Solving the N-Queen problem with Genetic Algorithms

Sergio Oyaga Iriarte

November 29, 2018

1 Introduction

In this project the N-Queen problem is solved using Genetic Algorithms (GAs). The problem consists of placing N queens in a NxN chessboard such that no queen attacks the rests. Although there are different techniques to solve this problem, we focus on GAs. GAs are useful solving optimisation problems. In our case, we need to optimize an N-Queen distribution on the chessboard, such that the number of checks is the smallest possible (zero in the best case). The GAs are based on genome evolution, which changes as the population reproduces in order to adapt to specific circumstances. Each individual of the population has his/her own genome, which determines his/her personal characteristics. These characteristics can be more or less suitable for a specific situation. The suitability of the genome affects the survival probability of the individual. The best individuals have more probabilities to pass their genome to their children.

Although nature does this evolution in a very complex way, the idea is no that complex and can be computationally implemented. We only need to specify some features:

1. The optimisation problem: Minimize the number of checks between queens. This is the fitness function that the genome minimizes.
2. The type of genome crossover:
   - Order Crossover.
   - Cycle Crossover.
3. The genome mutation: The mutation can be stronger or weaker, depending on the number of interchanged gene positions, e.g. two gene positions is weak, four gene positions is strong.
4. The specific parameters of the problem:
   - Population Size (size).
   - Number of Queens (n).
   - Probabilities of mutation(Pmut1 & Pmut2).

This structure of population, individuals, relations between individuals, mutations, generations, and etc, suggests the use of a programming object oriented structure. In this project, we decided to use a pure object oriented program like Java to solve the N-Queen using GAs.

2 Class structure and genetic methods

We first need to introduce the program structure, which is displayed in figure 1.

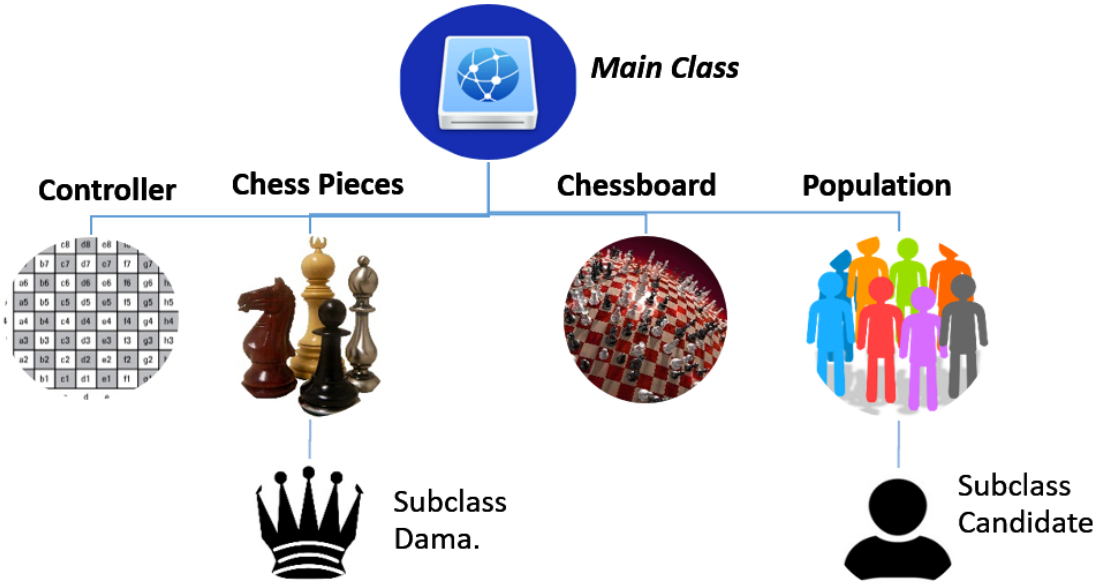


Figure 1: *Program structure*

In Figure 1, we see that the Population class contains the subclass Candidate, which refers to an individual in the population. The class and subclass are both responsible of the evolution, and allow the implementation of the GA. At the same level of the Population class, we find the class needed for the graphical representation (Chessboard), the class which stores the behaviour of the pieces (Chess Pieces) and the class responsible of managing the pieces and the graphical representation (Controller). Above these classes, there is the Main Class, where the main function is run. Here, we instance our objects and make the recursion to evolve the population.

Once the structure of the program is understood, we focus on the Population and Candidate class, which are shown in figure 2.

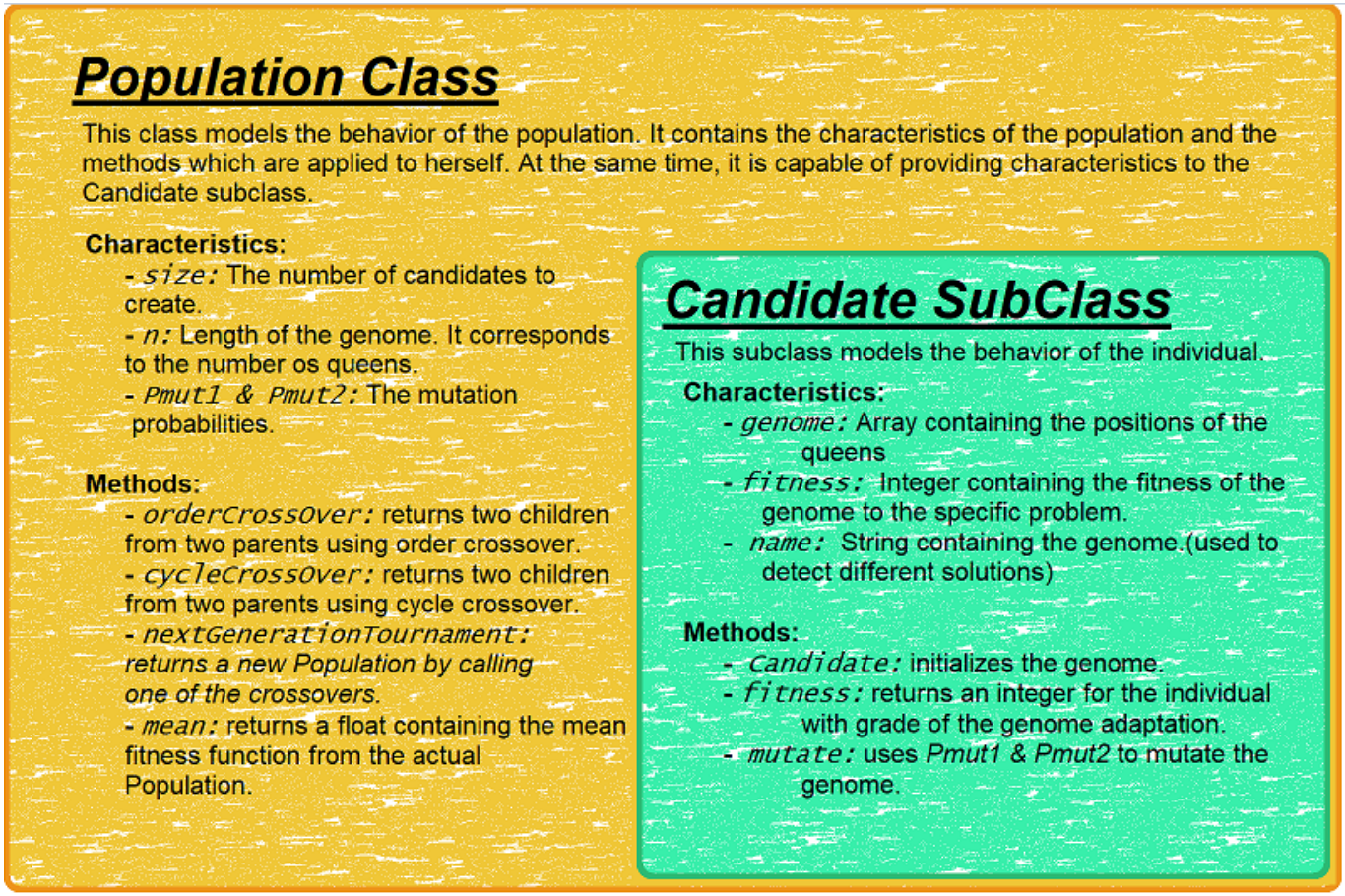


Figure 2: *Population and Candidate class structure specifying the most important characteristics and functions.*

The motivation of the structure presented in figure 2 is to mimic the natural process of gene selection. The genome, the fitness and the mutation method are particular of the individual. The population size, the genome length, the crossover and next generation methods are common to the whole population. Now, we focus on the specific methods used in the program.

2.1 Candidate methods

2.1.1 Candidate

The candidate method initialises a new individual by giving some initial values to his/her characteristics.

- The genome is the shuffle of an array going from $1 \rightarrow n$.
- The fitness is an integer obtained by calling the fitness function, described below, for this shuffled genome.
- The name consists of a String containing the genome characters.

2.1.2 Mutation

Mutation allows a specific individual to change randomly some features of his/her genome. In our program the mutation consists of interchanging gene positions. We stablish particular thresholds under which the mutation happens. These thresholds are $Pmut1$ and $Pmut2$ which interchange two or four genes, respectively. A graphical representation of this process can be found in figure 3.

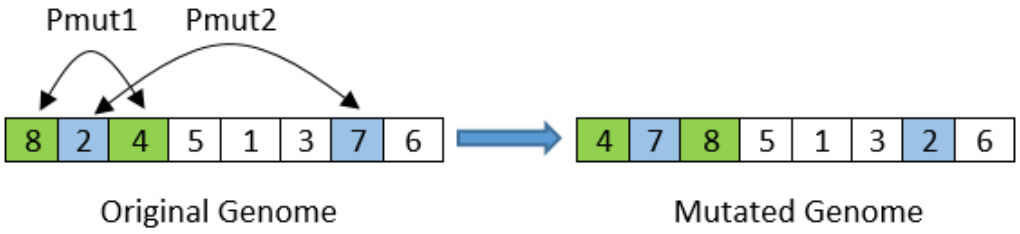


Figure 3: *Mutation process.*

In order to make the mutation as realistic as possible, we decide if the muation heppens randomly. We pick two pseudo-random normally distributed numbers. Each pseudo-random number is compared with each threshold, $Pmut1$ and $Pmut2$. If the value is lower than the threshold we randomly select the genes to be interchanged, picking again two pseudo-random normally distributed numbers.

2.1.3 Fitness

The fitness function consists of counting the number of checks between all the queens in the chessboard. This is performed by looking at the genome. We check every row, column, and diagonal for each queen. This can be simplified by considering that our genome cannot mix rows, because, by construction, each position of the genome is already the row position. The same happens with the columns, the genome cannot repeat numbers (we shuffled an $1 \rightarrow n$ array), which are the column position. So finally, we have an array where the position is the row and the column number. We only need to analyse the diagonals. This can be easily conducted since forward diagonals have the same index subtraction ($FW = Row_{idx} - Col_{idx}$), and backward diagonals have the same complementary subtraction ($BW = N_{queens} - Row_{idx} - Col_{idx}$). These changes are basically a coordinate transformations. The processes is shown in figure 4.

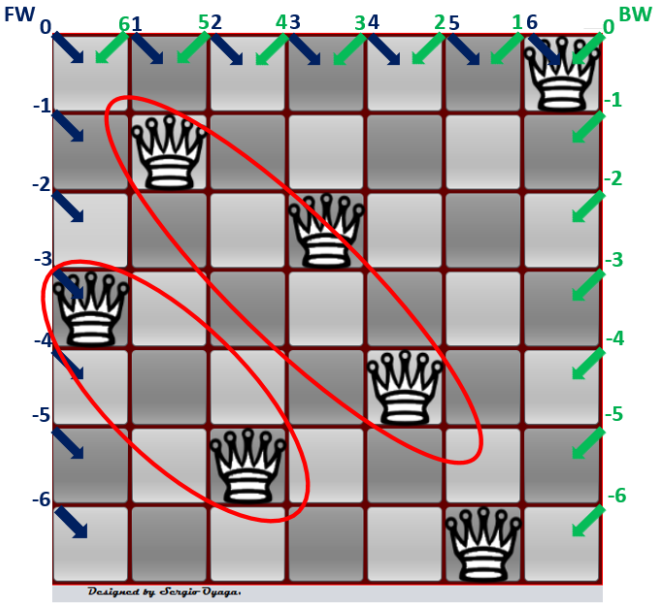


Figure 4: *Graphical representation of how Fitness function checks the diagonals. Forward in blue and backward in green.*

Figure 5 represents the same process for the genome.

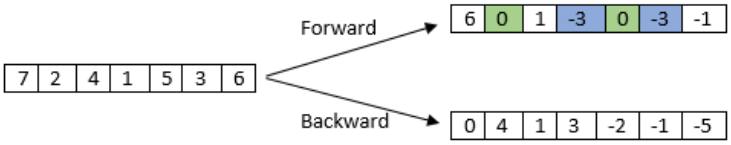


Figure 5: *Diagonal check in the genome.*

With this information, the fitness function can weigh the genome adaptation to the problem.

2.2 Population methods

2.2.1 Order crossover

The crossover is a technique where two individuals from a specific population (parents) generate the individuals for the next population (children). The crossover takes into account the nature of the problem, in this case the location of the queens in the chessboard. For the genome, this allocation is the same as to consider the order and the position in which the numbers appear in the genome.

The order crossover perform this process in the following way: it takes one block from one parent and passes it to the children without modifications (same place and same order). The remaining children's genome is filled with the genes from the second parent (the ones that have not yet been passed from the first parent). Figure 6 displays this process.

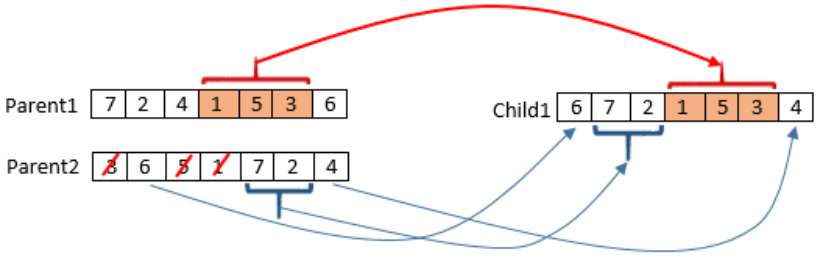


Figure 6: *Order crossover for the first child. The second child can be obtained by doing the same but changing the parent order.*

In order to make the crossover process as fair as possible, we select where the block starts and ends by throwing two pseudo-random uniformly distributed numbers.

2.2.2 Cycle crossover

The cycle crossover follows a different idea, the block to pass starts with one parent's random gene. The next gene to pass is given by the second parents gene which is in the same position as the gene passed before, but in the first parent position. Doing this, we define a closed cycle in the first parents genome. The children genome is completed by repetion of this process. Figure 7 depicts the described process.

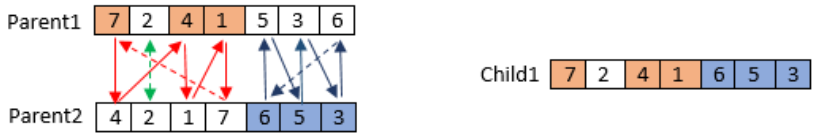


Figure 7: *Cycle crossover. In red, a cycle block starting with Parent1. In blue, a second cycle block from Parent2. The green cycle it is the same for both.*

In order to make the crossover as fair as possible, we choose the parent and the gene to start the

cycle by throwing two pseudo-random uniformly distributed numbers.

2.2.3 Next generation by tournament

The creation of a new population via crossover and mutation of the new individuals makes this technique suitable for a optimization problem. In nature, after each generation, the population fits better a specific problem. It can happen that the new individuals are worst than the previous ones, but the general tendency is to improve.

As shown in nature, the best individuals are the ones passing their genome to their children. This can achieved in several ways, but we decided to use the well-know technique of tournament.

This technique selects a random portion of the population and chooses the best individual as first parent. By repetion, the second parent can be obtained. Together, both parents will generate two children via crossover and mutation. Here, choosing the best means choosing the individual with better fitness value. Figure 8 displays a graphical description of the tournament.

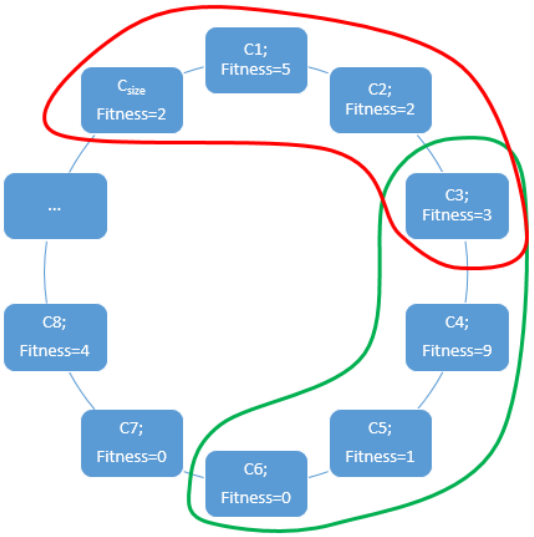
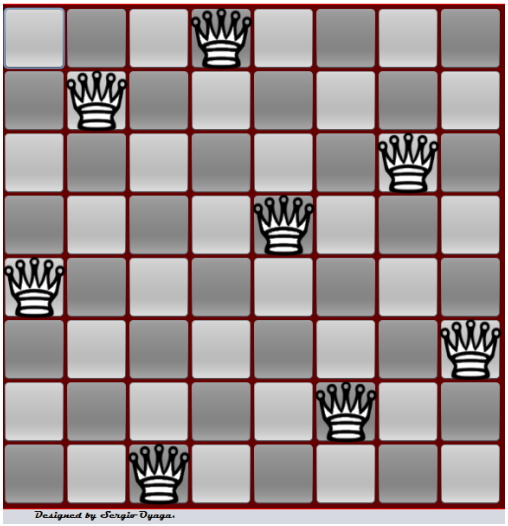


Figure 8: *Graphical description of how the Tournament works. The parents will be C2 and C6.*

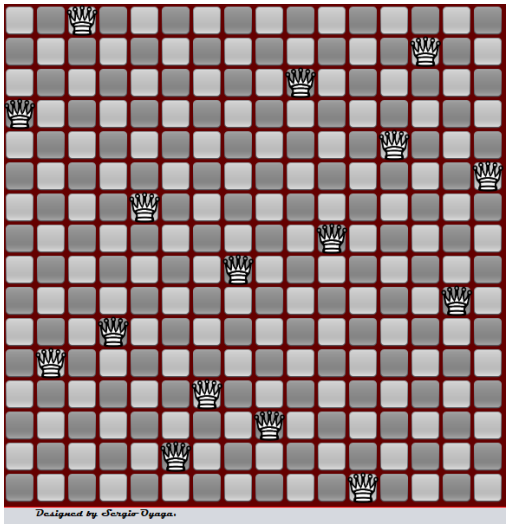
2.2.4 Mean

This function computes the mean of the population fitness by summing all the fitnesses for each individual in the population and dividing it by the population size.
This kind of parameters are usually use to study the convergence of the genome. If the problem have lots of close global minima, our genome will seem to converge while this is not true. For our specific objective, this is not a problem any more. When the mean is close to zero, the found local minima is highly replicated along the population. In the results, we extent this explanation further.

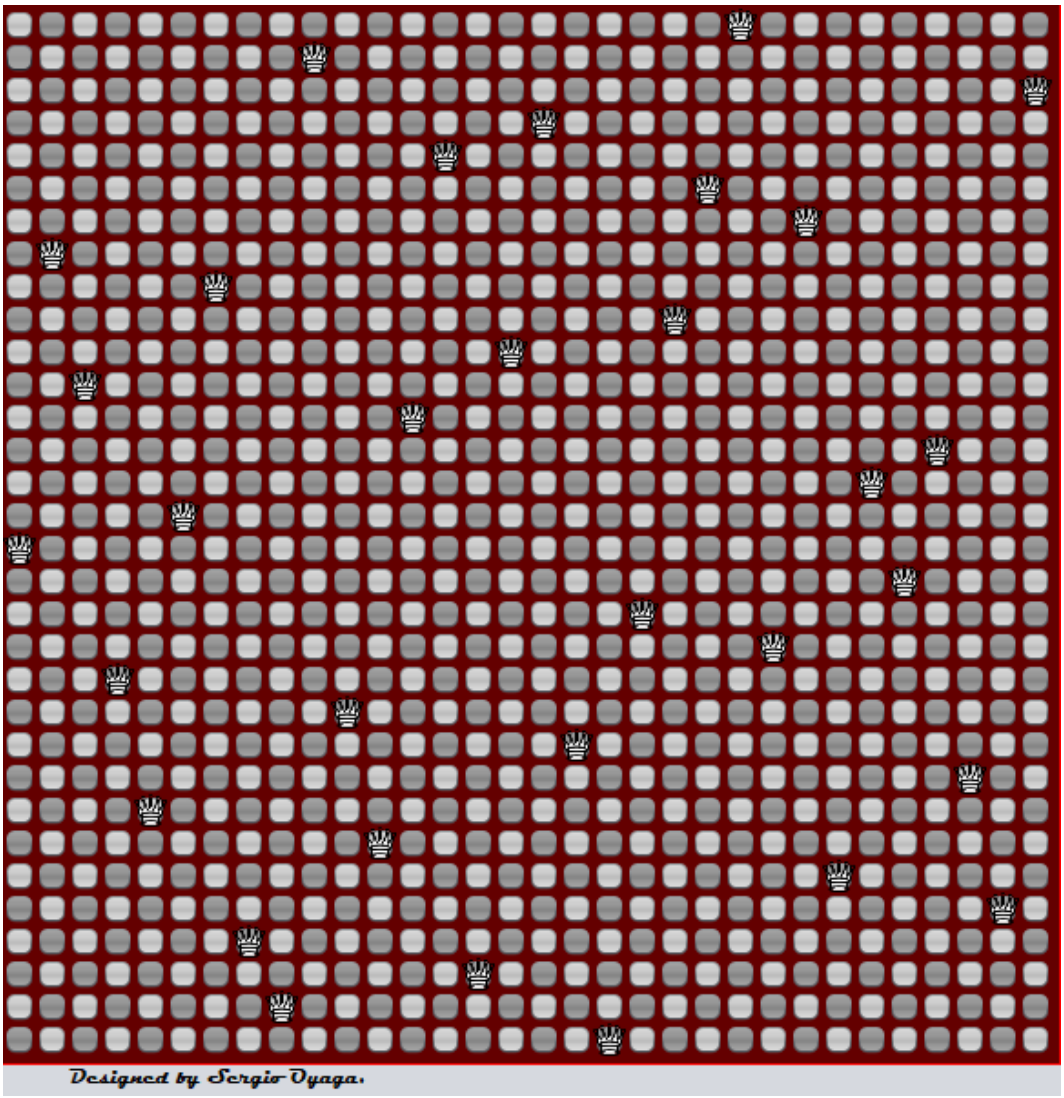
3 Results



(a) *Solution for 8 queens using order crossover.*



(b) *Solution for 16 queens using cycle crossover.*



(c) *Solution for 32 queens using cycle crossover and elitism.*

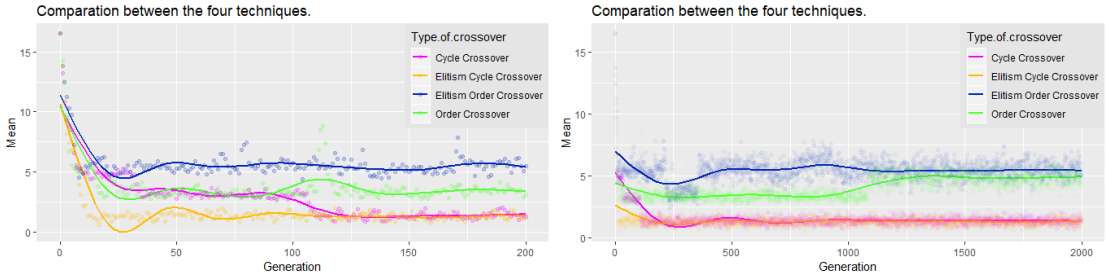
Figure 9: *Specific solutions for the N-queen problem.*

In Figure 9, we display some solutions obtained using the techniques described in the methods section. In addition, we included some kind of elitism, shown in figure 9c. This is performed by allowing the population to directly pass some of the best individuals to the next generation. The elitism level is chosen to be low, i.e. only the best two individuals pass.

We implemented four different methods to do the crossover.

1. Order Crossover.
2. Cycle Crossover.
3. Elitism Order Crossover.
4. Elitism Cycle Crossover.

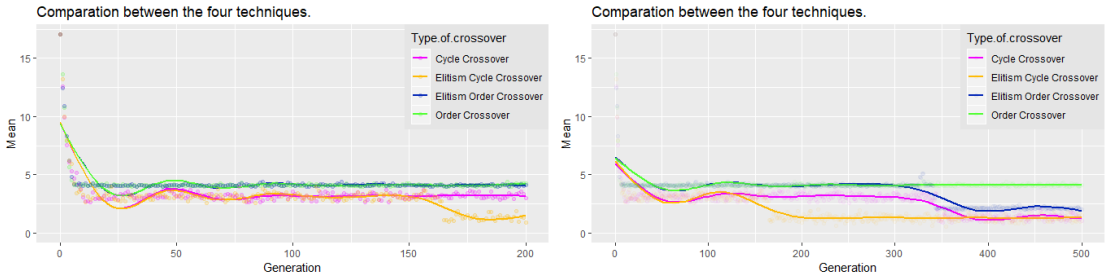
In order to compare each method, we represent graphically the population fitness mean. The representation can be seen in figure 10 (The global tendency is shown by the smooths). With the aim of making the comparison as fair as possible, we took the same parameters for every case.



(a) *Convergence for the 200 first generation.* (b) *Convergence for the 2000 first generation.*

Figure 10: *Population's size=100, Number of queens n=16, Pmut1=0.2 and Pmut2=0.05.*

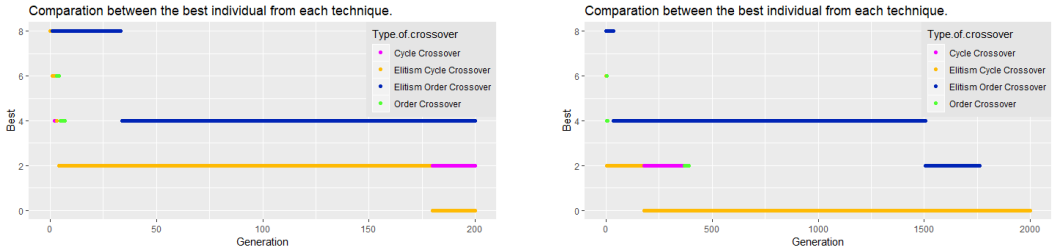
Although it seems like the ordered crossover did not converge, the genome has a unique solution, and the population is around it. However, this high mean value for the fitness is not good, and we will need to decrease the mutation probabilities. The change in the mutation's probabilities makes our algorithm less exploratory, which implies loosing capability of void local minima. We need to vary the size and the probability values, as displayed in figure 11.



(a) *Convergence for the 200 first generations.* (b) *Convergence for the 500 first generations.*

Figure 11: *Population's size=150,* $Pmut1_{order} = 0.2$, *and* $Pmut2_{order} = 0.05$.

In Figure 11, it seems that the population converged. The question is to confirm if it converged to a global minimum. This can be easily checked by plotting the best individual from each generation, as shown in figure 12.



(a) *Best individual fitness.* (b) *Best individual fitness.*

Figure 12: *Fitness of the best individual in each generation.*

In Figure 12, we observe when the first genome, that is a global minima, appears in the population. We also show that all techniques have enough mutation probabilities to reach the result.

We are left to check if the genome has converged around this minima or not. Hence, we plot histograms of the last populations genome, which are shown in figure 13.

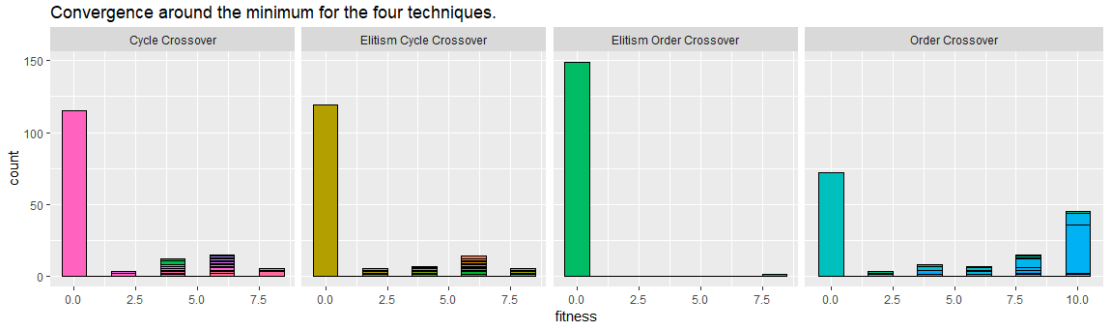


Figure 13: *Convergence around the minimum for all the techniques.*

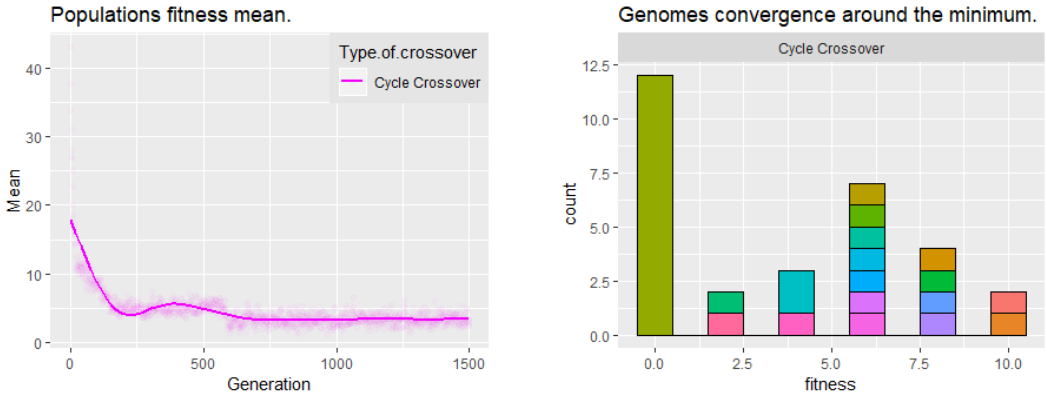
In Figure 13, an histogram of names from the last population is shown. Notice that the colour of each histogram box represents a different genome name. More than one colour per box can be found, which means that there are many genomes with the same fitness. However, we are interested in what happens in the zeroth box. This box is the largest in each subgraph, and it is characterised by a unique colour. Hence, we conclude that the genome has converged around one global minimum. The remaining boxes are due to bad crossovers or mutations.

4 Discussion

We considered the genome short enough to stablish only 4 genes interchanges at most. We believe that this is not the natural process, but however, the mutation level should remain low for these genome lengths. The mutation can be easily generalised to a random number of interchanges considering the genome length.

We implemented two crossover methods (with an elitist variant each one). We have shown their functionality. In the graphs presented, we proved that the fastest to converge is the Cycle Crossover method. Moreover, this method was better and easier to achieve. In order to equate the order crossover with the cycle crossover, we had to decrease the mutation probabilities of the order crossover, what made the algorithm less exploratory. This reinforces the fact that we did not find any global minima.

In addition, we studied the convergence of the genome around one global minimum. We have shown that the Elitism Order Crossover was the one that converged better. This could be due to the lowness of the mutation and the elitism makes the algorithm very exploitatory, and promotes no change of the genome. Therefore, we consider the cycle algorithms better in terms of convergence, because even if it has a not so low mutation probabilities, it can achieve very high convergence levels.
To conclude, it seems necessary to choose the most suitable method for this specific problem. We propose the Cycle Crossover as the best method, since the mutation probabilities can be considerably high until the fitness starts to diverge. In addition, avoiding elitism implies that the algorithm can void local minima easier. This encouraged us to decrease the population size while increasing the mutation probabilities, and therefore, to expect less generations until the genomes converges. Figure 14 displays a graphical example.



(a) *Populations fitness mean.* (b) *Genome convergence.*

Figure 14: *40-Queen problem convergence.*

The performance has been studied for:

- $n_{queens} = 40$
- $size_{population} = 50$
- $Pmut1 = 0.4$ & $Pmut2 = 0.1$

We see that the populations fitness mean converged for the 800^{th} generation and that the genome also converged to a specific one.

5 Conclusions

In conclusion, we would like to remark the utility of Genetic Algorithms for solving optimization problems. The strength of GA relies on the efficiency of searching for a solution in a high dimensional problem. GAs are nowadays one of the trending techniques in order to solve optimisation problems, and we could see this fact solving the N-queen problem.
We also proved how an object oriented program like Java is suitable for representing the idea of population, individual, crossovers, etc. We strongly believe that this programming paradigm allows the implementation of methods and characteristics, which reflect the real natural behaviour.