Towards self-organising action selection

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

Abstract

Systems with multiple parallel goals (e.g. autonomous mobile robots) have a problem analogous to that of action selection in ethology. Architectures such as the subsumption architecture (Brooks) involve multiple sensing-and-acting agents within a single robot, more than one of which is capable of controlling the robot on its own if allowed. Which to give control to at a given moment is normally regarded as a (difficult) problem of design. In a quest for a scheme where the agents decide for themselves in a sensible manner, I introduce a model where the agents are not only autonomous but are in full competition with each other for control of the robot. Interesting robots are ones where no agent achieves total victory, but rather a series of compromises are reached. Having the agents operate by the reinforcement learning algorithm Q-learning (Watkins) allows the introduction of an algorithm called 'W-learning', by which the agents learn to focus their competitive efforts in a manner similar to agents with limited spending power in an economy. In this way, the population of agents organises its own action selection in a coherent way that supports parallelism and opportunism. In the empirical section, I show how the relative influence an agent has on its robot may be controlled by adjusting its rewards. The possibility of automated search of agent-combinations is considered.

Keywords: reactive systems, action selection, autonomous mobile robots, reinforcement learning, multi-module learning

1 Action selection

It is generally accepted in ethology that animals have multiple competing desires or goals, among which attention must endlessly switch. The animal must solve the problem of *action selection* - the choosing of the appropriate goal to pursue or desire to satisfy at a given moment in time, given the state of the external world.

In recent years, roboticists (and other AI workers) have become interested in models of such systems with multiple parallel goals.

[*]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

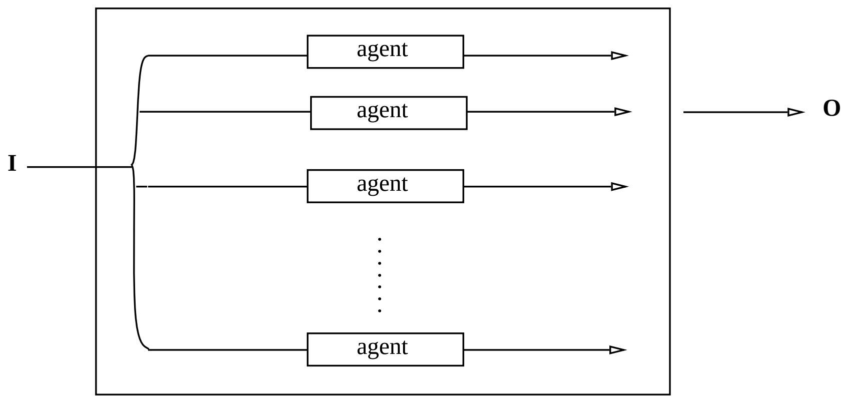

Figure 1: Competition among selfish peer agents in a horizontal architecture. Each agent suggests an action, but only one action is executed. Which agent is obeyed changes dynamically.

1.1 The subsumption architecture

Brooks [Brooks, 1986, Brooks, 1991] introduces an architecture for building autonomous mobile robots which he calls the *subsumption architecture*.

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. The subsumption architecture 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.
- 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 - one can imagine goals which are directly-competing peers. 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.

1.2 Competition among selfish agents

I introduce a model (Figure 1) with the following features:

- Make the layers peers, so that each can function in the absence of *all* the others. Now they are fully autonomous sensing-and-acting *agents* [Minsky, 1986], not ordered in any hierarchy, but rather in a loose *collection*.

- Have them 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.

A simple scheme would be one where each agent suggests its action with a strength (or *Weight*) W, expressing how important it is to their purposes that they be obeyed at this moment, and the robot executes the action that comes with the largest W.

To be precise, each agent A_i maintains a table of *W-values* $W_i(x)$. Given a state x, each agent A_i suggests some action $a_i(x)$ with weight $W_i(x)$, The robot 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.

We can draw a map of the state-space, showing for each state x, which agent succeeds in getting its action executed. Clearly, a robot in which one agent achieves total victory (wins the whole state-space) is not very interesting. Rather, interesting robots are ones where the state-space is fragmented among different agents.

1.2.1 W-values as action selection

This is a *winner-take-all* action selection scheme - that is, the winner gets its exact action executed (as opposed to a scheme where the actions of agents are merged). This corresponds to the classical view in ethology that at any one time, only one behavior system is being expressed in the movements of the animal.

The division of control here is state-based rather than time-based. 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. It is not clear however, that these 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.

For example, consider the conflict between feeding and body maintenance (discussed by Blumberg). Some action selection schemes assign priorities to entire activities, and then worry about how low-priority cleaning is ever going to be able to interrupt feeding. In our scheme, let $x = (e, i)$ be the state, where e is information from external sensors and $i = (f, c)$ is information from internal sensors, where f takes values 2 (very hungry), 1 (hungry) and 0 (not hungry) and c takes values 2 (very dirty), 1 (dirty) and 0 (clean). The 'Food' agent suggests actions with weight $W_f(x)$. The 'Clean' agent suggests actions with weight $W_c(x)$. We should find that for a given e, c:

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

and for a given e, f:

$$W_c((e,(f,2))) > W_c((e,(f,1))) > W_c((e,(f,0)))$$

A very strong 'Food', only rarely interrupted by 'Clean', would be represented by, for a given e:

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

For agents to be able to generate their own W-values, we need a scheme whereby they attach some kind of numerical 'fitness' value to the actions they wish to take. Previous work in action selection has regarded assigning such values as a problem of design. 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 (for example, see [Tyrrell, 1993, Ch.9]) until the designer is satisfied that the formulas deliver utility estimates that are fair.

In fact, there is 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.

2 Reinforcement Learning

2.1 Q-learning

Watkins [Watkins, 1989] introduces a method of reinforcement learning which he calls *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, it 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 not interested 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:

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

where the *learning rate* α, $0 \leq \alpha \leq 1$, takes decreasing (with each update) successive values $\alpha_1, \alpha_2, \alpha_3 \ldots$, such that $\sum_{i=1}^{\infty} \alpha_i = \infty$ and $\sum_{i=1}^{\infty} \alpha_i^2 < \infty$.

If each pair (x, a) is visited an infinite number of times, then Q-learning converges to a unique set of values $Q(x, a) = Q^*(x, a)$ which define a stationary deterministic optimal policy [Watkins and Dayan, 1992].

The optimal policy is defined by $\pi^*(x) = a^*(x)$ where:

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

2.2 Competition among selfish Q-learners

I use Q-learning as the mode of operation of the competing selfish agents in my model. Each agent is 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.

Consider what happens when agent A_i is *not* obeyed. If it 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.

2.2.1 W $= (\mathcal{P} - \mathcal{A})$

What we really need to express in W is the difference between predicted reward $\mathcal{P}$ (what is predicted if we are listened to) and actual reward $\mathcal{A}$ (what actually happened). What happens when we are not listened to depends on what the other agents are doing. I introduce an algorithm called *W-learning* for building up W-values that express this difference.

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:

$$Q_i(x,a_k) := (1-\alpha_Q)Q_i(x,a_k) + \alpha_Q(r_i + \gamma \max_{b \in A} Q_i(y,b)) \tag{2}$$

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

$$W_i(x) := (1-\alpha_W)W_i(x) + \alpha_W(Q_i(x,a_i) - (r_i + \gamma \max_{b \in A} Q_i(y,b))) \tag{3}$$

The reason why we do not update $W_k(x)$ is explained later (Section 3.3). In (object-oriented) pseudo-code, the W-learning system is, every time step:

```
state x := observe();
for ( all i )
 a[i] := A[i].suggestAction(x);
find k
execute ( a[k] );
state y := observe();

for ( all i )
{
 r[i] := A[i].reward(x,y);
 A[i].updateQ ( x, a[k], y, r[i] );
 if (i != k)
  A[i].updateW ( x, a[i], y, r[i] );
}
```

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 After Q has been (somewhat) learnt

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:

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

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 from the start to infinity, then:

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

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:

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

3.2 Convergence of W-learning

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:

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

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].

3.3 Scoring $W_k(x)$

Should we score W if obeyed as well? If we do, then:

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

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. So there will be no resolution of the competition ever.

4 Empirical Work

We can take hand-picked combinations of agents and see how they resolve their competition. Alternatively, we can use the fact that *any* combination of agents will resolve their competition, and do an automated search looking for interesting combinations.

Starting with any n agents with any reward functions (leading to any converged Q-values), W-learning will eventually converge and the agents will have permanently divided up state-space between them. So we can define spaces of possible robot architectures where every point in the space represents a robot that can be built and tried out (for the best exposition of this concept see [Dawkins, 1986, Ch.3]). Such a space can be searched in a systematic manner, or in a stochastic manner similar to evolution by natural selection [Langton, 1989].

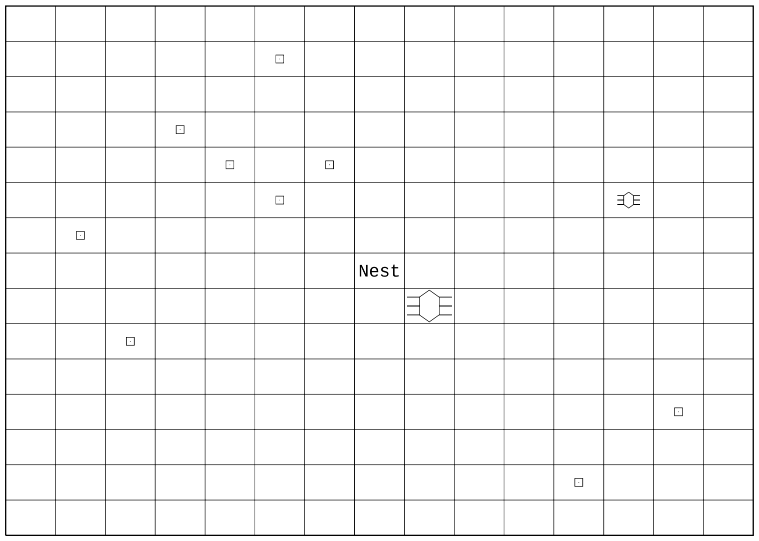

Figure 2: The toroidal gridworld.

4.1 The Simulated World

The problem I set for my W-learning simulated robot is the conflict between seeking food and avoiding moving predators on a simple simulated landscape (Figure 2). The world is a toroidal (SIZE x SIZE) gridworld containing a nest, a number (NOFOOD) of stationary, randomly-distributed pieces of food, and a number (NOPREDATORS) of randomly-moving dumb predators. World, evolution and learning are all implemented in C++.

Each timestep, the robot 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 robot is to forage food (i.e. find it, and bring it back to the nest) while avoiding the predators.

The robot senses $x = (i, n, f, p)$, where:

- i is whether the robot is carrying food or not, and takes values 0 (not carrying) and 1 (carrying).
- n is the direction (but not distance) of the nest, and takes values 0-7 (the eight main compass directions), 8 (when at the nest) and 9 (when the nest is not visible within a small radius of RADIUS squares).
- f is the direction of the nearest visible food, taking values 0-9.
- p is the direction of the nearest visible predator, also taking values 0-9.

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

4.2 Systematic search

We carry out a systematic search of various combinations of agents, looking for combinations whose W-converged situation is adaptive.

We want to define a space of agent-collections which will encompass a very broad range of robot behaviors. Consider the space of collections of 3 agents A_f, A_n, A_p, one of each of these types:

```
FoodSensingAgent
x = (i,f), state-space size 20
reward function:
 reward()
 {
  if (just picked up food) return r1
  else return 0
 }

NestSensingAgent
x = (n), state-space size 10
reward function:
 reward()
 {
  if (just arrived at nest) return r2
  else return 0
 }

PredatorSensingAgent
x = (p), state-space size 10
reward function:
 reward()
 {
  if (just shook off predator (no longer visible)) return r3
  else return 0
 }
```

for different values of r_1, r_2, r_3 in the range $r_{min} = 0$ to $r_{max} = 1$. That is, we're only interested in food-seekers, nest-seekers and predator-avoiders (we assume that food-avoiders, predator-seekers, etc. are of no interest). Even so, the range of possible behaviors of such a collection is vast. The behavior of the collection as a whole will depend upon the relative sizes of each agent's reward (hence how strong the agent will be in W-competitions, and how much of the statespace it will win).

We do a systematic search of all combinations of $r_1, r_2, r_3 \in \{0.1, 0.3, 0.5, 0.7, 0.9\}$ (a total of 5^3 combinations). In the test, we calculate the average amount of food foraged per 100 steps F and the average number of predator encounters per 100 steps P. Here are the 10 best foragers:

```
                                                          F       P
robot=[food(0.700),nest(0.100),predator(0.500)] scores: 7.220   0.050
robot=[food(0.700),nest(0.100),predator(0.300)] scores: 6.970   0.210
robot=[food(0.500),nest(0.100),predator(0.700)] scores: 6.670   0.090
robot=[food(0.900),nest(0.100),predator(0.300)] scores: 6.510   0.160
robot=[food(0.500),nest(0.100),predator(0.300)] scores: 5.860   0.050
robot=[food(0.500),nest(0.100),predator(0.900)] scores: 3.790   0.010
robot=[food(0.300),nest(0.100),predator(0.300)] scores: 3.300   0.030
robot=[food(0.700),nest(0.300),predator(0.500)] scores: 3.060   0.090
robot=[food(0.700),nest(0.300),predator(0.900)] scores: 2.380   0.080
robot=[food(0.900),nest(0.300),predator(0.700)] scores: 2.230   0.140
```

Here are the joint best predator-avoiders (all encountered no predators, they are sorted in order of food foraged):

```
                                                     F       P
robot=[food(0.300),nest(0.100),predator(0.900)] scores: 0.050   0.000
robot=[food(0.900),nest(0.100),predator(0.700)] scores: 0.060   0.000
robot=[food(0.300),nest(0.700),predator(0.900)] scores: 0.090   0.000
robot=[food(0.100),nest(0.300),predator(0.900)] scores: 0.150   0.000
robot=[food(0.300),nest(0.100),predator(0.700)] scores: 0.150   0.000
robot=[food(0.100),nest(0.100),predator(0.900)] scores: 0.300   0.000
robot=[food(0.100),nest(0.100),predator(0.500)] scores: 0.330   0.000
robot=[food(0.700),nest(0.100),predator(0.900)] scores: 0.340   0.000
robot=[food(0.700),nest(0.100),predator(0.700)] scores: 0.370   0.000
robot=[food(0.500),nest(0.500),predator(0.700)] scores: 1.090   0.000
```

4.2.1 MPEG Movie demo

An MPEG Movie demo of the best forager above can be viewed on the internet at `http://www.cl.cam.ac.uk/users/mh10006/w.html`

4.3 Evolutionary search

Alternatively, we can use a Genetic Algorithm (GA) (from [Holland, 1975], for an introduction see [Goldberg, 1989]) to search the space.

In this experiment, food does not grow. Instead, the robot makes a number of runs, each of length STEPSPERRUN timesteps. Writing F as the average amount of food foraged per run and P as the average number of predator encounters per run:

$$\text{fitness} = C_F F - C_P P$$

4.3.1 Evolved W-collections

We consider the space of collections of 3 agents A_f, A_n, A_p, one of each of these types:

```
FoodSensingAgent
 reward()
 {
  if (just picked up food) return r1
  else return r2
 }

NestSensingAgent
 reward()
 {
  if (just arrived at nest) return r3
  else return r4
 }

PredatorSensingAgent
 reward()
 {
  if (just shook off predator (no longer visible)) return r5
  else return r6
 }
```

for $r_1, \dots, r_6$ in the range $r_{min} = 0$ to $r_{max} = 2$.

After only quick evolutionary searches we found the following W-collections. The table shows, for the different fitness functions, the best solution (set of 6 rewards) found by evolution, and in each solution, the division of the full (size 1800) statespace between the agents:

CF	CP	max		best	evolved	W-collection				percentage			
										ownership			
		fitness		r1	r2	r3	r4	r5	r6	Af-An-Ap	F	P	fitness
100	1	400	EVO1	1.73	0.11	0.41	0.26	1.37	1.20	49-35-16	3.650	0.410	[364.590]
10	1	40	EVO1	1.73	0.11	0.41	0.26	1.37	1.20	49-35-16	3.650	0.410	[36.090]
1	1	4	EVO1	1.73	0.11	0.41	0.26	1.37	1.20	49-35-16	3.650	0.410	[3.240]
1	10	4	EVO2	1.93	0.28	0.26	0.07	1.99	0.76	35-16-49	2.990	0.015	[2.840]
1	100	4	EVO3	1.46	0.62	0.22	0.07	1.92	0.12	20-16-63	2.605	0.000	[2.605]

4.3.2 Analysis of EVO1

Our first level of analysis produces no surprises: $r_1 > r_2$, $r_3 > r_4$ and $r_5 > r_6$ throughout.

The next level of analysis looks at who finally wins each state. In EVO1, 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, 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
```

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
     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 robot 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 robot to wander, and then head for food when visible. A_n is constantly suggesting that the robot 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 robot 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.

In fact, 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.

4.4 Discussion

Given some collection of agents, there are a large number of ways in which they can divide up the statespace between them. This defines a large space of possible agent-combinations. As conditions change, we can move continuously through this space by varying the numerical rewards (their size, and the differences between them), so varying each agent's possession of state-space in a continuous manner. This is an alternative to the programmed approach where we have to specify and hand-code perhaps quite different logic for each situation.

5 Summary and conclusion

We have shown that for any given collection of Q-learners, there is what can be described as a 'natural' action-selection scheme. We avoid the problem of defining the flow of control by having it follow naturally once the collection of agents is specified.

W-learning resolves competition without any $W \rightarrow \infty$, 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

[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.

[Dawkins, 1986] Dawkins, Richard (1986), *The Blind Watchmaker*, Penguin Books.

[Goldberg, 1989] Goldberg, David E. (1989), *Genetic Algorithms: in search, optimization, and machine learning*, Addison-Wesley.

[Holland, 1975] Holland, John H. (1975), *Adaptation in Natural and Artificial Systems*, Ann Arbor, Univ. Michigan Press.

[Humphrys, 1995] Humphrys, Mark (1995), *W-learning: Competition among selfish Q-learners*, technical report no.362, University of Cambridge, Computer Laboratory.

[Langton, 1989] Langton, Christopher G. (1989), Artificial Life, in Christopher G.Langton, ed., *Artificial Life.*

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

[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.