NORWEGIAN UNIVERSITY OF SCIENCE AND TECHNOLOGY
IT3708 – SUBSYMBOLIC METHODS IN AI

Project 3:
Evolving Neural Networks for a Flatland Agent

Per Magnus Veierland
permve@stud.ntnu.no

April 14, 2016

Parameter	Value
Population size	100
Generation count	1000
Adult selection	Full replacement
Elitism count	5
Parent selection	Tournament
Tournament group size	10
Tournament random probability	0.1
Mutation rate	$0.01 \frac{\text{mutations}}{\text{bit}}$.
Crossover points	1

Table 1: Main EA parameters

1 Evolutionary Algorithm (EA) parameters

Table 1 shows the main EA parameters which were used for all evaluations described in this document. All parameters except the adult selection mechanism and crossover has been experimented with. By employing elitism with full generational replacement, it can be ensured that the best solutions are not lost while at the same time being able to tune some of the selection pressure in the system. Using an elitism count of 5 individuals (5% of the population size) was found to be helpful in helping guide exploitation, while allowing for exploration.

Using fitness proportionate- or rank adult selection was found to limit the population diversity greatly, and although sigma selection worked, tournament selection was found to yield the best results, and allows for more selection pressure tuning. A tournament group size of 10 and a random selection probability of 0.1 was found to adequately balance exploitation and exploration.

A single uniformly random crossover point was used for all experiments. The mutation rate was not tuned much, and values in the range of 0.005 to $0.01 \frac{\text{mutations}}{\text{bit}}$ provided enough exploration for successful results.

Increasing the population size notably did not prove beneficial, and a size of 100 was found to be sufficient in sustaining necessary diversity without too much cost. A good solution is usually found comfortably within an evolutionary run of 1000 generations, which takes about 30 seconds using a single Javascript webworker in a Firefox browser.

2 Fitness Function

The fitness function is input by the user as a text string and compiled to a function by the `math.js` library. When evaluating the function, the values for the number of a) food eaten b) total food c) poison eaten d) total poison, are exposed in its scope, such that various fitness strategies can easily be experimented with.

It was found that a simple fitness function (Equation 1) which subtracts the amount of poison eaten from the amount of food eaten, summed over the scenarios the individual is exposed to, works well in this context and is able to produce objectively fit individuals. A constant k_{pe} is used to tune the severity of eating poison. If eating poison is punished too harshly, exploration will be deterred, and no constructive evolution will take place. If it is too low, evolution will only value food eaten, no matter how much poison is also eaten. The consequences of eating poison is not described in the assignment but it is assumed that it should be avoided.

$$\text{Fitness}(i) = \sum_{s \in \textit{scenarios}} \Big(\text{FoodEaten}(i,s) - k_{pe} \cdot \text{PoisonEaten}(i,s) \Big) \tag{1}$$

A constant of $k_{pe} = 2$ was shown to work well in trials. To make comparisons between trials easier, all fitness values are normalized by the number of scenarios and the number of time steps. Given a 10×10 grid, $\frac{1}{3}$ food coverage, and 60 time steps, the normalized upper fitness boundary with this fitness function is 0.55.

3 Artificial Neural Network (ANN) implementation

For analysis, each hidden node can be treated as an AND-gate which matches against a conjunction of the binary inputs. With 6 binary inputs, there is a total of $2^6 = 64$ possible input cases. With one AND-gate to recognize each case, a maximum of 64 hidden nodes would be needed to match every input case, and each output node could be viewed as an OR-gate which matches a disjunction of the activated hidden nodes. Attempting to match every distinct input case is described in *Intelligence Emerging* as an *extensional* strategy, and although it can represent all possible mappings, it requires a large representation and it would be hard to discover *intensional* or *general* behavioral mappings.

Based on the analysis, a better intentional approach was designed. The genotype follows a *fixed-length, direct* encoding. Each weight is represented using 1 bit describing a value of -1 or 1, together with one bit per weight which turns the weight on or off. This switch bit allows both *neutral complexification* and intensional mappings to be represented. As both agent inputs and outputs are binary, both the hidden- and output neurons uses the *Heaviside* activation function, which will activate the neuron whenever the binary inputs are matched. Given that there are 6 binary inputs, the hidden layer bias has a range of -6 to $+6$, using a scaled gray code representation of $\lceil \log_2(6+6+1) \rceil = 4$ bits. Through trials, it was found that 6 hidden nodes were able to represent successful agents, such that the output layer bias also has a range of -6 to $+6$ represented as a scaled gray code using 4 bits.

Gray coding is used to improve the correlation between genotypes and phenotypes, and to give the effect of mutations a more gradual effect. Scaled integers are used such that values outside the effective ranges are not encountered.

If a single output neuron is activated, the corresponding action is chosen, otherwise no action is made by the agent.

The agent input describes the contents of the forward, left, and right locations relative to the agent. As each location can either be empty, contain food, or contain poison; there are 3 possible configurations for each location. Since there are three locations described by the input; there are a total of $3^3 = 27$ possible input combinations. For each of these 27 cases, an agent must decide one of three responses, (discounting the possibility of doing nothing), which yields a total of $3^{3^3} \approx 7.6 \cdot 10^{12}$ possible functionally distinct agents for the Flatland environment.

With 6 inputs, 6 hidden nodes, and 3 output nodes, there are 54 weight values (108 bits) and 9 bias values (36 bits), for a total genome length of 144 bits which has $2.2 \cdot 10^{43}$ possible permutations. Even discounting bloat in input representation and the 0.5 bit overhead per weight, this shows that the genome can represent a significant fraction of the number of possible agents.

4 Performance of the EA

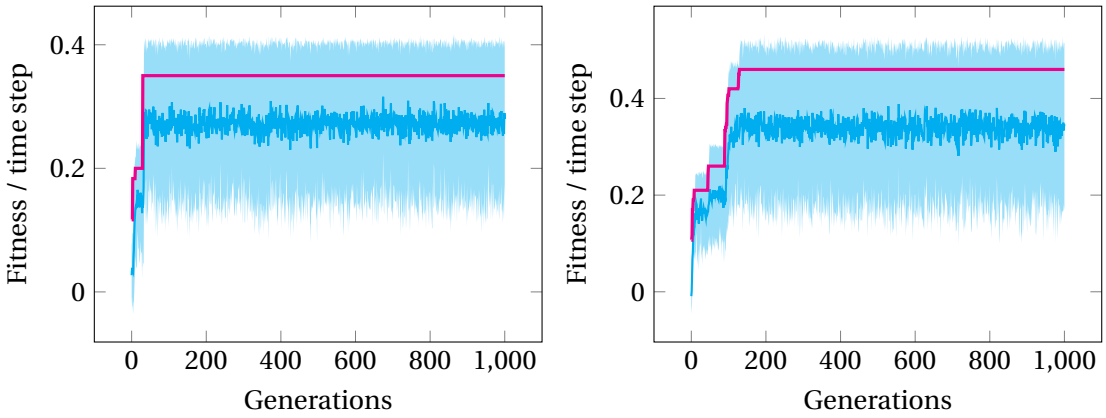

Figure 1: EANN performance when trained on 1 static scenario (left), and on 5 static scenarios (right). Mean population fitness is shown in blue with standard deviation shown in light blue, and the fitness of the best individual is shown in red.

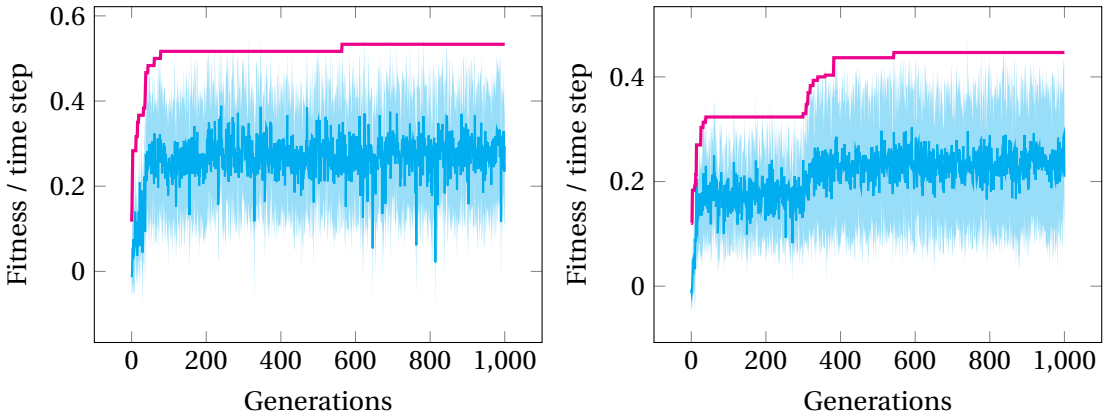

Figure 2: EANN performance when trained on 1 dynamic scenario (left), and on 5 dynamic scenarios (right). Mean population fitness is shown in blue with standard deviation shown in light blue, and the fitness of the best individual is shown in red.

1. **Agent evolved using 1 static scenario:** The best agent evolved achieved a fitness score of 0.35. When observing the scenario it was evolved with it performs quite well, consuming 25 (76 %) of the food and 2 (9%) of the poison over 60 time steps. Observing the behavior, it is clear that the agent has developed very basic behavior, where it mostly moves forward until some poison is met, then turning right and continuing. Due to the static environment and little amount of testing, the agent makes bad decisions such as moving to the right when there is food on the left and poison in front and on the right, however since it performs acceptably overall in the scenario, such behavior is sustained in the population.

   Testing the agent in a random scenario reveals significant deficiencies in the agent's strategy. It ended up getting stuck moving forwards after consuming 7 food and 1 poison, despite there being more food available on both sides of its path, resulting in a fitness score of 0.083.

2. **Agent evolved using 5 static scenarios:** The best agent evolved achieved a fitness of ≈ 0.46, consuming 29 food (88%) and 1 poison (5%), 29 food (88%) and 1 poison (5%), 32 food (97%) and 0 poison (0%), 27 food (82%) and 1 poison (5%), 27 food (82%) and 0 poison (0%), respectively in the 5 static scenarios.

   After observing the behavior of the agent in the five static scenarios, it can be seen that in all the instances where poison was consumed it was due to being surrounded by poison in all directions. There were no examples of bad behavior found, such as consuming poison when other options existed. The agent uses a strategy which involves a lot of agility, and instead of moving in a straight line for as long as possible, it chooses more rapid snake-like motions. This results in good coverage and the ability to consume most of the food available.

   Testing the agent in a random scenario shows that it has been able to generalize effective behavior, and it was able to consume 28 food (85%) and 0 poison (0%) through exploring a large section of the grid.

3. **Agent evolved using 1 dynamic scenario:** The best agent evolved achieved a fitness of 0.53, which is close to the theoretical maximum of 0.55. Since the agent was only tested on a single scenario, this high fitness is likely to be based on some luck rather than just a good strategy. When tested with a random scenario it performs fairly well, consuming 28 food (85%) and 3 poison (14%). It follows a strategy which involves agile movement without moving only in straight lines, and covers a large area in the grid without repeating locations. It does however perform a directly bad move as it consumes a poison in a situation where it would be possible to move to an empty cell.

4. **Agent evolved using 5 dynamic scenarios:** The best agent evolved achieved a fitness of 0.45. When evaluating across multiple scenarios, the fitness value becomes less dependent on luck and is more meaningful since the agent performs well in several environment configurations.

   Testing the best agent on a random scenario shows exploring behavior where the agent is able to cover a large part of the grid while consuming 26 food (79%) and 1 poison(5%). The single poison consumed occurred in a situation where the agent was surrounded by poison.

When comparing the four performance cases, it is clear that using multiple scenarios is necessary to achieve generalized behavior which will work in new scenarios. Using dynamic scenarios instead of static scenarios offers much greater opportunity to expose and test the population through evolution on a variety of edge cases. The behavior of the agent evolved using one dynamic scenario clearly shows more general behavior compared to the agent evolved using one static scenario.

The fitness plots, see Figure 1 and Figure 2, show that the fitness development is more gradual in the dynamic cases compared to the static cases, indicating that learning is more gradual and that new individuals in the population gradually performs better as the population adapts to different scenarios. The standard deviation for the mean population fitness is also visibly lower in the dynamic cases, with the best individual performing more than one standard deviation better than the mean, indicating that the population fitness is more stable than in the static cases.

NB: Due to an error in testing, all results described in this document uses 3 bits to represent the hidden- and output layer bias values, instead of the 4 bits determined by the analysis. This shows that a value of 3 bits per bias value is sufficient to achieve good results.