Co-evolution – with animats in pursuit-evasion

... and in an application to ‘sorting networks’

*** Think about how you would set about re-implementing either of these as a project of your own

---

D. Cliff and G. F. Miller
http://citeseer.ist.psu.edu/cliff95coevolution.html
Or on http://drop.io/alergic: as CliffMiller96.pdf

---

Two (or more) species evolve in a situation where the selection pressure on one species (eg the Predators or Pursuers) depends (at least in part) on the current fitness of the other species (Prey or Evaders) ... and vice versa

Arm’s Race, or ‘Red Queen effect’
-- you run as fast as you can yet stay in same place
(... figuratively !)

This provides very much an implicit fitness function rather than an explicit one.

---

Studied in deliberately simplified environment - a 2-D infinite plane with no walls or obstacles, just one pursuer, one evader

Animats (animal/robot) - term often used in SAB

Motors:
These animats have left and right wheels. Variable forces can be applied L and R, simple Newtonian physics

Fuel use (from limited fuel tank) proportional to square of force. Friction acts to slow you down.

---

Each animat has several (typically 2) simulated ‘photoreceptors’. Position (relative to straight-ahead) and angle of acceptance (wide/narrow) is genetically specified -- and hence can co-evolve with the ‘brain’

Each sensor returns proportion of its angle-of-view which is not obscured by any object on horizon

Hence simulation is a very simplified version of real physics, but still has some significant element of physical plausibility.
The control system is a CTRNN (continuous-time recurrent neural network model), of precisely the Beer type (see previous lecture).

Fully connected ANN, with (fixed) weights and biases that are genetically specified — ie evolved.

2 neurons connected to 'eyes', 2 to motors.

A Genotype for any one Animat specifies -
1) the sensory morphology
2) the architecture (weights etc) of the ANN

The Genetic Algorithm evolves 2 completely distinct populations ('species')

Spatially distributed GA — individuals in the population are spread out over a 'mating' grid, and will only mate, and replace, close neighbours on this grid.

Evaluation:
All in the population of pursuers are tested against the same best-of-last-generation evader.
And vice versa.

Several trials from random starts:
Evader fitness = how long before caught
Pursuer fitness = ++ for 'approaching evader'
   + bonus for hit, sooner the bigger

That last picture showed successful pursuers/evaders from generation 999
But at gen 0, there was a pursuer which failed to catch an evader, and at gen 999 likewise.

So in what sense has there been any 'advance'?
Possibility of 'no real advance' in coevolution — cf Stone Scissors Paper game, no strategy can be supreme for ever.
Test evader from gen 200 against pursuer from gen 999.
Later work extended this idea, of monitoring current gen against best of previous gens. Does this escape from the circular trap?

Can coevolution be used for engineering purposes?
Here is an example


Danny Hillis -- Connection machines --powerful very distributed parallel machines.
This work done in late 1980s, 64,536 processors, populations 500 to 1000000, 'about 100 to 1000 generations per minute'
Evolving minimal sorting networks

A sorting network is something that can be given any scrambled list of N objects with different values (here N=16) --- and it is in effect an algorithm that will systematically sort the list into order by a sequence of 'compare and maybe swap's.

The sorting network is a series of pairs of numbers, [a b] which can be interpreted as:-

✓ Compare the a\textsuperscript{th} and b\textsuperscript{th} items in your scrambled list.
✓ If in wrong order, swap, otherwise leave

Visual way to represent, 16 rows represent the 16 items to be re-ordered. Starting from left, the vertical bars show rows to be compared/ swapped. Numbering rows from 0 to 15, above swaps are:

[0 1] [2 3] ...[14 15] [0 2] [4 6] [8 10] ......
The previous diagram has a total of 60 swaps and was (in 1991) the shortest-known, discovered by MW Green. It is a perfect sorter, in that if you present it with any scrambled list, after going through all the 60 swaps from left to right then the list comes out perfectly ordered. [note: for swaps shown as bars in same vertical column, it will not matter which is done first]

The problem is to find the shortest network, ideally better than this 60, which still sorts anything.

Do you have to check if it sorts all possible combinations of numbers in the list – NO!

It can be shown that if a network correctly sorts any scrambled list of 0s and 1s (so that it finishes up with all the 1s at the top, all the 0s at the bottom), then the network will also sort any list of real-valued items.

So can test a 16-network exhaustively with only $2^{16}$ tests (about 32,000) – instead of 16 factorial (about $2 \times 10^{13}$).

But this is still a lot of tests -- can one save time? – YES!

We need a genetic encoding, so that strings of characters represent possible sorting networks. But we are not sure how long any sorting network will be before we start – after all, we are looking for the shortest.

Hillis chose a sort-of-diploid encoding

<table>
<thead>
<tr>
<th>haploid = 1 string</th>
</tr>
</thead>
<tbody>
<tr>
<td>diploid = 2 strings</td>
</tr>
</tbody>
</table>

A full genotype is 60 such codon-pairs, hence encoding between 60 and 120 test/swaps.

cf: homozygous / heterozygous (a bit different !)

The population is initialised with everyone having the same first 32 exchanges (that are known to be sensible), and thereafter randomised.

Then each network is tested on how well it sorts -- the percentage of input test scrambled lists which it sorts correctly.

Rather than testing on all $2^{16}$ test cases, it could be tested on a random sample.

OR (see later) the test cases could be chosen cleverly – coevolution.
Tournaments:
pick pairs of contestants in local neighbourhood
(Gaussian spread, nearer is more likely)

This was the really simple way to get a spatially distributed GA within the Microbial GA.
Just modify the way in which guys were selected for tournaments, so that they were 'close to each other'.

Last lecture I briefly mentioned demes – partially overlapping sub-populations with local breeding.
Idea from Lee Spector: easiest to do this with a ring-shaped geography of overlapping sub-popns -
Easy within the Microbial one-liner!

Just adjust the choice of b for each tournament. We keep
\[ a = \text{POP} \times \text{drand48}(); \]
but we change the original
\[ b = \text{POP} \times \text{drand48}(); \]
to ....

Demes v1: \[ b = (a + \text{deme_width} \times \text{drand48}) \mod \text{POP}; \]
Where the modulo POP (%POP) joins end of popn back to beginning, in a ring.

OR.....

Demes v2 (recommended): \[ b = (a + 1 + \text{deme_width} \times \text{drand48}) \mod \text{POP}; \]
Where the +1 ensures b different from a. Choose deme_width as some suitable fraction of POP, eg POP=30, deme_width=10.

Tournament: from pair of contestants, compare scores, winner over-writes loser (ie then has 2 copies).
Mating: then select mates locally, with same principles
Recombination to produce offspring
(Hillis actually had 15 crossover points '1 per chromosome')
Mutation: one bit-flip per 1000 sites.

Typical run like this -- without coevolution -- for up to 5000 generations, with a popn of 64536.
Best scores = sorting networks of 65 exchanges -- target was 60.
How can one improve this through coevolution?
Two main sources of inefficiency in the GA without coevolution:

1. Local optima -- once the population had found a 3/4 decent solution, quite probably all the neighbouring solutions (genetically similar) were less fit -- so the population would have to cross a valley to reach 'higher ground'.

2. Inefficiency in testing -- once popn was ¾ decent, they all passed most of the test cases, so little differences in scores.

The answer: co-evolve a separate population of parasite test-cases, which themselves have a fitness function designed to make them as hard as possible for the sorting networks.

This solves both inefficiencies (1) and (2).

Parasite coevolution can generate genetic diversity (cf. W Hamilton)

The population of sorting networks is already spatially distributed on one grid. Have a population of parasites likewise distributed on a similar grid, overlaid.

Each parasite is a genetically specified group of 10 to 20 test cases -- rather than all the 2¹⁶ possible ones.

Each sorting network is tested against the parasite that is on corresponding grid square. The score of the sorting network is 'what proportion of tests does it pass'

The score of the parasite is 'how many tests does it fail the sorter on'

Networks get selected, mated, and reproduce on their grid, parasites completely separately on theirs.

Results improved to a minimum size of 61 (has it been beaten since?)

Prevents getting stuck in local optima -- as soon as this happens, the parasites evolve to zap them.

Population is in a constant state of flux.

Second advantage: testing is more efficient -- need only test on a few difficult test cases, which themselves change appropriately according to circumstances.

Hence computationally more efficient.

The End