IT3708 Module 3

Johan Slettevold
Iver Egge

April 14, 2016

1 Evolutionary algorithm

1.1 Parameters

Parameter	Value	Comment
Generations	50	Higher is better
Population size	200	Higher is better
Number of children	200	Higher is better
Genotype length	$18 \cdot 16$	$3 \cdot 6$ weights stored as 16-bit vectors
Adult selection	Generation mixing	
Parent selection	Tournament selection	
Tournament ϵ	0.9	Set high to prevented stagnation
Tournament group size	20	Set low to prevent stagnation
Boltzmann temperature	1.0	
Boltzmann ΔT	0.01	
Crossover rate	0.1	Set low to prevent stagnation
Mutation rate	1.0	Set high to prevent stagnation

Table 1: Parameters

We found these values by first discovering that a static run would stagnate very quickly; To mitigate the effect, we tuned the ea for high variation.

1.2 Fitness function

The fitness of an individual is based on how much food and poison it ate and constants defined by the developer:

$F * F_b - P * P_p$

Where F denotes the food eaten by the agent, F_b denotes the food bonus constant, P dones the poison eaten by the agent and P_p denotes the poison penalty constant.

A F_b value of 1 and a P_p value of 2 seems well suited.

2 Implementation

The program was developed in the languages *D* and *Python3*, using the *PyD* library for Python/C API functionality. *Python3* is only responsible for calling the algorithms in *D* and the visual representation of the output.

2.1 Artificial neural network

We use a very simple neural network, as suggested by the assignment text. It is a fully connected network with no hidden layers, and no bias sources. Thus, the network has 18 weights. We use a basic sigmoid activation in the output layer.

2.1.1 Genotype and phenotype

The genotypes are stored in a bit vector, with 16 bits for each weight. These 16 bits represent the binary value of the phenotype. When converting the genotype to the phenotype, we first find the decimal value of the binary value and then divide it by 2^16. This gives us a phenotype with value between 0.0 and 1.0.

2.2 Process

To create a suitable network, we started simple. However networks with hidden layers, and different activation functions were tested, but were more difficult to tune. As we saw that our simple network would suffice, we stuck to it.

A network should be able to solve the flatland problem simply by sending low signals for poison fields, and high values for food fields. This simple solution would actually not require more than weights from the same direction to the same direction (for a total of 6 weights). Our network is bigger, and covers this simple idea. This is why our network should work.

The Artificial neural network (ANN) was developed in *Python3* using *Numpy*. The implementation was then ported to *D*, substituting *Numpy* with our own Matrix-class.

3 Performance

In the plots shown below, the blue line represents the highest fitness, the green line represents the average fitness and the red line represents the standard deviation. The y-axis represents the fitness and the x-axis the generation. The fitness of the agent is added together on runs with several scenarios.

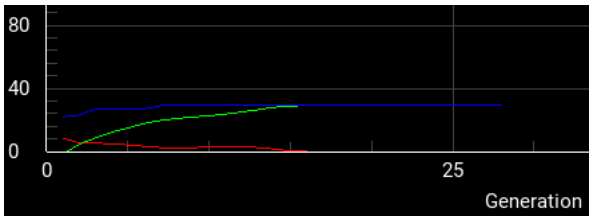

Figure 1: Static run, single scenario

For a static run with a single scenario the EA found a high fitness relatively quickly. The agent performed well on this board, eating 30 of 37 foods and 3 of 27 poisons. On a new, random board, the agent performed less well but still satisfactory, eating 29 of 40 foods and 4 of 24 poisons.

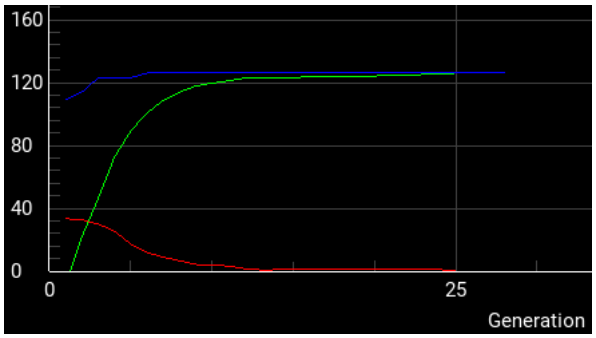

Figure 2: Static run, 5 different scenarios

For a static run with 5 different scenarios, the EA again quickly settled on a high fitness. The agent performs somewhat better on new random boards, as the training data now represents a broader set of situations as long as they are not too similar.

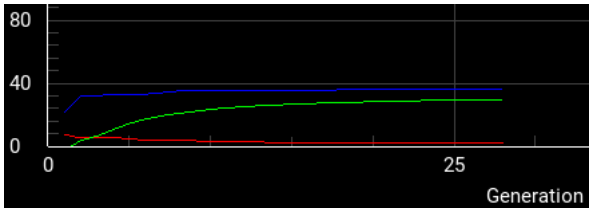

Figure 3: Dynamic run, single scenario

In dynamic mode with a single scenario the agent performs somewhat random on different boards, ranging from very good to very bad. We believe the EA does not get to evaluate an agent on a similar board enough times before

replacing it with another. The plot shows a fitness that fluctuates and never reaches a very high value.

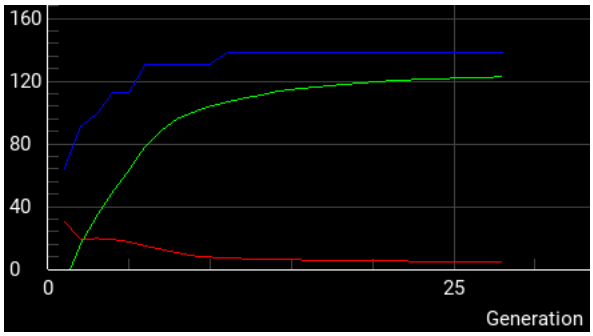

Figure 4: Dynamic run, 5 different scenarios

In the dynamic run with 5 different scenarios, the fitness fluctuates while evolving and the standard deviation never reaches zero. We were able to reach the highest fitness using this configuration. The agent performs similarly well on subsequent new random boards.

3.1 Summary

The static runs with 5 different scenarios provide an agent that solves the problem somewhat generally, given that the scenarios are not very similar. On a single scenario dynamic run the agent does not get to train enough before being replaced, and scores somewhat randomly on subsequent new random boards.

The 5 scenario dynamic run provides an agent that performs well on almost all boards. This agent has developed a fitness that solves the problem in general.

The plots confirms these assumptions, as discussed for each scenario.