is the difference between predicted reward (what is predicted if the agent is obeyed) and actual reward (what actually happened because we were not obeyed). What happens when we are not listened to depends on what the other agents are doing.

The predicted reward is $D = E(d)$, the expected value of the reward distribution. If we are not obeyed we will receive reward f with expected value $F = E(f)$. We can learn what this loss is by updating:

$$W \mapsto D - f$$

After repeated such updates:

$$W \to D - F$$

Using a $(D - f)$ algorithm, the agent learns what W is best without needing to know why. It does not need to know whether it has ended up with low W because this is Type 1 or because this is Type 2. It just concentrates on learning the *right* W.

Consider the case where our Q-learning agents all share the same suite of actions, so that if another agent is taking action a_k we have already got an estimate of the return in $F = Q_i(x, a_k)$. Then we can directly assign the difference:

$$W := D - F$$

where $D = Q_i(x, a_i)$ and $F = Q_i(x, a_k)$:

$$W_i(x) := Q_i(x, a_i) - Q_i(x, a_k)$$

That is:

$$W_i(x) := d_{ki}(x)$$

where $d_{ki}(x)$ is the *deviation* (expected difference between predicted and actual) or expected loss that A_k causes for A_i if it is obeyed in state x. We can show the losses that agents will cause to other agents in a $n_\times n$ matrix:

$$\begin{pmatrix} 0 & d_{21} & d_{31} & \dots & d_{n1} \\ d_{12} & 0 & d_{32} & \dots & d_{n2} \\ d_{13} & d_{23} & 0 & \dots & d_{n3} \\ \dots & & & & \\ d_{1n} & d_{2n} & d_{3n} & \dots & 0 \end{pmatrix}$$

where all $d_{ki} \geq 0$. Note that all $d_{kk} = 0$ (the leader itself suffers an expected loss of zero).

5.6 A walk through the matrix

Given such a matrix, can we search in some way for the 'right' winner? Consider first a situation where those agents not in the lead have to raise their W-values. If one of them gets a higher W-value than the leader then it goes into the lead, and so on.

Theorem 5.1 *Given variables $W_1, \ldots, W_n$, the process:*

> *start with leader $k :=$ random column and $W_k := 0$*
> *loop:*
>   *for all i other than k*
> $W_i := d_{ki}$
>   *$W_l :=$ highest of these W_i*
>   *if $W_l > W_k$, new leader l and goto loop*
>   *else terminate with winner k*

will terminate within n steps, and we will have found k such that:

$$d_{ki} \leq W_k \;\; \forall i$$

That is, there must exist at least one k such that *all* the values in the k'th column are less than or equal to a value in the k'th row.

Proof: The process goes:

random leader $W_k = 0$

first proper leader $W_l = d_{kl} > W_k$
that is, $0 < d_{kl}$

new leader $W_m = d_{lm} > W_l$
that is, $0 < d_{kl} < d_{lm}$

new leader $W_p = d_{mp} > W_m$
that is, $0 < d_{kl} < d_{lm} < d_{mp}$
...

We have a strictly increasing sequence here. Each element is the highest in its column, so we are using up one column at a time. So the sequence must terminate within n steps. ■

5.6.1 Multiple possible winners

For a given matrix, there may be more than one possible winner. For example, consider the matrix:

$$\begin{pmatrix} 0 & 3 & 0 \\ 0 & 0 & 9 \\ 0 & 0 & 0 \end{pmatrix}$$

Start with all $W_i = 0$. Choose at random agent A_2's action for execution. Then:

$$W_1 := 3$$

Now agent A_1 is in the lead, and:

$$3 > 0, 0$$

Agent A_1 is the winner. However, if we had started by choosing agent A_3's action, then:

$$W_2 := 9$$

Now agent A_2 is in the lead, and:

$$9 > 3, 0$$

Agent A_2 is the winner. We have a winner when *somebody* finds a deviation they suffer *somewhere* that is worse than the deviation they cause everyone else.

We do not examine all d_{ki} combinations and make some kind of global decision. Rather we 'walk' through the matrix and see where we stop. Where the walk stops may depend on where it starts. Later (§13) we will ask if we can make a global decision. But first we consider where the only possible strategy is to make a walk.

Chapter 6

W-learning (Minimize the Worst Unhappiness)

We do not assume in general that other agents' actions are meaningful to an agent. We do not assume that we have a handy estimate of $Q_i(x, a_k)$. If we don't, then all we can do is *observe* what happened when we were not obeyed and the unrecognised action was taken. We can observe the r_i and y it led to, and so build up a substitute for $Q_i(x, a_k)$.

Now there might be many different unrecognised actions being taken by the leaders of different states. Rather than adding them all to our action set and learning a huge $Q_i(x, a)$ for every new action for every state, we only learn the *minimum* that we actually need to compete. In each state x, we simply learn *how bad it is not to be obeyed in this state*. We learn $W_i(x)$, which will require less memory than learning $Q_i(x, a)$ for more than one new action. A change of leader in the state does not give us a new $Q_i(x, a)$ quantity to learn (i.e. we have to expand our memory dynamically). Instead we just change the quantity of the single W-value for the state.

Another complication avoided by this strategy is that the leading agent may be executing a confusing, non-deterministic policy from our viewpoint, perhaps because it perceives a different state to the state that we perceive. In this case, we could not find a single fixed action a_k with which to learn $Q_i(x, a_k)$. Instead we would have to learn $Q_i(x, k)$, where action k means "do whatever agent A_k would do now". We will look more closely in §6.6 at the confusion caused by agents perceiving different subspaces. For now we assume that all agents build up W-values for the full state x. But note in passing that the W-value again avoids such complication by its strategy of simply building up an averaged estimate of *how bad it is not to be obeyed*.

We will need more than one sample to build up a proper estimate. *W-learning* (introduced in [Humphrys, 1995]) is a way of building up such an estimate when agents do not share the same suite of actions. When agent A_k is the winner and has its action executed, all agents A_i *except* A_k update:

$$W_i(x) \mapsto (Q_i(x,a_i) - (r_i + \gamma \max_{b \in A} Q_i(y,b)))$$

for $i \neq k$, but note that the reward r_i and next state y were *caused* by A_k rather than by agent A_i itself. The reason why we do not update $W_k(x)$ is explained later (§6.3). When we see a difference term between predicted and actual, we expect that this 'error term' will go to zero, but note that here it goes to a positive number.

For example, in the discrete case of W-learning, where we store each $W_i(x)$ explicitly in lookup tables, we update:

$$W_i(x) := (1-\alpha)W_i(x) + \alpha(Q_i(x,a_i) - (r_i + \gamma \max_{b \in A} Q_i(y,b)))$$

where α takes successive values $1, \frac{1}{2}, \frac{1}{3}, \ldots$. Since the rewards and Q-values are bounded, it follows that the W-values are bounded (Theorem B.3).

Using such a measure of W, an agent will not need explicit knowledge about who it is competing with. Similar to how Q-learning works, a W-learning agent need only have local knowledge - what state x we were in, what action a it suggested, whether it was obeyed or not, what state y we went to, and what reward r that gave it. It will be aware of its competition only indirectly, by the interference they cause. It will be aware of them when they stop its action being obeyed, and will be aware of the y and r caused as a result. The agent will learn W by experience - by actually experiencing what the other agents want to do.

In fact, I considered not even telling the agent whether it was obeyed, but it seems there is no satisfactory way of doing this. §6.3 shows that the obeyed agent must be treated differently from the unobeyed.

In pseudocode, the W-learning method is, every time step:

```
observe state x
find Wk(x) = highest Wi(x)
execute ak

for all agents i other than k
 Wi(x) -> (Qi(x,ai) - reward for Ai)
```

The change of W's mean that next time round in state x there may be a different winner, and so on. Note that by `reward for Ai` we really mean the combination of immediate reward and next state.

6.1 Comparison of Q-learning and W-learning

The important thing to remember in comparing Q-learning and W-learning is that they are solving different problems. Q-learning is solving the RL problem (choosing the action in pursuit of one goal). W-learning is solving the Action Selection problem (choosing the agent in timeslicing of multiple goals).

W-learning is not another method of RL. It is not a competitor of Q-learning.

Consider Q-learning as the process:

$$D \mapsto d$$

where we are learning D, and d is caused by the execution of our action. Then W-learning is:

$$W \mapsto (D - f)$$

where D is *already* learnt, and f is caused by the execution of *another* agent's action. In the discrete case, Q-learning would be:

$$D := (1 - \alpha)D + \alpha d$$

and W-learning would be:

$$W := (1 - \alpha)W + \alpha(D - f)$$

this is confusing because it looks like a standard way [Sutton, 1988] of writing the Q-learning update:

$$D := D + \alpha(d - D)$$

where the expected value of the error term $(d - D)$ goes to zero as we learn. But this is not the same error term as in W-learning:

$$W := W + \alpha((D - f) - W)$$

6.2 Progress of competition

In general, we assume that Q is learnt before W. Either we delay the learning of W (see [Humphrys, 1995, §3.1]) or, alternatively, imagine a dynamically changing collection with agents being continually created and destroyed over time, and the surviving agents adjusting their W-values as the nature of their competition changes. Q is only learnt once, right from the start of the life of the agent, whereas W is relearnt again and again. The skill that A_i learns, expressed in its converged Q-values, remains intact through subsequent competitions for x. Once it learns its action $a_i^*(x)$ it will promote it in all competitions, only varying the strength with which it is promoted, as its competition varies (this is why we only need to keep $W_i(x)$ values, not $W_i(x,a)$ values). In the long term, the single-step update for A_i is approximated by:

$$W_i(x) \mapsto (V_i^*(x) - (r_i + \gamma V_i^*(y)))$$

where r_i and y are caused by the leader A_k. Note that even though A_k keeps suggesting the same action, we have probabilistic transitions so r_i and y are not constant but are random variables. We can write this as:

$$W_i(x) \mapsto d_{ki}(x)$$

where the deviation $d_{ki}(x)$ (the loss that A_k causes for A_i by being obeyed in state x) is now not a fixed quantity as in §5.5.1 but a *random variable*.

The function V_i^* is fixed, so any variation in $d_{ki}(x)$ is caused only by the variation in r_i and y, which is caused by the variation in $P_{xa}^i(r)$ and $P_{xa}(y)$ (where $a = a_k^*(x)$). We already know that $P_{xa}(y)$ is a stationary distribution. We will assume that, even though A_k's action a is unrecognised by A_i, we still have some stationary distribution $P_{xa}^i(r)$. If so, then $d_{ki}(x)$ will be stationary, with expected value:

$$\begin{aligned} E(d_{ki}(x)) &= V_i^*(x) - (E(r_i) + \gamma E(V_i^*(y))) \\ &= V_i^*(x) - \left(\sum_r r P_{xa}^i(r) \ + \ \gamma \sum_y V_i^*(y) P_{xa}(y)\right) \end{aligned}$$

We expect:

$$E(d_{kk}(x)) = 0$$

though we do not actually update $W_k(x)$, and we expect for $i \neq k$:

$$E(d_{ki}(x)) \geq 0$$

That is, if obeyed, we expect $f = D$. If not obeyed, we expect $f \leq D$.[1]

We cannot calculate $E(d_{ki}(x))$ analytically unless we know $P^i_{xa}(r)$ and $P_{xa}(y)$. So instead, as in Q-learning, we calculate it by sampling it. If A_k leads forever in state x, then by Theorem A.1:

$$\begin{aligned} W_i(x) &\rightarrow E(d_{ki}(x)) \\ &\geq 0 \end{aligned}$$

This convergence will be interrupted if some new agent takes the lead. $W_i(x)$ itself might increase so that $W_i(x) > W_k(x)$ and A_i then takes the lead in state x. If it does, W-learning stops for it until (if ever) it loses the lead. If another agent A_l takes the lead by building up a high $W_l(x)$, then A_i will suddenly be taking samples from the distribution $d_{li}(x)$. By Theorem A.2, if we update forever from *this* point, $W_i(x)$ converges to the expected value of the new distribution:

$$W_i(x) \rightarrow E(d_{li}(x))$$

and so on.

6.2.1 Convergence

W-learning is an *approximation* of the walk through the matrix that we saw in §5.6. Instead of direct assignments to the expected loss, we have to take samples of the distribution of losses.

W-learning does this walk because we cannot exhaustively search all combinations k, i to find the highest $E(d_{ki}(x))$ in the matrix (or whatever we decide would be the fairest winner). It would be impractical to let every agent experience what it is like with every other agent in the lead for long enough to build up an expected value for each one. And it would also require memory to build up a map of the matrix and return to a past leader. W-learning tries to get down to a winner quicker than that. We just put someone into the lead, and it's up to the others to raise their W-values to pass it out. Anyone who can manage a higher W-value is allowed overtake.

[1] Actually, although A_i has learnt the optimal action given its statespace, action set and reward distribution, an agent A_k that is connected to different senses or actions of the body may suggest things that are actually better for A_i, things that A_i is not able to learn itself because it is denied access to some senses or actions. In such a case it would positively pay A_i not to be obeyed (we will show such a case in §7.1.1 later). Our model can actually cope with this - A_i will simply develop negative W-values and not compete. We will return to this whole issue in §18. For the moment we will assume that agents know best what is good for themselves, that is, that they will suffer ≥ 0 if not obeyed.

Just as in Q-learning - where we don't actually have to wait for an infinite time for it to be useful, it will fixate on one or two actions fairly quickly - so in W-learning we don't actually require Q-learning to have converged and we don't have to wait to get to the expected value of $d_{ki}(x)$ for there to be a switch of leader. W-learning rapidly gets down to a competition involving only one or two agents. The switches of leader trail off fairly quickly as W rises.

In each state x, competition will be resolved when some agent A_k, as a result of the deviations it suffers in the earlier stages of W-learning, accumulates a high enough W-value $W_k(x)$ such that:

$$\forall i, i \neq k, \quad W_i(x) \rightarrow E(d_{ki}(x)) < W_k(x)$$

As in the walk, the reason why the competition converges is that for the leader to keep changing, W must keep rising. While there may be statistical variations[2] of $d_{ki}(x)$ in any finite sample, in the long run the expected values $E(d_{ki}(x))$ must emerge, and the walk of §5.6 will take place.

A_k wins because it has suffered a greater deviation in the past than any expected deviation it is now causing for the other agents. The agent that wins is the agent that would suffer the most if it did not win. Think of it as perhaps many agents 'wanting' the state, but A_k wanting it *the most*. Since all $E(d_{ki}(x)) \geq 0$, we normally expect $W_k(x) \gg 0$ for it to win. W-learning resolves competition without resorting to devices such as killing off agents that are disobeyed for time t, without any $W \rightarrow \infty$, and in fact with normally most $W \ll W_{\max}$.

There will be a different competition in each state x, each being resolved at different times. Eventually, the entire statespace will have been divided up among the agents, with a winner for each state x.

6.3 Scoring $W_k(x)$

So why don't we update the leader's W-value as well? The answer is that if we do, we would be updating:

$$\begin{aligned} W_k(x) &\rightarrow E(d_{kk}(x)) \\ &= 0 \end{aligned}$$

[2]For example, the one that takes the lead may not actually be the one with the *worst* expected loss. This is what I allowed for in [Humphrys, 1995, §4], where the longer walk will terminate within n^2 steps. In fact, the one that takes the lead mightn't even have an expected loss worse than $W_k(x)$, it might just be an unlucky sample (see §6.5).

The leader's W is converging to zero, while the other agents' W's are converging to $E(d_{ki}(x)) \geq 0$. They are *guaranteed* to catch up with it. We might think it would be nice to try and reduce all weights to the minimum possible, so as soon as you are obeyed, you start reducing your weight. But you can only *find* the minimum by reducing so far that someone else takes over. As soon as an agent gets into the lead, its W-value would start dropping until it loses the lead. We will have back and forth competition forever under any such system, whereas we want someone to actually *win* the state.

If we do for all i *including* k:

$$W := W + (D - f)$$

then W_k stays the same, while (unless they are suffering zero) the other $W_i \to \infty$ and overtake it. If we do for all i including k:

$$W \mapsto (D - f)$$

then $W_k \to 0$, while (unless they are suffering zero) the other $W_i \to$ some quantity > 0 and overtake it. So we can't have the same rule for the leader as for the others. The leader does nothing - it's up to the others to catch up with it. If they can't, we have a resolved competition.

6.4 Strict highest W

Not scoring the leader $W_k(x)$ leads to a potential problem however. If $W_k(x)$ somehow gets set to an unfairly large value, then it will never get corrected, since the other agents will be unable to catch up to it.

This can happen if it gets initialised randomly to some large value. Note that $W_{\min} < 0 < W_{\max}$ (Theorem B.3). Any W-value ≤ 0 will eventually be challenged since we expect $E(d_{ki}(x)) \geq 0$. However, a W-value in the range $0 < W \leq W_{\max}$ may never be challenged.

The solution is that we initialise all $W_i(x)$ to be in the range $W_{\min} \leq W \leq 0$. They can be random within that range since $(1 - \alpha)$ will wipe out the initial value in the first update.

6.5 Stochastic highest W

But an unfairly high $W_k(x)$ can also happen because of unlucky samples. Say one time A_k wasn't in the lead in state x, and it experienced:

$$W_k(x) \mapsto d_{jk}(x)$$

where $d_{jk}(x) \gg E(d_{jk}(x))$ is a sample from the very high end of the distribution with expected value $E(d_{jk}(x))$. A_k *normally* won't suffer this when A_j is in the lead, but this single unlucky update puts A_k into the lead and its W-value is never challenged after that. The other W's all converge to their expected values $E(d_{ki}(x))$, but $W_k(x)$ does not converge to anything. It remains representing this single unlucky sample.

The solution is still not to score the leader's W-value, at least not while it is in the lead. Rather, we make sure that its W-value is updated a few times before it can take the lead forever. It can't win based on just one sample.

The solution is to pick the highest W with a Boltzmann distribution that starts at a moderately high temperature (pick a stochastic winner centred on the highest W) and declines over time. We still only score the ones that aren't obeyed. We reduce the temperature until we end up with one winner, the others converging to the deviation it causes them. Again, as in §2.2.3, we aim to have the finished creature end up with just a low temperature Boltzmann rather than brittle strict-determinism.

To be precise, when we observe state x, we obey agent A_k with probability:

$$p(k) = \frac{e^{\frac{W_k(x)}{T}}}{\sum_i e^{\frac{W_i(x)}{T}}}$$

Note that $\sum_i p(i) = 1$. The advantage of this is that if we start with a high temperature (pick random winners), it allows W-values to start totally random. We don't need an initialisation strategy any more.

6.6 W-learning with subspaces

The analysis in §6.2 showed that if agent A_k leads in state x, then:

$$W_i(x) \to E(d_{ki}(x))$$

But this assumes that the x in $W_i(x)$ refers to the *full* state. What if our agents, which are already only learning Q-values in subspaces (§4.4.1) were to only build up their W-values in subspaces too?

The problem with this is that there may be many different leaders for the many different full states which A_i sees as all the one state. And even if there was just one leader, what A_i sees as one state is seen by the leader as a number of different states so it will be executing different actions, causing quite different deviations for A_i. From A_i's point of view, sometimes the leader A_k executes a good action, sometimes a bad one, for no apparent reason. But instead of this simply being samples from different ends of a distribution, we are actually taking samples from *different* distributions. In short, we are not repeatedly updating:

$$W_i(x) \mapsto d_{ki}(x)$$

for a single distribution $d_{ki}(x)$ with expected value $E(d_{ki}(x))$. Rather we are updating:

$$W_i(x) \mapsto d^{(j)}$$

for many different distributions $d^{(j)}$, representing different k's and different x's. By Theorem A.4:

$$W_i(x) \to \sum_j p(j) E(d^{(j)})$$

where $p(j)$ is the probability of taking a sample from $d^{(j)}$.[3] So we coalesce the expected values of multiple distributions into one W-value. This is a weighted mean (see §E) since $\sum_j p(j) = 1$.

It is a rather crude form of competition. While Q-learning with subspaces is just as good for learning the policy as Q-learning with full space, W-learning with subspaces might not be as 'intelligent' a form of competition as W-learning with full space. The agent doesn't need the full space to learn its own policy, but it may need it to talk to other agents.

We will return to this in §10. For now, we use both Q-learning and W-learning with subspaces for the sake of the tiny memory requirements this method will have. A_i's W-value is a weighted mean of all the separate W-values that it would have if it was able to distinguish the states. While all these W-values have been compressed into one W-value, it is done fairly. The more likely it is to experience a deviation $d^{(j)}$, the more biased $W_i(x)$ will be in the direction of $E(d^{(j)})$.

[3]Assuming this probability is stationary. What we are talking about here is, given that the agent observes a certain subspace state, what is the probability that this is *really* a certain full space state?

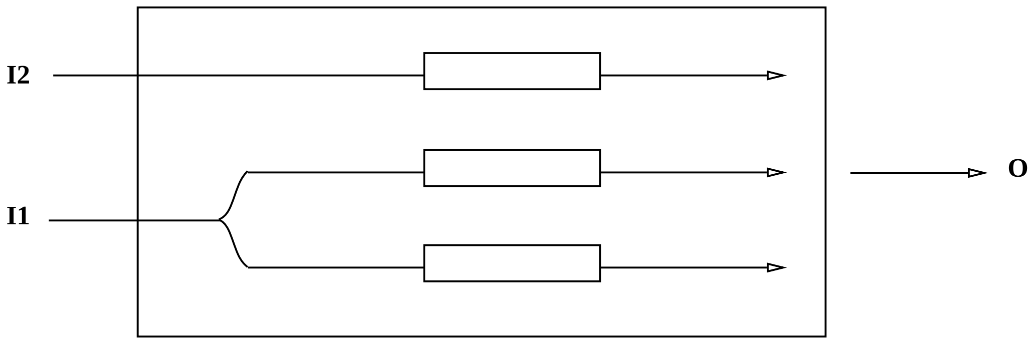


Figure 6.1: Agents may compete with each other despite sharing no common sensors. The abstract 'full' state here is $x = (I_1, I_2)$, though nothing in the creature works with this full state.

6.6.1 Agents with heterogenous sensory worlds

With W-learning with subspaces, we can add new sensors dynamically, with new agents to make use of them, and the already-existing agents will just react to it as more competition, of the strange sort where sometimes in the same state (as they see it) they are opposed and sometimes not (Figure 6.1).

Recall the 'map' of the statespace that we suggested drawing in §5, showing who wins each state. When agents work in subspaces only, the creature as a whole can still draw up a map of the full state-space. A W-value built up by an agent in a sub-state will typically be enough to win only *some* of the full states that the sub-state maps to, and lose others. The agent itself could not draw up a map of its statespace, since it could not classify its x as either 'won' or 'lost'.

Note how concepts are becoming distributed in this model. For example, in our House Robot problem, there is now no central Perception area where the concepts of both 'dirt' and 'smoke' coexist. One agent solves the perception problem of distinguishing dirt from *non-dirt*. Another agent solves the different problem of separating smoke from non-smoke. Each solves the problem of vision to the level of sophistication necessary for its own purposes - nobody directly solves the problem of separating dirt from smoke. Agents existing in different sensory worlds are competing against each other.

Hierarchical Q-learning with subspaces looks like this too except that the *switch* must know about the full state-space.

Chapter 7

W-learning with subspaces (preliminary test in Ant World)

Before we test W-learning in the House Robot problem, we first look at an implementation of it in a simpler problem. The reason we do this is because we want to draw a map of the full statespace, showing who wins each state. In the House Robot problem, the full statespace is too big to explicitly map. It is too big even to illustrate interesting subspaces. In this simpler problem, the full space is small enough to map.

The Ant World problem is the conflict between seeking food and avoiding moving predators on a simple toroidal gridworld (Figure 7.1). The world contains a nest, a number of stationary, randomly-distributed pieces of food, and a number of randomly-moving dumb predators. Each timestep, the creature can move one square or stay still. When it finds food, it picks it up. It can only carry one piece of food at a time, and it can only drop it at the nest. The task for the creature is to forage food (i.e. find it, and bring it back to the nest) while avoiding the predators.[1]

This is clearly the ancestor of the House Robot problem. The food/predators problem can be translated directly to the vacuuming/security problem - food becomes dirt, the nest becomes the plug, and the moving predators to avoid become moving family to avoid. Unlike in the House Robot problem, here the world is a proper torus, so the creature can always run away from the predator - it cannot get stuck in corners. The creature senses $x = (i, n, f, p)$

[1] **Problem Details -** For these results (from [Humphrys, 1995]), when a piece of food is picked up its square becomes blank. The creature makes repeated short runs of length 50 timesteps each. Each run starts with the creature, NOPREDATORS predators and NOFOOD new pieces of food placed randomly. Ideally the creature should manage to collect all NOFOOD pieces (so that the grid is empty) before the run ends. We have a SIZE x SIZE grid. For these results we had SIZE=10, NOPREDATORS=1 and NOFOOD=4.

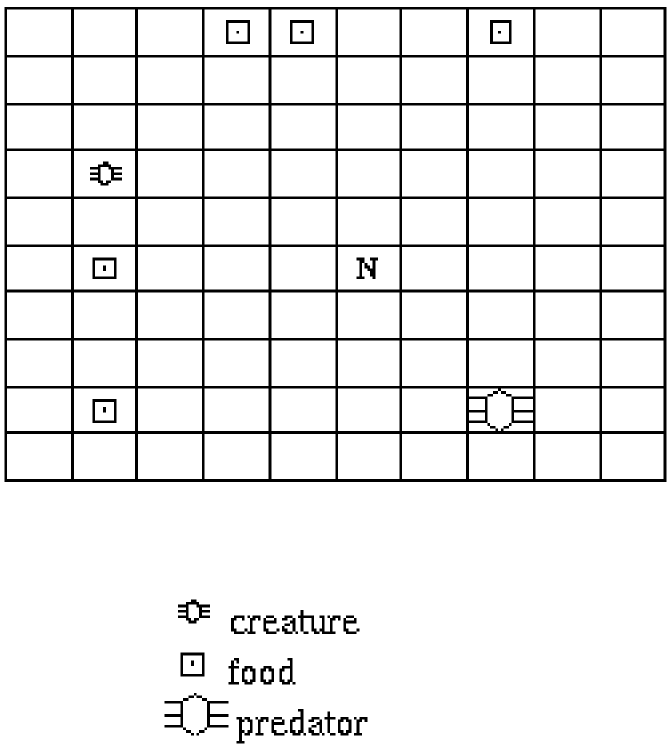


Figure 7.1: The Ant World problem. N is the nest.

where:

- i is whether the creature is carrying food or not, and takes values 0 (not carrying) and 1 (carrying).
- n (0-8) is the direction (but not distance) of the nest as before in Figure 4.2. Here the direction of the nest is known from any distance.
- f (0-9) is the direction of the nearest visible food. Unlike the nest, food and predators are only visible within a small radius.
- p (0-9) is the direction of the nearest visible predator.

As before, the creature takes actions a, which take values 0-7 (move in that direction) and 8 (stay still).

7.1 Analysis of best food-finding solution

We searched for a good food-finding solution and a good predator-avoiding solution. See [Humphrys, 1995] for the details of what exactly our search was. The important thing here is just to show how agents can successfully interact via W-learning. The best food-finding solution was this collection of agents:

A_f	senses: (i,f)	
	reward: if (picked up food)	1.62 else 0
A_n	senses: (n)	
	reward: if (arrived at nest)	0.15 else 0
A_p	senses: (p)	
	reward: if (just shook off predator)	0.17 else 0

This is the collection called EVO1 in [Humphrys, 1995], rewritten to take advantage of the fact that an agent with reward function `if (condition) r else s` is interchangeable with one with reward function `if (condition) (r-s) else 0`, in the sense that both its policy and its W-values will be the same (see §C.3).

Looking at who wins each state, A_f wins almost the entire space where $i = 0$ (not carrying). In the space where $i = 1$ (carrying), A_n wins if $p = 9$ (no predator visible). Where a predator is visible, A_n wins if the nest direction is along a diagonal (0,2,4,6), otherwise the space is split between A_n and A_p.

For example, here are the owners of the area of statespace where $p = 7$. The agents A_f, A_n, A_p are represented by the symbols `o, NEST, pred` respectively. States which have not (yet) been visited are marked with a dotted line:

```
---- p=7: ----
i=0:
f=0     o       o       o       o       o       o       o       o       o
f=1     o       o       o       o       o       o       o       o       o
f=2     o       o       o       o       o       o       o       o       o
f=3     o       o       o       o       o       o       o       o       o
f=4     o       o       o       o       o       o       o       o       o
f=5     o       o       o       o       o       o       o       o       o
f=6     o       o       o       o       o       o       o       o       o
f=7     o       o       o       o       o       o       o       o       o
f=8     ------------------------------------------------------------------------
f=9     o       o       NEST    o       NEST    o       o       o       o
        n=0     n=1     n=2     n=3     n=4     n=5     n=6     n=7     n=8
i=1:
f=0     NEST    pred    NEST    pred    NEST    NEST    NEST    NEST    --------
f=1     NEST    NEST    NEST    pred    NEST    pred    NEST    NEST    --------
f=2     pred    NEST    NEST    pred    NEST    pred    NEST    pred    --------
f=3     NEST    pred    NEST    NEST    NEST    pred    NEST    pred    --------
f=4     NEST    pred    NEST    NEST    NEST    NEST    NEST    pred    --------
f=5     NEST    pred    NEST    pred    NEST    pred    NEST    NEST    --------
f=6     NEST    pred    NEST    NEST    NEST    pred    NEST    pred    --------
f=7     NEST    pred    NEST    pred    NEST    pred    NEST    NEST    --------
f=8     NEST    NEST    NEST    pred    NEST    pred    NEST    NEST    --------
f=9     NEST    o       NEST    NEST    NEST    pred    NEST    NEST    --------
        n=0     n=1     n=2     n=3     n=4     n=5     n=6     n=7     n=8
```

The difference between diagonals and non-diagonals was an unexpected result. On analysis, it turned out that it was caused by the way in which compass directions were assigned. There are 4 diagonal directions and 4 non-diagonal directions. Anything which doesn't lie directly on one of the 8 primary or secondary directions is assigned to the nearest *non-diagonal* direction. So the non-diagonal directions gathered up all the messy angles while if you sensed something on a diagonal direction you knew exactly where it was. If the nest is sensed in a diagonal direction, then moving that way is *guaranteed* to hit it without a change of direction. If the nest is in a non-diagonal direction, then the creature may have to make a change of direction as it moves towards it.[2]

[2] For the House Robot problem, I changed this to use the nearest direction to the precise angle, so there was no bias between diagonals and non-diagonals.

A_n is more confident that it will get its reward when it sees the nest along a diagonal and moves along that diagonal, and it builds up higher W-values accordingly, causing it to win these states from A_p. This is certainly a more subtle way for the agents to give way to each other than a programmer would normally think of.

Drawing the complete map of the full statespace shows that A_f wins 49.27% of the states, A_n wins 34.97% and A_p wins 15.75%.

Here are *all* the W-values of all the agents, sorted to show who beats who. The W-values $W_f(i,f), W_n(n), W_p(p)$ are represented by `food.W(i,f)`, `nest.W(n)`, `predator.W(p)` respectively:

```
    food.W(0,0)      0.195
    food.W(0,1)      0.183
    food.W(0,7)      0.144
    food.W(0,5)      0.114
    food.W(0,2)      0.097
    food.W(0,6)      0.094
    food.W(0,3)      0.089
    food.W(0,9)      0.087
    food.W(0,4)      0.084
    nest.W(6)        0.083
    nest.W(2)        0.074
    nest.W(4)        0.058
    nest.W(0)        0.041
predator.W(1)        0.037
predator.W(0)        0.021
predator.W(2)        0.018
predator.W(3)        0.016
    nest.W(8)        0.016
predator.W(7)        0.013
    nest.W(7)        0.012
predator.W(5)        0.011
    nest.W(1)        0.006
predator.W(4)        0.004
predator.W(6)        0.001
    nest.W(3)        0.001
    food.W(0,8)      0.000    (never visited)
    nest.W(5)       -0.000
predator.W(8)       -0.003
predator.W(9)       -0.008
    food.W(1,9)     -0.008
    food.W(1,6)     -0.026
    food.W(1,1)     -0.027
    food.W(1,3)     -0.031
    food.W(1,5)     -0.032
    food.W(1,2)     -0.033
```

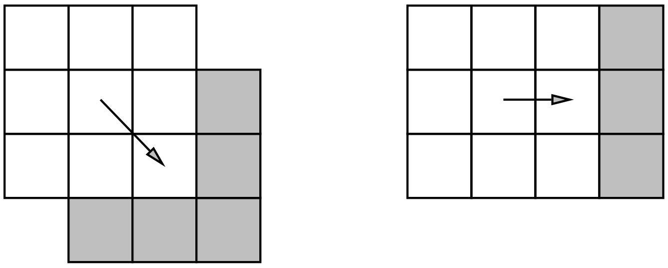


Figure 7.2: When the visible squares are simply the adjacent ones, a diagonal move reveals new squares along two sides of the box, whereas a non-diagonal move reveals new squares on one side only. A diagonal move is the better search strategy.

```
food.W(1,7)       -0.036
food.W(1,0)       -0.036
food.W(1,4)       -0.045
food.W(1,8)       -0.098
```

The next level of analysis is what actions the creature actually ends up executing as a result of this resolution of competition. When not carrying food, A_f is in charge, and it causes the creature to wander, and then head for food when visible. A_n is constantly suggesting that the creature return to the nest, but its W-values are too weak. Then, as soon as $i = 1$, A_f's W-values drop below zero, and A_n finds itself in charge. As soon as it succeeds in taking the creature back to the nest, $i = 0$ and A_f immediately takes over again. In this way the two agents combine to forage food, even though both are pursuing their own agendas.

EVO1 is a good forager partly because A_f turns out to have discovered a trick in searching for food. In [Humphrys, 1995], I hand-coded a creature for the Ant World (similar to the hand-coded program for the House Robot that we saw above in §4.3.3). When the creature couldn't see food, my hand-coded program just adopted the strategy of making a random move 0-7. One might think that there is no better memoryless strategy for searching. In fact, there is. At any point, a diagonal move (of distance $\sqrt{2}$) reveals on average slightly more *new* squares than a non-diagonal move (of distance 1). One can see this easiest when the visible squares are simply the adjacent ones (Figure 7.2) but it also holds true for the distance-based field of vision we used in the Ant World.

So moving around diagonally is a better search strategy. A_f gradually builds up higher Q-values along the diagonals (0,2,4,6), discovering something that might have easily escaped the programmer:

```
 (a)           (0)     (1)     (2)     (3)     (4)
Qf((0,9),a)  [0.724] 0.703  0.709  0.694  0.689

 (a)           (5)     (6)     (7)     (8)
Qf((0,9),a)   0.697  0.707  0.701 <0.680>
```

7.1.1 Negative W-values

The final level of analysis is why the W-values turn out the way they do. We can see, for example, that when $i = 1$ (carrying food), A_f is a long way off from getting a reward, since it has to lose the food at the nest first. And it cannot learn how to do this since (n) is not in its statespace. A_f ends up in a state of dependence on A_n, which actually knows better than A_f the action that is best for it. In the notation of §5.5.1, A_f regularly experiences $D < f$, and as a result the values $W_f(1, *)$ here are all negative. One can see negative W-values for the following reasons:

- The agent, as here, does not sense some information that would be useful to it. Or it may not have the full suite of actions that it needs. Another agent keeps doing things that help this agent, but this agent can't learn how to do them itself. It keeps getting negative W updates. W can be persistently negative.

- The world is not an MDP. If the world is not an MDP (recall §4.2), then the agent may not be able to learn the optimal Q^* policy for its own reward function. This is similar to the first point, since the agent probably needs more senses. W can be persistently negative.

- We have only taken a finite number of samples. Say we have $Q_i^*(x, a_k)$ only slightly less than $Q_i^*(x, a_i)$. We have only taken a small finite number of samples, which just happen to be from the *high* end of the distribution $> Q_i^*(x, a_i)$. After finite time, W is negative. After infinite time, it will end up positive.

So why not get rid of A_n altogether and simply supply A_f with the space $x = (i, n, f)$? Because it is more efficient if we can use two agents with statespaces of size 20 and 9 respectively (total memory required = 29) instead of one with a statespace of size 180 (total memory required = 180). Obviously this only really becomes important as these numbers get larger. As we scale up, addition (multiple agents with subspaces) is preferable to multiplication (one agent with full space).

7.2 Analysis of best predator-avoiding solution

Here is the predator-avoiding solution:

A_f	senses: (i,f)	
	reward: if (picked up food)	1.65 else 0
A_n	senses: (n)	
	reward: if (arrived at nest)	0.19 else 0
A_p	senses: (p)	
	reward: if (just shook off predator)	1.23 else 0

This is the collection EVO2 in [Humphrys, 1995]. The predator-sensing agent is much stronger, and the contrast in behavior is dramatic. Here is that same area of statespace:

```
---- p=7: ----
i=0:
f=0     o       o       pred    o       o       o       o       o       o
f=1     pred    pred    o       pred    pred    pred    pred    pred    pred
f=2     pred    pred    pred    pred    pred    o       pred    pred    pred
f=3     pred    pred    pred    o       pred    pred    pred    o       pred
f=4     o       o       o       o       o       o       pred    o       o
f=5     pred    o       o       pred    pred    pred    pred    pred    o
f=6     o       o       o       o       o       o       o       o       o
f=7     pred    o       o       o       o       pred    o       o       o
f=8     ------------------------------------------------------------------------
f=9     pred    pred    pred    pred    pred    pred    pred    pred    pred
        n=0     n=1     n=2     n=3     n=4     n=5     n=6     n=7     n=8
i=1:
f=0     pred    pred    pred    pred    pred    pred    pred    pred    --------
f=1     pred    pred    pred    pred    pred    pred    pred    pred    --------
f=2     pred    pred    pred    pred    pred    pred    pred    pred    --------
f=3     pred    pred    pred    pred    pred    pred    pred    pred    --------
f=4     pred    pred    pred    pred    pred    pred    pred    pred    --------
f=5     pred    pred    pred    pred    pred    pred    pred    pred    --------
f=6     pred    pred    pred    pred    pred    pred    pred    pred    --------
f=7     pred    pred    pred    pred    pred    pred    pred    pred    --------
f=8     pred    pred    pred    pred    pred    pred    pred    pred    --------
f=9     pred    pred    pred    pred    pred    pred    pred    pred    --------
        n=0     n=1     n=2     n=3     n=4     n=5     n=6     n=7     n=8
```

A_p mainly dominates when a predator is visible in directions 0-7. In particular, in that space A_f loses the crucial state $(0, 9)$ (not carrying food and no food visible). In the special case $p = 8$, all directions are equal as far as A_p is concerned, and A_f and A_n are allowed compete to take the action. When $p = 9$, A_f and A_n fight it out as if A_p wasn't there. They end up combining to forage.

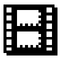
A_f's share of the statespace has dropped to 35.22%, A_n's has dropped to 15.78% and A_p's has risen to 49.01%. Percentage of statespace owned is of course only a very rough measure of the influence of the agent on the behavior of the creature - ownership of a few key states may make more difference than ownership of many rarely-visited states. Also, we must remember that ownership of a state does not imply that the agent had to fight other agents for it. A weak agent may be allowed to own lots of states, but only because they happen to coincide with what the dominant agent wants. The ownership of all these states by the weak agent masks the fact that the dominant agent really owns the entire statespace.

Here are all the W-values:

```
    food.W(0,4)        0.201
    food.W(0,6)        0.166
    food.W(0,0)        0.164
predator.W(4)          0.162
predator.W(5)          0.160
    food.W(0,7)        0.153
predator.W(7)          0.149
    food.W(0,5)        0.149
    food.W(0,1)        0.146
    food.W(0,2)        0.143
predator.W(1)          0.140
    food.W(0,3)        0.133
predator.W(0)          0.132
predator.W(3)          0.127
predator.W(6)          0.103
    nest.W(2)          0.099
    nest.W(0)          0.094
    nest.W(4)          0.079
    nest.W(6)          0.068
    food.W(0,9)        0.049
    nest.W(8)          0.045
    nest.W(1)          0.035
    nest.W(7)          0.029
    food.W(1,7)        0.025
    nest.W(5)          0.023
    nest.W(3)          0.019
predator.W(2)          0.002
    food.W(1,5)        0.001
    food.W(0,8)        0.000    (never visited)
```

food.W(1,9)	-0.007
predator.W(9)	-0.015
predator.W(8)	-0.015
food.W(1,1)	-0.020
food.W(1,3)	-0.037
food.W(1,4)	-0.051
food.W(1,0)	-0.052
food.W(1,2)	-0.066
food.W(1,6)	-0.082
food.W(1,8)	-0.096

7.3 MPEG Movie demo of basic W-learning



An MPEG Movie demo of a W-learning forager in the Ant World[3] can be viewed at the web page:

`http://www.cl.cam.ac.uk/~mh10006/w.html`

The MPEG Movie demo is actually of the best forager found in the Systematic search section (§4.2) of [Humphrys, 1995a]. The following graphics are screen shots of the web page. The text is worth reading here - though obviously the accompanying movies can't be played until one goes to the web page.[4]

7.3.1 Rewarding on transitions or continuously

In §2.1.3 we contrasted rewarding on transitions from x to y with just rewarding for being in state y. Note (see the text of the web page) how rewarding on transitions makes A_n happier to cooperate with the other agents. It does not resist leaving the nest since it only gets rewards for the moment of arrival.

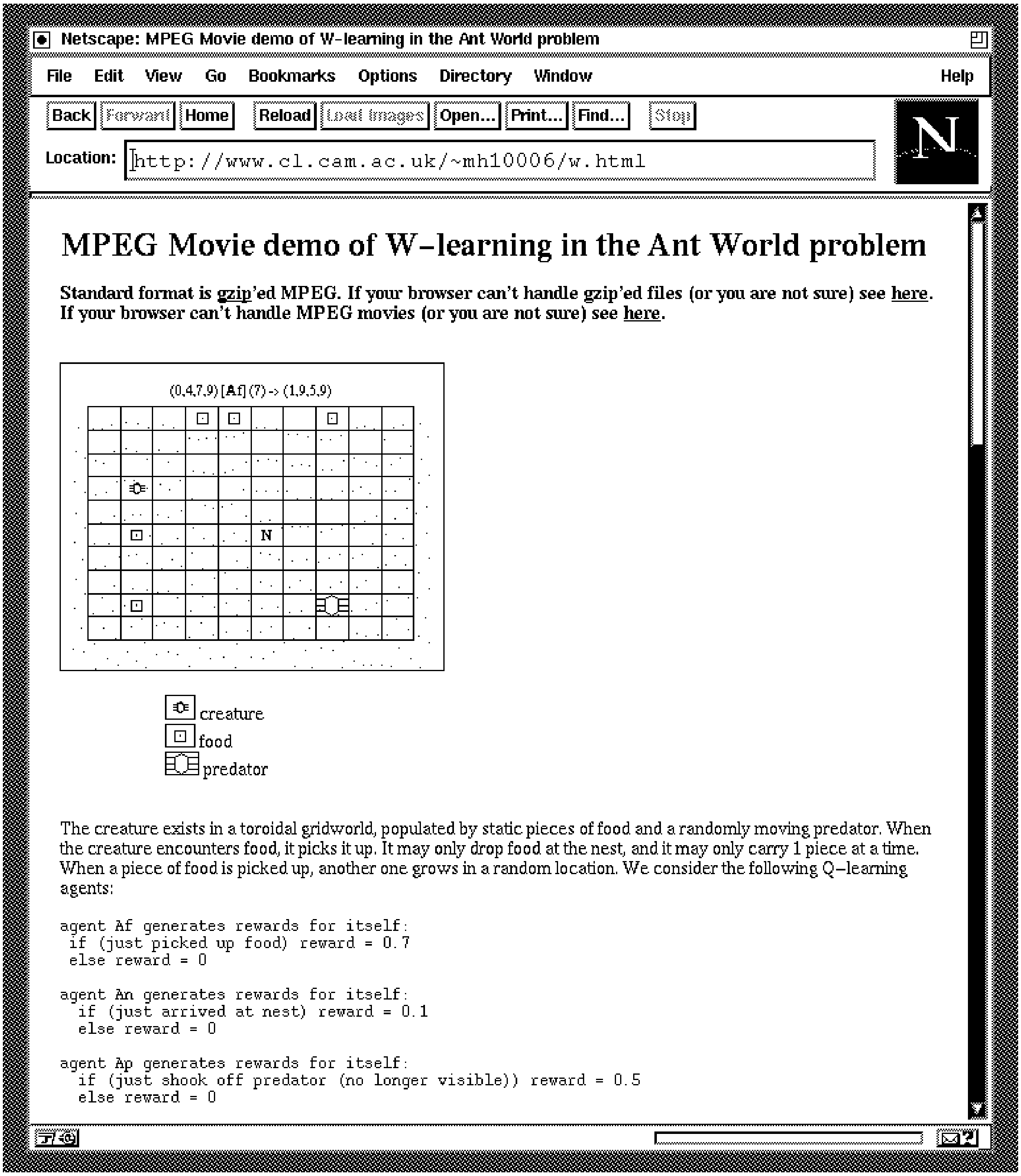
[3]**Problem Details -** For the MPEG Movie experiment (from [Humphrys, 1995a]), when a piece of food is picked up another one grows in a random location. So at all times there are fully NOFOOD pieces on the grid. There is no such thing as 'runs' - instead the world can run continuously for thousands of steps. Here SIZE=10, NOPREDATORS=1 and NOFOOD=5. Also, the nest is now only visible within a small radius (n takes values 0-9).

[4]Actually, the movies are also on a 'video appendix' that is deposited with this dissertation in Cambridge University Library. This VHS video tape plays 4 MPEG Movies in sequence. First, the creature under the control of agent A_f alone. Then A_n alone. Then A_p alone. Then all 3 competing together in the same body.

If we rewarded it continually for being in the nest, it would resist leaving. Similarly, A_f does not resist being taken back to the nest and losing food, since it only gets rewards at the moment of picking up food. If we rewarded it continuously for having food, it would resist going anywhere (that is, it would try to stay still).

In general, if an agent is rewarded only for arriving at a place, once it gets there it won't stay still, but will go the *minimum* distance away from it and then come back so it can get the reward again. If the agent is rewarded just for being in the place, when it arrives it will stay still.

In this thesis, I leave construction of these reward functions as a design problem. Rewarding on transitions was an easy way to think about the Ant World problem, with agents 'not caring' at different points, but we probably could have found a solution as well even if we rewarded continuously.



MPEG Movie demo of W-learning in the Ant World problem

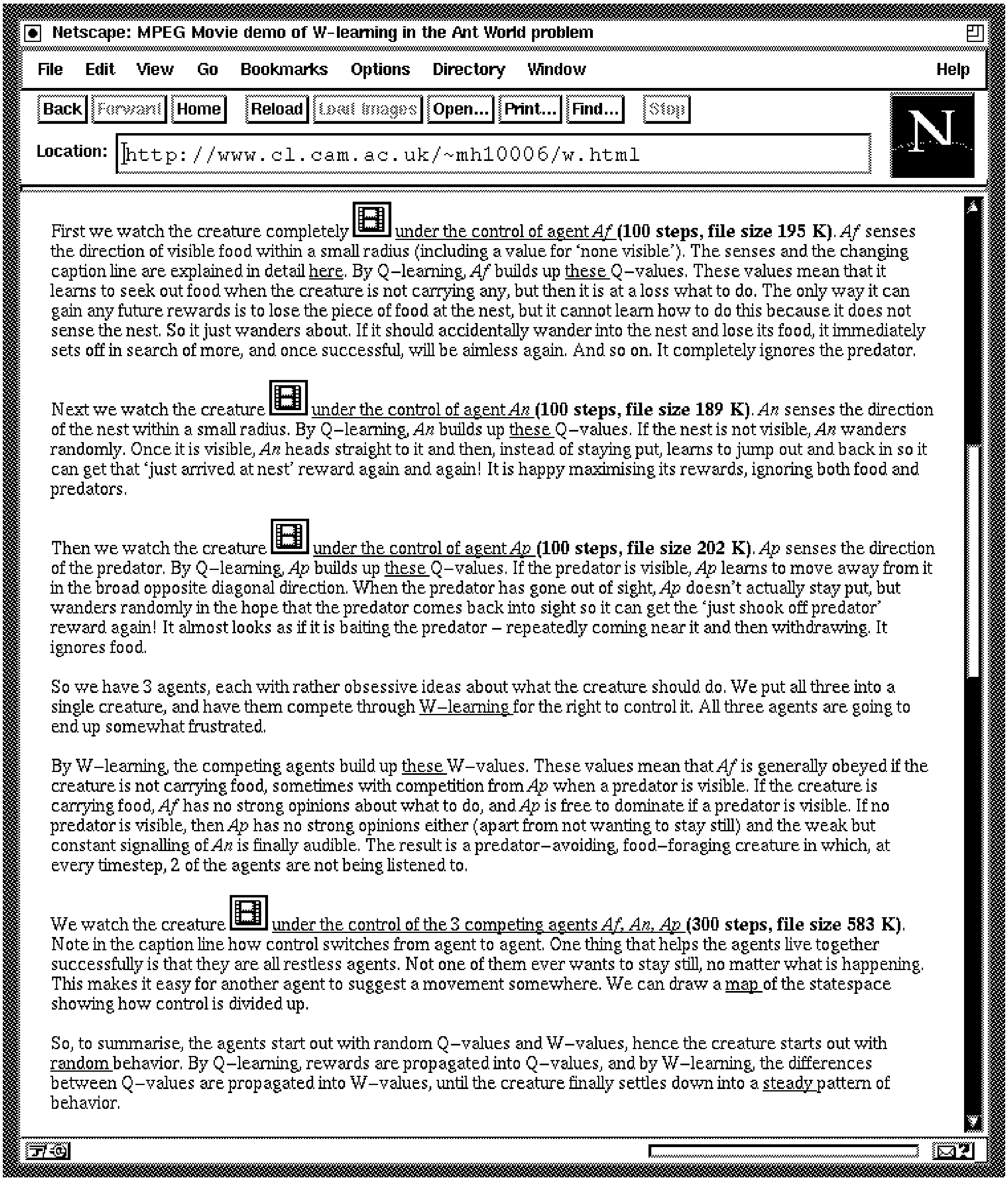
Standard format is gzip'ed MPEG. If your browser can't handle gzip'ed files (or you are not sure) see here. If your browser can't handle MPEG movies (or you are not sure) see here.

The creature exists in a toroidal gridworld, populated by static pieces of food and a randomly moving predator. When the creature encounters food, it picks it up. It may only drop food at the nest, and it may only carry 1 piece at a time. When a piece of food is picked up, another one grows in a random location. We consider the following Q-learning agents:

```
agent Af generates rewards for itself:
 if (just picked up food) reward = 0.7
 else reward = 0

agent An generates rewards for itself:
  if (just arrived at nest) reward = 0.1
  else reward = 0

agent Ap generates rewards for itself:
  if (just shook off predator (no longer visible)) reward = 0.5
  else reward = 0
```



First we watch the creature completely under the control of agent *Af* **(100 steps, file size 195 K)**. *Af* senses the direction of visible food within a small radius (including a value for 'none visible'). The senses and the changing caption line are explained in detail here. By Q–learning, *Af* builds up these Q–values. These values mean that it learns to seek out food when the creature is not carrying any, but then it is at a loss what to do. The only way it can gain any future rewards is to lose the piece of food at the nest, but it cannot learn how to do this because it does not sense the nest. So it just wanders about. If it should accidentally wander into the nest and lose its food, it immediately sets off in search of more, and once successful, will be aimless again. And so on. It completely ignores the predator.

Next we watch the creature under the control of agent *An* **(100 steps, file size 189 K)**. *An* senses the direction of the nest within a small radius. By Q–learning, *An* builds up these Q–values. If the nest is not visible, *An* wanders randomly. Once it is visible, *An* heads straight to it and then, instead of staying put, learns to jump out and back in so it can get that 'just arrived at nest' reward again and again! It is happy maximising its rewards, ignoring both food and predators.

Then we watch the creature under the control of agent *Ap* **(100 steps, file size 202 K)**. *Ap* senses the direction of the predator. By Q–learning, *Ap* builds up these Q–values. If the predator is visible, *Ap* learns to move away from it in the broad opposite diagonal direction. When the predator has gone out of sight, *Ap* doesn't actually stay put, but wanders randomly in the hope that the predator comes back into sight so it can get the 'just shook off predator' reward again! It almost looks as if it is baiting the predator – repeatedly coming near it and then withdrawing. It ignores food.

So we have 3 agents, each with rather obsessive ideas about what the creature should do. We put all three into a single creature, and have them compete through W–learning for the right to control it. All three agents are going to end up somewhat frustrated.

By W–learning, the competing agents build up these W–values. These values mean that *Af* is generally obeyed if the creature is not carrying food, sometimes with competition from *Ap* when a predator is visible. If the creature is carrying food, *Af* has no strong opinions about what to do, and *Ap* is free to dominate if a predator is visible. If no predator is visible, then *Ap* has no strong opinions either (apart from not wanting to stay still) and the weak but constant signalling of *An* is finally audible. The result is a predator–avoiding, food–foraging creature in which, at every timestep, 2 of the agents are not being listened to.

We watch the creature under the control of the 3 competing agents *Af, An, Ap* **(300 steps, file size 583 K)**. Note in the caption line how control switches from agent to agent. One thing that helps the agents live together successfully is that they are all restless agents. Not one of them ever wants to stay still, no matter what is happening. This makes it easy for another agent to suggest a movement somewhere. We can draw a map of the statespace showing how control is divided up.

So, to summarise, the agents start out with random Q–values and W–values, hence the creature starts out with random behavior. By Q–learning, rewards are propagated into Q–values, and by W–learning, the differences between Q–values are propagated into W–values, until the creature finally settles down into a steady pattern of behavior.

Chapter 8

W-learning with subspaces (test in House Robot)

Now we return to the House Robot problem to test W-learning. There being no global (x, i) statespace to worry about, we can expand the number of agents. To the previous five (§4.4), we add three more agents. The collection of agents is as follows:

A_d	senses: (d,i)	
	reward: if (picked up dirt)	r_d else 0
A_p	senses: (p)	
	reward: if (arrived at plug)	r_p else 0
A_c	senses: (w)	
	reward: if (lost sight of wall)	r_c else 0
A_w	senses: (w)	
	reward: if (wall same dir as last time)	r_w else 0
A_u	senses: (h,c)	
	reward: if (made ID)	r_u else 0
A_s	senses: (h,c)	
	reward: if (ID=stranger and visible)	r_s else 0
A_m	senses: (h,c)	
	reward: if (ID=family and here)	0 else r_m
A_f	senses: (f,w_f)	
	reward: if (put out fire)	r_f else 0

Rewards are in the range $0 < r \leq 1$. Both the $Q_i(x, a)$ values and $W_i(x)$ values refer to x in the subspaces, for which lookup tables can be used.

A_c should head for the centre of an open area while A_w should engage in wall-following. In fact, as we shall see, A_w turned out to be useless since its preferred action of course is to stay still. Perhaps its reward should just have been `if (wall visible) ..`

We can add more agents than probably needed - if they're not useful they just won't win any W-competitions and won't be expressed. In Hierarchical Q-learning, you can add extra agents that aren't chosen by the switch but you pay the price of a larger (x, i) statespace.

8.1 Searching for well-balanced collections

8.1.1 Making agents weaker or stronger

Since the exact values of these rewards $r_i > 0$ don't matter for the policy learned by Q-learning, why do we not set them all to 1 as we did before in §4.4?

The answer is that the size of the rewards affect the size of the *W-values*. Theorem C.2 shows that if we have an agent of the form:

A_i reward: if (good event) r else s

then A_i will present W-values:

$$W_i(x) = c_{ki}(x)(r - s)$$

where $c_{ki}(x)$ is a constant independent of the particular rewards. W is simply proportional to the subtraction $(r-s)$, so in particular we lose nothing by fixing $s = 0$ here and just looking at different values of r. [1] Then we have simply:

$$W_i(x) = c_{ki}(x) r_i$$

Increasing the size of r_i will cause A_i to have the same policy (the same disagreements with the other agents about what action to take), but higher W-values (an increased ability to compete). r_i is the parameter by which we make agents with the same logic stronger or weaker within the creature as a whole. We do not want all agents to be of equal importance within the creature. Rather, adaptive collections are likely to involve well-chosen combinations of weak and strong agents. For instance, a creature containing a strong version of the predator-avoiding agent: (if predator visible $r = -10$ else $r = 0$) will behave differently from one containing a weak version of the same thing: (if predator visible $r = -0.1$ else $r = 0$). The predator-avoiding

[1] This is something I neglected to exploit in [Humphrys, 1995, §5.4], where I used 2-reward functions with 2 non-zero rewards and needlessly evolved *both* rewards.

agent in both will be suggesting the same actions, but in the former creature it is more likely to actually win its competitions.

The set of agents above define a vast range of creature behaviors, depending on how strong each agent is. For example, if $r_w = 1$ and all other rewards are 0.001, then A_w will beat all other agents in competition since its absolute $(D - f)$ differences will be so large compared to theirs. In almost all states x it will build up a higher W-value $W_w(x)$ than any of the competitor $W_i(x)$'s. The house robot will be completely dominated by A_w, and will spend all its time wall-following - in fact, spend all its time stationary, with a wall in sight, since that is A_w's preferred position.

In conclusion, the kind of *scaling* we suggested in §5.4 is not of any use to us. We're quite happy to have unequal agents, to use the absolute differences between Q-values, not relative ones. This demands that their Q-values are all judged on the same scale, but this is no restriction - we have to adjust their rewards relative to each other anyway to make an adaptive collection.

8.1.2 Making agents weaker or stronger without re-learning Q

So we try out different combinations of the r_i's, either by hand or by an automated search, looking for adaptive collections. A trick that was employed to speed up our search is that we only have to learn the agents' Q-values once at the start. We learn the Q-values for reward 1:

A_i reward: if (condition) 1 else 0

Then to generate the Q-values for this:

A_i reward: if (condition) r_i else 0

we just multiply the base Q-values by r_i (Theorem C.3).

Figure 8.1 shows how multiplying the base Q-values by a factor does not change the Q-learning policy (the suggested action) but it does change the progress of W-learning (the differences $D - f$) by exaggerating (making the agent stronger) or levelling out (making it weaker) the Q-values, while still preserving their basic contour. This trick allows us make agents weaker or stronger without re-learning Q.

A *scaled* measure of W, however, would be indifferent to multiplication by a constant. Say we were using a standard deviation-based measure of W. Instead of the static, z-score version of §5.4.1, we would probably use a dynamic version depending on a_k:

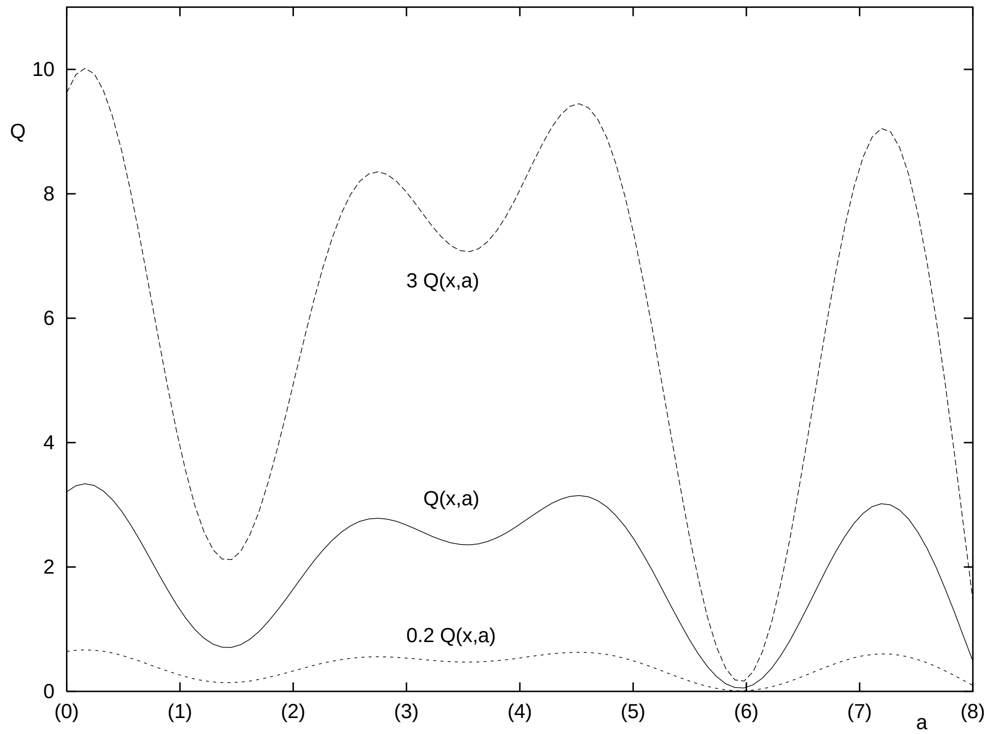


Figure 8.1: Multiplying the reward by a factor multiplies the Q-values by that factor, which either *exaggerates* or levels out the contour $Q_x(a) = Q(x,a)$. The agent will still rank the actions in the same order, and still suggest the same preferred action, but its W-values will be different.

$$W_i(x) = \frac{Q_i(x, a_i) - Q_i(x, a_k)}{\sigma}$$

Multiplying the base Q-values by a constant c would multiply both the mean and the standard deviation by c. But then our W-value would be:

$$W_i(x) = \frac{cQ_i(x, a_i) - cQ_i(x, a_k)}{c\sigma}$$

that is, unchanged. So we would have no easy way of making agents weaker or stronger. Scaling does *not* make agents all equal in strength to each other. But the problem is that it makes their inequalities *fixed*.

Finally, if we can just multiply all the Q-values by a constant, can't we just multiply the W-values by the constant instead of re-learning them? The answer is that a W-value depends on who the current leader is. If we increase r_i in:

$$W_i(x) = c_{ki}(x) r_i$$

then $W_i(x)$ increases, but as soon as it does, everything may change. A_i may go into the lead itself, causing such a huge loss to another agent that it increases its W-value and then takes over the lead, and then A_i's W-value will reflect a completely different loss $c_{li}(x)$. Once a W-value changes, we have to follow the whole re-organisation to its conclusion.

8.1.3 No Global reward function

The agents learn their Q-values from their local reward functions and then organise their action selection by W-learning, all without any reference to the global reward function of §4.3.1. In this work, for purposes of comparison with the other action selection methods, we will now want to test the fitness of the creature.

Obviously, if the global reward function of §4.3.1 still *defines* what we are looking for, we still need to use it to score the fitness of the collection. But it no longer need be available to the agents as an *explicit* function they can learn against. It is only used to test them. Hence the fitness function could be just *implicit* in the environment, as in the best Artificial Life research [Ray, 1991, Todd et al., 1994].

As has been said many times in contrasting natural evolution with the standard Genetic Algorithm, living things *don't have* an explicit global reward defined or available to learn from. Their fitness test is only implicit in their environment - whether they manage to live or die. How exactly they replicate is up to them, and what persists over time is often a surprise to the observer.

Similarly, as we give our artificial creature more complex multi-goal tasks, the global reward functions become much harder to design than the local ones (as we argued in §4.4.3).

Imagine that we know what behavior we want when we see it, but we're having trouble designing a suitable multi-reward global reward function. So we adopt the strategy of tweaking the r_i's by hand, letting the agents re-organise themselves, and seeing the result. Say agent A_i is not being expressed in the creature's behavior. We slowly increase r_i until it first starts to win the one or two absolutely crucial states that it needs. As we increase r_i further, it will win more and more extra states until eventually it would dominate the creature. And so on.

We don't want to design all the explicit behavior, but at the same time we do have some ideas as to what is suitable behavior and what is not, so we do not want to simply design an abstract set of global values as in §4.3.1 and hope for the best. What our decomposition into multiple reward functions defended by competing agents gives us is effectively an *adjustable* global reward function. We increase the strength *in general* of particular agents, while still letting the agents sort out the details.

8.2 Analysis of best solution

So it would not necessarily be difficult to design the collection of r_i's by hand, increasing ones that aren't being expressed, and so on. To test a combination of r_i's, we multiply the base Q-values by them, and then re-run the W-competition.

In fact, in this work I use a simple Genetic Algorithm to automate the search. The genotype encodes a set of r_i's in chromosomes of length 4 (quite a coarse-grained evolutionary search). We have a population of size 60, initially randomised, evolving for 30 generations. The advantage of an automated search is that I can be confident that I have put the same amount of effort into finding good solutions for each method.

For this test (W-learning with subspaces on the House Robot problem) the best combination of r_i's found by GA search was:

$r_d = 0.93$
$r_p = 0.01$
$r_c = 0.41$
$r_w = 0.01$
$r_u = 0.54$
$r_s = 0.60$
$r_m = 0.67$
$r_f = 0.67$

which averages 13.446 per 100 steps, slightly less than we got with Hierarchical Q-learning, but achieved with a reduction in memory requirements from 9.6 million to 1600.

We have solved the problem not with one complex entity, but via the complex *interactions* of multiple simple entities.

Looking closer at our solution, note that A_w, as predicted, is useless. With such a tiny reward it will not be expressed at all. Neither, interestingly, will A_p - obviously other agents are managing to take the creature back to the plug often enough for it to not be needed. The creature as a whole works by interleaving all its goals. Here are the strongest few W-values:

```
Ws(7,2)        0.499
Ws(3,2)        0.413
Ws(0,2)        0.337
Ws(0,0)        0.257
Ws(2,2)        0.243
Wu(7,0)        0.240
Wd(4,0)        0.177
Wd(5,0)        0.176
Wd(1,0)        0.163
Wd(2,0)        0.131
Wd(0,0)        0.126
Ws(5,0)        0.119
Ws(6,2)        0.088
Wd(7,0)        0.085
Wd(6,0)        0.084
Wf(2,0)        0.078
Wf(4,0)        0.076
Ws(3,0)        0.070
Wf(6,0)        0.068
Wd(3,0)        0.068
Wf(0,0)        0.062
Ws(1,0)        0.061
Wf(1,0)        0.061
Ws(2,0)        0.048
Wf(5,0)        0.042
```

```
Wf(3,0)        0.042
Ws(4,0)        0.040
Wf(8,0)        0.037
Wf(7,0)        0.031
Ws(9,2)        0.030
Wd(8,0)        0.028
Wf(7,1)        0.017
Wc(3)          0.017
...
```

We can see that there is a complex intermingling of W_s, W_d and W_f. The states $(*, 2)$ (as seen by A_s) are those where a human has been identified as a stranger. These are the crucial states for A_s since it can pick up a continuous reward if it keeps the human in sight. The states $(*, 0)$ (as seen by A_f) are those where fire is visible without a wall in the way. These build up higher W-values (A_f is more confident about what to do) than when there is a wall in the way, where A_f will need some kind of stochastic policy.

Here are the probabilities of each action being suggested by A_f when the fire is in direction 4, behind a wall. The higher probability actions under a soft max control policy (§2.2.3) are highlighted.

```
 (a)             (0)       (1)       (2)       (3)       (4)
Qf((4,1),a)     0.028     0.029     0.036     0.033     0.034
 p(a)           0.078     0.084   [ 0.154] [ 0.121] [ 0.127]

 (a)             (5)       (6)       (7)       (8)
Qf((4,1),a)     0.035     0.036     0.030     0.025
 p(a)         [ 0.135] [ 0.147]    0.091     0.063
```

We can see that A_f builds up a broad front in approach to the wall. Moving at right angles to the direction of the fire (directions 2 and 6) is good because it is more likely to see the end of the wall. In any case, when the route to the fire is blocked by a wall, A_f is amenable to suggestions by other agents, in particular by the combination of A_c and A_d, who drive the house robot in a strong wandering behavior otherwise.

A_p with its tiny reward is irrelevant - its job is done for it by A_c bringing the creature towards the centre and hence quite often past the plug. A_u has a not insignificant reward, but finds its job is done for it by A_s, which also wants to investigate unidentified humans (in case they turn out to be strangers). So A_u lets A_s do all the work for it, and as long as A_s is being obeyed by the creature, A_u is happy:

```
Wu(0,0)        -0.182
Wu(1,0)        -0.193
Wu(2,0)        -0.179
Wu(3,0)        -0.152
Wu(4,0)        -0.123
Wu(5,0)        -0.112
Wu(6,0)        -0.130
Wu(7,0)         0.240
```

The sole exception is state $(7, 0)$, which for some reason fell to A_u to be responsible for, while A_s took its turn at dropping out of the competition:

```
Ws(0,0)         0.257
Ws(1,0)         0.061
Ws(2,0)         0.048
Ws(3,0)         0.070
Ws(4,0)         0.040
Ws(5,0)         0.119
Ws(6,0)         0.013
Ws(7,0)        -0.108
```

Chapter 9

W=Q (Maximize the Best Happiness)

The first response to W-learning is to ask if we need such an elaborate value of W. Why not simply have actions promoted with their Q-values, as we originally suggested back in §5.3. The agent promotes its action with the same strength no matter what (if any) its competition:

$$W_i(x) = Q_i(x, a_i)$$

and we search for an adaptive combination of r_i's as before. To test a particular combination of r_i's, we just multiply the base Q-values by them and then see how the creature performs under the rule W=Q. There are no W-values to learn.

If the agents share the same suite of actions, W=Q is equivalent to simply finding the action:

$$\max_{a \in A} \max_{i \in 1,\ldots,n} Q_i(x, a)$$

since agents suggest their best Q over a and we take the highest W=Q over i. That is, we are only interested in the best possible *individual* happiness. We are going to start drawing economic analogies to our various approaches. In economic theory, this would be the equivalent of a *Nietzschean* social welfare function [Varian, 1993, §30], where the value of an allocation depends on the welfare of the best off agent.

The counterpart of this method would be:

$$\min_{a \in A} \min_{i \in 1,\ldots,n} (Q_i(x, a_i) - Q_i(x, a))$$

that is, find the action which leads to the smallest unhappiness for someone and take it. This approach is pointless because it means just obey one of the agents and cause unhappiness zero for them.

I have not seen an example of straightforward use of W=Q in Reinforcement Learning, but it can hardly be an original idea. What look like examples [Rummery and Niranjan, 1994] turn out only to be using multiple neural networks for storing Q-values $Q_a(x)$ in a monolithic (single reward function) Q-learning system (§4.3.2) and then letting through the action with the highest Q-value.

Searching for combinations of r_i's under W=Q works very well, and finds the following collection which achieves a score of 15.313. Further, the memory requirements are even less, since no W-values at all are kept.

$r_d = 0.93$
$r_p = 0.41$
$r_c = 0.41$
$r_w = 0.08$
$r_u = 0.80$
$r_s = 0.14$
$r_m = 0.08$
$r_f = 1.00$

9.1 Discussion

So have we wasted our time with measures of W that make compromises with the competition? Would we have been better off ignoring the competition completely?

It seems on paper that W=Q should not perform so well, since it maximizes the rewards of only one agent, while W-learning makes some attempt to maximize their collective rewards (which is roughly what the global reward is). Consider the following scenario, where there are two possible actions (1) and (2). The agents' preferred actions are highlighted:

```
a          (1)      (2)
Q1(x,a)    [1.1]    1
Q2(x,a)    0        [0.9]
```

If we use W=Q, then agent A_1 wins (since $1.1 > 0.9$), so action (1) is executed, A_1 gets reward 1.1, and A_2 gets 0. If we use the $W = (D - f)$ method, then A_2 wins (since it would suffer 0.9 if it didn't, while A_1 would only suffer 0.1 if disobeyed), so action (2) is executed, A_1 gets 1, and A_2 gets 0.9. If the global reward / fitness is roughly a combination of the agents' rewards, then $W = (D - f)$ is a better strategy. In short, this is the familiar ethology problem of *opportunism* - can A_2 force A_1 into a small diversion from its plans to pick up along the way a goal of its own?

There's one way our W=Q search will find to solve this - by just finding a high r_2 so that it becomes:

```
a          (1)      (2)
Q1(x,a)    [1.1]    1
Q2(x,a)    0        [1.2]
```

But this is an unsatisfactory solution because it assumes that it is A_2 that always needs high Q-values in order for the two agents to behave opportunistically. What if in another state y, the situation is reversed and it is A_1 trying to ask A_2 for a slight diversion:

```
a          (1)      (2)
Q1(x,a)    [1.1]    1
Q2(x,a)    0        [0.9]

a          (1)      (2)
Q1(y,a)    0        [0.9]
Q2(y,a)    [1.1]    1
```

Ideally we would take action (2) in both states. But W=Q will be unable to prevent action (1) being taken in at least one of the states. Currently, agent A_2 is losing state x and winning state y. We want it to win state x and lose state y. If we increase r_2 to make it win state x, we increase *all* Q-values across the board and make it even less likely to lose state y.

W=Q will not be able to find the opportunistic solution in cases like this, whereas W-learning will. And cases like this will be typical. Agents that ask for opportunities from other agents will themselves be asked for opportunities at other times.

In fact, any of our static measures of W from §5.4, such as:

$$W_i(x) = Q_i(x, a_i) - \min_{b \in A} Q_i(x, b)$$

would fail to be opportunistic in situations where W-learning would be. When there are more than two actions, the other agent might not be taking the *worst* action for A_i, perhaps only the second best.

So, if we agree that W-learning will find opportunism where W=Q (or any static measure) cannot, why did W-learning not perform better? The answer seems to be that the House Robot environment does not contain problems of the nature above. It contains situations where in state x, A_2 wants to slightly divert A_1 alright, but only in situations where A_2 itself doesn't mind being diverted - the 0 above becomes a 0.8. This is because all behaviors here are essentially of the form 'if some feature is in some direction, then move in some direction' with rewards for arriving at the feature or losing sight of it. So if $Q_1(x,1) = 1.1$ is similar to $Q_1(x,2) = 1$, it is because actions (1) and (2) are movements in roughly the same direction, in which case $Q_2(x,1)$ and $Q_2(x,2)$ will end up similar.

9.2 Happiness and Unhappiness

Despite its name, Minimize the Worst Unhappiness (W-learning) does *not* mean we're always avoiding disaster. Expected reward and expected disaster are two sides of the same coin, because if the leader is *not* obeyed it will be unhappy. Say we have an agent who if obeyed will gain a high reward. If not obeyed, it won't suffer a punishment, just nothing interesting happens. But it might as well be a punishment since it lost the chance of that reward. It will build up a high W-value under any $(D - f)$ scheme.

So it would be mistaken to think that the difference between Minimize the Worst Unhappiness and Maximize the Best Happiness is that one is concerned with "Unhappiness" and the other with "Happiness". As just noted, these are really the same thing. The real difference between the two approaches is that Minimize the Worst Unhappiness consults with other agents while Maximize the Best Happiness does not consult. Minimize the Worst Unhappiness tries out other agents' actions to see how bad they are. An agent in Maximize the Best Happiness only ever considers its best action.

Chapter 10

W-learning with full space

A further reason why W-learning underperformed is that we still haven't found the ideal version of W-learning. Remember from §6.6 that using only subspaces for $W_i(x)$ results in a loss of accuracy. Using the full space for $W_i(x)$ would result in a more sophisticated competition.

Consider the competition between the dirt-seeker A_d and the smoke-seeker A_f. For simplicity, let the global state be $x = (d, f)$. A_d sees only states (d), and A_f sees only (f). When the full state is $x = (d, 5)$, A_f simply sees all these as state (5), that is, smoke is in direction 5. Sometimes A_d opposes it, and sometimes, for no apparent reason, it doesn't. But $W_f(5)$ *averages* all these together into one variable. It is a crude form of competition, since A_f must present the same W-value in many different situations where its competition will want to do quite different things. The agents might be better able to exploit their opportunities if they could tell the real states apart and present different W-values in each one.

If we are to make the x in the $W_i(x)$ refer to the full state, then each agent needs a single neural network to implement the function. The agent's neural network takes a vector input x and produces a floating point output $W_i(x)$. The Q-values can remain as subspaces of course. We are back basically to the same memory requirements as Hierarchical Q-learning - subspaces for the Q-values and then n times the full state x.

10.1 Strict highest W

Recall (§6.4) that if the winner is to be the strict highest W we start with W random negative, and have the leading $W_k(x)$ unchanged, waiting to be overtaken. This works for lookup tables, but will not work with neural networks. First because trying to initialise W to random negative is pointless since the network's values will make large jumps up and down in the early stages when its weights are untuned. Second because even if we do not update it, $W_k(x)$ will still change as the other $W_k(y)$ change. And if the net doesn't see $W_k(x) \mapsto d$ for a while, it will forget it.

We could think of various methods to try to repeatedly clamp $W_k(x)$, but it seems all would need extra memory to remember what value it should be clamped to.

10.2 Stochastic highest W

The approach we took instead was: Start with W random. Do one run of 30000 steps with *random* winners so that everyone experiences what it's like to lose, and remembers these experiences. Then they each replay their experiences 10 times to learn from them properly. Note that when learning W-values in a neural network, we are just doing updates of the form $W(x) \mapsto d$. No W-value is referenced on the right-hand side, unlike the case of learning the Q-values. Hence there is no need for our concept of backward replay.

With a similar neural network architecture as before, the best combination of agents found, scoring 14.871, was:

$r_d = 0.67$
$r_p = 0.01$
$r_c = 0.80$
$r_w = 0.08$
$r_u = 0.14$
$r_s = 0.60$
$r_m = 0.21$
$r_f = 1.00$

which is better than W-learning with subspaces, but still not as good as W=Q. A problem with this method of random winners is that it will actually build up each $W_i(x)$ to be the average loss over *all* other agents in the lead:

$$W_i(x) = \frac{1}{n-1} \sum_k (Q_i(x, a_i) - Q_i(x, a_k))$$

for $k \neq i$. So what we are doing is in fact finding:

$$\max_i \sum_k (Q_i(x, a_i) - Q_i(x, a_k))$$

This sum doesn't really *mean* anything (see the discussion in §F). For example, it is certainly not the loss that the *current* leader is causing for the agent.

Using random winners is equivalent to a stochastic highest W strategy (§6.5) with fixed high temperature. We would probably have got better results if we had used a more normal stochastic highest W - one with a declining temperature. This would have multiple trials, replay after each trial, and a declining temperature over time as in §4.3.2. But we have some confirmation that telling states apart is a good thing. In the next section, we find out what happens when we can tell states apart perfectly.

Chapter 11

Negotiated W-learning

If other agents' actions are meaningless to it, all an agent can do is observe what r and y they generate, as W-learning does. It could perhaps assume that unknown actions have the effect of 'do nothing' or 'stay still', if they have a Q-value for such an action (§2.1.2), but it might be unwise to assume without observing.

However, if agents share the same suite of actions, and the other agent's action *is* recognised by the agent, it already has built up an estimate of the expected reward in the value $Q_i(x, a_k)$. So rather than learning a W-value from samples, it can assign it directly if the successful action a_k is communicated to it. We can do this in the House Robot problem, since all agents share the same suite of actions ('move' 0-8). In other words, we can follow the walk in §5.6 exactly, we do not have to approximate it.

In Negotiated W-learning, the creature observes a state x, and then its agents engage in repeated rounds of negotiation before resolving competition and producing a winning action a_k. It is obviously to be preferred that the length of this 'instant' competition will be very short. In pseudocode, the Negotiated W-learning method is, each time step:

```
observe state x

start with leader k := random agent and Wk := 0
loop:
  for all agents i other than k
   Wi := Qi(x,ai) - Qi(x,ak)
  if highest Wi > Wk, new leader and goto loop
(loop terminates with winner k)

execute ak
```

This algorithm discovers explicitly in one timestep what W-learning only learns over time.

It also gives us the high accuracy of telling states apart, as in W-learning with full statespace, yet without needing to store such a space in memory. In fact its accuracy will be better than W-learning with full statespace since the latter method has to use a generalisation, whereas Negotiated W-learning can tell states apart perfectly. We deal with each full state x as it comes in at run-time. The agents recognise different components of x and compete based on this one-off event.

We have no memory requirements at all for W. The W_i are just n temporary variables used at run-time. In fact, note that 'Negotiated W-learning' is actually not learning at all since nothing permanent is learnt.

The best combination found, scoring 18.212, was:

$r_d = 0.87$
$r_p = 0.01$
$r_c = 0.54$
$r_w = 0.01$
$r_u = 1.00$
$r_s = 0.08$
$r_m = 0.74$
$r_f = 0.34$

As noted in §5.6.1, the Negotiated W-learning walk may have a different winner depending on which random agent we start the walk going with. Since we run a new competition every timestep, this means that Negotiated W-learning has a *stochastic* control policy. Note that W-learning may have multiple possible winners too, depending on who takes an early lead. But *once a winner is found* it has a deterministic control policy.

We could make Negotiated W-learning have a deterministic control policy by recording the winners $k(x)$ for each full state x so that we don't have to run the competition again. On the other hand, we might have dynamic creation and destruction of agents (see §17.5 later), in which case we would want to re-run the competition every time in case the collection of agents has changed.

Negotiated W-learning could also be used during the learning of Q itself, in which case we will want to re-run the competition each time round with better Q estimates.

11.1 Reactiveness

Theorem 5.1 shows that the instant competition (or walk through the matrix) is bounded. To be precise, in the algorithm described above, the shortest possible loop would be: Start with random leader. All other $W_i = 0$, e.g. all agents agree about what action to take, $a_k = a_i \ \forall i$. Loop terminates, length 1.

The longest possible loop would be: Start with random leader, whose W-value is zero. Some other agent has $W_i > 0$ and takes the lead. Try out all other agents, only to come back to the original leader, whose W-value is now non-zero, and it wins. We tried out the original agent, then $(n-1)$ other agents, then the original agent again. Total length $(n+1)$.

So the competition length is bounded by 1 and $(n+1)$. Here $n = 8$ so competitions will be of length 1 to 9. With the combination above, the competition lengths seen over a run of 40000 steps[1] were:

1	234	0.6%
2	27164	68.0%
3	11978	30.0%
4	558	1.4%
5	10	0.025%
6	0	
7	0	
8	0	
9	0	

This gives a (reasonably reactive) average competition length of 2.3, as illustrated in Figure 11.1.

Figure 11.1 also gives us some idea of how quickly our original method of W-learning (§6) actually gets down to a competition between only one or two agents.

[1] Actually, for uninteresting technical reasons, 39944 steps.

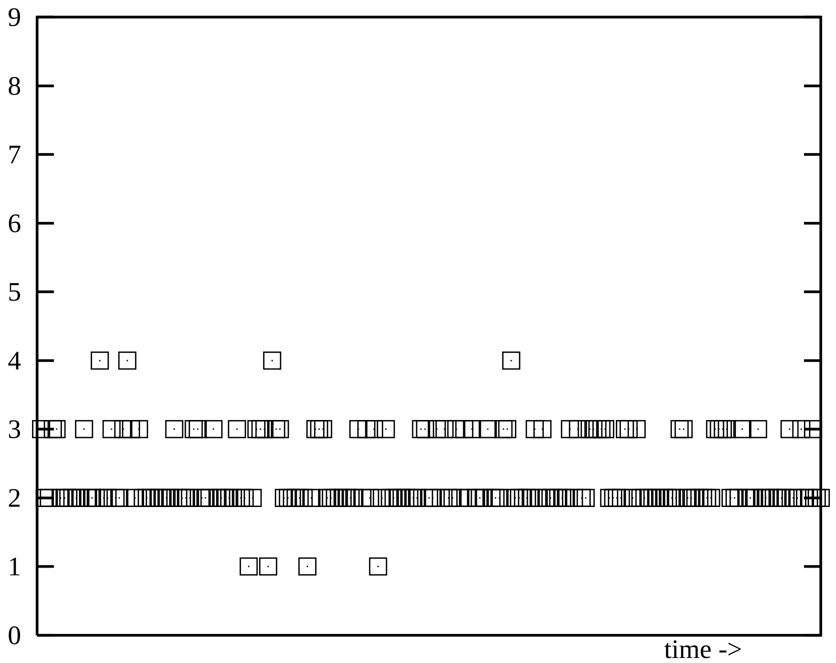


Figure 11.1: The 'reactiveness' of Negotiated W-learning. This is a typical snapshot of 200 timesteps, showing how long it took to resolve competition at each timestep. The theoretical maximum competition length here is 9.

Chapter 12

Collective methods

For completeness we now describe various *Collective* methods, though they have not been tested. As will be discussed, we don't expect these Collective methods to perform better, but it is still instructive to compare them with the singular methods.

12.1 Maximize Collective Happiness

First, if the global reward is roughly the sum of the agents' rewards, maybe we should *explicitly* maximize collective rewards. If the agents share the same suite of actions, we can calculate:

$$\max_{a \in A} \left[\sum_{i=1}^{n} Q_i(x,a) \right]$$

Note that this may produce *compromise* actions. The winning action may be an action that none of the agents would have suggested. In economics, this method would be equivalent to the classic *utilitarian* social welfare function [Varian, 1993, §30] (the greatest happiness for the greatest number).

If the agents don't share the same suite of actions, it's hard to see what we can do. We can't *predict* the happiness of other agents if one agent's action is taken. We can only try it and observe what happens. This leads to the following method.

12.2 Collective W-learning (Minimize Collective Unhappiness)

The counterpart of the above is:

$$\min_{a \in A} \left[\sum_{i=1}^{n} (Q_i(x, a_i) - Q_i(x, a)) \right]$$

That is, if agents share the same set of actions. We call this pure Minimize Collective Unhappiness.

If agents don't share the same actions, we can approximate this by a process we will call Collective W-learning. Each agent builds up a value $W_i(x)$ which is the sum of the suffering it causes *all the other agents* when it is being obeyed. We look for the *smallest* $W_i(x)$. Like W-learning, agents observe r_i and y, and build up their deficits over time. We *only* update the leader $W_k(x)$. In pseudocode, the Collective W-learning method is, each time step:

```
observe state x
find Wk(x) = lowest Wi(x)
execute ak

Wk(x) ->   sum    (Qi(x,ai) - reward for Ai)
       over all agents i
       other than k
```

The change of $W_k(x)$ means that there might be a different lowest W next time round in state x, and so on. As with the W-values in singular W-learning, $E(W_k(x)) \geq 0$.

Remember that in singular W-learning, we don't mind if an agent never experiences the lead because agents outside the lead are still being updated, and it obviously hasn't taken the lead because its W-value isn't strong enough. In Collective W-learning, on the other hand, we *do* want every agent to experience the lead since that's the only way we get any estimate of the W-value.

12.2.1 Strict lowest W

Because we update the leader's W-value only, there is again a problem with initialisation of W-values. In singular W-learning, if an agent has a high initial W-value, it wins for no good reason. In Collective W-learning, if an agent has a high initial W-value, it never gets to try being the leader, for no good reason. If *all* agents except one have unluckily high, positive, initial W-values, the leader converges to its true W-value somewhere lower and wins for no good reason.

So for different reasons, we have the same initialisation strategy for both - start with all $W_i(x)$ zero or random negative.

12.2.2 Stochastic lowest W

Alternatively, as in singular W-learning, we could avoid the initialisation strategy by choosing the winner stochastically. Here this is the *lowest* W we choose stochastically. To be precise, when we observe state x, we obey agent A_k with probability:

$$p(k) = \frac{e^{\frac{-W_k(x)}{T}}}{\sum_i e^{\frac{-W_i(x)}{T}}}$$

Note that $\sum_i p(i) = 1$.

12.2.3 Collective W-learning with subspaces

Consider the plug-seeking agent A_p, building up a W-value $W_p(5)$, of the total deficits it causes other agents when it is obeyed when the plug is in direction 5 and it moves in direction 5. What would this total deficit be? It would just be the average deficit over *all* states for the other agents of taking action 5, which is likely to be meaningless. It is a far cruder form of competition even than W-learning with subspaces.

Collective W-learning needs the W-values to refer to the full space to work.

12.2.4 Negotiated Collective W-learning

If agents *do* share common actions, we can do an instant 'Negotiated' version rather than waiting for W-values to build up over time. In pseudocode, the Negotiated Collective W-learning method is, each time step:

```
observe state x

for all agents k
  Wk :=   sum    (Qi(x,ai) - Qi(x,ak))
        over all agents i
        other than k
find lowest Wk

execute ak
```

But this would not really have any advantages over the pure Minimize Collective Unhappiness. This leads to the question of can we replace Negotiated W-learning itself by a pure Minimize the Worst Unhappiness, which we shall ask in §13.

12.3 Expected performance of Collective methods

We expect that any collective method will generate a similar sort of behavior - keeping the majority of agents happy at the expense perhaps of a small minority. Collective approaches are probably a bad idea if there are a *large* number of agents. The creature will choose safe options, and no one agent will be able to persuade it to take risks. Even if one agent is facing a *non-recoverable* state (where if it is not obeyed now, it cannot ever recover and reach its goal in the future), it may still not be able to overcome the opinion of the majority. Consider how Maximize Collective Happiness deals with this:

a	(1)	(2)
Q1(x,a)	[10]	0
Q2(x,a)	3	[5]
Q3(x,a)	3	[5]
Q4(x,a)	3	[5]
Q5(x,a)	3	[5]
Q6(x,a)	3	[5]
Q7(x,a)	3	[5]

This is a crucial state for agent A_1. To get such a big difference between its Q-values this must be a non-recoverable state. If it could take action (2) and then return to this state fairly quickly its Q-value would be higher, something like:

$$Q_1(x, a) = 0 + \gamma 0 + \gamma^2(10)$$

Obviously if A_1 fails now, it can't return here easily. But Maximize Collective Happiness chooses action (2). It ends up not doing very much, giving mediocre rewards to the other 6 agents at the expense of losing the first agent, perhaps forever. It's possible that, even if the global reward is roughly the sum of the agents' rewards, Collective Happiness may still not be the best strategy, because losing A_1 means that it can't contribute to the total reward in the future. It's the whole creature equivalent of going for short-term gain even at the expense of long-term loss. At moments like this, our action selection scheme should keep A_1 on board.

As in the discussion about W=Q (§9.1) we could say why not increase r_1 until A_1 tips the balance in favour of action (1). and again the answer is that this would increase all of A_1's Q-values, not just those in state x.

Collective W-learning (Minimize Collective Unhappiness) will give the same result as Maximize Collective Happiness here. If A_1 is in the lead, the sum $W_1(x) := 12$. If any other agent A_i is in the lead, its sum $W_i(x) := 10$ and it will win. Action (2) gets executed. The basic problem with a collective approach is that an individual agent must effectively beat the *sum* of the other agents. This is alright if its Q-values are on a different (larger) scale to theirs. But we can't have *all* agents' Q-values on a different scale. In W-learning, individuals only have to beat other individuals.

Finally, situations can be found which are favourable to a Collective approach. Consider this case:

a	(1)	(2)	(3)	(4)
Q1(x,a)	[1]	0	0	0.99
Q2(x,a)	0	[1]	0	0.99
Q3(x,a)	0	0	[1]	0
Q4(x,a)	0	0	0	[1]

Here a Collective approach is best (action (4)). If we start listening to agent A_3, we will jump to action (3), but then other agents will start complaining and in general we risk ending up with an action that causes disaster for three agents instead of just one.

This is the question, can we allow agent A_i, in pursuit of its action, to cause a loss less than or equal to its own for *all* the other agents? What may answer this question is that it is probably more likely that just *one* agent is facing disaster while all the others are living normally, then it is that all agents are in danger of disaster at the same time. So it will very rarely be a case of one agent dragging everyone else down with it, more likely just one agent saving itself.

In other words, situations like the above will tend not to happen. The agents' disaster zones will be spread over different states. A crucial state to one will be a humdrum state to the others. So listening to individual stories may not be a dangerous strategy.

12.4 Collective Equality

Note that maximizing the sum of happiness (or the average happiness, which is the same thing, divided by n) is not necessarily the same as spreading the happiness round lots of agents. If an agent A_i's Q-value *is* big enough it can outweigh all the others. One agent receiving 100 with nine other agents receiving zero is exactly equivalent under the Maximize Collective Happiness method to all agents receiving 10 each. If we wanted to favour the second of these, we would have to take some kind of *standard deviation*-based approach. For example, we could calculate:

$$\min_{a \in A} \left[\sum_{i=1}^{n} (Q_i(x,a) - M)^2 \right]$$

where the mean $M = \frac{1}{n} \sum_{i=1}^{n} Q_i(x,a)$. But just minimising the standard deviation of the distribution is still not quite what we want. We don't want all Q the same if they're all going to be low.[1] Similarly for other equality measures (such as minimising the difference between the richest and poorest). We can see how in the trade off between equality and wealth in the Action Selection problem, Minimize the Worst Unhappiness is beginning to look like what we need.

[1] Continuing our economic analogy, this would presumably be *communism*. We may (or may not) be interested in equality at all costs in real human society - I have no interest in making statements about that. But in the Action Selection problem, it is clearly not what we want.

Chapter 13

Minimize the Worst Unhappiness (revisited)

The consensus of the last few chapters is that a Minimize the Worst Unhappiness approach is expected to be a better Action Selection strategy than either Maximize the Best Happiness or a Collective method. But we mentioned in §12.2.4 that we may not yet have found our ideal Minimize the Worst Unhappiness method.

Consider why W-learning approximates a *walk* through the matrix rather than a global decision. In W-learning, agents don't share the same actions, so they can only see how bad things are by trying them out. It's impractical to let every agent experience what it is like when every other is in the lead, and experience it for long enough to build up the *expected* loss. W-learning gets down to a winner a lot quicker by just putting someone in the lead and leaving it up to the others to overtake.

Negotiated W-learning concentrates on copying W-learning in one timestep. But once we share common actions, and we can draw up the matrix, we can do anything. So let us return to that question of §5.6: Given a $E(d_{ki}(x))$ matrix, who *should* win?

For example, should we search for the highest $E(d_{ki}(x))$ in the matrix? Should we search for the worst any agent could *possibly* suffer and then let that agent win in case it does. For example, here agents A_1 or A_2 would always win to prevent the other one winning (and loss of 1):

```
a          (1)       (2)      (3)      (4)
Q1(x,a)   [1]         0        0.7      0.99
Q2(x,a)    0        [1]        0.7      0.99
Q3(x,a)    0.5       0.5     [1]        0.5
Q4(x,a)    0.7       0.7       0.7     [1]
```

But consider what happens with the W-learning walk. Say we start with A_4 in the lead. A_3 is suffering 0.5, and goes into the lead with $W_3 = 0.5$. All other agents are now suffering 0.3, so A_3 wins.[1] Agents A_1 and A_2 never got to experience each other's disasters since neither ever tried out the lead. The other agents gave them a way of avoiding each other. Of course, here the outcome of the walk depends on which random agent we start with. If we did start with A_1 or A_2 then we would get a straight competition between them.

So searching for the worst $E(d_{ki}(x))$ is a *pessimistic* method. W-learning checks whether these hypothetical $E(d_{ki}(x))$ will actually come to pass. Instead of asking what is the worst deviation an agent could possibly suffer, W-learning asks what it is *likely* to suffer if it does not win. We don't want to be afraid of an event that is not going to happen.

So what should our global decision be? How about instead of starting the walk with a random action, start with the action that satisfies Maximize Collective Happiness and let W-learning run from there. But this doesn't help because W-learning will (almost) always switch from the initial leader who has $W = 0$. So how about we look into the future before we switch. Start with the action that satisfies Maximize Collective Happiness and only switch if the agent currently suffers a loss bigger than any it will cause when it gets the lead. That is, only switch if you can guarantee you will win. Competition lengths will all be 1 or 2.

To summarise, the W-learning walk is similar to a town hall meeting where everyone is agreed except for one person. You can't just ignore their problem, but if they get their way, someone else will be even more annoyed. And so on, and you follow a chain leading who knows where. It would be simple if you could just take a majority vote, but here we're trying to respect individual stories.

13.1 Pure Minimize the Worst Unhappiness

If we could make a global decision, the decision we want is probably something like John Rawls' *maximin principle* from economics [Varian, 1993, §30]. This says that the social welfare of an allocation depends only on the welfare of the worst off agent. In our terminology this would be:

$$\max_{a \in A} \min_{i \in 1,\ldots,n} Q_i(x,a)$$

[1] Note in passing here that one agent suffering 0.5 counts more than 3 agents suffering 0.3. Numbers of agents don't count in W-learning - we only look at individual stories.

which would be a fairly sensible method, although for an agent A_i to make sure it wins state x, its strategy has to be to have the lowest absolute Q-values in all the actions *other* than a_i to wipe these actions out from the competition. A single other agent A_j with very low Q-values across the board could spoil its plans. The creature would end up taking random actions, controlled by the agent A_j that can't score anything anyway.

The counterpart is really what we want:

$$\min_{a \in A} \max_{i \in 1,\ldots,n} (Q_i(x, a_i) - Q_i(x, a))$$

We call this pure Minimize the Worst Unhappiness. Changing to any other action will cause a worse worst-sufferer. This will choose action (3) above (worst sufferer 0.3). W-learning approximates this when actions are not shared. After the W-learning competition is resolved, changing to the *previous* action will cause a worse worst-sufferer though changing to some other untried action may not.

In W-learning, an agent is suffering, so we have a change of leader, *hopefully* to a situation where no one is suffering as much but in fact sometimes to a situation where someone is now suffering *more*. They will then go into the lead, and so on. Only at the very last change of leader is the worst suffering reduced by the change. Change continues until an optimum is reached where the current worst sufferer cannot be relieved without probably creating an even worse sufferer. The W-learning walk may stop in a local optimum. Minimize the Worst Unhappiness can simply look at the matrix and calculate the global optimum.

And while Negotiated W-learning restricts itself to suggested actions only, the pure Minimize the Worst Unhappiness may pick a different, *compromise* action. It doesn't have to be a compromise - a single agent can still take over if the differences between its Q-values are big enough, in which case it can wipe out all actions other than a_i from the competition. Like Negotiated W-learning, Minimize the Worst Unhappiness gives us all the accuracy of the full space without any memory requirements (as indeed do all the explicit methods where actions are shared).

In short, if agents share the same suite of actions, then taking the Negotiated W-learning walk is of no advantage. Instead we should use the global decision of pure Minimize the Worst Unhappiness.

If agents don't share the same suite of actions, all we can do is repeatedly try things out and observe the unhappiness. We must use W-learning with subspaces or full space to get an *approximation* of pure Minimize the Worst Unhappiness. Using stochastic highest W (§6.5), being a simulatedannealing-like process of a high temperature cooling down over time, will make the walk more likely to settle on the global optimum (pure Minimize the Worst Unhappiness) and less likely to stop on some local optimum.

13.2 Global v. Decentralised Calculations

Somehow Negotiated W-learning still seems more *plausible* than pure Minimize the Worst Unhappiness, since it involves decentralised communication among agents rather than a global-level calculation.

Negotiated W-learning might be better if the number of actions is very large (or continuous) - pure Minimize the Worst Unhappiness has to cycle through them. However, the answer there might be that Minimize the Worst Unhappiness could restrict itself to just the suggested actions (that's what Negotiated W-learning does). And the global decision on these would still be better than the walk:

$$\min_{a \in a_1, \ldots, a_n} \max_{j \in 1, \ldots, n} (Q_j(x, a_j) - Q_j(x, a))$$

Chapter 14

Summary

14.1 The four approaches

There are four basic approaches to organising action selection without reference to a global reward. When the agents share the same set of actions, we can calculate all four approaches immediately. They are:

- **Maximize the Best Happiness**

$$\max_{a \in A} \max_{i \in 1,\ldots,n} Q_i(x, a)$$

  This is equivalent to W=Q and can be implemented in that form when actions are not shared (or indeed when they *are* shared).

- **Minimize the Worst Unhappiness**

$$\min_{a \in A} \max_{i \in 1,\ldots,n} (Q_i(x, a_i) - Q_i(x, a))$$

  When actions are not shared, this can be approximated by W-learning.

- **Minimize Collective Unhappiness**

$$\min_{a \in A} \left[\sum_{i=1}^{n} (Q_i(x, a_i) - Q_i(x, a)) \right]$$

  When actions are not shared, this can be approximated by Collective W-learning.

- **Maximize Collective Happiness**

$$\max_{a \in A} \left[\sum_{i=1}^{n} Q_i(x,a) \right]$$

  This approach can only be implemented at all if actions are shared. But in fact, Minimize Collective Unhappiness is pretty much the same thing, and that can be approximated by Collective W-learning when actions are not shared.

There are of course other combinations of maximizing, minimizing and summing which do *not* make any sense. For the full list see §F.

The first approach above has a counterpart discussed in §9. The second approach has a counterpart discussed in §13. The third and fourth approaches are counterparts of each other.

	Memory requirements n no.agents x subspace X full space	No. updates per timestep when learning	No. updates per timestep when exploiting	Ability to discriminate states
Q-learning	1.Xa	1	0	Partial
Hierarchical Q-learning	n.xa + 1.Xn	n.1 + 1	0	Partial
Max Best Happ. (W=Q)	n.xa	n.1	n.1	Full
Min Worst Unhapp.	n.xa	n.1	—	Full
W-learning (subspaces)	n.xa + n.x	n.1 + (n-1).1	0	Poor
W-learning (full space)	n.xa + n.X	n.1 + (n-1).1	0	Partial
Negotiated W-learning	n.xa	n.1	1 to n+1	Full
Min Coll. Unhapp.	n.xa	n.1	—	Full
Coll. W (subspaces)	n.xa + n.x	n.1 + 1.(n-1)	0	Poor
Coll. W (full space)	n.xa + n.X	n.1 + 1.(n-1)	0	Partial
Negotiated Coll. W	n.xa	n.1	n.(n-1)	Full
Max Coll. Happ.	n.xa	n.1	—	Full

General comparisons between the action selection methods.

A dash indicates 'not applicable' here.

'Number of updates per timestep' is modelled on Watkins [Watkins, 1989] where he wanted to impose limited computational demands on his creature per timestep. The format here is (number of agents) times (updates per agent). We are looking for updates that can be carried out in parallel in isolation.

In the 'Ability to discriminate states' column, 'Full' indicates complete ability to discern the full state. 'Partial' indicates that the ability to discern the full state depends on the generalisation. 'Poor' indicates that agents see subspaces only.

Are there any missing methods here? How about W=Q with full space? That would be pointless since that's just Q with full space. The agent will just learn the same Q-values repeated (§4.4.1). Similarly, Maximize Collective Happiness with full space is just Q with full space. The W-learning methods are approximations of the approach to which they belong when actions are not shared. In which case, the practical solution is to try out being disobeyed and observe the Unhappiness, as opposed to expanding our action set and learning new Q-values. There is no equivalent of a W-learning method for the Happiness approaches.

	Memory requirements	No. updates per timestep (per agent) when learning	No. updates per timestep (per agent) when exploiting	Best solution found
Hand-coded program (strict hierarchical)	—	—	—	8.612
Q-learning	10800000	1	0	6.285
Hierarchical Q-learning	9601440	2	0	13.641
Max Best Happ. (W=Q)	1440	1	1	15.313
Min Worst Unhapp.	1440	1	—	n/tested
W-learning (subspaces)	1600	2	0	13.446
W-learning (full space)	9601440	2	0	14.871
Negotiated W-learning	1440	1	average 2.3	18.212
Min Coll. Unhapp.	1440	1	—	n/tested
Coll. W (subspaces)	1600	8	0	n/tested
Coll. W (full space)	9601440	8	0	n/tested
Negotiated Coll. W	1440	1	7	n/tested
Max Coll. Happ.	1440	1	—	n/tested

Comparisons between the methods as applied in the House Robot problem.

A dash indicates 'not applicable' here.

Full details of the experiments, and in particular how we end up comparing a single score for each method, are in §G. For discussion of the *spread* of scores for each method see §16.3.

Here we show the Number of updates per timestep per agent if the system is totally parallelised (each agent has its own processor). Remember here $n = 8$.

Actually for testing Hierarchical Q-learning, I used $n = 5$ to reduce the size of the $Q(x, i)$ space. The memory requirements shown here are for $n = 8$.

	Need an explicit global reward function?	Full statespace must exist somewhere?	Agents must share same suite of actions?
Q-learning	Yes	Yes	Yes
Hierarchical Q-learning	Yes	Yes	No
Max Best Happiness (W=Q)	No	No	No
Min Worst Unhappiness	No	No	Yes
W-learning (subspaces)	No	No	No
W-learning (full space)	No	Yes	No
Negotiated W-learning	No	No	Yes
Min Collective Unhappiness	No	No	Yes
Collective W-learning (subspaces)	No	No	No
Collective W-learning (full space)	No	Yes	No
Negotiated Collective W-learning	No	No	Yes
Max Collective Happiness	No	No	Yes

Restrictions on Decentralisation.

'No' is good here.

There is 'no free lunch' in decentralisation. If we want a totally decentralised model, with separate state and action spaces, we have to use either a static method like W=Q, with its inability to support opportunism, or W-learning with subspaces, with its inaccurate competition. In a world where opportunism was more important, W-learning with subspaces wouldn't necessarily be worse than W=Q, as it was here.

14.2 Winner-take-all v. Compromise actions

The action selection methods can be categorised based on the action they choose:

- Hierarchical Q-learning, Maximize the Best Happiness (W=Q), W-learning (all types) and Collective W-learning (all types) are *winner-take-all* action selection schemes. There is a winner A_k that gets its exact action a_k executed. That is, we only search among the suggested actions a_i.

  In fact these are qualified winner-take-all schemes because control may switch from agent to agent per timestep. The winner isn't guaranteed control of the body for some n timesteps.

  In W-learning, agents suggest only their favourite actions, and we pick one of these. They do not reveal their second or third favourites. In fact though, with multiple agents all suggesting different actions a_i, something in effect like a search through multiple actions looking for a compromise may occur (although we do not necessarily try out even all a_i before getting down to a winner).

- Minimize the Worst Unhappiness, Minimize Collective Unhappiness and Maximize Collective Happiness are schemes where we may get a *compromise action*. The executed action may be an action that none of the agents would have suggested. That is, we search among all actions a.

  For a compromise action, agents really need to share the same suite of actions because if they don't share actions, they can't *predict* how another action will affect them. All they can do is *observe* what happens when the action is taken. And then it would be rather impractical to try out executing *every* single action a. W-learning tries out executing only the a_i's (and even then, normally only some of them) before reaching a decision.

  Consider what an elaborate system we would need to find compromise actions (try out executing every single a) when agents don't share the same set of actions. The leader could execute its second best action. Other agents observe how bad that was, and so on. But we would need to keep a memory, since there is nothing to say that the leader's second best action is better for the other agents - it might be even worse!

  Finally, compromise actions are useful for those cases where otherwise we can't avoid total disaster for some agent, but it's important that our scheme doesn't *always* take a compromise action, otherwise nothing will get completed. It should be just a *possibility* under the scheme. In Minimize the Worst Unhappiness, one agent can still force the creature to listen to its exact action if it has big enough differences between Q-values for all other actions.

- Monolithic Q-learning doesn't fit into either of these two groups since there are no competing agents.

Note that none of these are schemes where the actions of agents are *merged*. Merging or averaging actions does not make sense in general but only with certain types of actions. For example, see [Scheier and Pfeifer, 1995], where all agents suggest a speed value for the left motor and one for the right motor. These type of actions are amenable to addition and subtraction.

14.3 Single-Mindedness

We will attempt to summarise both this chapter and the preceding discussion in one diagram.

Collective methods (§12) trample on the individual by trying to keep the majority happy. They are likely both to ruin some individual plans, while at the same time leading to problems with *dithering*, in which no goal is pursued to its logical end.

W=Q (§9) goes to the other extreme in having only one agent in charge, and perhaps suffers because it does not allow *opportunism*. It won't compromise with the second most needy agent.

W-learning may be a good balance between the two, allowing opportunism without the worst of dithering. One agent is generally in charge, but will be corrected by the other agents whenever it offends them too much. W-learning tries to keep everyone on board, while still going somewhere. For a further analysis of how Minimize the Worst Unhappiness gets tasks completed (enforces persistence) see §15.1.3.

[Maes, 1989] lists desirable criteria for action selection schemes, and in it we see this tension between wanting actions that contribute to several goals at once and yet wanting to stick at goals until their conclusion. We can represent this in a diagram of 'single-mindedness' (Figure 14.1).

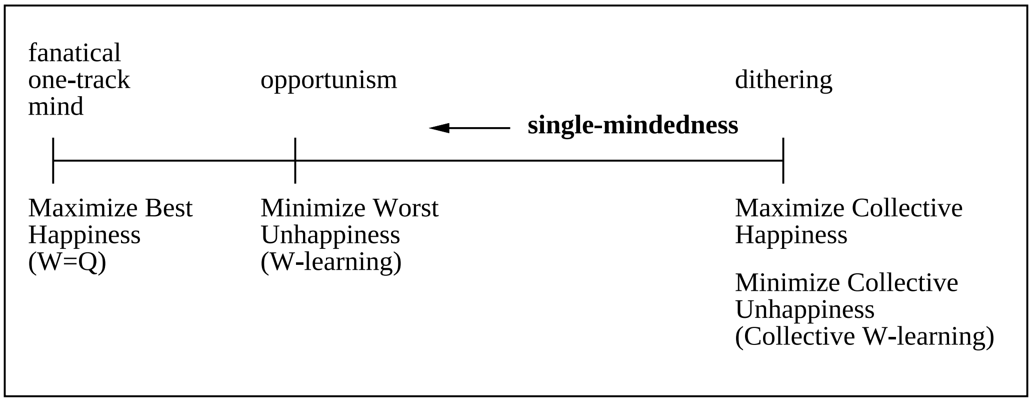


Figure 14.1: The 'single-mindedness' of the methods that organise action selection without reference to a global reward.

Chapter 15

Related work

The action selection methods introduced in this thesis will now be compared to a range of alternative work. Mostly we will be contrasting the work with the Minimize the Worst Unhappiness strategy, and in particular with W-learning.

Many of these action selection schemes mix together what we might call the 'Q-problem' (actions to take in pursuit of a single goal) with the 'W-problem' (choice between conflicting goals). Our methods rigorously separate these two different problems.

15.1 Ethology

First we show how our model addresses various classic problems of animal behavior.

15.1.1 Time-based action selection

The division of control used in all the action selection methods in this thesis is state-based rather than time-based. In interleaving different behaviors, various authors have argued for time-based switching (e.g. see [Ring, 1992]). Blumberg [Blumberg, 1994] argues the need for a model of fatigue, where a switch of activity becomes more likely the longer an activity goes on. He points out that animals sometimes appear to engage in a form of time-sharing.

This is the same philosophy as Lorenz's 'Psycho-Hydraulic' model in ethology. Lorenz's agents have a constant pressure to get executed, increasing over time. This can lead to *vacuum activity* - where an agent has to get expressed just because it's been frustrated for a long time, even if it is irrelevant to the current situation x. Similarly, pressure is reducing on agents that *are* being expressed, which may stop them even though they are not finished their task. While some animals do indeed appear to engage in vacuum activity, Tyrrell [Tyrrell, 1993] argues convincingly that vacuum activity should be seen as a *flaw* in any action selection scheme. Control should switch for a reason.

It is not clear anyway that time-sharing effects cannot be achieved by a suitable state representation x. If an activity goes on for long enough, some internal component of x (that is, some internal sense, e.g.'hunger') may change, leading to a new x and a potential switch in activity.

The advantage of a state-based approach is that x contains a reason *why* control has switched, which we can then analyse. We can discuss who is winning particular states and why, and so on. The analysis of time-based action selection seems much more complex and arbitrary.

In W-learning, control never switches without a reason grounded in the present (in the current state x). I am unconcerned about agents being frustrated for long stretches of time, endlessly suggesting their actions and being disappointed (such as the nest-seeking agent A_n in §7.3). It's not viewed as pain, or Lorenz's water pressure building up. It's only information.

15.1.2 Low-priority activities

A classic ethology problem is: if priorities are assigned to entire activities, how does a low-priority activity ever manage to interrupt a higher-priority one? For example, the conflict between high-priority feeding and low-priority body maintenance discussed by [Blumberg, 1994]. Here is how W-learning would solve it.

The Food-seeking agent A_f suggests actions with weight $W_f(x)$. The Cleaning agent A_c suggests actions with weight $W_c(x)$. Both see the full state $x = (e, f, c)$ where:

- e is external senses
- f is an internal sense, taking values 2 (very hungry), 1 (hungry) and 0 (not hungry)
- c is an internal sense, taking values 2 (very dirty), 1 (dirty) and 0 (clean)

The sense c is irrelevant to the rewards that the Food-seeking agent A_f receives. Both e (whether food is visible) and f are relevant to its strategy. We expect that A_f's reward function $r_f(x, y)$ is a function partly of *internal* sense, giving higher rewards for eating when very hungry (consider it as a chemical rush - then it would be stronger when the creature is very hungry). Then the W-values will work out higher when very hungry too. We will find that for a given e, A_f will rank its W-values:

$$W_f(e,2,*) > W_f(e,1,*) > W_f(e,0,*)$$

Why would the W-values work out like this? Because if it is very hungry and eats, it gets a high reward R, otherwise zero. Hence W (the difference between the two) is high. If it is not hungry and eats, it gets a low reward r, otherwise zero. Hence W (the difference between the two) is low. Actually, because the Q-values represent a discounted look at future steps, the situation will be slightly more complex. It will be more like:

Q(very-hungry,eat) $= R$
Q(very-hungry,not-eat) $= \gamma R$ (we can eat on the next step)
Q(not-hungry,eat) $= r$
Q(not-hungry,not-eat) $= \gamma r$
Hence:
W(very-hungry) $= (1-\gamma)R$
W(not-hungry) $= (1-\gamma)r$

W is still higher when very hungry. Here the W-value is influenced simply by the *size* of the reward.

Similarly for the Cleaning agent A_c. Here both e and f are irrelevant to its rewards. It can clean at any time, irrespective of what its external senses e are. It will rank its W-values:

$$W_c(*,*,2) > W_c(*,*,1) > W_c(*,*,0)$$

So how would the low-priority activity get expressed? A very strong Food-seeker A_f, only rarely interrupted by a Cleaner A_c, would be represented by, for a given e:

$$\begin{aligned} W_f(e,2,*) &> W_f(e,1,*) > W_c(e,*,2) \\ &\quad > W_f(e,0,*) > W_c(e,*,1) > W_c(e,*,0) \end{aligned}$$

in which case A_f wins all the states $(e,2,*)$ and $(e,1,*)$. A_c wins the state $(e,0,2)$. And A_f wins the states $(e,0,1)$ and $(e,0,0)$. The creature only cleans itself when it is very dirty *and* not hungry. Otherwise, it feeds.

The typical weak, low-priority agent only manages to raise its head in one or two extreme states x which are its very last chance before total disaster.

It has been complaining all along but only at the last moment can it manage a W-value high enough to be obeyed.

Note that by feeding continuously, the state may change from $(e, 2, 2)$ to $(e, 0, 2)$, in which case there is a switch of control. But this switch of control has a reason - it is not fatigue with the feeding action, it is the movement into a new state x. The *effect* of time-based switching occurs when the state-based creature is in continuous interaction with a changing world.

Having internal hunger that increases over time does not necessarily break our MDP model. As [Sutton, 1990a] points out, we simply redraw the boundary between the creature and its environment. Where x is 'hungry', y is 'very hungry', and a is 'do nothing', we might have say $P_{xa}(x) = 0.9$ and $P_{xa}(y) = 0.1$ for any timestep.

To enable a state-based approach to action selection, we do in fact assume that goals generally decline in urgency after they have been in charge of the creature (and thus able to satisfy themselves) for a long time. Or at least that other goals increase in urgency if not pursued for a long time. This is actually quite a weak assumption. Most problems with any potential for goal-interleaving can be made to fit into this scheme, with the reward-generation of an agent having the potential to be at least *temporarily* exhausted if the agent is in charge for a very long time. The need for time-based action selection (above) seems to come when behaviors can go on receiving high rewards indefinitely.

[Whitehead et al., 1993] make this explicit in their scheme, where goals are inactive (don't compete), start up at random (are activated), compete until they are fulfilled, and then fall inactive again. In this work, goals are implicitly activated and deactivated by suitable accompanying internal and external senses.

15.1.3 Dithering and Persistence

Minsky [Minsky, 1986] warns that too simple forms of state-based switching will be unable to engage in opportunistic behavior. His example is of a hungry and thirsty animal. Food is only found in the North, water in the South. The animal treks north, eats, and as soon as its hunger is only partially satisfied, thirst is now at a higher priority, so it starts the long trek south before it has satisfied its hunger. Even before it has got south, it will be starving again. One solution to this would be time-based, where agents get control for some minimum amount of time.

Again, however, time-based switching is not the only answer. As Sahota [Sahota, 1994] points out, the real problem here is having actions based only on the urgencies of the goal, independent of the current situation. Opportunistic behavior is possible with state-based switching where agents can tell the difference between situations when they are likely to get an immediate payoff and situations when they could only *begin* some sequence of actions which will lead to a payoff later. In Minsky's example, let the full state $x = (f, w, h, t)$ where:

- f is an external sense, taking values $\blacksquare$ (food visible) and $\square$ (no food visible)
- w is an external sense, taking values $\blacksquare$ (water visible) and $\square$ (no water visible)
- h is an internal sense, taking values 2 (very hungry), 1 (slightly hungry) and 0 (not hungry)
- t is an internal sense, taking values 2 (very thirsty), 1 (slightly thirsty) and 0 (not thirsty)

The food-seeking agent A_f presents W-values $W_f(x)$. The water-seeking agent A_w presents W-values $W_w(x)$. When we can see food but no water we would expect something like:

$$W_f(\blacksquare, \square, 2, 2) > W_f(\blacksquare, \square, 1, 2) > W_w(\blacksquare, \square, 1, 2) > W_f(\blacksquare, \square, 0, 2)$$

We start off very hungry and very thirsty. We don't stop feeding (and start dithering) when we get down to slightly hungry. We only stop when we get down to not hungry, and only then is A_w able to persuade us to leave, in the absence of visible water. If water was visible, A_w could have persuaded us to leave earlier with a higher W-value:

$$W_w(\blacksquare, \blacksquare, 1, 2) > W_w(\blacksquare, \square, 1, 2)$$

It is not just a vain hope that W-values will be organised like this. It is in the very nature of the way we have designed them on top of Q-values. The agents we use *can* tell the difference between immediate and distant likely payoff, and will present different W-values accordingly. Consider how A_w might present higher W-values when water is visible. If water is visible, it predicts reward 1 if obeyed and reward γ if not obeyed (not reward 0, since even if not obeyed now, it can get it on the next step). If no water is visible, it predicts reward γ if obeyed (assume we move to a state where water is visible and we can get it on the next step) and reward γ^2 if not obeyed (again, we can be obeyed here on the next step):

Q(water-visible,obeyed) $= 1$
Q(water-visible,disobeyed) $= \gamma$
Q(no-water-visible,obeyed) $= \gamma$
Q(no-water-visible,disobeyed) $= \gamma^2$
Hence:
W(water-visible) $= (1 - \gamma)$
W(no-water-visible) $= (\gamma - \gamma^2) = \gamma(1 - \gamma)$

So its W-value is *lower* than for when water is visible. It might be very thirsty in both situations, but it presents W-values based also on whether it thinks it has a chance of *satisfying* its thirst. Here the W-value is influenced by *how many steps away* the reward is.

We see here how W-learning will support *persistence*, or sticking on a goal. When one agent A_i's goal is in sight, the differences if it is not obeyed are between immediate rather than delayed rewards. Hence the differences between Q-values are larger (not discounted). As its goal approaches, the agent will generate higher W-values until satisfied.

Ethologists have generally proposed *positive feedback* models to explain persistence (see [McFarland, 1989, §1]), where the strength W_f of the feeding agent is being increased by the act of feeding itself (while at the same time being *decreased* by the reduction in hunger, so that it is still probably decreasing overall). The above analysis shows that such a *time-based* mechanism is not necessary to explain persistent behavior at least. A state-based explanation is also possible. The fact that we are eating means that food exists close by, and it will be easy (probably) to eat some more in the next few immediate timesteps. Other goals may be a number of timesteps away from fulfilment. Persistence happens because even though rewards are low, they are close by.

Note that our persistence mechanism, where W-values increase as the goal is approached, can also be seen as a positive feedback process (since being obeyed makes you even more likely to be obeyed on the next step, when you will have even higher W-values), but it is a different type of positive feedback process, state-based rather than time-based.

Our method factors in what McFarland [McFarland, 1989] calls the *cost* of switching from one activity to another, where nothing useful may happen for some time. Here, the Q-values are a *promise* of what will happen in the future, suitably discounted, and the W-value scheme makes an action selection decision based on the different agents' promises. High rewards far away may compete against low rewards close by.

15.1.4 McFarland

Looking closer at McFarland's work, a W-value can be seen as equivalent to the 'tendency' to perform a behavior in his model of *motivational competition* [McFarland, 1989].

His *motivational isoclines* are contour lines on a map of the state space (similar to Figure 5.6 but for a single agent A_i), joining together states $x_1, x_2, \dots$ whose W-values are equal: $W_i(x_1) = W_i(x_2) = \cdots$

In his model, the continuous performance of an activity traces a trajectory through the statespace, which may cross isoclines, leading to a switch of behavior. That is, performing an activity may cause us to enter a new state x in which there is a new winner $W_k(x) = \max_i W_i(x)$ and hence a switch of behavior. Note that my model does not assume the trajectory has to be *continuous* - we may make leaps from point to point in the statespace.

We now classify the list of specific models into three main groups - hierarchical, serial and parallel.

15.2 Hierarchical models

First, by hierarchical models we mean multi-level systems where the modules are built into some kind of *structure*. Some modules have precedence over others, and control flows down to their submodules. While many problems seem to lend themselves to this kind of solution, there is normally a considerable burden of hand-design. This thesis has consistently argued that there is no future in models that are not capable of some self-organisation.

15.2.1 Brooks' Subsumption Architecture

Brooks' *Subsumption Architecture* was introduced in §5, and his ideas used as the basic inspiration for our model. A Brooks-like *hierarchy* would be a special case of our more general model. To implement a hierarchy, we would have *strong* higher level agents and *weak* lower ones such that when nothing is happening relevant to the higher agent, it does not compete against the lower ones but when it competes it will *always* win.

A hierarchy is only one of many possible ways that W-learning agents might automatically divide up control. In this problem, the best solutions were not strict hierarchies. Note also that with W-learning, a hierarchy may form only temporarily at run-time, and be dissipated when new agents are created that disrupt the existing competition.

Hierarchies may themselves be what Brooks [Brooks, 1991a] criticised traditional AI for - structures whose main attraction is that we find it easy to think about them. Just as we find it easier to think of central control than distributed, so we find it easier to think of hierarchies than of collections of agents that cannot be ranked in any single order. One agent sometimes forbids the other, then later vice versa.

Brooks acknowledges this to a certain extent in [Brooks, 1994, §3], but then avoids the question of who *should* win in a competition between peers by moving on (in the Cog project) to models of spreading excitation and inhibition through networks.

15.2.2 Hierarchies in Ethology

Hierarchies have also long been popular in ethology. Baerends, for example, put a great deal of work into breaking down the behaviors of the digger wasp and the herring gull into multi-level hierarchical structures. See [Tyrrell, 1993, §8] for a survey of this and Tinbergen's hierarchical models.

15.2.3 Nested Q-learning

Nested Q-learning will be introduced in §18.1. It *can* be used as a hierarchical model, but that chapter shows that it can also be built into a more general scheme, of which a traditional hierarchy is only a special case.

15.2.4 Feudal Q-learning

Feudal Q-learning will be introduced in §18.2. Again, it is easy to think of it as a hierarchical scheme, but I show that that is only a special case of the more general model.

15.3 Serial models

Self-organising models divide into two groups - serial (agents must terminate before other agents start) and parallel (agents endlessly interrupt each other). The Action Selection problem is essentially about the latter. Serial or sequential models have already been criticised in §3.

15.3.1 Singh's Compositional Q-learning

Singh's *Compositional Q-learning* [Singh, 1992] was addressed in §3.

15.3.2 Wixson

Wixson's model (to be introduced in §18.1) is basically a serial form of Nested Q-learning, where we must wait for agents to terminate at their goal states before control may switch to a new agent.

15.3.3 Maes' Spreading Activation Networks

Maes' *Spreading Activation Networks* [Maes, 1989, Maes, 1989a] or *Behavior Networks* consist of a network of agents (or *nodes*) which are aware of their preconditions. Nodes can be linked to from other nodes that can help to make those preconditions come true, or be inhibited by other nodes who will cause their preconditions to *not* hold. They can in turn link to other nodes whose preconditions their behavior can affect.

Agents in our system do not have explicit preconditions. It's just that if we are in a state x where agent A_i cannot take any meaningful action, its W-value $W_i(x)$ will be low since there will be little difference between its Q-values whatever action is taken. So it will in effect not compete. Our system could also be seen as more decentralised in the sense that agents have no explicit concept of what other agents' goals might be. They only discover when the other agents start taking their actions.

Maes' *spreading activation mechanism* spreads excitation and inhibition from node to node. As in other methods, we see that the 'Q-problem' and the 'W-problem' are mixed together. For example, the desire to pursue a goal sends excitation to the consummatory action node, which if not executable propagates excitation back to appetitive action nodes. In this way links encode temporal appetitive-consummatory relationships between nodes. But in our system, basic Q-learning takes care of this on its own (§2.1.6) and it is not seen as part of the problem of action selection.

The advantage of mixing the 'Q-problem' and the 'W-problem' here is that appetitive nodes can be *shared* by multiple goals. But Tyrrell then shows [Tyrrell, 1993, §9.3.3] that there may be problems caused if we can't tell if the 'Explore' node is excited multiple times because it is serving multiple goals or because it is serving multiple paths to the same goal. Ideally, if it was serving multiple goals, we would allow it to be highly excited, whereas if it was only serving multiple paths to the same goal we would divide its excitation by the number of paths to express that it was only serving one goal. But because we can't tell the difference at a local level, we can't define a local rule to divide up the excitation or not.

In my architecture, each goal is taken care of by a separate agent. Each has to implement its *own* explore. The equivalent of 'Explore' serving multiple goals would be multiple agents wanting to take the same action. If an action serves multiple goals it will be defended by multiple W-values. If it serves one goal via different paths it will only be represented by one W-value.

It might seem wasteful that I make each agent implement its own 'Explore' but this does not mean that each agent actually wants its 'Explore' to be *obeyed*. All I mean is that each is a complete sensing-and-acting machine which can suggest an a for each x. If there is an efficient specialised Explore agent present, agents find they do better when their own rudimentary explore or random motion action is *not* taken. So they do not try to compete (as long as Explore is winning).

Say we have 5 agents, and 3 of them want to do the *same* thing. In Maes' scheme they would divide up their excitation but as Tyrrell points out they then can't compete on an equal footing with the other 2.

In W-learning, all 5 would initially compete all against each other. If one of the 3 starts winning the whole competition, the other two of the 3 will back off. Agents back off when someone else is fighting their battles for them, but *only* if they are winning.

15.4 Parallel models

In parallel, or Pandemonium, models we have multiple agents trying to control the body simultaneously, and some intelligent (or dumb) switch deciding which one of them (or combination of them) to let through.

15.4.1 Lin's Hierarchical Q-learning

Lin's *Hierarchical Q-learning* has already been introduced (in §3) and subsequently tested.

15.4.2 Pandemonium

The ancestor of our abstract decentralised model of mind is Selfridge's *Pandemonium* model [Selfridge and Neisser, 1960]. This is a collection of "demons" shrieking to be listened to. The highest shriek wins. Selfridge's implementation was in the area of pattern recognition or classification, where the demons examine different features, and their weight (shriek) is the similarity of the input to the structure of the demon. For example, each demon might be a character-recogniser. Demons' weights might be hand-coded, or they might be learnt by Supervised Learning. When the correct output is given, weights could be strengthened or weakened.

While there are many descendants of this kind of model in pattern recognition, it is not clear how it translates to the Action Selection problem, where demons become control programs, suggesting actions to execute, each pursuing different goals. What should the weights represent? Similarity to what? What is the "correct" demon? How should the weights change?

15.4.3 Competitive Learning

The *Competitive Learning* algorithm, an unsupervised learning technique in connectionism, can be seen as a detailed implementation of a pandemonium model.

In a neural network, the input pattern must be somehow encoded in the weights on the links leading into the hidden layer. Normally one finds that the representation is spread over a number of hidden units cooperating. In Competitive Learning, a *single* hidden unit will represent the input pattern. Given a particular input, a large number of hidden units *compete* to represent it. The winner is the one whose incoming weights are most similar to the input. The winner then modifies its weights to be even more similar to the input. We do not (as we normally would) modify the weights of *all* hidden units, only those of the winner.

In this way, each hidden unit ends up winning a different *cluster* of input patterns and its incoming weights converge to the centre of the cluster.

Again though, while a detailed implementation is useful, this is still directed towards the pattern recognition or category formation problem. It hasn't answered the question of how to apply this to Action Selection.

15.4.4 Jackson

Jackson [Jackson, 1987] attempts to apply a pandemonium model to Action Selection. He has demons taking actions on a playing field, forging links with successor demons in the stands by exciting them, to get temporal chains of demons.

This is partly temporal chains of actions, which we saw is taken care of in pursuit of a single goal by basic Q-learning (§2.1.6). It is also partly temporal sequencing of agents, or time-based action selection, that I criticised in §15.1.1. As we can see, the 'Q-problem' and the 'W-problem' are mixed together here.

Jackson's paper is full of interesting ideas but it is a conceptual model so it is hard to know whether it would really work as claimed. It would be interesting to see a model detailed enough for implementation.

15.4.5 The DAMN Architecture

The *Distributed Architecture for Mobile Navigation* or 'DAMN' architecture [Rosenblatt, 1995, Rosenblatt and Thorpe, 1995] is a pandemonium-like model for Action Selection. Agents vote for actions, and the action which receives the most votes is executed. The action selection method is similar to Maximize Collective Happiness. Agents must share the same suite of actions, and be able to provide votes for actions other than their first choice. To find the best action to execute, the creature calculates the equivalent of:

$$\max_{a \in A} \left[\sum_{i=1}^{n} w_i Q_i(x,a) \right]$$

where the w_i are a set of weights reflecting which agents are currently priorities of the system. The idea is that these weights can be used to adjust the strength of an agent relative to its fellows to avoid the pitfalls of a collective method, such as inability to be single-minded (§12.3). But one couldn't multiply all of A_i's Q-values by the same weight, as we saw in §9.1. We need $w_i(x)$, a weight specific to the state. Rosenblatt acknowledges this, and suggests the w_i weightings can be changed dynamically as the creature moves through different states x. But he has no automatic way of generating these weights. This is all part of the burden of hand-design.

The utility theory developed in [Rosenblatt, 1995] develops a hand-designed equivalent of a static W = importance scheme (as in §5.4), where the agent's vote for an action is calculated relative to all alternative actions. One difference is that the agent has a vote for every action $W(x,a)$. His normalised equation:

$$U_b(c) = (g_b(c) - g_{\min})/(g_{\max} - g_{\min})$$

is the equivalent of the normalised equation:

$$W(x,a) = (Q(x,a) - \min_{b \in A} Q(x,b))/(\max_{b \in A} Q(x,b) - \min_{b \in A} Q(x,b))$$

There is a special case where the denominator is zero (all Q-values are the same) but then the numerator is zero as well. We would just set $W = 0$.

The trouble with the above measure is that we are really only interested in the best Q-value, for which $W(x) = 1$ always. We could do a *dynamic* form of scaling perhaps:

$$W_i(x) = (Q_i(x, a_i) - Q_i(x, a_k))/(\max_{b \in A} Q_i(x, b) - \min_{b \in A} Q_i(x, b))$$

but it is unclear what the advantage would be. If we multiply the Q-values by a constant, the W-value would remain unchanged. We have no easy mechanism for making agents stronger or weaker. The argument against scaling was presented in §8.1.

15.4.6 The BSA Architecture

The *Behavioral Synthesis Architecture* [Aylett, 1995] is another example of hand-designed utility functions for behaviors. For example, Aylett's *behavior pattern* equation:

$$bp_t = \{r_t = f_r(s_t)\}, \{u_t = f_u(s_t)\}$$

is equivalent to the agent being able to generate for any x an action and an associated utility:

$$\{a = a^*(x)\}, \{Q = V^*(x)\}$$

The action selection is similar to Maximize Collective Happiness. Again, the difference is that these numbers are all designed by hand.

15.4.7 Drives

As was mentioned when it was introduced (§5.2), our basic model of parallel internal competition is a form of *drives* model from ethology. Here the difference is we provide an answer to the question of where the drive strengths come from.

15.4.8 Tyrrell

Implementing a $W = D - F$ measure is essentially what Tyrrell is trying to do in his parameter tuning to get an implementation of drives working

[Tyrrell, 1993, §9.3.1]. The amount of hand-design he needs to do only points out the value of self-organisation of these numbers.

For example, if a predator is visible he designs his 'Avoid predator' drive to have a high drive strength (it must be listened to). If no predator is visible he gives it a low drive strength (it doesn't matter if it is not listened to).

When avoiding a moving predator, the predator-avoiding agent A_p must be listened to *precisely* (the creature must move in exactly the opposite direction). When avoiding a stationary hazard, any action is alright so long as it isn't the one action in the direction of the hazard. So it doesn't matter if the hazard-avoiding agent A_h is not listened to if other agents don't want to go in that exact direction anyway.

Under W-learning, these numbers would have been built automatically. A_p would learn a high W-value if (as probable) other agents wanted to do anything else other than move away from the predator. A_h would know that only that particular direction was bad, and would learn a low W-value (would not compete) if other agents were going in some other direction anyway.

In [Tyrrell, 1993, §11.1.2], Tyrrell's *hierarchical decision structure* implements a form of Maximize the Best Happiness, while his *free-flow hierarchy* implements a form of Maximize Collective Happiness.

15.4.9 Modular Q-learning (University of Rochester)

Work at the University of Rochester [Whitehead et al., 1993, Karlsson, 1997] addresses the same question as this thesis - given a collection of competing peers, who should win? Their *nearest neighbor* strategy is effectively Maximize the Best Happiness (W=Q). Their *greatest mass* strategy is Maximize Collective Happiness. A Minimize the Worst Unhappiness strategy was not tried.

[Karlsson, 1997] argues that a modular approach learns a sub-optimal policy quicker and with less memory than a monolithic approach. While this is true, I further point out that the monolithic approach may not even learn the optimal policy itself.

[Ono et al., 1996, §2], in their implementation of greatest mass, implement exactly the Maximize Collective Happiness strategy.

15.4.10 W-learning

For completeness, I point out which category this work itself belongs to. In their basic form, W-learning and related strategies are standard, one-layer, parallel models. §18 will show how they can scale up into more complex forms of societies, still self-organising.

15.5 Classifier Systems

Finally we consider other related work which looks to be relevant but on closer examination may not be quite so relevant.

In this thesis we have agents with reward functions of the form:

if (condition) then reward r_1 else reward r_2

and we use a genetic algorithm to search through different combinations of r_1 and r_2. This may remind some of *classifier systems* [Holland, 1975], which run genetic search on *production rules* of the form:

if (condition) then execute (action)

The comparison is misleading. In this thesis we are not searching the space of reward function rules. We are not inventing new reward functions, only modifying the strengths of the existing ones. All we are doing is testing all different possible parameters - something any good empirical test has to do.

The classifier system rule operates at the low level of actually specifying behavior. The reward function rule operates at a higher level of specifying a value system, that must be translated into behavior. In this thesis I have been content to hand-design the reward functions and let the behavior be learnt from them. The genetic search on the reward functions' parameter values is then actually a search for different *combinations* of weak and strong agents. We will return to this in §17.2.

Note that classifier systems are really searching through rules of the form:

if x then execute a

They are solving the Q-problem, not the W-problem.

15.5.1 Grefenstette

Grefenstette's SAMUEL system [Grefenstette, 1992] uses competition among agents, where an agent (or *strategy*) is a set of rules.[1] A rule is of the form: `IF (condition) THEN suggest (set of actions with strengths)`. Hence 'strength' here is the equivalent of a Q-value for the action. One difference with Q-values is that Grefenstette looks for high expected reward and a *low variance* in those rewards. Q-learning just looks at expected reward irrespective of variance.

[1] Note that Grefenstette himself uses the word agent to refer to the whole *creature*.

'Competition among strategies' however turns out to mean only a genetic search for good strategies to solve *the same problem*, deleting the bad ones, and mutating good ones. There is no competition between two stategies in the same body at run-time which is what competition between agents means in this thesis. In other words, Grefenstette uses competition as a *device* to solve the single-goal Q-problem, rather than it being an unavoidable run-time feature of the problem, as it is in the multi-goal Action Selection W-problem.

A rule fires when its condition is met - which would seem equivalent to the rule representing the Q-values for a particular state x. An important difference though is that the conditions can be quite general, for example something like: `IF fuel=[low or medium] AND speed=[600 to 800] THEN...` So a rule builds up the equivalent of Q-values for actions in a *region* of statespace. These regions may overlap. That is why multiple rules may fire at the same time - something that cannot happen if rules are of the form `IF x THEN..` where each x is a unique state. For states x lying in overlapping regions, there will be *multiple* estimates of $Q(x, a)$.

15.5.2 Product Maximize Collective Happiness

Grefenstette's method of dealing with conflicting suggestions is interesting because it can be adapted for use in the Action Selection problem (which he does not address). It is a Maximize Collective Happiness strategy, but instead of summing the Q-value-equivalents (as in §12.1), it multiplies them. His two bids (suggested actions with strengths) of:

```
( right(0.9),left(0.4) )
( right(0.8),left(0.9) )
```

Are combined to give a bid of:

```
( right(0.72),left(0.36) )
```

So the action "move right" is chosen for execution. This is the equivalent of having two agents with Q-values:

```
a          (R)    (L)
Q1(x,a)   [0.9]   0.4
Q2(x,a)    0.8   [0.9]
```

To select an action for execution, we calculate:

$$\max_{a \in A} [\ Q_1(x,a) Q_2(x,a) Q_3(x,a) \dots Q_n(x,a)\ ]$$

As before this may produce compromise actions, that none of the agents would have suggested. This method requires agents to share the same suite of actions. Note that we can't implement this method if Q-values can be negative or close to zero. Consider agents with Q-values:

a	(1)	(2)	(3)	(4)
Q1(x,a)	0.9	0.1	-0.2	0.9
Q2(x,a)	0.9	0.1	-0.2	0.9
Q3(x,a)	0.9	0.1	-0.2	0.9
Q4(x,a)	0	0.1	-0.2	-0.1

Using a naive product method of Maximize Collective Happiness, action (3) will be chosen, strangely enough. Action (1) doesn't get chosen because the 0 cancels out the 0.9's. Action (4) doesn't get chosen because the -0.1 makes the product negative. And then action (3) beats action (2) because the minus signs cancel each other out.

Obviously, we must make all Q-values positive, which is accomplished by making all rewards positive (Theorem B.2). Q-values arbitrarily close to zero can still cancel out an entire column, but perhaps this is alright, since if all rewards are > 0, a Q-value very close to zero means the action will not only give the agent no reward but also take it to a state from which nothing it can do can expect to bring it a reward for quite some time into the future. So perhaps it is right that the agent vetoes this action. Note that in the example above it is *not* good that the 0 vetoes the action because action (1) is actually quite a good action for agent A_4!

This method of Action Selection will still suffer though from the drawbacks of any collective approach, as discussed in §12.3. In a crucial state for it, one agent can get drowned out by the sheer weight of numbers of other agents, none of whom care very much about this state:

a	(1)	(2)	(3)	(4)
Q1(x,a)	2.5	0.1	0.1	0.1
Q2(x,a)	0.1	0.3	0.3	0.3
Q3(x,a)	0.1	0.3	0.3	0.3
Q4(x,a)	0.1	0.3	0.3	0.3

Here, agent A_1 is drowned out and action (1) is not taken. Note that the summation method of Maximize Collective Happiness would have chosen action (1) in this case.

15.5.3 The Economy of Mind

Baum [Baum, 1996] introduces an Economy of Mind model ("A Model of Mind as a Laissez-Faire Economy of Idiots"), where new agents can be created dynamically, and agents that do not thrive die off.

An "agent" here though is simply a rule (a condition-action pair). There is a single goal, and the economy selects agents over time to find a set of rules that best satisfy that goal. That is, as with Grefenstette (§15.5.1), competition here is a *device* to solve the single-goal Q-problem, rather than referring to the run-time competition of the W-problem.

There are large numbers of these simple agent-rules (in one typical run, at any given time between 20 and 250 agents were alive). Agents *pay* for the right to take an action (they pay the previous winner). When an action is taken only the winner receives a reward. So an agent pays to win, but then receives two payoffs - one from the action it takes, and one in the next step when it is paid off by the next winner. What it pays should be less than or equal to on average what it receives, otherwise it will eventually be bankrupted and deleted. The agent who expects to gain the largest rewards by taking its action will outbid the others.

But these bids are not analogous to W-values. They are an estimate of expected reward, used to control the entry of new rules. New agents can only enter and thrive if their estimates are more accurate. Bids are analogous to *Q-values* - they are a way of learning estimates of reward in pursuit of the single goal.

It would be interesting to extend Baum's model to the Action Selection problem. 'Agents' do not actually have to be simple condition-action pairs. The concept of bidding and paying for actions can still be applied even when the agents are our Q-learning agents pursuing different goals. In a strict economic model, though, agents would have no interest in letting other agents take their actions for them, since they won't receive the reward if that happens. Agents wanting to do exactly the same thing will still bid against each other. And when agents want to do similar but not exactly the same things, there can be no concept of compromise. An agent either receives its full reward or nothing. The system would seem to be incapable of opportunism.

Perhaps we should relax the strict economic analogy and allow all agents to profit if one of them does good.

15.6 Operating Systems Theory

An interesting comparison can be made between how control switches in a W-learning creature and in an Operating System. Operating Systems implement a sort of time-based action selection (recall §15.1.1), but different assumptions make OS ideas difficult to apply to our problems.

Agents get CPU time for some time slice. There is no concept of whether or not this is an appropriate state x for them to take over. Agents can be given a priority, which gets them more frequent timeslices, but still independent of x, although the concept of priority for mouse and keyboard responsive routines could be seen as a simple form of priority based on x.

There seems no way to express the loss an agent suffers by not being obeyed as being dependent on x and on who was obeyed. In an Operating System all losses are assumed the same - the agent just has to wait some time. There is also no concept that a non-winner can profit from what the winner is doing.

15.7 Ethernet protocols

Another interesting comparison is with how the Ethernet [Metcalfe and Boggs, 1976] works. In an ethernet, stations only transmit when they find the Ether is silent. When multiple stations transmit at the same time, a collision occurs and they back off at different rates, so that one will be left to transmit while the others defer their transmission. When the winner is finished, the others will start up again. It has been compared to a civilized meeting where if someone wants to speak they raise their voice and everyone else quietens down.

In W-learning, agents raise their voices (W-values) to be heard, but if the lead is taken by an agent favourable to their plan of action they will be found lowering their voices again. At any moment, all agents may have low W-values, but only because they are all happy with the current leader.

For example, in our artificial world, when no interesting features of the world are visible, almost any agent with a 'wander'-type behavior can take control relatively unopposed. In fact, the wander behavior may be split among a number of agents, depending stochastically on which got into the lead first in each state.

15.8 Game Theory

Interesting examples of competition can be set up where an agent can do well if it is selfish, but even *better* if it cooperates, so long as the other agents cooperate too. Such problems include the Prisoner's Dilemma.

In our model, an agent always does best when it is selfish and is normally expected only to do as well or worse when it is not obeyed. While action selection here is not *exactly* a zero-sum game - nonobeyed agents still collect rewards up to and sometimes equal to their highest possible - the best strategy for an agent is still always to try to be obeyed.

A more complex strategy might be if the agent accepts it can't win, but has the choice of saying who it would support to win instead. If it votes for a winner and votes are added, though, we might again run the risk of the collective methods, where sheer weight of numbers can drown out an unpopular minority opinion.

15.9 Economic Theory

The problem of reconciling independent wills in a society is an old problem of economic and political theory. Because of different assumptions though, much of the work is not easily transferable to the Society of Mind.

In economics, *utility* [Varian, 1993, §4] is used as a way of ranking an agent's preferences. It can be seen as analogous to Q-values for actions. In the standard *ordinal utility* theory, the precise numbers are not important. What matters is only the *order* in which things are ranked. This is analogous to a single Q-learning agent on its own. It is interesting how difficult it often is to find such an ordering in economics. Many apparently arbitrary functions are used, once the requirement is dropped that the precise numbers be meaningful.

In *cardinal utility* theory some meaning is attached to the numbers. The utility of some choice can be said to be twice that of another, and so on. Here we can introduce an analog of W-values, where the difference between the Q-values is measured. We can ask the question: 'You want choice A. If we force you to take choice B, how bad is that?'

Welfare theory [Varian, 1993, §30] is the branch of economics with the most relevance to this thesis. Welfare theory makes ethical judgements about what kind of economic society we want, and suggests various social welfare functions to be optimised. We have noted in the text that some of our action selection methods are equivalent to optimising the following social welfare functions:

- The classic *utilitarian* or Benthamite social welfare function (the greatest happiness for the greatest number) is equivalent to Maximize Collective Happiness (see §12.1).

- A "Nietzschean" social welfare function (the value of an allocation depends on the welfare of the best off agent) is equivalent to Maximize the Best Happiness (see §9).

- A Rawlsian social welfare function (the value of an allocation depends on the welfare of the worst off agent) is a Minimize the Worst Unhappiness method, though not quite the one that we use (see §13).

Rawls sides with his agents, and takes note of their individual stories, out of ethical concern for their unhappiness. We side with ours not because we 'care' about them (they are only parts of the mind) but because we want individual agents to have the ability, when they really need it, to raise their voices and be heard above the others.

One considerable difference between the Society of Mind and economic societies is that economic agents don't benefit from someone else's success. They won't be happy to see another win if they could have won themselves.

15.10 Political Theory

The problem of reconciling independent wills has also of course long been addressed by political theory, although here the focus tends to be on rights and freedoms rather than on wealth. In fact, the analogies with our systems of action selection are even stronger:

- The Collective methods are analogous to old-fashioned majority-rule democracy (or 'majoritarianism'). This is a simple majority vote where minority rights can be ignored.

- W-learning is analogous to liberal democracy, where the test of a democracy is how well it treats its minorities. Liberal democracy here is the attempt to minimize *individual* hard-luck stories. Journalism in a liberal democracy focuses endlessly on individual problems and individual denials of rights, rather than focusing on the majority or the common good. A single individual's denial of rights can change the law for the whole population.

- W=Q is analogous to anarchy, which is survival of the fittest, where no one is safe. Anarchy here means that *individuals* can trample on your rights, analogous to how the winner in W=Q will not compromise with anyone.

This is summarised in Figure 15.1. Note that an *authoritarian* system would not be on this scale since it would not involve action selection at all - there would be a constant winner independent of state x. In an authoritarian system, a strong agent wins all states x. In W=Q (anarchy), strong agents win, but different ones in each x.

What is good for the Society of Mind of course may not necessarily be good for the Society of Humans, and I have no interest in making any statements about the latter. I think it would be uncontroversial though to say that, whether one is in favour of it or not, liberal democracy is a more *complex* idea than simple majority-rule democracy, which is perhaps why the two are often confused even in much of the democratic world.[2] Similarly in action selection, Minimize the Worst Unhappiness is a more complex idea than Maximize Collective Happiness, which is why perhaps it has been generally overlooked. In the action selection methods in this chapter, we have noted the recurring popularity of Maximize Collective Happiness schemes.

So whatever about the Society of Humans, this thesis argues that the Society of Mind (or at least our first draft of a Society of Mind) should probably be a liberal democracy.

[2]For example, in Northern Ireland 'democracy' is repeatedly assumed to mean simple majority-rule democracy (with the opposing sides defining different territories over which to take the majority vote). I am not, of course, saying there is anything *right or wrong* with any particular model of society, but only noting that people confuse their terms.

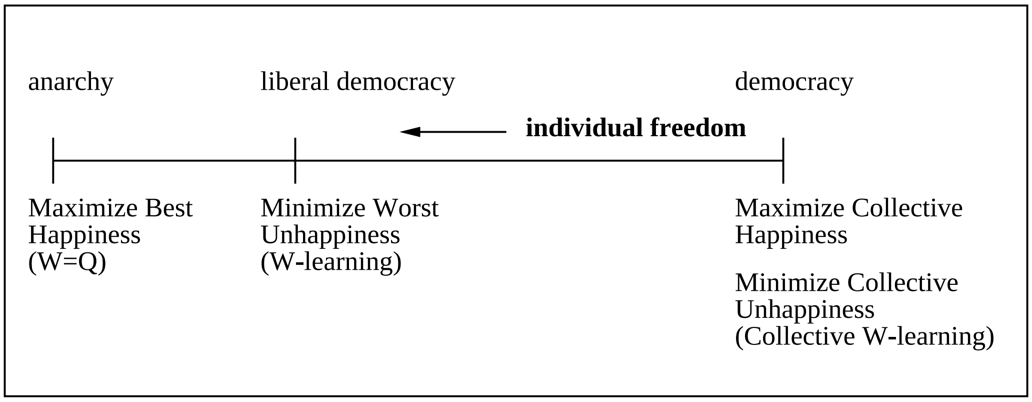


Figure 15.1: Political analogs of the action selection methods. Note that individual freedom is not identical to individual happiness.

Chapter 16

Conclusion

16.1 Empirical results

Looking at the summary of empirical results in §14, we note that Q-learning and Hierarchical Q-learning were outperformed by methods that do not refer to the global reward. How could this be? One can see the advantage of self-organising methods if we can't (or don't want to) design an explicit global reward function. But surely if a global reward exists, you can't do better than learning from it.

The answer is that yes, in a Markov Decision Process with lookup tables you *can't* do better than learning from the global reward. But if the world was that simple, Q-learning would do fine and we wouldn't even be interested in Hierarchical Q-learning let alone any of the self-organising, decentralised methods. It's because we are not in such a simple world that Q-learning (and indeed Hierarchical Q-learning too) can be beaten. [Lin, 1993] has already pointed out that the reason Hierarchical Q-learning beat Q-learning was because Q-learning could not learn the optimal solution in its neural network.

Q-learning and Hierarchical Q-learning suffer from a similar flaw to the Collective methods (§12.3) - they will tend to lose minority agents. Because they use a generalisation, when a reward comes in as part of the global reward function it affects the values of other (x, a) pairs. Small or infrequent rewards can get drowned out by the pursuit of large or frequent rewards. The generalisation learns a cruder policy than is possible with say Negotiated W-learning. It is liable to optimise only the one or two main components of the global reward.

In the self-organising methods, we devolve powers to the agents themselves, allowing rarely used and normally-quiet agents to rise up and take over at crucial, but infrequent, times. Agents can spot better than global functions opportunities to pick up rewards, so devolving power to the agents is a better strategy for keeping them all on board. Minimize the Worst Unhappiness (W-learning) can listen to individual stories in a way that Q-learning and Hierarchical Q-learning cannot.

It is interesting that even simple W=Q was better than Hierarchical Q-learning here. That large statespace is the problem, and Lin hasn't found a way of getting rid of it. It does seem wrong that there should exist such a structure, for what does it mean? Is it not merely a number of unrelated observations from different senses combined together. In Tyrrell's Simulated Environment [Tyrrell, 1993] the full space would have to be of size 10^{100}. Across all our methods, those huge full spaces are mostly wasted. The two best methods were both with tiny subspaces.

Some of the results were as expected. As expected, Negotiated W-learning was better than W-learning with full space, which itself was better than W-learning with subspaces. W-learning with full space is more accurate than with subspaces, but the neural network means we can't get full accuracy. Negotiated W-learning finally delivers full accuracy.

Also as expected, the results support Lin's basic finding of the advantages of Hierarchical Q-learning over Q-learning.

Finally, while the experiments clearly support decentralisation of some form, there are a number of reasons why they are inconclusive as to what are the very *best* methods. First of all, the pure version of Minimize the Worst Unhappiness would probably have outperformed Negotiated W-learning and been the best method of all had it been tested. W-learning with full space would probably have performed better than it did had we used a declining Boltzmann for W instead of random W winners (see §10), and so would probably have been next up after these two. Of course these two require agents to share a common suite of actions. If agents don't share actions, W-learning with full space might be the best method. The simple, static W=Q method performed very well, but probably only because opportunism is not important enough in this world (see §9.1). If we were to increase the benefits of opportunism, we might see static W=Q drop to the level of W-learning with subspaces - the two low-memory, inaccurate-competition methods.

16.1.1 Hand-coded programs

The final comment on the experiments is that it is remarkable how difficult it is to hand-code the goal interleaving necessary to perform well in this world. The solutions generated by learning are far superior, though difficult to visualise or translate into a concise set of instructions.

In the much simpler Ant World problem, W-learning with subspaces performed similar to hand-coded programs (see [Humphrys, 1995]), and the statespace was small enough to map, so we could translate the winning strategy into a set of rules (as in §7.1).

In the more complex House Robot problem, hand-coded solutions are far inferior, and at the same time the statespace is too big to map so it is difficult to analyse how the good solutions actually work. Since it is hard to enumerate all the states and see what the system does, it is hard to translate our solution into a concise program. When comparing a state-space versus a set of thousands of `IF .. THEN ..` rules, one might as well stay with the state-space - it is hardly less comprehensible.

16.2 Related work

A general comment about the survey of related work (§15). It would be easier to compare action selection methods if the two problems, the 'Q-problem' and the 'W-problem', were not mixed together. Some of the problems that various action selection methods try to solve are in fact solved by standard methods for single goals, such as short-term loss for long-term gain, which is solved by basic Q-learning (§2.1.6). One can't expect one mechanism to do everything. For larger systems, we must separate Q and W or we will drown in complexity.

In fact, it would be helpful if there was a more standard terminology - if all authors, irrespective of whether their scheme was hand-designed, learnt or evolved, used say $Q(x, a)$ for goodness/fitness/utility of an action relative to other actions in an ideal world where the agent acts alone, and $W(x)$ for strength/bid *relative to the other agents.*

16.3 Searching for the best solution relative to some global score

For the four new methods that we tested in this work (the 'global reward function - free' methods) we ran a Genetic Algorithm search on combinations of rewards r_i, and compared the *best* solution found by the GA with the solutions found by Q-learning and Hierarchical Q-learning (see §G for summary of all experiments). Is this unfair? It looks as if we are doing more parameter-tweaking with the four methods than we allowed for Q-learning and Hierarchical Q-learning.

First of all, remember that here we are trying artificially to compare all the methods on the same scale. So even though the four methods do not *need* a global reward function to organise their action selection, we still must search for combinations that *happen* to organise well according to the global reward function's judgement if we are to compare them directly to Q-learning and Hierarchical Q-learning.

Imagine that we have no explicit global reward function. The four methods organise with reference to local rewards only, and then we will have some judgement about the final creature. There are three possible strategies of searching for well-balanced combinations:

Strategy 1 - Run an automated search using a standard GA with an explicit global fitness function.

Strategy 2 - Run an automated search in an open-ended Artificial-Life type experiment where fitness is implicit in the environment.

Strategy 3 - Observe the actual *behavior* and then use human judgement to adjust the broad strength of particular agents up and down (recall §8.1.3). The agents still work out the details of the action selection themselves.

In some ways, Strategies 2 and 3 are *a search for the global reward function itself*. Global reward functions don't come out of thin air, and just because Q-learning and Hierarchical Q-learning have maximized one doesn't mean that the resultant creature has the balanced behavior you want.

Further experiments could be done using Strategies 2 and 3. Using Strategy 1 in this work was good from the point of view of a fair comparison, but bad from other points of view. It gives the impression that a perfect global reward function will normally exist (and hence that we can use Q-learning and Hierarchical Q-learning at all). It also gives the impression that *if* there is such a function, the four methods take more effort to learn from it than Q-learning and Hierarchical Q-learning do (a full GA search - essentially a search to *re-find* the global function again). But we didn't have to use a GA - we could have used Strategy 3. The decision not to was again for reasons of a fair test (see §8.2).

16.3.1 The Adaptive Landscape

Further work could be done with Strategy 1 as well, to analyse the nature of the *adaptive landscape* that it is exploring. It would be interesting to see how sensitive the performance of the creature could be to small changes in the relative strengths of its competing agents.

The landscape here is hard to illustrate since it is a 9-dimensional surface (8 parameters leading to 1 fitness value). Certainly there could be sharp cliffs on the landscape - there could be some state x which a specific agent A_i must win if the creature is to be remotely adaptive. At the boundary where A_i is just too weak to win the competition for x, there will be a sharp change in the fitness value. A sharp *peak* on the landscape might be less likely than a cliff.

Informally, the search space for the GA seemed fairly tractable. The best solutions found levelled off quite rapidly, while the population was continuing to explore diverse points (i.e. this was not simply quick fixation of the whole population at a single local maximum).

More formally, the only proper analysis of the adaptive landscape during this work was, ironically, for the case of the W-learning with subspaces in §4.1 of [Humphrys, 1996] where a badly-designed global reward function caused the optimal behavior to be to jump in and out of the plug non-stop.[1]

For what it's worth, here is an illustration of that landscape. The global reward function was the same as in §4.3.1 except that arrival at the plug scores 1 point. This allowed scores of up to 29.590 points per 100 steps to be achieved. Figure 16.1 shows how quickly the GA finds good solutions.

To try to understand whether good solutions are clustered together or scattered on narrow peaks, we need some measure of *distance* between two solutions where solutions are n-dimensional vectors. The Euclidean distance metric provides such a measure. Here, given some combination of rewards $c = \{r_1, r_2, \ldots, r_n\}$, the Euclidean distance of this from another combination of rewards $c' = \{r'_1, r'_2, \ldots, r'_n\}$ is:

$$d(c, c') = \sqrt{\sum_{i=1}^{n} (r_i - r'_i)^2}$$

Figure 16.2 shows the Euclidean distance of all solutions relative to the single best solution $\{0.93, 0.34, 0.47, 0.08, 0.47, 0.74, 0.34, 1\}$, which scored 29.590. It seems clear that there are multiple peaks with near-optimal solutions - although there is still some kind of downward slope as you get very distant from the best point, so the multiple peaks are to *some* extent in the same general region of space.

[1]Remember that our suggested cure for problems like this in designing global reward functions is to do away with the global reward function altogether.

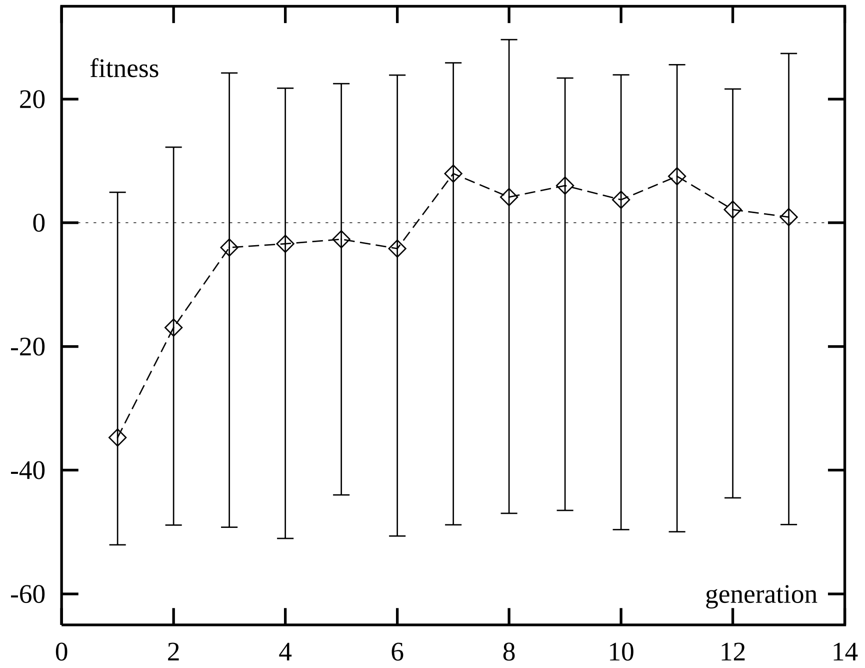


Figure 16.1: The average fitness of the population for each generation of evolution. The errorbars show the best and worst individuals in each generation.

In fact, we can see that the global maximum *cannot* be a single point, it must be a ridge. Simply multiply all rewards by the *same* positive constant and all W-values are multiplied by the constant and the competition is unchanged. If $W_i(x) < W_k(x)$, then $cW_i(x) < cW_k(x)$. We get a combination with an identical resolution of competition, and hence identical fitness, but at a different point in space (at a different Euclidean distance). For example, $\{0.093, 0.034, 0.047, 0.008, 0.047, 0.074, 0.034, 0.1\}$ would have identical fitness to the best combination here. Figure 16.2 is probably not a good way to illustrate the landscape then, since for any high fitness point we can easily find points of similar fitness at high Euclidean distance from it.

Confusing as this n-dimensional fitness landscape may be, it is the automated search program's job (Strategies 1 and implicitly 2) to navigate it. The searching by hand strategy (Strategy 3) does not deal with a fitness landscape at all, but rather observes *behavior*.

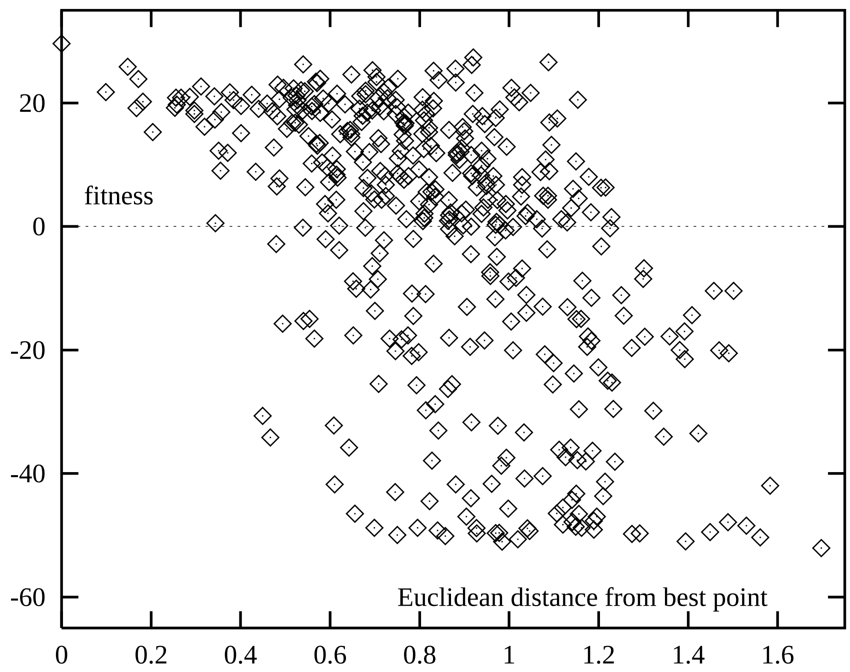


Figure 16.2: A plot of all combinations of rewards tested, showing the fitness of each combination plotted against its Euclidean distance from the single best combination.

16.4 What global policy have we learnt?

An agent presents certain Q-values not necessarily because of immediate reward but also because of what is a few steps away. So it may not be much use to the agent to get obeyed now if it won't get obeyed after that and never picks up the reward. Is it possible that in our action selection schemes, an agent is fighting for a state only to never see the long-term rewards that were the reason it wanted the state in the first place?

Say we want to maximise, as the global fitness function for the whole creature, the sum of rewards at each timestep:

$$\left(\sum_i r_i\right)_t + \gamma \left(\sum_i r_i\right)_{t+1} + \gamma^2 \left(\sum_i r_i\right)_{t+2} + \cdots$$

Strangely enough, this does *not* translate into picking the sum of the Q-values each timestep. Because an agent's Q-value expresses what the agent expects if it takes action a and thereafter *follows its own policy*. For example, an agent might need to go down a path of a few steps to get its reward:

$$Q_i(x, a) = 0 + 0 + \gamma^2 r$$

So if we follow this agent for under three steps and then switch control, the sum of rewards it contributes will actually be zero. Maximising $(\sum_i r_i)_t$ only translates to maximising $\sum_i Q_i$ for the special case $\gamma = 0$. In general, the sum of the agents' Q-values is a meaningless term because it expresses expected reward if a number of different policies are followed simultaneously:

$$\sum_i Q_i(x_t, a_t) = \sum_i \left[(r_i)_t + \gamma (r_i)_{t+1} + \gamma^2 (r_i)_{t+2} + \cdots \right]$$

Maximising this sum of the Q-values may not lead to maximising the sum of the rewards - it may just lead to dithering. As we put it in §12.3, it's the whole creature equivalent of going for short-term gain at the expense of long-term loss. The problem arises because agents' Q-values are being *optimistic* about what happens if they are obeyed. For instance, agent A_1 is near its goal. If obeyed, its Q-value is r, if not γr since it can be obeyed on the next step. The other agents are miles away from their goal. If obeyed, their Q-value predicts $\gamma^n r$ since it is being optimistic that the agent's policy will be followed for n steps afterwards, which of course it won't be. If disobeyed, their Q-value is $\gamma^{n+1} r$:

a	(1)	(2)
$Q_1(x,a)$	r	γr
$Q_2(x,a)$	$\gamma^{n+1} r$	$\gamma^n r$
$Q_3(x,a)$	$\gamma^{n+1} r$	$\gamma^n r$
$Q_4(x,a)$	$\gamma^{n+1} r$	$\gamma^n r$
...	...	...

If there are enough other agents, Maximize Collective Happiness will take action (2) and A_1 will never see its reward. If there is a similar profile for all the agents - with all of them wanting to go down paths that none of the others want to go down - then the same thing will happen to them when they try to get their rewards. Maximize Collective Happiness, following those optimistic contradictory Q-values, will always dither and never pick up any actual rewards. The sum of rewards will be zero, even though we appear to be following the Q-values.

16.4.1 How do we maximise the ongoing sum of rewards?

If we want to maximise the sum of the rewards, the first solution is of course to simply go back to a global reward function and a monolithic Q-learner, where the reward each time step is $\sum_i r_i$. Another solution (discussed in [Whitehead et al., 1993]) is to use a *model* to predict what state our Action Selection policy will lead to next, what decision Action Selection will take then, what state that will lead to, and so on. Or the Q-values, so as not to be so misleading, would start to express the expected reward given how we expect to be *obeyed* in the future (i.e. we lose the idea of the Q-value expressing something independent of the collection). Alternatively, perhaps we should give agents control for a minimum number of timesteps.

None of these methods is very satisfactory. In this thesis we can offer instead a local-information, model-free solution whereby it is a property of the action selection mechanism that *chains* of actions from the same agent are likely to be followed and consummated. §15.1.3 showed how Minimize the Worst Unhappiness implements an automatic, positive-feedback, *persistence* mechanism whereby an agent that wins a state will be even more likely to win the next state, and so on until its reward is achieved. There is much interruption, but it is more likely at some moments than at others. Goals are followed and the promised rewards do tend to be picked up before interruption. Note that W-learning would have allowed A_1 to reach its reward in the example above. It would have a much higher W-value than any other individual.

Our four approaches to action selection all decide competition based only on the current state x and the Q-values the agents have for x. [Whitehead et al., 1993] call this a *simple merging strategy*, but are too quick to move on to more complex methods without considering what properties a Minimize the Worst Unhappiness simple merging strategy could have.

16.4.2 What the Global reward function cannot express

So we have argued that Minimize the Worst Unhappiness is the best way of maximising the ongoing sum of rewards for the whole creature using local information only - and hence the best way of maximising the sum of rewards at all, since monolithic methods are impractical.

Now we argue that the sum of rewards, or even any linear combination of them (like the global reward function in §4.3.1), *cannot* fully express the behavior of the whole creature that we want. Because we don't want to see this global score maximised if the solution *drops* any individual subgoal. We want a good score according to the global reward and at the same time ensure that *all* subgoals are at some time attended to. This is what is hard to express within the RL framework.

It is difficult to express this constraint in a $P_{xa}(r)$ distribution which only considers the current state x. It is also difficult to express this constraint when we use the global reward function, as we did here, as a fitness function for a GA, generating a single numeric fitness score. If we want to make sure that every subgoal is attended to, we need to use Strategies 2 or 3 of the previous section, not Strategy 1.

But where does that leave the empirical results of this thesis? We're arguing that maximising the global reward function of §4.3.1 does *not* define completely how we want the creature to behave. But we argued in favour of our methods because of their success in maximising this global reward function.

In defence of this, first, we presented various examples and analysis to show that our methods will actually take care of individual subgoals where other methods will not. Also, we argued that in this case the reason our methods performed well on the global reward was in fact because they picked up rewards from all the subgoals, whereas monolithic methods learnt a cruder policy. And finally, just because the global test was flawed doesn't mean that the other methods would have performed *better* under a more ideal global test. Our analysis suggests they would have performed even worse.

It still shows that the comparison of the methods was not ideal though. We used Strategy 1 so that all methods would have a fair comparison. But this was still somewhat unsatisfactory because we lack a way of explicitly penalising methods that ignore subgoals.

Chapter 17

Further work

Finally, we examine possible further work to consider where this approach to action selection is leading.

17.1 Further empirical work

This thesis may look like some kind of definitive empirical comparison of these methods but it is in fact far from the last word. The real contribution of this thesis is to invent these methods in the first place, show their motivation, their expected strengths and weaknesses, place them in context relative to each other, and show how the initial empirical results support what the theory would predict. There are a number of untested methods (though their expected performance has been discussed in the text):

- **Untested methods that we expect to be useful**
  - The pure Minimize the Worst Unhappiness when actions are shared (§13).
  - W-learning with full space with a declining Boltzmann distribution for W instead of random W winners (§10).
- **Untested methods that we do *not* expect to be interesting**
  - Other static measures of W, such as W = importance (§5.4). We expect them to have the same problem as static W=Q, of not being able to take opportunities (see §9.1).
  - Any *scaled* static or dynamic measures of W (§8.1.2,§15.4.5). These won't provide any advantage because we do not *want* to fix agents' inequalities relative to each other (see §8.1.2).

- The Collective methods (§12). [Whitehead et al., 1993] found that Maximize Collective Happiness performed not much different from W=Q (while both were better than monolithic Q-learning).
- The *product* method of Maximize Collective Happiness (§15.5.2) - once the correction is made that rewards (and hence Q-values) are all made > 0.
- The *standard deviation* method of Maximize Collective Happiness (§12.4).

The artificial world I used increased in complexity throughout the thesis, but as was revealed in §9.1, still proved too simple to properly separate the two approaches of Maximize the Best Happiness and Minimize the Worst Unhappiness. To properly contrast the two approaches, they need to be tested in an environment in which the rewards of opportunism (and the costs of not being opportunistic) are greater.

When testing the Collective methods, the world should also be one where some agents have non-recoverable states. This world probably doesn't have enough such states. If not obeyed at some point, an agent can almost always recover its position within a small number of steps.

As just discussed in §16.4.2, the definitive empirical test probably needs more than just a global fitness function as a linear combination of rewards. It should also include an *explicit* penalty for solutions that ignore subgoals.

17.2 Scaling up Reinforcement Learning

In scaling up Reinforcement Learning to complex problems (such as robot control), ways of breaking up the problem and its statespace are clearly needed. The problem is normally phrased (e.g. see [Kaelbling et al., 1996]) as that of task decomposition - breaking up the problem into subtasks that must be achieved in serial or in parallel.

It would be misleading to see this thesis as providing a solution to the general problem of decomposing problems into sub-tasks. Rather, its contribution is, *given* a collection of agents, find automatic ways of resolving their competition so that they can share the same creature in a sensible manner. Once put together, the agents in W-learning automatically find each other's weak spots, and discover the areas on which they can and cannot compromise.

Then the problem becomes one of designing a suitable collection of agents. This is not at all trivial, but it is hoped that it will be easier than designing the details of their action selection. We do have some rules of thumb to help in designing these collections - for example, to increase the influence of an agent in the creature's mind, slowly increase the differences between its rewards (recall §8.1.3).

This system has the property of being able to 'absorb' change - increasing an agent's strength doesn't guarantee that it will do better in every single state. In some states, a change of leader might reawaken a dormant agent, who suddenly finds that the agent that used to fight its battles for it is no longer winning. In other words, an agent increasing its strength will be unlikely to *suddenly* ruin all the other agents. They will give up their ground slowly, giving up less important states but holding on to the end to *their* crucial states. This is all done automatically.

The promise of RL is that designing a reward function will be easier than designing the behavior itself. Similarly, in addressing the Action Selection problem, it is hoped that designing well-balanced collections (and letting the competition resolution be automatic) will be easier than designing the detailed action selection itself.

17.2.1 What problems can this be applied to?

In general, we cannot expect to apply these techniques to *all* multi-goal problems. If goals are so different that they effectively *can't* be interrupted or interleaved, the ideas in this thesis may not be of much help.

Note however that the W-learning algorithm does not *force* goal-interleaving. It only *discovers* whether goal-interleaving is possible. Competition can actually result in serial behavior in practice. Imagine agents that have almost no areas of agreement, where one agent's best action is always the other's worst action, and vice versa. As soon as one agent gets started (gets to take a few actions down its goal path) it will be too strong compared to the others that are not started, since it will have taken us into states where they can get no reward, yet at the same time they would cause it a considerable loss if they take over. Their loss by not getting their goal started cannot compete with its loss if it loses the lead. So it will fight them off and run until completion. Then another agent gets a chance, and runs to termination. And so on. In practice, goals are satisfied serially in this case. This is simply an extreme version of the persistence-enforcing positive-feedback mechanism (§15.1.3).

17.2.2 Competition in Single Goal problems

This thesis has been concerned with the Action Selection problem - the problem of choice between multiple conflicting goals at run-time. I now turn all this on its head by applying it to the problems that Reinforcement Learning has been primarily interested in - problems having one single goal.

Nothing in our model forces the competing agents to be actually trying to solve *different* goals. To solve a single-goal problem, we put together a large number of agents with different reward functions but all roughly trying to solve the same thing - each perhaps taking radically different approaches, with statespaces and actionspaces that share nothing in common - and we let them struggle via W-learning to solve it. If there are multiple unexpressed behaviors, and lots of overlap, this may be a small price to pay if there is a robust solution.

This has some resemblance to Minsky's "accumulation" of multiple different ways of solving a problem in his Society of Mind [Minsky, 1986]. Some may be more efficient than others, but we keep them all for robustness. Or more subtly, it might be hard to rank them in a single order. Some may be more efficient some times (for some x) but *less* efficient other times (for other x).

Having a noisy collection of competing agents to solve the problem may seem a dangerous strategy, but remember that if the creature is looking as if it is going to make some serious error, individual agents will come in with large W-values and take it all over for a while. We positively want collections that single agents can take over and dominate (such as Minimize the Worst Unhappiness) if they feel strongly enough.

Note that we could try something like this in Hierarchical Q-learning too (but would pay the price of a large (x, i) statespace).

17.3 Different source of numbers

In this thesis, I introduced Reinforcement Learning first, and then showed how these numerical values could be propagated further to solve the action selection problem. But I could just as easily have introduced the action selection methods and left open where the numerical values came from. They could also be evolved randomly, designed explicitly, or come from some other learning method.

17.3.1 Symbolic W-learning

In particular, any system that builds up 'fitness' values for actions $F(x, a)$ can be used as the basis of W-learning type schemes. If a symbolic AI system can answer either the question before the fact: "You want to take action a_i. If I force you to take action a_k, how bad will that be?" or the question after the fact: "OK I ignored you and did something. How bad was it?" then a symbolic AI W-learning system can be built.

17.4 Parallel W-learning

As should have been obvious throughout, and as is illustrated by Figures 5.3 and 6.1, W-learning almost demands a parallel implementation. Agents can work out their suggested action and update their Q and W values independently, there being no serial links between them at all.

Most parallel multi-agent systems postulate low-bandwidth communications between agents anyway. With basic (not Negotiated) W-learning there is *perfect parallelism*, where there is no communication between agents at all. What communication there is takes place between agents and the switch. Similar to schemes such as Maes' Spreading Activation Networks [Maes, 1989, §7.2], no problem-specific agent language is used. Only numbers are communicated.

Consider again Watkins' motivation [Watkins, 1989] of finding learning methods that might plausibly occur in nature, in particular methods that impose limited computational demands per timestep. Once the Q-values have been learnt, Q-learning provides a fast control policy - ignore rewards, don't learn, and simply act on the best Q-value. A Q-learning-controlled creature is a pure stimulus-response machine. Parallel W-learning would retain all the speed of stimulus-response in a more sophisticated architecture.

How to parallelise is explained in more detail above in §14. See the 'Number of updates required per timestep per agent' and also the 'Restrictions on Decentralisation'.

17.5 The Bottom-up Animat approach

The whole motivation for the behavior-based AI project (see [Dennett, 1978]) is to understand simple complete creatures and gradually move on to more complex complete creatures (as opposed to the traditional AI approach of trying to understand sub-parts of already-complex creatures in the hope of combining these parts into an understanding of a whole complex creature).

The behavior-based AI approach involves a certain *discipline* in not moving on to more complex models until simpler ones have been exhausted and understood. There is a great temptation in the House Robot problem to start equipping the creature with internal working memory, context, world models, map-building ability, a concept of time, plans, explicit goals, representations, symbols, reasoning, and so forth. One ends up with a hand-designed ad-hoc system from which very few if any general principles can be learnt.

I take the broad approach of Wilson [Wilson, 1990] in pushing the limits of simple animat models, and resisting as long as possible the temptation to introduce more complex models:

> *"The essential process is ... given an environment and an animat with needs and a sensory/motor system that satisfies these needs to some criterion, increase the difficulty of the environment or the complexity of the needs - and find the minimum increase in animat complexity necessary to satisfy the needs to the same criterion."* [Wilson, 1990]

For example, Todd et al. [Todd et al., 1994] show that even the possibilities of sensor-less(!) memory-less creatures have not yet been fully explored. Their creatures find (by evolution) the optimum $p(a)$, the probability of generating an action (independent of the state of the world).

Certainly it is clear from this thesis that the possibilities of stimulus-response have not remotely been exhausted. In particular, *societies* of stimulus-response agents. Instead of building more complex components, I'm heading in the other direction, of *more complex societies* of relatively simple components. Actually I call these relatively simple components because they are stimulus-response, but in fact making hierarchies or societies out of components as complex as entire learning behaviors has only been tried relatively recently.

Where all this is leading is towards ecosystem-like minds, with structure in their collections, multiple competitions, and dynamic creation and destruction of agents. But I have been building carefully from the bottom up, instead of jumping direct to such complex systems.

17.6 Dynamically changing collections of agents

17.6.1 The Ecosystem of Mind

This work was partly inspired by Edelman's vision of a "rainforest" mind, with life and death, growth and decay, in a dynamic ecosystem inside the head [Edelman, 1989, Edelman, 1992]. In his model, called *Neural Darwinism*, competition is between structures called *neuronal groups*.

The idea is appealing, but it is difficult to tell whether neuronal groups are meant to be action-producing agents or just pattern classifiers. Edelman presents no explicit algorithm to show how the model is implemented. In fact, presenting arguments only about symbolic AI, he draws the familiar conclusion that no computer algorithm[1] could implement his ideas. No mention is made of self-modifying, reinforcement-learning agents embedded in the world, to which his criticisms do not apply.

Here, with W-learning, we have the basis for a full living and dying ecosystem inside the creature, where what comes into existence, struggles, and perhaps dies is not mere connections (as Edelman may in fact mean) but entire autonomous goals. In standard W-learning, there is one competition between the agents, this is resolved, and the creature is then fixed. In a *dynamic* collection, agents would continually adjust their W-values as new agents came into existence, old agents died, and the nature of their competition changed.

17.6.2 Invasion of strong new agents

In the earlier version of this document [Humphrys, 1996a, §17.5.1] I briefly discussed the possibilities for constructing dynamically changing collections. The basic problem with a dynamic collection though, is, if there is an automatic generator of random new agents, that it can just generate a single new, extremely strong agent (large differences between its rewards) which immediately takes over the whole creature, turning a carefully-built creature into a zombie overnight.

In fact, there would be an 'arms race' between such strong agents over the lifetime of the creature, as the agent-generator generated an even stronger agent which took over in its turn, and so on. The creature would change from being one type of zombie to another over its life. Our dilemma is caused by the fact that we want the mind to be driven by strong individuals in each state (we don't want woolly collective decision making). But then one strong individual can take over.

There are a number of possible ways of making collections hard to invade without going back to majority-rule, but at this point we must ask how useful this line of thinking really is from the point of view of engineering.

When thinking about animal minds, it is attractive to think of new agents (ideas, memes) coming into existence during the life history, and trying to claim some space in the already-crowded mind. But from the point of view of engineering, concepts like 'individual' are only useful myths. If a collection is

[1] This may all be a matter of terminology. Edelman's co-worker Olaf Sporns says [Sporns, 1995] that they do not like to use the word *algorithm* for a self-modifying algorithm embedded in the real world. A computer scientist would say that it still is an algorithm, even if it has effectively become part of the world.

invaded and destroyed, we simply note that the new collection didn't work, reset it to what it used to be and try something else. Resisting invasion may be important in biology to preserve the integrity of the individual, but in engineering this is just another way of building things. Only if we sent our robot out into the world, with the ability to communicate, imagine, and form new goals *unsupervised* over long periods of time, might questions of maintaining individual integrity start to arise. Even then, an artificial system can restore a past backup in a way that a natural system cannot.

So for the moment, dynamic collections are not so interesting since in their basic form they are just another way of finding combinations of agents in the same collection. More interesting for now is to discuss whether the collection itself can start to have any structure. This is the subject of the final chapter.

Chapter 18

The general form of a Society of Mind based on Reinforcement Learning

We can produce one final summary diagram, showing how the methods in this thesis might scale up to large numbers of agents. This is really part of the Further work chapter, but I have separated it off since it is self-contained, and summarises the direction that all this work is heading in.

Consider what a contrast the idea of multiple agents, with overlapping behaviors, driven by strong individuals, is to normal hand-designed systems, where task decomposition is accomplished in a parsimonious manner, with functions clearly parcelled out to different modules and a reluctance to allow any waste or overlap.

Consider the waste involved in a society of multiple competing agents solving the same problem. Or indeed, in a society of multiple agents pursuing different goals but needing to learn common skills to solve them. Where there is common functionality needed by two agents, e.g. both need to know how to lift the left leg, the normal engineering approach would be to encode this as a single function that both can call. Here in contrast, how to lift the left leg is learnt (in perhaps different ways) by both agents during development, and the information learnt is stored (perhaps redundantly) in two different places.

But recall from the discussion in §15.3.3 that just because an agent has learnt a skill doesn't mean that the agent actually wants to be obeyed. If it finds that another agent has learnt the same skill more efficiently, it will find that it does better when its own rudimentary action is *not* taken. Its W-values will automatically drop so that it does not compete with the more efficient agent (as long as the other agent is winning). The rudimentary skill serves no purpose except as a back-up in case of "brain damage". If the winning agent is lost or damaged, another agent will rouse itself (bring its W-values back up) to do the task (in its own slightly different way) since the task is not being done for it any more.

Instead of brittle task decomposition we have wasteful, overlapping task decomposition. The skill might even be *distributed* over a number of overlapping, winning agents, depending on which one got ahead first in each state (for example, A_u and A_s in §8.2). If one of them is lost through brain damage, the distribution of the skill among the remaining agents shifts slightly.

Let us formalise this notion that another agent might know better than you how to do your job. Throughout this thesis, we have generally assumed that agents know best how to maximize their own rewards, and therefore that their W-values are positive and they try to win. In a Markov Decision Process, agent A_i maximizes its rewards, but *only* relative to a particular state space, action space and reward structure $P^i_{xa}(r)$. These 3 things have to be designed, and the designer might omit senses or actions that could contribute towards rewards (as we saw in §7.1.1), or the designed reward structure might not be rich enough to make the rewards at subgoals distinguishable from noise. One might argue that this means the world is not really an MDP, but the answer is that almost all complex worlds aren't MDP's anyway - we assume already that we are just trying to approximate one (recall §4.2). So we have from A_i's point of view, there are some states x in which it would be better for it if it let another agent A_k win. A_i may actually have the same suite of actions as A_k, but may for some reason be unable to learn what A_k is doing (e.g. A_k may appear to have a non-deterministic policy, because it sees many states where A_i sees only one, as in §7.1.1). In any case, if A_i *observes* the average reward when A_k wins, then it will see that it would pay on average to let it win.

Under my system, A_i will drop out of the competition if A_k is winning. But if A_k is *not* winning, then my model has a problem. A_i cannot use A_k's trick, whatever it is, but has to try and compete itself, using its less efficient strategy. For instance, when I need to wander, ideally I give way to a highly-specialised Explore agent which does nothing else. But if Explore isn't able to win on its own, I have to come in with my own rudimentary version of the skill.

18.1 Nested W-learning

Digney's *Nested Q-learning* [Digney, 1996] shows us an alternative way, where A_i can *force* A_k to win, even if A_k couldn't manage to win on its own. In the basic setup (Figure 18.1), each agent is a combination *both* of a normal Q-learning agent and a Hierarchical Q-learning switch. Each agent has its own set of actions $Q_i(x,a)$ *and* a set of actions $Q_i(x,k)$ where action k means "do whatever agent A_k wants to do".

Wixson [Wixson, 1991] seems to have invented an earlier, but more restricted, form of Nested Q-learning, where the called agent is called not just to take an action for this timestep, but to take actions repeatedly from now on until its goal state is reached. Digney's model (like ours) does not demand the existence of terminating goal states. The action selection is revisited every time step. In Digney's notation the choice of possible actions is:

$$u \in \{Q^{a_1}, \dots Q^{a_J}, Q^{f_1}, \dots Q^{f_I}\}$$

In our notation this would mean that we build up Q-values for J real actions plus I 'actions' of calling other agents:

$$a \in \{a_1, \dots a_J, 1, \dots I\}$$

Each agent has its own statespace and its own actionspace. It can learn $Q_i(x,a)$ as normal, and can also learn $Q_i(x,k)$ in a normal manner. Every time A_k wins state x, A_i can update $Q_i(x,k)$ based on the state y we go to and the reward r_i that is received. It might find that in some states there is a k such that $Q_i(x,k)$ is greater than any $Q_i(x,a)$.

Of course the difference between the agent and a Hierarchical Q-learning switch is that the agent may not be *obeyed* when it says "do whatever A_k wants to do". It has to fight for A_k using its own W-value $W_i(x)$. The advantage of this model is that agents could specialise, and use function from other agents without the other agent needing to be strong enough to win by its own rights. For example, the efficient, specialised Explore agent might almost *never* win on its own, but regularly win because it was being promoted by somebody else.

Digney's actual model is more like a hierarchy [Digney, 1996, §2.3] with the Action Selection among the agents decided similar to a Hierarchical Q-learning switch with a global reward function. What makes our model Nested W-learning rather than Nested Q-learning is that we will retain the Action Selection ideas of this thesis, while applying them to agents that are Digney's Nested Q-learners.

Here the Action Selection in Figure 18.1 would still be decided on the basis of Minimize the Worst Unhappiness. The winning agent either takes an action itself, or immediately orders someone else to take an action. But the principle of Action Selection is unchanged - the winning agent is using

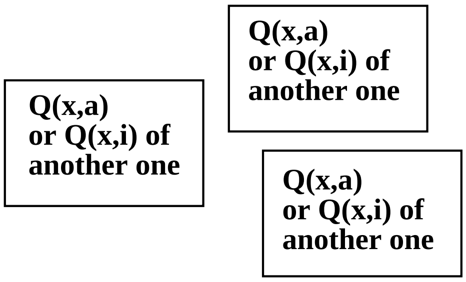


Figure 18.1: The simplest form of Nested W-learning. Each agent either suggests its own action $Q(x, a)$ or learns to suggest the action of another agent $Q(x, i)$ (without necessarily understanding the other agent's state or actionspace). The Action Selection between the agents is still on the basis of Minimize the Worst Unhappiness. That is, we apply the ideas of this thesis to a collection of agents that are *Nested* Q-learners.

function from other agents for *its own* purposes, not for the purposes of the other agent (much as the other agent will be happy to be thus used, and will drop out of the competition if being defended by someone else's W-value).[1]

W-learning learns in effect a *single* $Q_i(x, k)$ value for whoever the leader happens to be. What actual action the leader takes may seem to vary (e.g. because A_k sees different states where A_i sees only one) so W-learning concentrates on just building up an average $Q_i(x, k)$ instead of trying to find out what action(s) the leader is actually executing and building up $Q_i(x, a_k)$ values. The agent understands what action k means, but only for a *single* k.

Nested W-learning learns detailed $Q_i(x, k)$ values for many other agents and then can suggest which it would prefer to win. Again their actions may seem to vary so it concentrates on building up a Q-value for action k rather than for any particular real action a_k. Once all the agents have learnt such detailed $Q_i(x, k)$ values, we need to do Action Selection on these as our base Q-values. We can now actually do pure Minimize the Worst Unhappiness, instead of W-learning, even if agents do not share the same actions. Because now even if A_i does not understand action a_k, it certainly understands action k.

An interesting question that arises in Figure 18.1 is whether an infinite loop could develop. For example, A_i normally takes action a and A_j normally takes action b. A_i starts to notice that b is actually better for it, just as A_j

[1]So this is *not* like the scheme we suggested in §15.8, where the agent wants to win, but accepts it can't and votes for who it would like to win instead. Here, taking another agent's action may *look* like a compromise, but in fact the winning agent hasn't compromised at all.

notices that a is actually better for *it*. A_i switches to calling A_j just as A_j's preferred action becomes to call A_i. At this point, if either of them wins, an infinite loop will ensue. One solution is to choose preferred actions using only a soft max (§2.2.3). Another is to design the possible $Q(x, i)$ interactions so that a chain of commands cannot loop back on itself (see Figure 18.3 shortly).

18.1.1 The generic form of Nested W-learning

In the generic scheme (Figure 18.2) there are fundamentally two classes of agents, those competing for Action Selection and those not. In Figure 18.2, Action Selection takes place only among the agents in the top layer. The lower layer agents only *ever* execute their actions when called by a winning top layer agent (when called thus they simply execute their best action according to their Q-values). For instance, in the collection of 8 agents for the House Robot problem as defined in §8, we could have 6 of them in the competitive top layer and put A_c and A_w in the lower layer. At the lower level, we have specialised Explore agents. At the top level, we follow some goal or else call the specialised Explore.

In the generic case, agents may consist of $Q(x, a)$ only, $Q(x, i)$ only (the agent gets *all* its work done by calling other agents), or a combination of both. Note that if we take agents out of the Action Selection competition, then we only get to see what happens when they win when agents in the top layer get around to exploring different $Q(x, i)$.

Figure 18.1 can now be seen to be a special case of the generic scheme with the lower layer empty. Another special case would be the more familiar 2-layer hierarchy of Figure 18.3 (note that this is designed so that an infinite loop of commands is impossible). Another special case would be to have the same agents as in Figure 18.3 but include all of them in the Action Selection, the (formerly-lower) $Q(x, a)$ ones hardly ever winning on their own (this is more like what we actually had in §8). The basic Hierarchical Q-learning is also just another special case, with a single $Q(x, i)$-only agent in the top layer.

In a *multi-layer* hierarchy, agents in the Action Selection layer call agents in a lower layer, which call agents in a further lower layer, and so on. This is actually just another special case with all layers other than the Action Selection layer fitting inside the single 'lower layer' of the generic case. The agents in the 'lower layer' are then divided into groups with rules about which other agents an agent's $Q(x, i)$ can refer to. The advantage of setting up such rules (apart from ensuring again that no infinite loop can occur) would be to reduce the size of i in the $Q(x, i)$ statespace, that is, reduce the number of 'useful' agents that any one agent has to know about. As we have

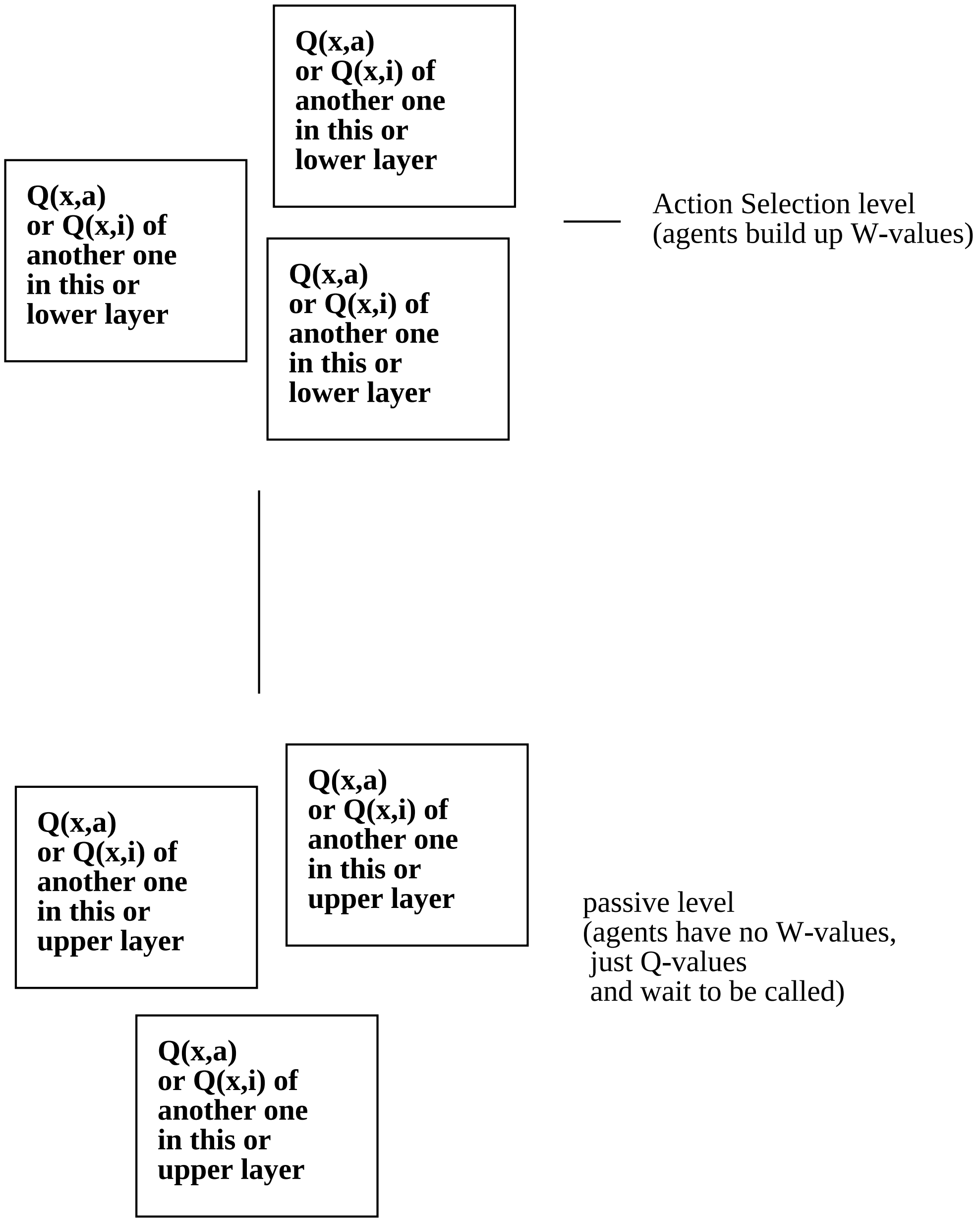


Figure 18.2: The generic form of Nested W-learning. Action Selection (on the basis of Minimize the Worst Unhappiness) takes place among the agents in the top layer only. The winner may or may not call an agent in the lower layer.

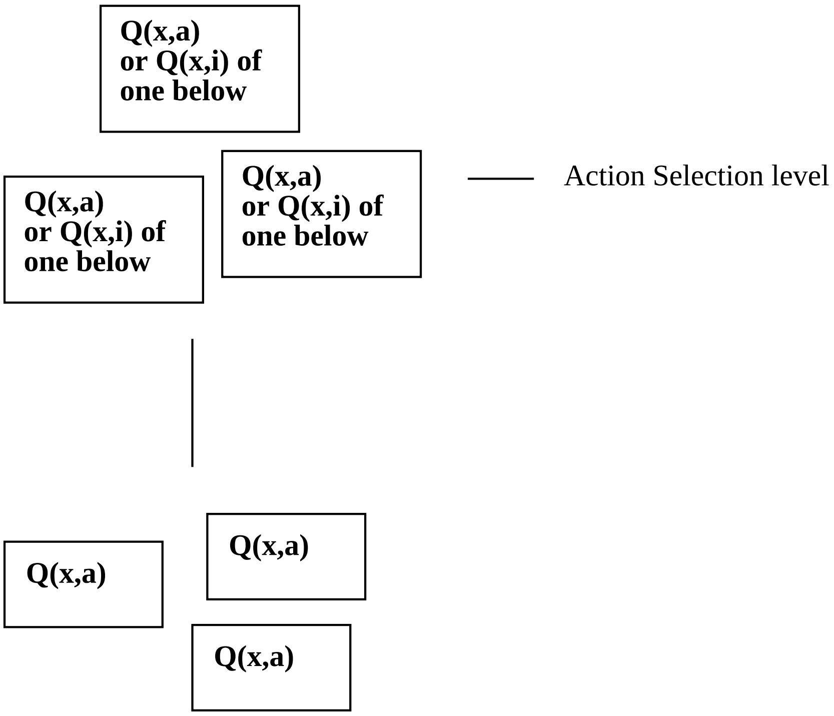


Figure 18.3: A typical form of Nested W-learning.

said before though (§15.2.1), hierarchies are only particular cases of the more general model, and not necessarily the most interesting cases either.

18.2 Feudal W-learning

Watkins' *Feudal Q-learning* [Watkins, 1989, §9] shows another way of having agents use other agents - by sending explicit orders.

In Feudal Q-learning, a *master* agent sends a command to a *slave* agent, telling it to take the creature to some state (which the master agent may not know how to reach on its own). The slave agent receives rewards for succeeding in carrying out these orders. To formalise, the master has a *current command* c in operation. This actually forms part of the 'state' of the slave. Using the notation `(x,c),a -> (y,c)`, the slave will receive rewards for transitions of the form:

```
(*,c),(*) -> (c,c)
```

Note that immediately the command c changes, we jump to an entirely new state for the slave and it may start doing something entirely different.[2]

Given a slave that has learnt this, the master will learn that it can reach a state by issuing the relevant command. Using the notation `x,a -> y`, it will note that the following will happen:

```
(*),(c) -> (c)
```

That is, the master learns that the world has state transitions $P_{xa}(y)$ such that $P_{xc}(c)$ is high. It then learns what actions to take based on whatever it wants to get done. Note that it can learn a chain - that $P_{xc}(y)$ is high and then $P_{yc}(c)$ is very high. So the model does not fail if the slave takes a number of steps to fulfil the order.

The master may be able to have a simpler state space as a result of this delegation. For example, in §7.1.1 A_f can order A_n to get it to the state where it is not carrying food (so that it can then set off for a new reward). A_n senses $x = (i, n, c)$ and takes 9 move actions. A_f senses $x = (i, f)$ and takes 9 move actions plus 2 command actions (when the creature reaches the nest, A_f will want to change the command to do nothing). The combined state and action memory requirements of A_f and A_n is 544, whereas for a monolithic A_f sensing $x = (i, n, f)$ and taking 9 move actions it would be 1620. For the W-learning agents in that section the combined state and action space was 261, but recall that A_f could not explicitly call A_n (it could only drop out of the competition and hope that A_n would win).

Again, Action Selection must take place somewhere, and the basic principle of Action Selection is unchanged. The master is using the slave for its own purposes. The slave indeed has no purposes of its own, and so cannot be in the Action Selection loop. In Feudal W-learning (as distinct from Feudal Q-learning) Action Selection will take place between master agents on the basis of Minimize the Worst Unhappiness. The winner will either take an action itself or send a command to a slave agent.

We can combine Nested and Feudal W-learning and the Action Selection ideas of this thesis in one grand diagram showing the general form of a Society of Mind based on Reinforcement Learning (Figure 18.4). Note that this structure is not a hierarchy in any meaningful sense. While the flow of control must originate with some winner in the Action Selection layer, after that control may flow in any direction.

[2] Actually there is a special case where a change of command happens just as the slave fulfilled the old command. We should still reward it - it's not its fault that the state has suddenly changed to `(c,d)`. So we should actually reward for the transition: `(*,c),(*) -> (c,*)`

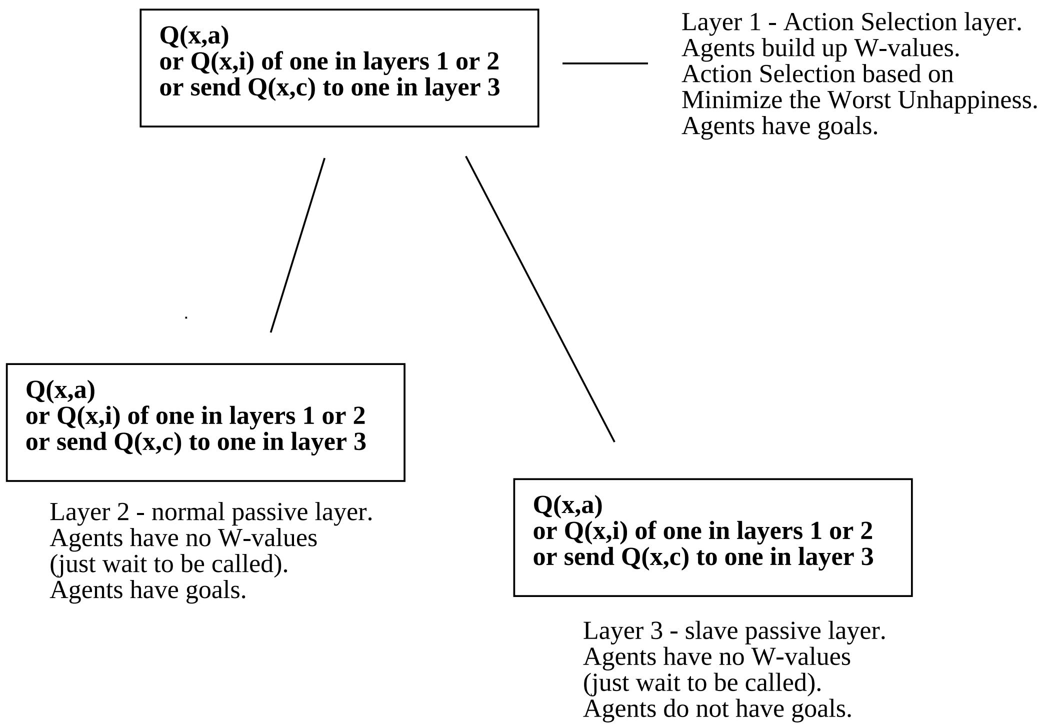


Figure 18.4: The general form of a Society of Mind based on Reinforcement Learning. This is *not* a hierarchy.

18.3 The wasteful, overlapping mind

Neither Nested nor Feudal approaches affect the basic analysis we made in this thesis of how Action Selection between competing peers should be resolved. The basic ideas running throughout this thesis, of overlap driven by strong individuals, of multiple unexpressed behaviors and agents dropping out of competitions, all have a parallel in Minsky's "If it's broken don't fix it, suppress it" maxim in the Society of Mind [Minsky, 1986]. In Minsky's model, the normally-good behavior is allowed expression most of the time, except in some states where a *censor* overrides it and prevents its expression. This would be like W-learning between a general solver of a problem and a highly-specific agent introduced to focus on one small area of statespace. The specialist remains quiet (has low W-values) for most of the statespace and only steps in (has high W-values) when the state x is in the area of interest to it.

For example, we could have specialised agents which are trained to look for disasters, combined with a more careless agent dedicated to solving the main problem. The main agent is allowed a free run except at the edges near to disasters, at which the previously quiet disaster-checkers will raise their W-values to censor it. The looming disaster might be obvious in the disaster-checker's sensory world but invisible in the main agent's sensory world. We end up with one agent A_k generally in charge, but being 'shepherded' as it goes along its route by a variety of other agents A_i, who are constantly monitoring its actions and occasionally rising up to block them.

Where all this is leading is away from the simplistic idea of a single thread of control. Any complex mind should have alternative strategies constantly bubbling up, seeking attention, wanting to be given control of the body. As Dennett [Dennett, 1991] says, the Cartesian Theatre may be officially dead, but it still haunts our thinking. We should not be so afraid of multiple unexpressed behaviors:

> *"We can suppose that all of this happens in swift generations of 'wasteful' parallel processing, with hordes of anonymous demons and their hopeful constructions never seeing the light of day ..."*
> [Dennett, 1991, §8.2]

The concept of ideas having to fight for actual expression is of course not original. The idea of competition between selfish sub-parts of the mind is at least as old as William James and Sigmund Freud. But what I have tried to do in this thesis is to provide some fully-specified and general-purpose algorithms rather than either unimplementable conceptual models or ad-hoc problem-specific architectures.

Acknowledgements

I am most indebted to Barney Pell for introducing me to Reinforcement Learning. Thanks to my supervisor John Daugman for his consistent support and advice. Thanks to Richard Prager, Tony Prescott and a number of anonymous referees for much essential criticism.

Credit to Ian Lewis and Gavin Rummery for important contributions, and thanks also to Ralph Becket, Marcel Hernandez, Steve Hodges, Clare Jackson, Dave Palfrey, Tony Robinson, Michael K.Sahota and Chen K.Tham for useful comments on this work. There are many other people that I will not list, with whom I have had useful and inspiring conversations about my work and related topics. They have made me so glad I chose to come to Cambridge.

Thanks to John Murphy for encouraging me to do a PhD. Sincere thanks to William Clocksin and Roger Needham for getting me started. This work was supported mainly by the British Council and by the University of Cambridge Computer Laboratory. I am grateful to the United States Office of Naval Research and the M.R. Bauer Foundation for a student travelship. I am also grateful for contributions from Trinity Hall, Cambridge, from the Cambridge Philosophical Society and from the University of Cambridge Board of Graduate Studies.

Thanks to Frank for his comments, thanks to Richard for astonishingly reading the whole thesis, and thanks in general to Richard, Frank and Joe for many years of entertainment. Much love to Elizabeth, and thanks for being there for the good part.

Appendix A

Incremental sampling of random variables

Most of the results in these appendices are either well-known or trivial, but they are included here for completeness, and so they can be referred to from the main body of the text.

A.1 Single variable (average)

Let $d_1, d_2, d_3, \ldots$ be samples of a stationary random variable d with expected value $E(d)$. Then the following update algorithm provides an elegant way of sampling them. Repeat:

$$D := (1-\alpha)D + \alpha d_i$$

Theorem A.1 *If α takes successive values $1, \frac{1}{2}, \frac{1}{3}, \ldots$, then $D \rightarrow E(d)$, independent of the initial value of D.*

Proof: D's updates go:

$$\begin{array}{ll} D := 0D_{init} + 1d_1 & = d_1 \\ D := \frac{1}{2}d_1 + \frac{1}{2}d_2 & = \frac{1}{2}(d_1 + d_2) \\ D := \frac{2}{3}\frac{1}{2}(d_1+d_2) + \frac{1}{3}d_3 & = \frac{1}{3}(d_1+d_2+d_3) \\ \ldots & \\ D = \frac{1}{t}(d_1 + \cdots + d_t) & \end{array}$$

i.e. D is simply the average of all d_i samples so far.
As $t \rightarrow \infty$, $D \rightarrow E(d)$. ■

More generally:

Theorem A.2 *If α takes successive values $\frac{1}{t}, \frac{1}{t+1}, \frac{1}{t+2}, \ldots$, then $D \to E(d)$, independent of the initial value of D, and independent of t.*

Proof: D's updates go:

$$\begin{array}{lll}
D & := \frac{t-1}{t} D_{init} + \frac{1}{t} d_1 & \\
D & := \frac{t}{t+1}\left(\frac{t-1}{t} D_{init} + \frac{1}{t} d_1\right) + \frac{1}{t+1} d_2 & = \frac{t-1}{t+1} D_{init} + \frac{1}{t+1}(d_1 + d_2) \\
D & := \frac{t+1}{t+2}\left(\frac{t-1}{t+1} D_{init} + \frac{1}{t+1}(d_1 + d_2)\right) + \frac{1}{t+2} d_3 & = \frac{t-1}{t+2} D_{init} + \frac{1}{t+2}(d_1 + d_2 + d_3) \\
\ldots & & \\
D & = \frac{t-1}{t+n} D_{init} + \frac{1}{t+n}(d_1 + \cdots + d_{n+1}) & \\
 & = \frac{t-1}{t+n} D_{init} + \frac{n+1}{n+t}\frac{1}{n+1}(d_1 + \cdots + d_{n+1}) &
\end{array}$$

As $n \to \infty$:

$$D \to 0 + 1E(d)$$

that is, $D \to E(d)$. ■

One way of looking at this is to consider D_{init} as the average of all samples before time t, samples which are now irrelevant for some reason. We can consider them as samples from a different distribution f:

$$D_{init} = \frac{1}{t-1}(f_1 + \cdots + f_{t-1})$$

Hence:

$$\begin{array}{ll}
D & = \frac{1}{n}(f_1 + \cdots + f_{t-1} + d_t + \cdots + d_n) \\
 & = \frac{1}{n}(f_1 + \cdots + f_{t-1}) + \frac{1}{n}(d_t + \cdots + d_n) \\
 & \to 0 + E(d)
\end{array}$$

as $n \to \infty$.

A.2 Single variable (time-weighted)

An alternative approach (e.g. see [Grefenstette, 1992]) is to build up not the average of all samples, but a *time-weighted* sum of them, giving more weight to the most recent ones. This is accomplished by setting α to be constant, in which case D's updates go:

$$
\begin{aligned}
&D := (1-\alpha)D_{init} + \alpha d_1 \\
&D := (1-\alpha)^2 D_{init} + (1-\alpha)\alpha d_1 + \alpha d_2 \\
&D := (1-\alpha)^3 D_{init} + (1-\alpha)^2\alpha d_1 + (1-\alpha)\alpha d_2 + \alpha d_3 \\
&\ldots \\
&D = (1-\alpha)^t D_{init} + \alpha\left((1-\alpha)^{t-1} d_1 + (1-\alpha)^{t-2} d_2 + \cdots + (1-\alpha) d_{t-1} + d_t\right)
\end{aligned}
$$

Since $(1-\alpha)^n \to 0$ as $n \to \infty$, this weights more recent samples higher. Grefenstette uses typically $(1-\alpha) = 0.99$.

A.3 Multiple variables

Consider sampling alternately from two stationary random variables d and f:

$$
\begin{aligned}
&D := 0 D_{init} + 1 d_1 \\
&D := \tfrac{1}{2} D + \tfrac{1}{2} f_1 \\
&D := \tfrac{2}{3} D + \tfrac{1}{3} d_2 \\
&D := \tfrac{3}{4} D + \tfrac{1}{4} f_2 \\
&D := \tfrac{4}{5} D + \tfrac{1}{5} d_3 \\
&\ldots
\end{aligned}
$$

Theorem A.3 $D \to \frac{1}{2}E(d) + \frac{1}{2}E(f)$

Proof: From Theorem A.1, after $2t$ samples:

$$
\begin{aligned}
D &= \tfrac{1}{2t}(d_1 + f_1 + d_2 + f_2 + \cdots + d_t + f_t) \\
&= \tfrac{1}{2}\tfrac{1}{t}(d_1 + \cdots + d_t) + \tfrac{1}{2}\tfrac{1}{t}(f_1 + \cdots + f_t) \\
&\to \tfrac{1}{2}E(d) + \tfrac{1}{2}E(f)
\end{aligned}
$$

as $t \to \infty$. ■

More generally, consider where we have multiple stationary distributions $d^{(1)}, \ldots, d^{(n)}$, where each distribution $d^{(i)}$ has expected value $E(d^{(i)})$. At each time t, we take a sample $d_t^{(i)}$ from one of the distributions. Each distribution $d^{(i)}$ has probability $p(i)$ of being picked. Repeat:

$$D := (1-\alpha)D + \alpha d_t^{(i)}$$

Theorem A.4 *If the distribution p is stationary, then $D \to p(1)E(d^{(1)}) + \cdots + p(n)E(d^{(n)})$.*

Proof: For a large number of samples t we expect to take $p(1)t$ samples from $d^{(1)}$, $p(2)t$ samples from $d^{(2)}$, and so on. From Theorem A.1, we expect:

$$\begin{aligned}
D &= \tfrac{1}{t}(d_1^{(1)} + \cdots + d_{p(1)t}^{(1)} + d_1^{(2)} + \cdots + d_{p(2)t}^{(2)} + \cdots + d_1^{(n)} + \cdots + d_{p(n)t}^{(n)}) \\
&= p(1)\tfrac{1}{p(1)t}\left(d_1^{(1)} + \cdots + d_{p(1)t}^{(1)}\right) + \cdots + p(n)\tfrac{1}{p(n)t}\left(d_1^{(n)} + \cdots + d_{p(n)t}^{(n)}\right) \\
&\to p(1)E(d^{(1)}) + \cdots + p(n)E(d^{(n)})
\end{aligned}$$

as $t \to \infty$. ■

Appendix B

Bounds

B.1 Bounds with a learning rate α

Let D be updated by:

$$D := (1-\alpha)D + \alpha d$$

where d is bounded by $d_{\max}$, $d_{\min}$, and the initial value of $\alpha = 1$. Then:

Theorem B.1 *D is also bounded by $d_{\max}$, $d_{\min}$.*

Proof: The highest D can be is if it is always updated with $d_{\max}$:

$$\begin{array}{ll} D := 0D_{init} + 1d_{\max} & = d_{\max} \\ D := (1-\alpha)d_{\max} + \alpha d_{\max} & = d_{\max} \\ \cdots & \end{array}$$

so $D_{\max} = d_{\max}$. Similarly $D_{\min} = d_{\min}$. ■

B.2 Bounds of Q-values

Theorem B.2

$$\begin{array}{l} Q_{\max} = \frac{r_{\max}}{1-\gamma} \\ Q_{\min} = \frac{r_{\min}}{1-\gamma} \end{array}$$

Proof: In the discrete case, Q is updated by:

$$Q(x,a) := (1-\alpha)Q(x,a) + \alpha(r + \gamma \max_{b \in A} Q(y,b))$$

so by Theorem B.1:

$$\begin{aligned} Q_{\max} &= (r+\gamma Q)_{\max} \\ &= r_{\max} + \gamma Q_{\max} \\ Q_{\max} &= \tfrac{r_{\max}}{1-\gamma} \end{aligned}$$

This can also be viewed in terms of temporal discounting:

$$\begin{aligned} Q_{\max} &= r_{\max} + \gamma(r_{\max} + \gamma(r_{\max} + \cdots)) \\ &= r_{\max} + \gamma r_{\max} + \gamma^2 r_{\max} + \cdots \\ &= (1+\gamma+\gamma^2+\cdots) r_{\max} \\ &= \tfrac{1}{1-\gamma} r_{\max} \end{aligned}$$

Similarly:

$$\begin{aligned} Q_{\min} &= r_{\min} + \gamma Q_{\min} \\ Q_{\min} &= \tfrac{r_{\min}}{1-\gamma} \\ &= r_{\min} + \gamma r_{\min} + \gamma^2 r_{\min} + \cdots \end{aligned}$$

■

For example, if $\gamma = 0$, then $Q_{\max} = r_{\max}$. And (assuming $r_{\max} > 0$) as $\gamma \to 1$, $Q_{\max} \to \infty$.

Note that since $r_{\min} < r_{\max}$, it follows that $Q_{\min} < Q_{\max}$.

B.3 Bounds of W-values

Theorem B.3

$$\begin{aligned} W_{\max} &= Q_{\max} - Q_{\min} \\ W_{\min} &= -(Q_{\max} - Q_{\min}) \end{aligned}$$

Proof: In the discrete case, W is updated by:

$$W(x) := (1-\alpha)W(x) + \alpha(Q(x,a) - (r + \gamma \max_{b \in A} Q(y,b)))$$

so by Theorem B.1:

$$\begin{aligned} W_{\max} &= (Q - (r+\gamma Q))_{\max} \\ &= Q_{\max} - (r+\gamma Q)_{\min} \\ &= Q_{\max} - Q_{\min} \end{aligned}$$

by Theorem B.2.

Similarly:

$$W_{\min} = Q_{\min} - Q_{\max}$$

■

Note that since $Q_{\min} < Q_{\max}$, it follows that $W_{\min} < 0 < W_{\max}$.

Appendix C

2-reward reward functions

Consider an agent of the form:

A_i reward: if (good event) r else s

where $r > s$.

C.1 Policy in Q-learning

Theorem C.1 *Q-learning returns the same policy irrespective of the exact values of r and s, provided only that the inequality $r > s$ is maintained.*

Proof: Let us fix r and s and learn the Q-values. In a deterministic world, given a state x, the Q-value for action a will be:

$$\begin{aligned} Q(x,a) &= r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} + \cdots \\ &= \textstyle\sum_i \gamma^i r + \sum_j \gamma^j s \ \text{ for various } i,j \\ &= cr + ds \end{aligned}$$

for some real numbers $c + d = 1 + \gamma + \gamma^2 + \cdots$. The Q-value for a different action b will be:

$$\begin{aligned} Q(x,b) &= \textstyle\sum_k \gamma^k r + \sum_l \gamma^l s \ \text{ for various different } k,l \\ &= er + fs \end{aligned}$$

where $e + f = 1 + \gamma + \gamma^2 + \cdots$. That is, $e + f = c + d$.

So whichever one of c and e is bigger defines which is the best action (which gets the larger amount of the 'good' reward r), irrespective of the sizes of $r > s$. ■

Note that these numbers are not integers - it may not be simply a question of the worse action receiving s instead of r a finite number of times. The worse action may also receive r instead of s at some points, and also the number of differences may in fact not be finite.

To be precise, noting that $(c-e) = (f-d)$, the difference between the Q-values is:

$$\begin{aligned} Q(x,a) - Q(x,b) &= (c-e)r + (d-f)s \\ &= (c-e)r - (c-e)s \\ &= (c-e)(r-s) \end{aligned}$$

where the real number $(c-e)$ is constant for the given two actions a and b in state x. $(c-e)$ depends only on the probabilities of events happening, not on the specific values of the rewards r and s that we hand out when they do. Changing the relative sizes of the rewards $r > s$ can only change the *magnitude* of the difference between the Q-values, but not the *sign*. The ranking of actions will stay the same.

For example, an agent with rewards (10,9) and an agent with rewards (10,0) will have different Q-values but will still suggest the same optimal action a_i.

In a probabilistic world, we would have:

$$E(r_t) = pr + qs$$

where $p + q = 1$, and:

$$\begin{aligned} E(r_{t+1}) &= \sum_y P_{xa}(y)(p_y r + q_y s) \quad \text{where each } p_y + q_y = 1 \\ &= \sum_y p_y P_{xa}(y) r + \sum_y q_y P_{xa}(y) s \\ &= p'r + q's \end{aligned}$$

for some $p' + q' = 1$. Hence:

$$\begin{aligned} Q(x,a) &= (pr + qs) + \gamma(p'r + q's) + \gamma^2(p''r + q''s) + \cdots \\ &= (p + \gamma p' + \gamma^2 p'' + \cdots)r + (q + \gamma q' + \gamma^2 q'' + \cdots)s \\ &= cr + ds \end{aligned}$$

for some $c + d = 1 + \gamma + \gamma^2 + \cdots$ as before.

C.2 Strength in W-learning

Theorem C.2 *The strength of the agent in W-learning is simply proportional to* $(r - s)$.

Proof: From the proof of Theorem C.1:

$$\begin{aligned} W_i(x) &= Q_i(x, a_i) - Q_i(x, a_k) \\ &= c_{ki}(x)(r - s) \end{aligned}$$

where $c_{ki}(x)$ is a constant independent of the particular rewards. ■

Using our 'deviation' definition, for the 2-reward agent in a deterministic world:

$$d_{ki}(x) = c_{ki}(x)(r - s)$$

The size of the W-value that A_i presents in state x if A_k is the leader is simply proportional to the difference between its rewards. If A_k wants to take the same action as A_i, then $c_{ki}(x) = 0$ (that is, $(c - e) = 0$). If the leader switches to A_l, the constant switches to $c_{li}(x)$.

Increasing the difference between its rewards will cause A_i to have the same disagreements with the other agents about what action to take, but higher $W_i(x)$ values - that is, an increased ability to compete. So the progress of the W-competition will be different.

For example, an agent with rewards (8,5) will be stronger (will have higher W-values and win more competitions) than an agent with the same logic and rewards (2,0). And an agent with rewards (2,0) will be stronger than one with rewards (10,9). In particular, the strongest possible 2-reward agent is:

A_i reward: if (good event) $r_{\max}$ else $r_{\min}$

C.3 Normalisation

Any 2-reward agent can be normalised to the form:

A_i reward: if (good event) $(r - s)$ else 0

From Theorem C.1, this will have different Q-values but the same Q-learning policy. And from Theorem C.2, it will have identical W-values. You can regard the original agent as an $(r - s), 0$ agent which also picks up an automatic bonus of s every step no matter what it does. Its Q-values can be obtained by simply adding the following to each of the Q-values of the $(r-s), 0$ agent:

$$\begin{aligned} & s+\gamma s+\gamma^2 s+\cdots \\ & =\tfrac{s}{1-\gamma} \end{aligned}$$

We are shifting the *same* contour up and down the y-axis in Figure 8.1.

The same suggested action and the same W-values means that for the purposes of W-learning it is the same agent. For example, an agent with rewards (1.5,1.1) is identical in W-learning to one with rewards (0.4,0). The W=Q method would treat them differently.

C.4 Exaggeration

Say we have a normalised 2-reward agent A_1:

A_1 reward: if (good event) r else 0

where $r > 0$.

Theorem C.3 *If agent A_2 is of the form:*

A_2 *reward: if (good event) cr else 0*

where the condition is the same, then:

$$Q_2^*(x,a) = cQ_1^*(x,a) \;\; \forall x, a$$

Proof: We have just multiplied all rewards by c, so all Q-values are multiplied by c. If this is not clear, see the general proof Theorem D.1. ■

A_2 will have the same policy as A_1, but different W-values. We are *exaggerating* or levelling out the contour in Figure 8.1. In particular, the strongest possible normalised agent is:

A_i reward: if (good event) $r_{\max}$ else 0

Appendix D

3-reward (or more) reward functions

For 3-reward (or more) agents the relative sizes of the rewards *do* matter for the Q-learning policy. Consider an agent of the form:

A_i reward: if (best event) r_1 else if (good event) r_2 else r_3

where $r_1 > r_2 > r_3$.

D.1 Policy in Q-learning

We show by an example that changing one reward in this agent while keeping others fixed can lead to a switch of policy. Imagine that currently actions a and b lead to the following sequences of rewards:

(x, a) leads to sequence r_3, r_3, r_3, r_3, r_1 and then $r_2, \ldots$ forever
(x, b) leads to sequence r_2, r_2, r_2, r_2, r_2 and then $r_2, \ldots$ forever

Currently action b is the best. We lose $r_2 < r_1$ on the fifth step certainly, but we make up for it by receiving the payoff from $r_2 > r_3$ in the first four steps. However, if we start to increase the size of r_1, while keeping r_2 and r_3 the same, we can eventually make action a the most profitable path to follow and cause a switch in policy.

D.2 Strength in W-learning

Because increasing the gaps between rewards may switch policy, we can't say that in general it will increase W-values. In the example above, say the leader A_k was suggesting (and executing) action a all along. By increasing the gaps between our rewards, we suddenly want to take action a ourself, so $W_i(x) = 0$.

Increasing the difference between its rewards may cause A_i to have *new* disagreements, and maybe new agreements, with the other agents about what action to take, so the progress of the W-competition may be radically different. Once a W-value changes, we have to follow the whole re-organisation to its conclusion.

What we can say is that multiplying all rewards by the *same* constant (see §D.4 shortly), and hence multiplying all Q-values by the constant, will increase or decrease the size of all W-values without changing the policy.

D.3 Normalisation

Any agent with rewards $r_1, r_2, \ldots, r_{n-1}, r_n$ can be normalised to one with rewards $(r_1 - r_n), (r_2 - r_n), \ldots, (r_{n-1} - r_n), 0$. The original agent can be viewed as a normalised one which also picks up r_n every timestep no matter what.

The normalised agent will have the same policy and the same W-values.

D.4 Exaggeration

Theorem D.1 *If agent A_1 has some reward function with rewards $r_1, r_2, \ldots, r_n$ and agent A_2's reward function has the same logic, but with rewards $cr_1, cr_2, \ldots, cr_n$, where c is some constant, then:*

$$Q_2^*(x,a) = cQ_1^*(x,a) \ \ \forall x, a$$

Think of it as changing the "unit of measurement" of the rewards.

Proof: When we take action a in state x, let $P_{xa}(i)$ be the probability that reward r_i is given to A_1 (and therefore that reward cr_i is given to A_2). Then A_2's expected reward is simply c times A_1's expected reward:

$$\begin{aligned} E_2(r_t) &= \sum_i (cr_i) P_{xa}(i) \\ &= c \sum_i r_i P_{xa}(i) \\ &= cE_1(r_t) \end{aligned}$$

It follows from the definitions in §2.1 that $V_2^*(x) = cV_1^*(x)$ and $Q_2^*(x,a) = cQ_1^*(x,a)$. ■

A_2 will have the same policy as A_1, but larger or smaller W-values.

Appendix E

Weighted sum and weighted mean

Consider a weighted sum of quantities d_i:

$$D = \rho_1 d_1 + \cdots + \rho_t d_t$$

A *weighted mean* is a weighted sum where $0 \leq \rho_i \leq 1$ such that $\sum_i \rho_i = 1$. The ordinary mean is a special case of this with $\rho_i = \frac{1}{t}$. A weighted mean may be greater than or smaller than the mean.

Appendix F

Full list of action selection methods

We only consider here what numbers one might generate at a single timestep where some action is being taken. That is, we only use the Q-value an agent has for the executed action or the loss it is suffering. We omit methods that use terms that don't mean anything in this timestep, such as (from §10):

$$\max_i \sum_k (Q_i(x, a_i) - Q_i(x, a_k))$$

We also only consider maximizing, minimizing and summing. We omit other possible methods, such as product (§15.5.2) or standard deviation (§12.4).

Many methods below make no sense, such as maximizing unhappiness. Ones that make some sense are in bold.

F.1 Search for compromise action

Search for this, and take action a:

$\max_a$	$\max_i$	$Q_i(x,a)$	**Maximize the Best Happiness (W=Q)**
$\max_a$	$\max_i$	$(Q_i(x,a_i) - Q_i(x,a))$	find worst loss and cause it
$\max_a$	$\min_i$	$Q_i(x,a)$	**Minimize the Worst Unhappiness (but with problems, see §13)**
$\max_a$	$\min_i$	$(Q_i(x,a_i) - Q_i(x,a))$	find worst minimum loss and cause it
$\max_a$	$\sum_i$	$Q_i(x,a)$	**Maximize Collective Happiness**
$\max_a$	$\sum_i$	$(Q_i(x,a_i) - Q_i(x,a))$	maximize collective unhappiness

$\min_a$	$\max_i$	$Q_i(x,a)$	find worst maximum reward and take it
$\min_a$	$\max_i$	$(Q_i(x,a_i) - Q_i(x,a))$	**Minimize the Worst Unhappiness**
$\min_a$	$\min_i$	$Q_i(x,a)$	find worst reward and take it
$\min_a$	$\min_i$	$(Q_i(x,a_i) - Q_i(x,a))$	**cause the smallest unhappiness (just obey someone, see §9)**
$\min_a$	$\sum_i$	$Q_i(x,a)$	minimize collective happiness
$\min_a$	$\sum_i$	$(Q_i(x,a_i) - Q_i(x,a))$	**Minimize Collective Unhappiness**

F.2 Use only suggested actions

Search for this, and take action a_k:

$\max_k$	$\max_i$	$Q_i(x, a_k)$	**Maximize the Best Happiness (W=Q)**
$\max_k$	$\max_i$	$(Q_i(x, a_i) - Q_i(x, a_k))$	find worst loss and cause it
$\max_k$	$\min_i$	$Q_i(x, a_k)$	**Minimize the Worst Unhappiness (but with problems, see §13)**
$\max_k$	$\min_i$	$(Q_i(x, a_i) - Q_i(x, a_k))$	find worst minimum loss and cause it (assume $i \neq k$)
$\max_k$	$\sum_i$	$Q_i(x, a_k)$	**Maximize Collective Happiness**
$\max_k$	$\sum_i$	$(Q_i(x, a_i) - Q_i(x, a_k))$	maximize collective unhappiness

$\min_k$	$\max_i$	$Q_i(x, a_k)$	find worst maximum reward and take it
$\min_k$	$\max_i$	$(Q_i(x, a_i) - Q_i(x, a_k))$	**Minimize the Worst Unhappiness (W-learning)**
$\min_k$	$\min_i$	$Q_i(x, a_k)$	find worst reward and take it
$\min_k$	$\min_i$	$(Q_i(x, a_i) - Q_i(x, a_k))$	**cause the smallest unhappiness (just obey someone, see §9)**
$\min_k$	$\sum_i$	$Q_i(x, a_k)$	minimize collective happiness
$\min_k$	$\sum_i$	$(Q_i(x, a_i) - Q_i(x, a_k))$	**Minimize Collective Unhappiness (Collective W-learning)**

Search for this, and take action a_i:

$\max_i$	$Q_i(x, a_i)$	**Maximize the Best Happiness (W=Q)**
$\min_i$	$Q_i(x, a_i)$	find worst reward and take it

In fact, we also omitted combinations of two of these terms. For instance, Tony Prescott suggested the following:

$$\max_k \left[Q_k(x, a_k) - \sum_{i, i \neq k} (Q_i(x, a_i) - Q_i(x, a_k)) \right]$$

That is, the gain by the winner, minus the losses it causes the others. Note that:

$$\sum_{i, i \neq k} (Q_i(x, a_i) - Q_i(x, a_k)) = \sum_i (Q_i(x, a_i) - Q_i(x, a_k))$$

This method seems also to inhabit that desirable middle ground between Maximize the Best Happiness and the Collective methods. It somewhat answers the criticism of Maximize the Best Happiness, in that if the winner would make a gain no matter what action is taken, then it will prefer actions that cause smaller losses to the other agents. We see that it would successfully be opportunistic in the example in §9.1. And it is still more individual-driven than the pure Collective methods - it successfully picks A_1 in the first example in §12.3.

But we only have to increase the number of agents to show that it is still vulnerable to the criticisms of any Collective method. It will fail to pick A_1 here:

a	(1)	(2)
Q1(x,a)	[10]	0
Q2(x,a)	3	[5]
Q3(x,a)	3	[5]
Q4(x,a)	3	[5]
Q5(x,a)	3	[5]
Q6(x,a)	3	[5]
Q7(x,a)	3	[5]
Q8(x,a)	3	[5]
Q9(x,a)	3	[5]

The only other method of combining two terms that makes sense is:

$$\max_k \left[\sum_i Q_i(x, a_k) - \sum_i (Q_i(x, a_i) - Q_i(x, a_k)) \right]$$

which again is just another Collective method.

Appendix G

Experimental Details

The experiments in this dissertation should not be regarded as completely definitive. As noted in §17.1, the complexity of the artificial world needs to increase if we are to properly separate the methods. Then comparisons can be run similar to the experiments here, the full details of which follow. The experiments here were all implemented in C++.

Q-learning (§4.3) - Monolithic Q-learner learns $Q(x,a)$ using the global reward function of §4.3.1. Q-values are stored in a neural network (in fact, for convenience, this is broken into one network per action a). 100 trials, each trial interacting with the world 1400 times and then replaying experiences 30 times. Policy improves over time using Boltzmann distribution (§2.2.3). Test over 20000 timesteps (interactions with world) to yield score according to global reward function of §4.3.1. The architecture of the network and coding of inputs was adjusted to get the best score of average 6.285 points per 100 timesteps.

Hand-coded program (§4.3.3) - A range of strictly-hierarchical programs were designed, with both deterministic and stochastic policies. Test over 20000 timesteps (interactions with world) to yield score according to global reward function of §4.3.1. The best scored average 8.612 points per 100 timesteps.

Hierarchical Q-learning (§4.4) - 5 small agents, with rewards 1 or 0, as in §4.4. Agents learn Q-values for their local reward functions by random exploration of 300000 timesteps (interactions with world), all learning together (§4.4.2) with random winner each step. The switch then learns $Q(x,i)$ using the global reward function of §4.3.1. Switch's Q-values are stored in a neural network (in fact, for convenience, this is broken into one network per action i). 100 trials, each trial interacting with the world 1400 times and then replaying experiences 30 times. Policy improves over time using Boltzmann distribution (§2.2.3). Test over 20000 timesteps (interactions with world) to yield score according to global reward function of §4.3.1. The architecture of the network and coding of inputs was adjusted to get the best score of average 13.641 points per 100 timesteps.

W-learning with subspaces (§8) - 8 small agents, with rewards r_i or 0, as in §8. Agents learn Q-values for their local reward functions by random exploration of 300000 timesteps (interactions with world), all learning together (§4.4.2) with random winner each step. Agents learn Q-values *once* with all $r_i = 1$. Genetic algorithm genotype is a set of r_i's. Population size 60. For each individual genotype, multiply base Q-values by r_i (§8.1.2), then re-learn W-values by W-learning (without reference to global reward) over 50000 timesteps (interactions with world), then test. Test resultant creature over 20000 timesteps (interactions with world) to yield score according to global reward function of §4.3.1. This score is the fitness function to decide who is allowed reproduce. Evolution for 30 generations found best combination of r_i's scoring average 13.446 points per 100 timesteps.

W=Q (§9) - 8 small agents, with rewards r_i or 0, as in §8. Agents learn Q-values for their local reward functions by random exploration of 300000 timesteps (interactions with world), all learning together (§4.4.2) with random winner each step. Agents learn Q-values *once* with all $r_i = 1$. Genetic algorithm genotype is a set of r_i's. Population size 60. For each individual genotype, multiply base Q-values by r_i (§8.1.2), then test. No W-values to learn since W is simply Q. Test creature over 20000 timesteps (interactions with world) to yield score according to global reward function of §4.3.1. This score is the fitness function to decide who is allowed reproduce. Evolution for 30 generations found best combination of r_i's scoring average 15.313 points per 100 timesteps.

W-learning with full space (§10) - 8 small agents, with rewards r_i or 0, as in §8. Agents learn Q-values for their local reward functions by random exploration of 300000 timesteps (interactions with world), all learning together (§4.4.2) with random winner each step. Agents learn Q-values *once* with all $r_i = 1$. Genetic algorithm genotype is a set of r_i's. Population size 60. For each individual genotype, multiply base Q-values by r_i (§8.1.2), then re-learn W-values (without reference to global reward) then test. Each agent's W-values are stored in a neural network (one network for each agent). To learn W-values, do one run of 30000 timesteps (interactions with world) with random winners. Each agent then replays its experiences 10 times to learn its W-values. Test resultant creature over 20000 timesteps (interactions with world) to yield score according to global reward function of §4.3.1. This score is the fitness function to decide who is allowed reproduce. Evolution for 30 generations found best combination of r_i's scoring average 14.871 points per 100 timesteps.

Negotiated W-learning (§11) - 8 small agents, with rewards r_i or 0, as in §8. Agents learn Q-values for their local reward functions by random exploration of 300000 timesteps (interactions with world), all learning together (§4.4.2) with random winner each step. Agents learn Q-values *once* with all $r_i = 1$. Genetic algorithm genotype is a set of r_i's. Population size 60. For each individual genotype, multiply base Q-values by r_i (§8.1.2), then test. No W-values to learn since competition is resolved on the fly by Negotiated W-learning. Test creature over 20000 timesteps (interactions with world) to yield score according to global reward function of §4.3.1. This score is the fitness function to decide who is allowed reproduce. Evolution for 30 generations found best combination of r_i's scoring average 18.212 points per 100 timesteps.

Bibliography

[Aylett, 1995] Aylett, Ruth (1995), Multi-Agent Planning: Modelling Execution Agents, *Papers of the 14th Workshop of the UK Planning and Scheduling Special Interest Group.*

[Baum, 1996] Baum, Eric B. (1996), Toward a Model of Mind as a Laissez-Faire Economy of Idiots, *Proceedings of the Thirteenth International Conference on Machine Learning.*

[Blumberg, 1994] Blumberg, Bruce (1994), Action-Selection in Hamsterdam: Lessons from Ethology, *Proceedings of the Third International Conference on Simulation of Adaptive Behavior (SAB-94).*

[Brooks, 1986] Brooks, Rodney A. (1986), A robust layered control system for a mobile robot, *IEEE Journal of Robotics and Automation* 2:14-23.

[Brooks, 1991] Brooks, Rodney A. (1991), Intelligence without Representation, *Artificial Intelligence* 47:139-160.

[Brooks, 1991a] Brooks, Rodney A. (1991), Intelligence without Reason, *Proceedings of the 12th International Joint Conference on Artificial Intelligence (IJCAI-91).*

[Brooks, 1994] Brooks, Rodney A. (1994), Coherent Behavior from Many Adaptive Processes, *Proceedings of the Third International Conference on Simulation of Adaptive Behavior (SAB-94).*

[Charpillet et al., 1996] Charpillet, Francois; Chevrier, Vincent; Foisel, Remy and Haton, Jean-Paul (1996), *Organizing a Society of Softbots for World Wide Web Applications*, workshop on Artificial Intelligence-based tools to help W3 users, Fifth International World Wide Web Conference.

[Clocksin and Moore, 1989] Clocksin, William F. and Moore, Andrew W. (1989), Experiments in Adaptive State-Space Robotics, *Proceedings of the 7th Conference of the Society for Artificial Intelligence and Simulation of Behaviour (AISB-89).*

[Dennett, 1978] Dennett, Daniel C. (1978), Why not the whole iguana?, *Behavioral and Brain Sciences* 1:103-104.

[Dennett, 1991] Dennett, Daniel C. (1991), *Consciousness Explained*, Allen Lane, The Penguin Press.

[Digney, 1996] Digney, Bruce L. (1996), Emergent Hierarchical Control Structures: Learning Reactive/Hierarchical Relationships in Reinforcement Environments, *Proceedings of the Fourth International Conference on Simulation of Adaptive Behavior (SAB-96).*

[Edelman, 1989] Edelman, Gerald M. (1989), *The Remembered Present: A Biological Theory of Consciousness*, Basic Books.

[Edelman, 1992] Edelman, Gerald M. (1992), *Bright Air, Brilliant Fire: On the Matter of the Mind*, Basic Books.

[Grefenstette, 1992] Grefenstette, John J. (1992), The Evolution of Strategies for Multi-agent Environments, *Adaptive Behavior* 1:65-89.

[Holland, 1975] Holland, John H. (1975), *Adaptation in Natural and Artificial Systems*, Ann Arbor, Univ. Michigan Press.

[Humphrys, 1995] Humphrys, Mark[1] (1995), *W-learning: Competition among selfish Q-learners*, technical report no.362, University of Cambridge, Computer Laboratory.

[Humphrys, 1995a] Humphrys, Mark (1995), Towards self-organising Action Selection, *Papers of the 14th Workshop of the UK Planning and Scheduling Special Interest Group.*

[Humphrys, 1996] Humphrys, Mark (1996), Action Selection in a hypothetical house robot: Using those RL numbers, *Proceedings of the First International ICSC Symposia on Intelligent Industrial Automation (IIA-96) and Soft Computing (SOCO-96).*

[Humphrys, 1996a] Humphrys, Mark (1996), *Action Selection methods using Reinforcement Learning*, PhD thesis (first version), University of Cambridge, Computer Laboratory.

[Jackson, 1987] Jackson, John V. (1987), Idea for a Mind, *SIGART Newsletter*, Number 101, July 1987.

[Kaelbling, 1993] Kaelbling, Leslie Pack (1993), *Learning in Embedded Systems*, The MIT Press/Bradford Books.

[Kaelbling, 1993a] Kaelbling, Leslie Pack (1993), Hierarchical Learning in Stochastic Domains, *Proceedings of the Tenth International Conference on Machine Learning.*

[Kaelbling et al., 1996] Kaelbling, Leslie Pack; Littman, Michael L. and Moore, Andrew W. (1996), Reinforcement Learning: A Survey, *Journal of Artificial Intelligence Research* 4:237-285.

[Karlsson, 1997] Karlsson, Jonas (1997), *Learning to Solve Multiple Goals*, PhD thesis, University of Rochester, Department of Computer Science.

[Lin, 1992] Lin, Long-Ji (1992), Self-Improving Reactive Agents Based On Reinforcement Learning, Planning and Teaching, *Machine Learning* 8:293-321.

[Lin, 1993] Lin, Long-Ji (1993), Scaling up Reinforcement Learning for robot control, *Proceedings of the Tenth International Conference on Machine Learning.*

[Maes, 1989] Maes, Pattie (1989), How To Do the Right Thing, *Connection Science* 1:291-323.

[Maes, 1989a] Maes, Pattie (1989), The dynamics of action selection, *Proceedings of the 11th International Joint Conference on Artificial Intelligence (IJCAI-89).*

[Mataric, 1994] Mataric, Maja J. (1994), Learning to behave socially, *Proceedings of the Third International Conference on Simulation of Adaptive Behavior (SAB-94).*

[McFarland, 1989] McFarland, David (1989), *Problems of Animal Behaviour*, Longman.

[Metcalfe and Boggs, 1976] Metcalfe, Robert M. and Boggs, David R. (1976), Ethernet: Distributed Packet Switching for Local Computer Networks, *Communications of the ACM* 19:395-404.

[Minsky, 1986] Minsky, Marvin (1986), *The Society of Mind*, Simon and Schuster, New York.

[Moore, 1990] Moore, Andrew W. (1990), *Efficient Memory-based Learning for Robot Control*, PhD thesis, University of Cambridge, Computer Laboratory.

[1] All my publications are at:
`http://www.cl.cam.ac.uk/~mh10006/publications.html`

[Ono et al., 1996] Ono, Norihiko; Fukumoto, Kenji and Ikeda, Osamu (1996), Collective Behavior by Modular Reinforcement-Learning Animats, *Proceedings of the Fourth International Conference on Simulation of Adaptive Behavior (SAB-96).*

[Ray, 1991] Ray, Thomas S. (1991), An Approach to the Synthesis of Life, *Artificial Life II.*

[Ring, 1992] Ring, Mark (1992), Two Methods for Hierarchy Learning in Reinforcement Environments, *Proceedings of the Second International Conference on Simulation of Adaptive Behavior (SAB-92).*

[Rosenblatt, 1995] Rosenblatt, Julio K. (1995), DAMN: A Distributed Architecture for Mobile Navigation, *Proceedings of the 1995 AAAI Spring Symposium on Lessons Learned from Implemented Software Architectures for Physical Agents.*

[Rosenblatt and Thorpe, 1995] Rosenblatt, Julio K. and Thorpe, Charles E. (1995), Combining Multiple Goals in a Behavior-Based Architecture, *Proceedings of the 1995 International Conference on Intelligent Robots and Systems (IROS-95).*

[Ross, 1983] Ross, Sheldon M. (1983), *Introduction to Stochastic Dynamic Programming*, Academic Press, New York.

[Rummery and Niranjan, 1994] Rummery, Gavin and Niranjan, Mahesan (1994), *On-line Q-learning using Connectionist systems*, technical report no.166, University of Cambridge, Engineering Department.

[Sahota, 1994] Sahota, Michael K. (1994), Action Selection for Robots in Dynamic Environments through Inter-behaviour Bidding, *Proceedings of the Third International Conference on Simulation of Adaptive Behavior (SAB-94).*

[Scheier and Pfeifer, 1995] Scheier, Christian and Pfeifer, Rolf (1995), Classification as Sensory-Motor Coordination, *Proceedings of the 3rd European Conference on Artificial Life (ECAL-95).*

[Selfridge and Neisser, 1960] Selfridge, Oliver G. and Neisser, Ulric (1960), Pattern recognition by machine, *Scientific American* 203:60-68.

[Singh, 1992] Singh, Satinder P. (1992), Transfer of Learning by Composing Solutions of Elemental Sequential Tasks, *Machine Learning* 8:323-339.

[Singh et al., 1994] Singh, Satinder P.; Jaakkola, Tommi and Jordan, Michael I. (1994), Learning without state-estimation in Partially Observable Markovian Decision Processes, *Proceedings of the Eleventh International Conference on Machine Learning.*

[Sporns, 1995] Sporns, Olaf (1995), personal communication.

[Steels, 1994] Steels, Luc (1994), A case study in the Behavior-Oriented design of Autonomous Agents, *Proceedings of the Third International Conference on Simulation of Adaptive Behavior (SAB-94).*

[Sutton, 1988] Sutton, Richard S. (1988), Learning to Predict by the Methods of Temporal Differences, *Machine Learning* 3:9-44.

[Sutton, 1990] Sutton, Richard S. (1990), Integrated Architectures for Learning, Planning and Reacting Based on Approximating Dynamic Programming, *Proceedings of the Seventh International Conference on Machine Learning.*

[Sutton, 1990a] Sutton, Richard S. (1990), Reinforcement Learning Architectures for Animats, *Proceedings of the First International Conference on Simulation of Adaptive Behavior (SAB-90).*

[Tan, 1993] Tan, Ming (1993), Multi-Agent Reinforcement Learning: Independent vs. Cooperative Agents, *Proceedings of the Tenth International Conference on Machine Learn-*

ing.

[Tesauro, 1992] Tesauro, Gerald (1992), Practical Issues in Temporal Difference Learning, *Machine Learning* 8:257-277.

[Tham and Prager, 1994] Tham, Chen K. and Prager, Richard W. (1994), A modular Q-learning architecture for manipulator task decomposition, *Proceedings of the Eleventh International Conference on Machine Learning.*

[Todd et al., 1994] Todd, Peter M.; Wilson, Stewart W.; Somayaji, Anil B. and Yanco, Holly A. (1994), The blind breeding the blind: Adaptive behavior without looking, *Proceedings of the Third International Conference on Simulation of Adaptive Behavior (SAB-94).*

[Tyrrell, 1993] Tyrrell, Toby (1993), *Computational Mechanisms for Action Selection*, PhD thesis, University of Edinburgh, Centre for Cognitive Science.

[Varian, 1993] Varian, Hal R. (1993), *Intermediate Microeconomics*, W.W.Norton and Co.

[Watkins, 1989] Watkins, Christopher J.C.H. (1989), *Learning from delayed rewards*, PhD thesis, University of Cambridge, Psychology Department.

[Watkins and Dayan, 1992] Watkins, Christopher J.C.H. and Dayan, Peter (1992), Technical Note: Q-Learning, *Machine Learning* 8:279-292.

[Weir, 1984] Weir, Michael (1984), *Goal-Directed Behaviour*, Gordon and Breach.

[Whitehead et al., 1993] Whitehead, Steven; Karlsson, Jonas and Tenenberg, Josh (1993), Learning Multiple Goal Behavior via Task Decomposition and Dynamic Policy Merging, in Connell and Mahadevan, eds., *Robot Learning*, Kluwer Academic Publishers.

[Wilson, 1990] Wilson, Stewart W. (1990), The animat path to AI, *Proceedings of the First International Conference on Simulation of Adaptive Behavior (SAB-90).*

[Wixson, 1991] Wixson, Lambert E. (1991), Scaling reinforcement learning techniques via modularity, *Proceedings of the Eighth International Conference on Machine Learning.*