To appear in *From Animals to Animats 4: Fourth Int. Conf. on Simulation of Adaptive Behavior* (SAB-96), Cape Cod, Massachusetts, USA, September 9-13, 1996.

Action Selection methods using Reinforcement Learning

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/users/mh10006/

Abstract

Action Selection schemes, when translated into precise algorithms, typically involve considerable design effort and tuning of parameters. Little work has been done on solving the problem using learning. This paper compares eight different methods of solving the action selection problem using Reinforcement Learning (learning from rewards). The methods range from centralised and cooperative to decentralised and selfish. They are tested in an artificial world and their performance, memory requirements and reactiveness are compared. Finally, the possibility of more exotic, ecosystem-like decentralised models are considered.

1 Action Selection

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.

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 some difficulties in translating the conceptual models into computational ones). The models that do lend themselves to algorithmic implementation (e.g. see [Brooks, 1991, 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, Ch.9], [Aylett, 1995]) 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) [Kaelbling, 1993]. Reinforcement Learning propagates numeric rewards into behavior patterns. The rewards may be external value judgements, or just internally generated numbers. This paper compares eight 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.

1.1 Multi-module Reinforcement Learning

In general, Reinforcement Learning work has concentrated on problems with a single goal. For complex problems, that need to be broken into subproblems, most of the work 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]. Lin has devised a form of multi-module RL suitable for such problems [Lin, 1992], and this will be the second method tested below.

2 The House Robot problem

We will demonstrate six of the methods in use in the hypothetical world of a 'house robot' (the final two are merely described). The house robot is given a range of multiple parallel and conflicting goals and must partially satisfy them all as best as it can. It doubles as a mobile security camera, mobile smoke alarm and occasional vacuum cleaner.

The artificial gridworld of Figure 1 was not constructed to be a *simulation* of a robot environment but

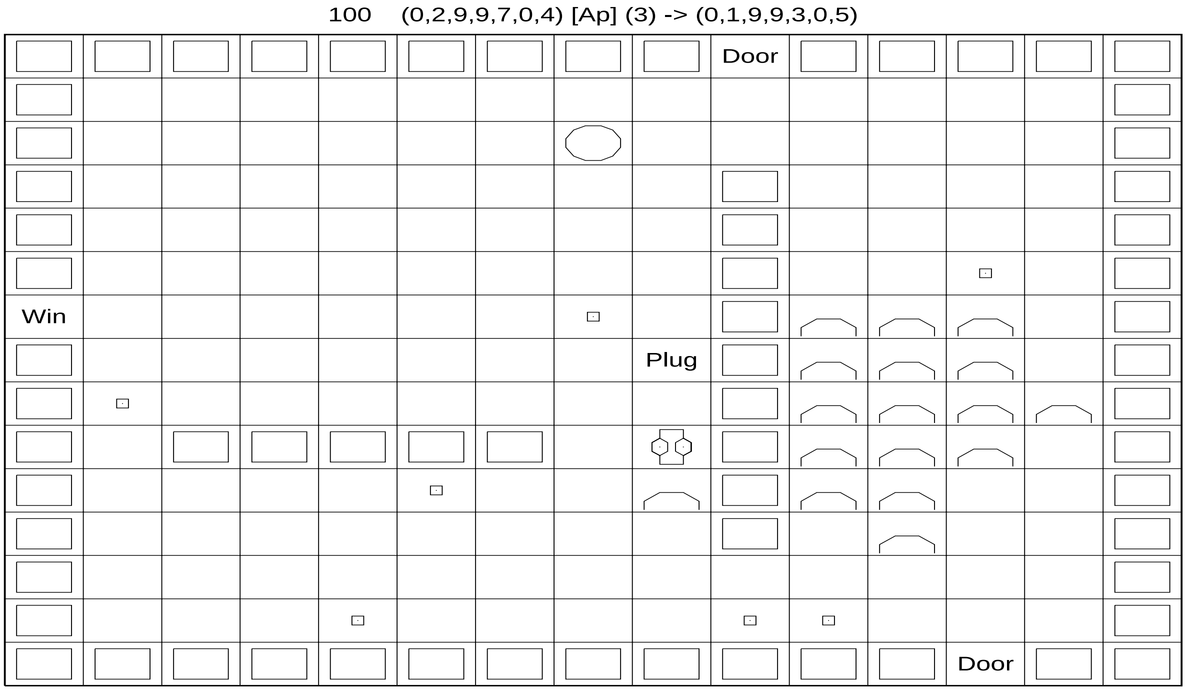

Figure 1: The House Robot's world. Here, the building is on fire, dirt is scattered everywhere, and an unidentified human has just come in the top door.

just to be a dynamic world in which to explore multiple conflicting goals. The positions of entrances and internal walls are randomised on each run. Humans are constantly making random crossings from one entrance to another. The robot or animat should follow strangers, and stay out of the way of family. It must go up close first to identify the human as family or stranger. Dirt trails after all humans. The animat 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 animat puts out the fire on a square by moving onto it. Each time step, the animat 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), 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).
- f (0-9) is the direction of the nearest visible smoke. Smoke is the only thing that it can detect through walls - otherwise if something is blocked by a wall it is not visible.
- w_f is whether the smoke is being detected through a wall, and takes values 0 and 1.

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

Given this sensory information, the animat needs to develop a purely reactive strategy to put out fire, clean up dirt, watch strangers, and regularly return to base to re-charge. When we specify precisely (see next section) 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.

3 Q-learning

The first method we apply is a single monolithic agent learning from rewards. Watkins [Watkins, 1989] introduced the method of reinforcement learning called *Q-learning*. Each discrete time step, the agent observes state x, takes action a, observes new state y, and receives immediate reward 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 strategy that the Q-learning agent adopts is to build up *Quality-values* (Q-values) for each pair (x, a). In 1-step Q-learning, after each experience, we update:

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

where $Q(x,a) \longrightarrow d$ means that we adjust the estimate $Q(x,a)$ in the direction of d. For example, if we store each $Q(x,a)$ explicitly, we may update: $Q(x,a) := (1-\alpha)Q(x,a) + \alpha d$, where α is the learning rate. Or if the function is being approximated by a neural network, we backpropagate the error $Q(x,a) - d$.

All Q-values start randomised. Given some restrictions, [Watkins and Dayan, 1992] proved that for lookup tables Q-learning converges to a unique set of values $Q^*(x,a)$ which define a stationary deterministic optimal policy, namely to always take the action with the highest Q^*-value.

3.1 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] for an example of accidentally designing one in which the optimum solution was to jump in and out of the plug nonstop). Later 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
```

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 impractical. So instead of using lookup tables, we need to use some sort of generalization - here, multi-layer neural networks.

Following [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. We have 9 separate nets acting as function approximators. Each takes a vector input x and producing a floating point output $Q_a(x)$

Furthermore, as in [Rummery and Niranjan, 1994], we note here that 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 much 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.

As [Tesauro, 1992] notes, learning here is not like ordinary supervised learning where we learn from (Input,Output) exemplars. Here we're not presenting $(x, Q^*(x))$ exemplars to the network but instead we are learning from *estimates* of Q^*. We need to repeat the updates as the estimate improves. 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 estimates. Like Lin, we use *backward* replay as more effective (update $Q(y)$ before updating the $Q(x)$ that led to it). Throughout the experiments in this paper, we use $\gamma = 0.6$, and all lookup tables and neural networks (and hence Q-value estimates) start randomised.

Adjusting the amount of replay, and the architecture of the networks, 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. This 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. Q-learning did not find an optimal policy because we are not using lookup tables, and do not have time anyway to experience each full state.

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.

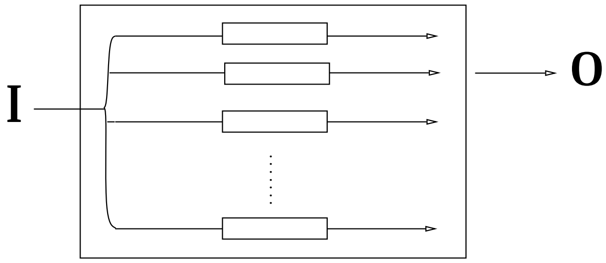

Figure 2: Competition among peer agents in a horizontal architecture. Each agent suggests an action, but only one action is executed. Which agent is obeyed changes dynamically.

4 Hierarchical Q-learning

[Lin, 1993] suggests breaking up a complex problem into sub-problems, having a collection of Q-learning agents [1] $A_1, \ldots, A_n$ learn the sub-problems, and then a single controller 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 animat 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 sub-agents simply compete and interfere with each other, that is, to classic action selection problems. In this decentralised approach (Figure 2), each behavior will drive the body on its own if allowed, but 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 (a constant action a_i). Later, we will make the switch simple, and what is sent more complex.

We set the following 5 agents to build up personal $Q_i(x,a)$ values, where the x they sense ranges over a subspace small enough to use lookup tables, and they learn by reference to personal reward functions r_i. The switch then learns $Q(x,i)$ from the global reward function. The switch still sees the full state x, and needs to be implemented as 5 neural networks.

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

[1] I still use the word *agent*, even though we now have multiple agents inside the same body. This is similar to the use of the word in [Minsky, 1986].

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. We try to keep the number of agents low, since a large number of agents will require a large (x,i) statespace. Note that having all rewards set to 1 doesn't reduce our options - if we replace 1 above by 0.5, or 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 true with a 3-reward (or more) reward function, where relative differences between rewards matters.

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.

5 W-learning (Minimize the Worst Unhappiness)

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.

W-learning [Humphrys, 1995] is an attempt to define a sensible way for the agents to organise themselves in the absence of a global reward. The agents learn by compromising, and the agent that wins is the agent that would suffer the most if it did not win. Given a state x, the agents suggest their actions with strengths or *Weights* $W_i(x)$. The switch in Figure 2 becomes a simple gate letting through the highest one. When the animat observes state x, the switch finds the winner k such that:

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

and 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 agents then *modify* their $W_i(x)$ values based on whether they were obeyed (and what happened if they weren't), so that next time round there may be a different winner. Schemes using such 'importance' values are common in multi-behavior models (e.g. see the 'utility' functions in [Aylett, 1995]), but are normally hand-designed.

W-learning builds up the difference between predicted reward $\mathcal{P}$ (what is predicted if the agent is obeyed) and actual reward $\mathcal{A}$ (what actually happened). Consider Q-learning as the process:

$$\mathcal{P} \longrightarrow \mathcal{A}$$

where we are *learning* $\mathcal{P}$, and $\mathcal{A}$ is caused by the execution of our action. Then W-learning is:

$$W \longrightarrow (\mathcal{P} - \mathcal{A})$$

where $\mathcal{P}$ is *already* learnt, and $\mathcal{A}$ is caused by the execution of *another* agent's action. $(\mathcal{P} - \mathcal{A})$ is the 'error' or loss that the other agent is causing for this one by being obeyed in its place. When we see a difference term between predicted and actual, we expect that this 'error term' will go to zero, but here it goes to a positive number. To be precise, when agent A_k is the winner and has its action executed, all agents *except* A_k update:

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

where the reward r_i and next state y were caused by A_k rather than by the agent itself. W-learning does not assume that A_k's actions are meaningful to this agent - it does not assume we can look at $Q_i(x, a_k)$. But we can always observe what happened in terms of r_i and y. The error distribution we sample from will abruptly change if there is a change of leader in state x, but competition will eventually be resolved for this state when some agent, as a result of losses it has suffered in the past, builds up an unassailable W-value for x (see [Humphrys, 1995] for details).

Note that if we update the winner's W-value as well, we would be updating $W_k(x) \longrightarrow 0$, since if you are obeyed, your expected error is zero. So as soon as an agent gets into the lead, its W-value starts dropping until it loses the lead again. Competition is never resolved. This is why the leader does nothing - it's up to the others to catch up with it. If they can't, we have a resolved competition. Not updating $W_k(x)$ though, means that if it gets arbitrarily set to an unfairly high value, it will never be challenged. So we initialise all $W_i(x)$ to zero or random negative values.

5.1 Making agents weaker or stronger

There being no global (x, i) statespace to worry about, we can expand the number of agents. To the previous five, we add three more agents. A_c should head for the centre of an open area while A_w should engage in wall-following. 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. 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 little subspaces, for which lookup tables can be used.

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

Unlike in the previous method, here the values of the rewards r_i *do* matter. An agent with rewards 1 and 0 will end up with higher W-values than an equivalent agent with rewards 0.1 and 0 and the same logic, since its absolute $(\mathcal{P} - \mathcal{A})$ differences will be greater. It will win a greater area of state-space in competitions. We make agents *stronger* (more influential in the collection) by increasing the differences between their rewards. Adaptive collections are likely to involve well-chosen combinations of weak and strong agents. The best combination found above was:

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

This was discovered by running a genetic algorithm search on combinations of r_i's. Given a particular combination, the agents learn their behaviors by Q-learning, and then organise their action selection by W-learning, without reference to a global reward. Obviously, if the global reward function still *defines* what we are looking for, we still need to use it - as the fitness function to guide our search. But it no longer need be available explicitly to the agents. It need only be used to test them. Hence the fitness function could be just *implicit* in the environment, as in the best Artificial Life research [Ray, 1991].

This combination scored 13.446, slightly less than we got with Hierarchical Q-learning, but achieved with a reduction in memory requirements from 9.6 million to 1600. For analysis of how the agents interact to achieve the task see [Humphrys, 1996]. Note that all GA searches in these experiments were population size ≤ 60, initially randomised, evolving for ≤ 30 generations.

6 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. 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 just search for adaptive combinations as before. This works very well, and the following collection achieved a score of 15.313. Further, the memory requirements are even less, since no W-values at all are kept.

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

But we are not finished with W-learning yet. It seems on paper that W=Q should not perform so well, since it maximises the rewards of only one agent, while W-learning makes some attempt to maximise their collective rewards (which is roughly what the global reward is). Consider the following scenario, where there are two 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 $1.1 > 0.9$), action (1) is executed, A_1 gets reward 1.1, and A_2 gets 0. If we use $W = (\mathcal{P} - \mathcal{A})$, then A_2 wins (since it would suffer 0.9 if it didn't), (2) is executed, A_1 gets 1, and A_2 gets 0.9. If the global reward / evolutionary fitness is roughly a combination of the agents' rewards, then $W = (\mathcal{P} - \mathcal{A})$ is a better strategy. Note that 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?

So 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 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.

7 W-learning with full space

W-learning performed similarly to Hierarchical Q-learning, but with far less memory requirements. But note that using subspaces for $W_i(x)$ results in a loss of accuracy.

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.

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. Recall that the W-learning strategy for lookup tables is to start with W random negative, and have the leading $W_k(x)$ unchanged, waiting to be overtaken. This 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 $W_k(x) \longrightarrow d$ 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. The simplest approach is probably: 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 all replay their experiences 10 times to learn from them properly. We use a similar network architecture as before. The best combination of agents found, scoring 14.871, was:

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

8 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. However, if other agents' actions mean something to 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 Negotiated W-learning, the animat 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 competition will be very short. The 'instant' competition operates as follows. 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 has terminated 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 any memory requirements at all for W. 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:

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

It is easily seen [Humphrys, 1995] that the competition length will be bounded by 1 and $n+1$ (remember here $n = 8$). With the combination above, the competition lengths seen over a run of 40000 steps (actually, for technical reasons, 39944 steps) 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 (Figure 3).

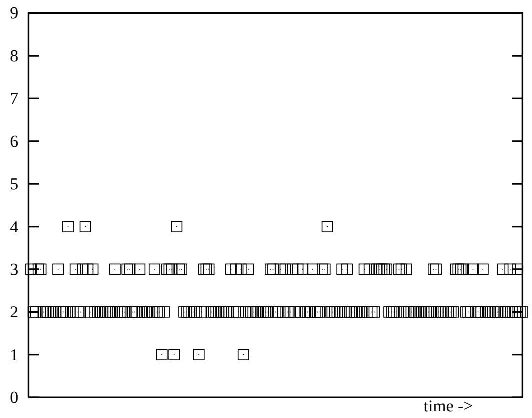

Figure 3: The 'reactiveness' of Negotiated W-learning. This is a typical snapshot of 200 steps of the best solution found, showing how long it took to resolve competition at each timestep. The theoretical maximum competition length is 9 (the number of agents plus 1).

9 Maximize Collective Happiness

For completeness we describe two final methods, though they have not yet been tested empirically. 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 executed action may be an action that none of the agents would have suggested.

10 Minimize Collective Unhappiness

Obviously, the final missing method is to *minimize collective unhappiness*. In this method, 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 do not need to share common actions. Rather, they observe r_i and y, and build up their deficits over time. We start with all $W_i(x)$ zero or random negative. Each time step:

```
observe state x
find Wk(x) = lowest Wi(x)
execute ak
all agents i other than k add to Wk(x)
 (so that it might not win next time round)
```

Again like W-learning, if agents *do* share common actions, we can resolve this immediately rather than waiting for W-values to build up over time.

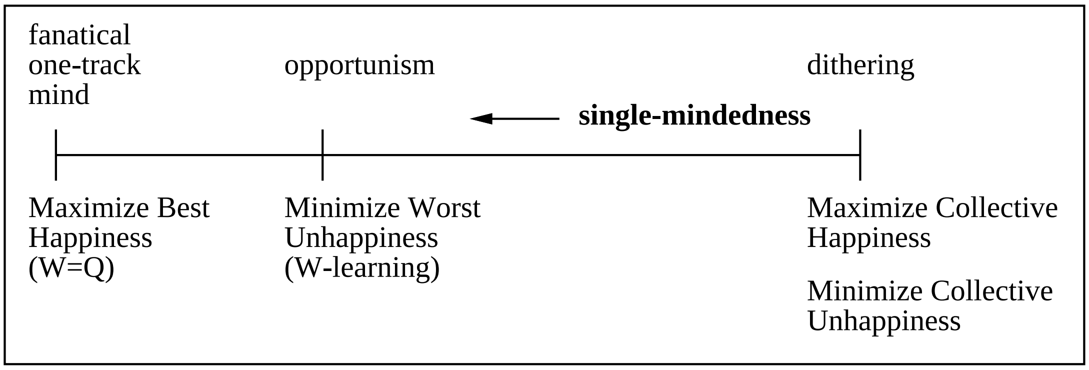

Figure 4: The 'single-mindedness' of the methods that organise action selection without reference to a global reward.

	Memory requirements n no.agents x subspace X full space	No. updates per timestep (per agent) when learning	No. updates per timestep when exploiting
Q-learning	1.Xa	1	0
Hierarchical Q-learning	n.xa + 1.Xn	2	0
W=Q	n.xa	1	1
Max Collective Happ.	n.xa	1	n/applicable
W-learning (subspaces)	n.xa + n.x	2	0
W-learning (full space)	n.xa + n.X	2	0
Negotiated W-learning	n.xa	1	1 to n+1
Min Collective Unhapp.	n.xa + n.x	n	0

General comparisons between the methods.

	Memory requirements	No. updates per timestep when learning	No. updates per timestep when exploiting	Best solution found
Hand-coded program (strict hierarchical)	n/applicable	n/applicable	n/applicable	8.612
Q-learning	10800000	1	0	6.285
Hierarchical Q-learning	9601440	2	0	13.641
W=Q	1440	1	1	15.313
Max Collective Happ.	1440	1	n/applicable	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 Collective Unhapp.	1600	8	0	n/tested

Comparisons between the methods as applied in the house robot problem.

10.1 Expected performance of the collective methods

The two collective methods will generate the same 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 animat 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.

One can easily set up situations in which trying to keep the majority of agents happy will lead to problems with *dithering*, in which no goal is pursued to its logical end. W=Q goes to the other extreme in having only one agent in charge, and perhaps suffers because it does not allow *opportunism*. 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.

[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 4).

11 Conclusion

In conclusion, it seems that selfishness on the part of agents is justified - compare (see tables) the stronger performance of the various W methods. Agents can spot better than global functions opportunities to pick up rewards, so letting agents be selfish is more likely to maximise their combined rewards.

Whether it is better to build up differences ($\mathcal{P} - \mathcal{A}$) (the three W-learning methods) or just use W=Q is perhaps still an open question. Although the best one was one of the W-learning methods, W=Q was second best. Perhaps, as discussed in that section, opportunism isn't important enough in this particular problem world to separate the two approaches.

Finally, 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.

12 Further work

As discussed, to further compare these methods, they could be tested in an environment in which the rewards of opportunism (and the costs of not being opportunistic) are greater. The collective methods should be tested in situations where some agents have non-recoverable states.

12.1 Competition in Single Goal problems

Note that with the W methods, agents don't actually have to be solving different goals. This may be a novel approach to classic single-goal problems. To solve a problem, we put together a large number of agents, all with different reward functions (and perhaps even different senses), all roughly trying to solve the same thing, and let them struggle 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. We could be similarly profligate with the number of agents in Hierarchical Q-learning too, but would pay the price of a large (x, i) statespace.

12.2 Ecosystem Minds

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]. The idea is appealing, but Edelman presents no explicit algorithm to show how it could be implemented. [2] Here, with W-learning, we have the basis for a full living and dying ecosystem inside the animat, where what comes into existence, struggles, and perhaps dies is not mere connections (as Edelman may in fact mean) but entire autonomous goals.

In both W=Q and Negotiated W-learning, nothing needs to be re-learnt when new agents arrive or old ones leave, but the dynamics of the system will immediately change in both. It seems Negotiated W-learning would be more robust since agents may generate higher W-values in the face of new competition, while in W=Q they are stuck with fixed W-values no matter what the competition. We could even imagine collections which are difficult for new agents to invade - where W-values are currently low but would all rise, as if in defence, on arrival of the invader. Such would be not only an ecosystem mind, but a stable, elderly ecosystem mind, set in its ways.

References

[Aylett, 1995] Aylett, Ruth (1995), Multi-Agent Planning: Modelling Execution Agents, in Sam Steel, ed., *Papers of the 14th Workshop of the UK Planning and Scheduling Special Interest Group.*

[2] In fact, presenting arguments only about classical AI, he draws the familiar conclusion that no computer algorithm could implement his ideas. No mention is made of self-modifying, reinforcement-learning agents embedded in a world, to which his criticisms do not apply.

[Blumberg, 1994] Blumberg, Bruce (1994), Action-Selection in Hamsterdam: Lessons from Ethology, in Dave Cliff et al., eds., *Proceedings of the Third International Conference on Simulation of Adaptive Behavior (SAB-94)*.

[Brooks, 1991] Brooks, Rodney A. (1991), Intelligence without Representation, *Artificial Intelligence* 47:139-160.

[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.

[Humphrys, 1995] Humphrys, Mark (1995), *W-learning: Competition among selfish Q-learners*, technical report no.362, University of Cambridge, Computer Laboratory. `http://www.cl.cam.ac.uk/users/mh10006/publications.html`

[Humphrys, 1996] Humphrys, Mark (1996), Action selection in a hypothetical house robot: Using those RL numbers, in Peter G.Anderson and Kevin Warwick, eds., *Proceedings of the First International ICSC Symposia on Intelligent Industrial Automation (IIA-96) and Soft Computing (SOCO-96)*.

[Kaelbling, 1993] Kaelbling, Leslie Pack (1993), *Learning in Embedded Systems*, The MIT Press/Bradford Books.

[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* vol.1, no.3.

[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.

[Ray, 1991] Ray, Thomas S. (1991), An Approach to the Synthesis of Life, in Christopher G.Langton et al., eds., *Artificial Life II*.

[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, in Dave Cliff et al., eds., *Proceedings of the Third International Conference on Simulation of Adaptive Behavior (SAB-94)*.

[Singh, 1992] Singh, Satinder P. (1992), Transfer of Learning by Composing Solutions of Elemental Sequential Tasks, *Machine Learning* 8:323-339.

[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*.

[Tyrrell, 1993] Tyrrell, Toby (1993), *Computational Mechanisms for Action Selection*, PhD thesis, University of Edinburgh, Centre for Cognitive Science.

[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.

MPEG Movie demo

A series of MPEG Movies of W-learning operating in a simple 'antworld', where an animat must collect food while avoiding moving predators, can be viewed at: `http://www.cl.cam.ac.uk/users/mh10006/w.html`