Genetic Algorithms
Developing the survival instinct of a population of individuals

Besana Emanuele Francesco

1 Genetic Algorithms

Genetic algorithms (GA) belong to the family of evolutionary algorithms (EA), which are computational research methods inspired by Darwin's theory of natural selection. They simulate Darwinian evolution of individuals that are part of a population.
These individuals can be of all kinds: digital organisms, strings, computer programmes, financial strategies etc.. Evolution influences these individuals and better versions of the population, called *generations*, evolve with the 'goal' of maximising fitness, i.e. their ability to survive in a certain environment (perform a certain task, maximise a certain performance...). In this way, EAs transform initially unsophisticated individuals into complex entities.
AGs form a distinct sub-family of EAs because they explicitly use a genetic representation of individuals, which are formalised in a programme in a way similar to the world of genetics, i.e. DNA; the representation of these individuals can then be a sequence of characters, or a vector of 0 and 1 ..., and this is their *genotype*, which defines their behaviour entirely.
In nature, evolution favours entities that are able to reproduce because of some advantage they might possess over the rest of the population, e.g. it might be a physical advantage (strength, stature, beauty), they might be better adapted to that particular environment, etc... In GAs, fitness provides a measure of the performance of a given individual who, having selected a certain task to perform, seeks to survive in the environment in which he or she finds himself or herself. Whereas in nature, fitnesses are somewhat implicit, in GAs they are made explicit through the use of a *function of fitness*, which can be calculated directly from what the entity accomplishes. (This type of reasoning involving fitness functions has a counterpart in the 'cost function' in machine learning.)
The fitness function can be as simple as counting the number of zeros or ones in the genotype, or as complicated as, for example, optimising a multi-objective problem. Population evolution thus seeks to identify individuals with the highest possible fitness: one looks for peaks in the fitness function, and the difficulty or simplicity of this search lies in the specific way in which the programme has been implemented, and of course in the fitness function itself.

Genetic algorithms proceed according to a precise pattern of steps. Initially, a population representing the first generation is created: a fixed number of randomly created individuals, so that the distribution of genotypes is as uniform as possible, so that there is a lot of variety. This is why the parameter 'number of individuals' is particularly important in this step (e.g. some individuals may already be close to the maximum fitness function).
Then the following steps are repeated a fixed number of times: the individuals, having done their task, possess a fitness (calculated via the fitness function) that allows them to rearrange themselves in order of increasing fitness; in this way, the most 'suitable' ones mate, producing offspring, which constitute the next generation (how many parents mate and how many offspring are produced is usually a parameter of the programme. For example, it is very popular to use so-called *roulette wheel sampling* among the parents, who would then mate with a probability proportional to their fitness). By doing so, the genotypes of the best parents are mixed, gradually producing generations that are more and more adapted to the environment. The mating method is a *crossover* between genotypes, in analogy to biology, and again there are many methods, each relevant to the problem itself. Finally, a specified number of offspring, before the selection process begins again, are 'mutated', i.e. the genome of a hypothetical offspring acquires a random characteristic (always in the space of possible genomes) not present in the parents.
So these steps, evaluation according to the fitness function, selection, mating and mutation, are repeated a specified number of times, which gives the population the opportunity to exhaustively explore the parameter space (genotypes).

2 The Problem

In a territory (a two-dimensional grid with periodic boundary conditions) live a number of individuals. In the same territory, a certain number of food and poison are placed in random locations each generation, and individuals can move one square at a time. The move that a certain individual can make depends on the "configuration" around them, and as there is food and poison in the territory, there are 3^4 possible configurations: empty,empty,empty,empty,food,food,food,empty,food,food,food,poison,empty,food,food etc.. Each configuration therefore corresponds to a move, which is encoded in the individual's genome. Every time the individual moves he loses one point of energy, he also loses another if he falls on poison (total energy loss -2) and instead gains one if he falls on food (total energy gain 0). The first generation has a fixed energy value for each entity, and the generation ends when all individuals have made a number of moves.
Individuals that reach 0 energy are eliminated, and once all individuals have moved on, provided they have not all been eliminated, the best individuals are selected who, reproducing with probability proportional to their fitness, replace the entire population. Each offspring has the average of its parents' fitness, rounded up if necessary.
The 'best' individual will then be the one who learns to move towards food and avoid poison. The aim is to write a genetic algorithm in Python that encodes this problem, also equipped with a graphical interface.

2.1 The implementation

The programme is divided into two main classes. The first is the `Environment` class, a grid of user-specified size with periodic boundary conditions, within which the positions of a number (still as input to the programme) of poison and food are stored. A fundamental concept is that of the 'configuration' of a given point in the environment, i.e. the presence or absence of food (represented by a 1) and poison (represented by a -1) at the points at the top, right, bottom and left of the point. Some examples might be:

- Food,empty,food,venom → `[1,0,1,-1]`
- Food,food,food,venom → `[1,1,1,-1]`
- empty,empty,empty,venom → `[0,0,0,-1]`

where 'empty' means a box in which neither food nor poison appears; as mentioned at the beginning, the possible configurations will be 3^4. This concept is important precisely for the second main class `Individual`. An individual is an array of possible moves, each corresponding to a certain configuration, in the following sense: in an array of the class there are all possible configurations, and the individual list has (corresponding to the indices of this latter array) within it the moves, i.e. one of 'u', 'r', 'd', 'l' (up,right,down,left). Each individual also has its own energy, which varies over time as explained in the previous section. Fitness is the energy of the individual.
So the programme starts by initially creating an environment with randomly distributed food and poison, and then initialises a population (the first generation) of individuals, each randomly placed in the grid. The function `move_all()` then moves all individuals for a specified number of moves, always checking to remove food or poison if an individual is on it, and changing the energy of each individual accordingly (if someone has 0 energy, they are removed from the population). Then calculating the fitness of the population, the parents mate with a roulette sampling, interlacing their genome (list of moves) with a two-point crossover, and 1/3 of the offspring are mutated (with a certain probability), replacing one letter of their genome.
These steps are repeated a number of generations fixed at the beginning. In case the individuals all die, the programme ends.The input parameters are thus: the number of individuals in each generation, the probability of mutation, the number of generations, the number of food and poison, the size of the environment and the initial (first generation) energy of the individuals.
The graphical interface was generated with the `AnnotationBbox` package of `Matplotlib`. Individuals

are represented by a Mario figurine, poison by a banana peel and food by fughi figures (as in the original game):

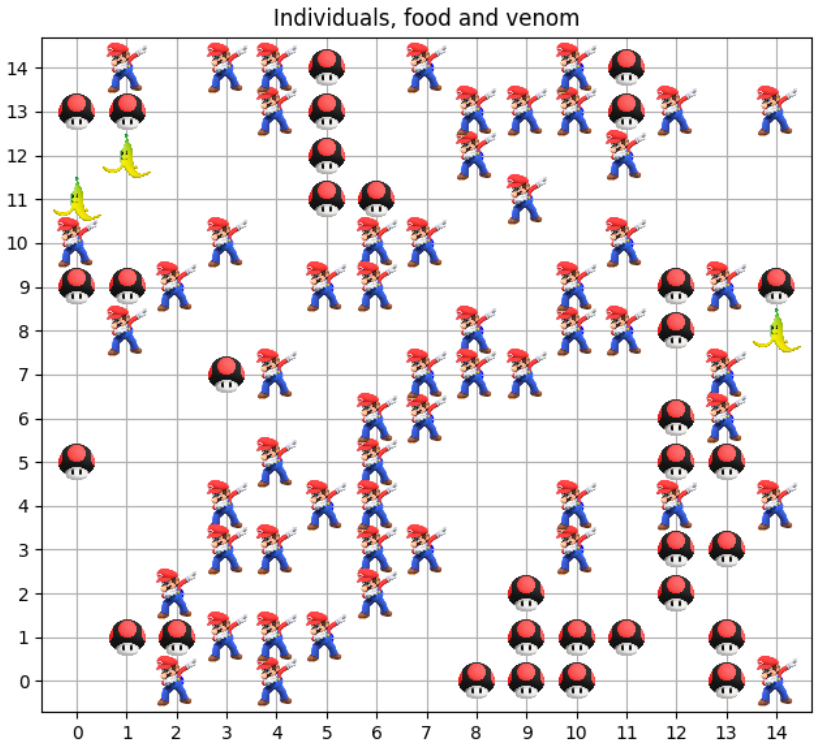


Figure 1: Graphic interface of the programme. Mario is the representation of the individual, the banana peel is the venom and the mushroom is the food. The dimensions of the grid are in this case 15×15

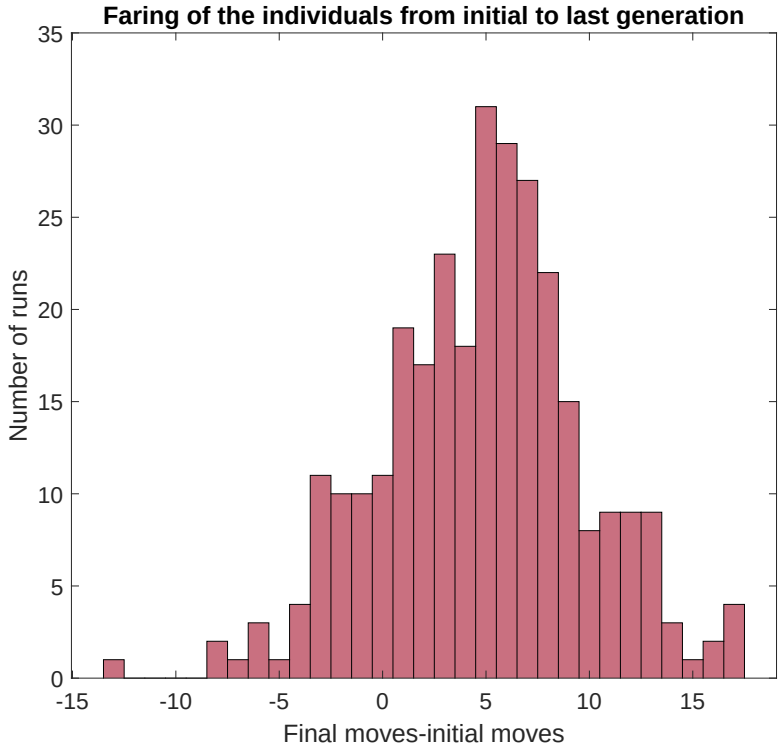
2.2 Results

The goal of the genetic algorithm is for the best individual in the last generation to be able to move towards the food and avoid the poison, in case they are around it. We can exploit this very fact to try to understand whether generations are learning or not: we consider all the configurations with only one 1 and we make sure that the individual's move is to go right towards the 1, then we do the same thing when there are two 1, and in this case the plausible moves will be two; instead, the configurations with three 1 were not used, because the moves could be 3 and therefore the probability that an individual has the correct move (regardless of having learned or not) is too high. The same reasoning was made with the configurations with three -1 and one 0 (the individual must move towards the 0, avoiding the poison), and the moves with two -1 and two 0; similarly to the previous case, the configurations with only one -1 were not considered. Of course, configurations with all numbers equal are also discarded: in that case any move is allowed. Some examples clarify the ideas:

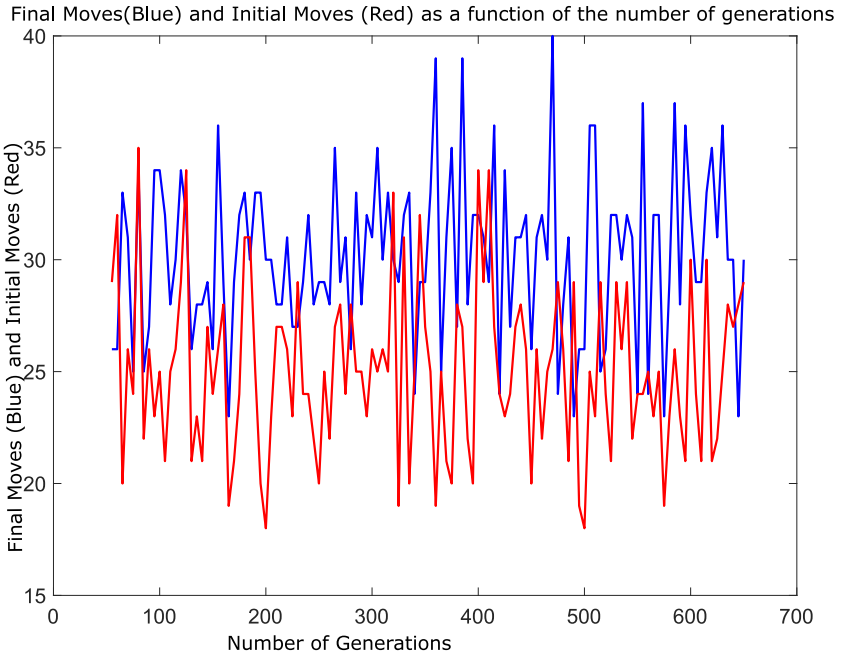
- Configuration `[1,0,0,0]` needs to correspond to move "u"
- Configuration `[1,1,0,0]` can correspond to "u" o "r"
- Configuration `[-1,-1,-1,0]` needs to correspond to move "l"
- Configuration `[-1,-1,1,0]` needs to correspond to move "d"

In total the configurations considered are 66 out of $3^4 = 81$. The perfect individual at the last generation would then have identified 66 moves out of 66.

We then introduce a function `check_individual()` that checks, at the end of each generation, how many correct moves the best individual has in its genome. The results are as follows:



(a)



(b)

Figure 2

As for Figure 2(a), we ran the programme 300 times with the following parameters:

```
Number of individuals: 60
Mutation probability: 0.3.
Number of generations: 420.
Number of Foods: 150.
Number of venom: 10
Initial Energy: 10.
Environment Size: 15 x 15.
```

and we calculated the difference between the number of moves detected by the best individual in the last generation and those in the first generation. In order to determine whether individuals are actually learning, one only has to check this difference, because the initial (also random) and final number of moves are irrelevant. It can therefore be seen from this graph that in most runs there has been an improvement (the graph is shifted to the right of 0).
In Figure 2(b), on the other hand, the programme was run by varying the number of generations from 55 to 650 with a step size of 5 and with the previous parameters.
Since it was only concluded in Figure 2(a) that individuals learn over generations, to give a quantitative estimate of how much they are actually learning, the number of detected moves from the best individual to the last generation in blue and the initial generation in red were plotted in Figure 2(b). It can be seen from the graph that as the generations increase, the number of detected moves does not necessarily increase, the only exception being the growth of the blue curve from 55 to $\sim$ 300 generations. Furthermore, it can be seen that the maximum number of detected moves was 40 out of 66! From this it can be deduced that the genetic algorithm is quite random (it depends a lot on which individual is close to as much food as possible) such that the growth of learning is stopped. Finally, it is also instructive to check the range of food and poison for which individuals learn to survive. The results are:

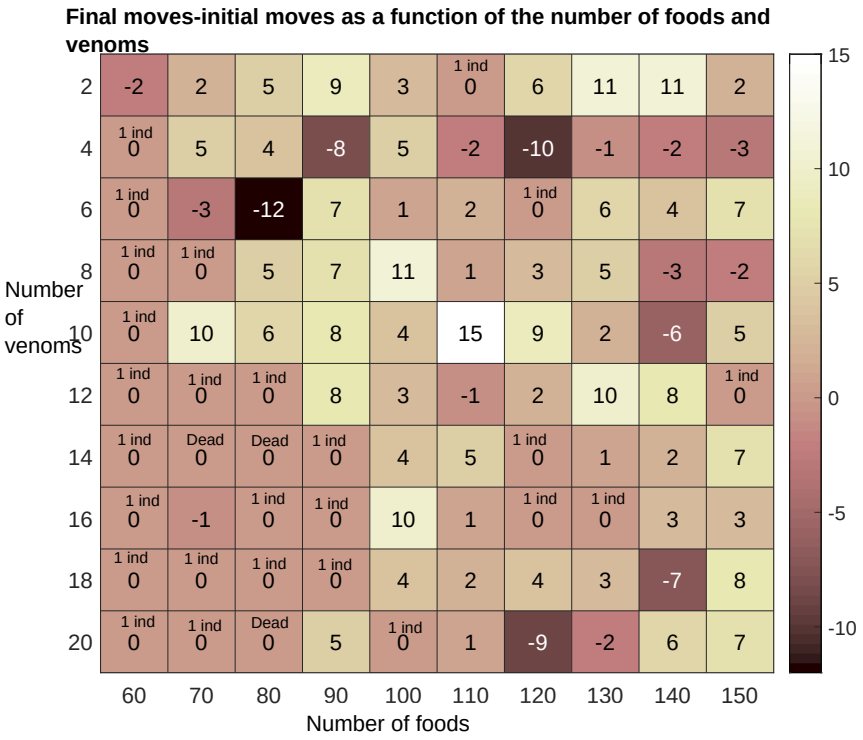


Figura 3

Figure 3 represents the same judgement metric used in Figure 1(a) but this time in a range of

finished food and poison, and the parameters are the same as those mentioned above. Where "1 ind" appears, it means that only one individual is left, while "Dead" means that the individuals have all been eliminated.
We note in particular that in the bottom left-hand corner, where there is little food (compared to the number of individuals) and a lot of poison, all but one individual is eliminated, so the programme is forced to stop. In the upper right, on the other hand, we find an environment rich in food but with little poison, so the poor performance must be attributed to the fact that the small number of poison does not allow individuals to learn to avoid it. For these reasons, the best parameters lie somewhere in between: enough food for individuals to procreate, but not too much to make the programme entirely random, and enough poison for individuals to learn but not so much that the population becomes extinct.

3 Conclusions

Individuals show an improvement in general, although not marked, as can be seen in Figure 2(a). On the other hand, this improvement does not appear to be quantitatively substantial, as at most 40 correct moves with about 500 generations, and increasing the number of generations does not lead to surpassing this maximum, as seen in Figure 2(b). This means that the algorithm is very susceptible to sudden changes in fitness, simply due to the fact that some individuals can be born close to a large amount of food despite having a non-performing genome.