Action selection in a hypothetical house robot: Using those RL numbers

Mark Humphrys
University of Cambridge, Computer Laboratory *
http://www.cl.cam.ac.uk/users/mh10006

Abstract

Reinforcement Learning (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. It is clear that many interesting systems possess multiple parallel and conflicting goals, all demanding attention, and none of which can be fully satisfied except at the expense of others. Animals are the prime example of such systems. In [Humphrys, 1995], I introduced the W-learning algorithm, showing one method of resolving competition among behaviors automatically by reference to their RL values. The scheme has the unusual feature that behaviors are at all times in selfish pursuit of their own goals and have no explicit concept of cooperation, despite residing in the same body. In this paper, I apply W-learning to the world of a hypothetical house robot, which doubles as family toy, mobile security camera, mobile smoke alarm and occasional vacuum cleaner. I show how a W-learning community of behaviors inside the robot will support a robust behavior pattern, capable of opportunistic behavior, avoiding dithering, and allowing for the concept of default behavior and expression of low-priority goals.

Keywords: reactive systems, action selection, reinforcement learning, multi-behavior learning

*postal address: University of Cambridge, Computer Laboratory, New Museums Site, Pembroke St., Cambridge CB2 3QG, England. tel: +44 1223 335443. fax: +44 1223 334678 or 334679. email: Mark.Humphrys@cl.cam.ac.uk

1 Action selection

It is clear that many interesting systems possess multiple parallel and conflicting goals, among which attention must endlessly switch. In the study of animal behavior, ethologists call this the problem of *action selection* - the choosing by the animal of the appropriate goal to pursue or desire to satisfy at a given moment in time, given the state of the external world.

Ethologists have proposed various models of action selection in the animals they observe (see [Tyrrell, 1993] for a survey). When engineers try to build systems according to similar models, they normally find themselves in the position of having to adjudicate the trade-offs between the goals (see [Tyrrell, 1993, Ch.9], [Aylett, 1995]). To try to escape from this difficult design problem, I introduced in [Humphrys, 1995] the following model in which the behaviors would decide among themselves in a logical manner.

1.1 Competition among selfish agents

The basic model of competing behaviors is shown in Figure 1. There are multiple behaviors sharing the same body. I call these *agents* [1] since each is

[1] I use the word *agent* to emphasise that these are autonomous actors, not mere procedure calls, not structured in any hierarchy and not even necessarily cooperative. This is similar to the use of the word in [Minsky, 1986], and this model can be seen as a society of mind in which every agent can deal with the external world. For somewhat-autonomous, somewhat-competing modules

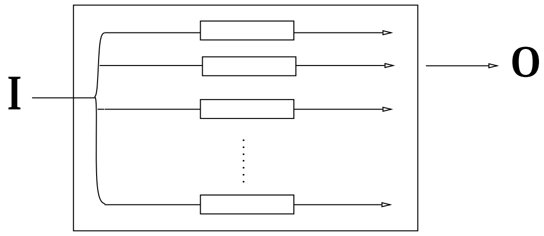

Figure 1: Competition among selfish peer agents in a horizontal architecture. Each agent suggests an action, but only one action is executed. Which agent is obeyed changes dynamically.

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 want to use the body to implement *their* plans.

Think of it as a *creature* with a fully decentralised mind, a sort of logical development of decentralised models of control such as the subsumption architecture [Brooks, 1986, Brooks, 1991].

Agents compete for control, having to make a case that they should be given it. One will *win*, having its action executed, then they will compete again for the next action to be executed, and so on indefinitely. The simplest scheme would be for each agent to suggest its action with a strength (or *Weight*) W, expressing how important it is to their purposes that they be obeyed at this moment, and have the creature execute the action that comes with the largest W. To be precise, we have a collection of agents $A_1, \ldots, A_n$ inside the one body. Each discrete time step, the creature observes the world to be in some state x. Each agent A_i suggests an action $a_i(x)$ with weight $W_i(x)$, and the creature executes action $a_k(x)$ where:

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

We call A_k the *leader* in the competition for state x at the moment, or the *owner* of x at the moment. The actions $a_i(x)$ will be whatever actions the agent has learnt to take to pursue its goals. It is where the *W-values* $W_i(x)$ come from, and how they change in response to not being obeyed, that is the interesting bit. Schemes using such 'importance' values are common in multi-behavior models (e.g. see the 'utility' functions in [Aylett, 1995]), and are normally hand-designed. To get them generated for free, we look to reinforcement learning.

within a single body, [Brooks, 1986] uses *layer* (though [Brooks, 1994] also uses *process*), [Blumberg, 1994] uses *activity* and [Sahota, 1994] uses *behavior*.

2 Reinforcement Learning

2.1 Q-learning

Watkins [Watkins, 1989] introduces 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. $P_{xa}(y)$ is the probability that doing a in x will lead to state y and $P_{xa}(r)$ is the probability that doing a in x will generate 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:

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

Given some restrictions (see details in [Watkins and Dayan, 1992]), Q-learning converges to a unique set of values $Q^*(x,a)$ which define a stationary deterministic optimal policy, namely to always takes the action with the highest Q^*-value:

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

A RL agent learns not only 'what' it wants to do, but also '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 in multi-behavior systems, these latter numbers are just what we are looking for.

2.2 Competition among selfish Q-learners

In the model in Figure 1, I make each agent into a Q-learning agent, with its own set of Q-values and more importantly, with its own reward function. To formalise, each agent A_i receives 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 maintains personal Q-values $Q_i(x,a)$ and W-values $W_i(x)$.

The basic criterion for generating W-values is that we don't want all $W \rightarrow W_{max}$. We want the agents to not fight equally for all states, but rather discriminate depending on x. We need something akin to an *economy*, where agents have finite spending power and must choose what to spend it on.

One way of achieving this is to have W express the difference between predicted reward $\mathcal{P}$ (what is predicted if we are listened to) and actual reward $\mathcal{A}$ (what actually happened).

3 W-learning

Consider Q-learning as the process:

$$\mathcal{P} := (1-\alpha_Q)\mathcal{P} + \alpha_Q(\mathcal{A})$$

Then W-learning is:

$$W := (1-\alpha_W)W + \alpha_W(\mathcal{P} - \mathcal{A})$$

For updating the Q-values, only one agent (the leader A_k) suggested the executed action a_k. However, all agents can learn from the transition (under their own different reward functions). We update for all i:

$$\begin{aligned} Q_i(x,a_k) &:= (1-\alpha_Q)Q_i(x,a_k) \\ &\quad +\alpha_Q(r_i + \gamma \max_{b \in A} Q_i(y,b)) \end{aligned}$$

For the W-values, we only update the agents that were not obeyed. We update for $i \neq k$:

$$\begin{aligned} W_i(x) &:= (1-\alpha_W)W_i(x) \\ &\quad +\alpha_W(Q_i(x,a_i) - (r_i + \gamma \max_{b \in A} Q_i(y,b))) \end{aligned}$$

Alternatively [Sutton, 1988], consider Q-learning as the process:

$$\mathcal{P} := \mathcal{P} + \alpha_Q(\mathcal{A} - \mathcal{P})$$

Then W-learning is:

$$W := W + \alpha_W((\mathcal{P} - \mathcal{A}) - W)$$

3.1 Convergence of W-learning

As Q is learnt, the update for A_i, $i \neq k$, is approximated by:

$$W_i(x) := (1-\alpha_W)W_i(x) + \alpha_W(V^*_i(x) - (r_i + \gamma V^*_i(y)))$$

where r_i and y are caused by the leader A_k. We can write this as:

$$W_i(x) := (1-\alpha_W)W_i(x) + \alpha_W d_{ki}(x)$$

where the random variable $d_{ki}(x)$ is the 'deviation' (difference between predicted $\mathcal{P}$ and actual $\mathcal{A}$) that A_k causes for A_i in state x if both are converged to their respective Q^*. Note that:

$$\begin{aligned} E(d_{ki}(x)) &= V^*_i(x) - (E(r_i) + \gamma E(V^*_i(y))) \\ &= V^*_i(x) - (\textstyle\sum_r r P^i_{xa}(r) \\ &\quad + \gamma \textstyle\sum_y V^*_i(y) P_{xa}(y)) \end{aligned}$$

where $a = a^*_k(x)$. We expect:

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

and we expect for $i \neq k$:

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

If A_k leads forever:

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

Of course, it may be interrupted, as some new agent takes the lead. If A_i itself takes the lead, then W-learning stops for it until (if ever) it loses it. If another agent A_l takes the lead, then A_i will suddenly be taking samples from the distribution $d_{li}(x)$. If we update forever from *this* point, then $W_i(x)$ eventually converges to the expected value of the new distribution:

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

Essentially, the reason why W-learning converges is that for the leader to keep on changing, W must keep on rising. And while there may be statistical variations in any finite sample, in the long run the expected values must emerge. 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:

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

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. For more detailed analysis see [Humphrys, 1995].

3.2 MPEG Movie demos

The concept of W-learning has been tested in a simple 'antworld' where a creature must collect food while avoiding moving predators. A series of MPEG Movies of the results can be viewed at: `http://www.cl.cam.ac.uk/users/mh10006/w.html`

4 The House Robot problem

This leads to the question of the commercial usefulness of systems with multiple parallel, partially-satisfied goals. Inspired by a familiar such system, the common household dog, I am looking at the action selection issues that might be faced by a hypothetical, multi-purpose 'house robot'. The question is: 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. It could also double as a mobile wandering smoke alarm, and default perhaps to a vacuum cleaner when nothing was happening. Ignoring the practicalities of this kind of scenario, I use it as inspiration to build an artificial world to test the action selection issues.

In the artificial gridworld of Figure 2, 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 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 robot 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 robot puts out the fire on a square by moving onto it.

The robot senses the direction, but not distance, of things in a small radius around it (the eight main compass directions, numbered 0-7, also 8 for 'here' and 9 for 'not visible'). Fire is the only thing that it can detect through walls - otherwise if something is blocked by a wall it is not visible. The robot's actions are simply move in direction 0-7 or stay still. The number of possible states x is 1.2 million. We populate the robot with the following collection of agents, each driven by a different reward function. Note A_c should head for the centre of an open area while A_w should engage in wall-following. Rewards are in the range $0 < r \leq 1$.

A_d	if (picked up dirt)	return r_d else return 0
A_p	if (arrived at plug)	return r_p else return 0
A_c	if (lost sight of wall)	return r_c else return 0
A_w	if (wall same dir as last time)	return r_w else return 0
A_u	if (made ID)	return r_u else return 0
A_s	if (ID=stranger and visible)	return r_s else return 0
A_m	if (ID=family and here)	return 0 else return r_m
A_f	if (put out fire)	return r_f else return 0

This defines a vast range of robot behaviors, depending on what these rewards are. 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 $(\mathcal{P} - \mathcal{A})$ differences will be so large. 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 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. By varying the relative size of the rewards, we can vary the relative influence of each agent in the collection.

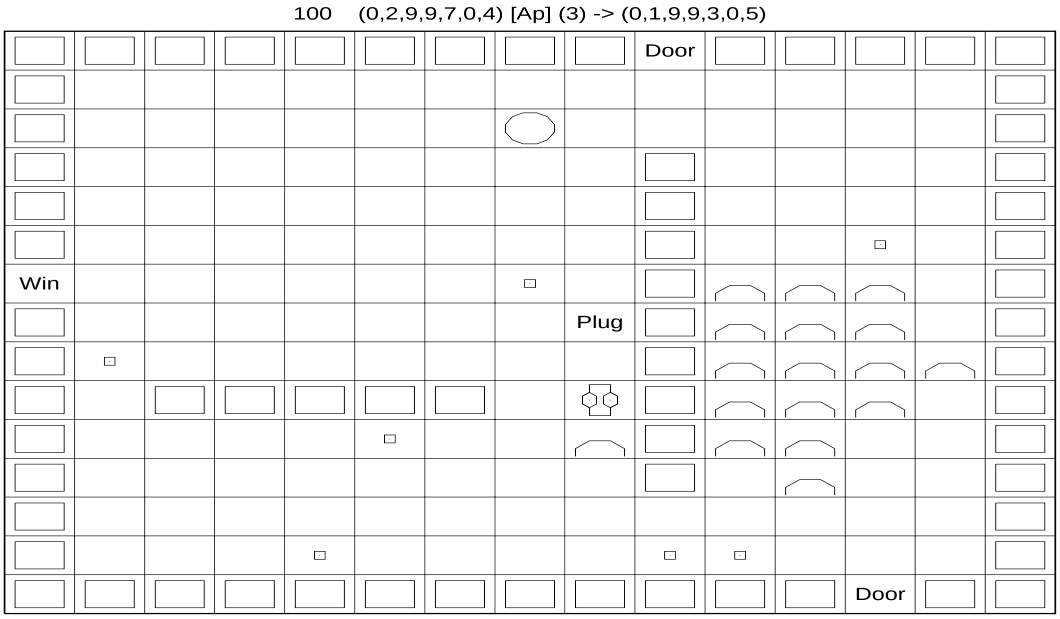

Figure 2: The House Robot's world. Here, the building is on fire, dirt is scattered everywhere, and an unidentified human has just come in the top door.

4.1 Defining good solutions

We obviously need to define somehow what type of collection we are looking for i.e. how one robot is better than another. Our judgement is made by totting up points during a run:

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

We could use this *global reward function* to just learn the full behavior with one agent, but we would be dealing with a statespace of size 1.2 million. Instead, using the decentralised approach, each agent need only sense what is relevant to its reward function. Here, none of the eight agents deals with a statespace larger than 30. We use the global score as the *fitness function* for testing different combinations of rewards. Did you spot the problem with this fitness function? Genetic algorithm search did. It produced the solution:

$r_d = 0.93$
$r_p = 0.34$
$r_c = 0.47$
$r_w = 0.08$
$r_u = 0.47$
$r_s = 0.74$
$r_m = 0.34$
$r_f = 1$

which picks up a remarkable average of 29.590 points every 100 steps. It does this by jumping in and out of the plug non-stop, picking up 1 point every alternate go. A_p dominates its behavior (though you might not expect this just looking at the rewards). It doesn't get the maximum of 50 because if it completely ignores fire, any fire that starts will rapidly become large and then subtract 1.2 points every 2 steps, overwhelming the points scored at the plug. This illustrates that even specifying what we are looking for is difficult. We revise the global score to make hitting the plug a continuous-type score:

```
...
 if (arrived at plug)       add 0.1 points
...
```

4.2 Analysis of evolved solution

Evolving with this new fitness function produced the best solution:

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

which averages 13.446 per 100 steps. It seems to do this by interleaving all its goals. 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, resulted in a score of 8.612 under this fitness function. One might argue that the fitness function is responsible - but if we knew in advance that what we wanted was a strict hierarchy, there would be no problem to solve.

It is clear that in many situations we do not want a strict hierarchy but rather want to see *opportunistic* behavior, where different goals are partially satisfied on the way to solving other goals. *Dithering* [Minsky, 1986, Sahota, 1994] in general is avoided in W-learning collections since 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. The agents will present different W-values accordingly.

Looking closer at our evolved solution, 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 $(i, 2)$ (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 $(i, 0)$ (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 are highlighted.

```
              (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]

              (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 robot in a strong wandering behavior otherwise.

A_p with its tiny reward is irrelevant - its job tends to be done for it anyway by A_c bringing the robot towards the centre. 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
```

5 Summary and conclusion

To conclude, W-learning is a method of learning action selection rather than designing it, based on RL, a method of learning behavior rather than designing it. It resolves competition in a manner which allows the expression of low-priority goals, avoids dithering, and allows for opportunism and a concept of default behavior, all qualities that one would try to build into a hand-designed action selection mechanism.

W-learning resolves competition without resorting to devices such as killing off agents that are disobeyed for time t, without any $W \to \infty$, and in fact with normally most $W \ll W_{max}$.

Finally, W-learning is fair resolution of competition - the most likely winner of a state is the agent that is most likely to suffer the highest deviation if it does not win.

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.*

[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, 1986] Brooks, Rodney A. (1986), A robust layered control system for a mobile robot, *IEEE Journal of Robotics and Automation* vol.RA-2, no.1, Mar 1986.

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

[Brooks, 1994] Brooks, Rodney A. (1994), Coherent Behavior from Many Adaptive Processes, in Dave Cliff et al., eds., *Proceedings of the Third International Conference on Simulation of Adaptive Behavior (SAB-94).*

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

[Minsky, 1986] Minsky, Marvin (1986), *The Society of Mind*, Simon and Schuster, New York.

[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).*

[Sutton, 1988] Sutton, Richard S. (1988), Learning to Predict by the Methods of Temporal Differences, *Machine Learning* 3:9-44.

[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.