

**SOLVING THE N QUEENS PROBLEM USING GENETIC ALGORITHMS**

**Computational Intelligence for Optimization**

**2023/2024**

**Group Name – Queen**

20230525 Alícia Pinho Santos

20230522 Jaime Simões

20230983 José Pedro Cruz Fernandes

20222125 Nuno Sousa

**Problem**

The challenge of the **N Queens Problem** is to find a solution so that n queens fit without conflicting with each other in an n-by-n chessboard.

This means that a solution cannot have two queens in the same row, column or diagonal. If we think for a second, we can see that for n=1 there is a single obvious solution and for n equaling 2 or 3 there is no possible solution. However, for any integer n bigger than 3, there is at least one possible combination of positions that meets the criteria. With the increase of n, the number of possible combinations for all the positions increases exponentially, requiring the optimization process to be efficient to solve the problem. When the algorithm finds one of the solutions, the problem is considered as solved.

The developed algorithm generates an initial population of possible random solutions, and different possibilities for selection, crossover and mutation functions will execute over the population over a number of iterations (generations) until it finds a solution (or reaches a specified maximum of generations). The solutions coming from different implementations of these functions will then be compared using statistical tests in order to find what is the “best” combination of these functions for this specific problem. We decided not to use the given library, as we wanted to develop everything from scratch, so that fully understanding the implementation would help us envision potential improvements. In this report, the implementation structure will be explained, all the different functions and parts of the code will be explained, and the statistical tests and results got will be analyzed.

**Representation**

The first thing that had to be decided is how the problem would be represented in code (what would the format of individuals be, etc.). We decided to not represent the board. Since we only need the work on the positions of the queen pieces, we decided each board would only be n tuples of 2 integers. Each tuple contains the coordinates of a position and the list of n tuple is the whole n positions. This brought forward at least three things that should be considered:

* The order within an individual is irrelevant;
* Through mutation and crossover functions we need to verify that the resulting individual had no duplicated positions, and had to have size n;
* And maybe the biggest thing which is that this structure lead us to work with two random factors, the x and y coordinates. This is important because an alternative could have to represent an individual as a simple list of integer, where the index would be the other coordinate, and that would have made things significantly easier.

**Initialization and Structure**

The algorithm begins by initializing a population of random individuals, that is, n positions of random values x and y up to n. The size of this population is specified as a parameter beforehand and remains constant over the generations. Afterwards, fitness is evaluated for every individual. As we will see later, the fitness function is made so that that this challenge, in terms of fitness becomes a minimization problem. For this reason, the initial fitness value is initially set as absurdly high vale, so that any first random position has a better fitness than that.

**Fitness:**

The fitness value is what defines the quality an individual (a set of positions). It is defined as the number of conflicts present in that board, so a lower number means it’s a better solution, turning it into a minimization problem for this value. If the fitness reaches 0, it means there are no conflicts and therefore algorithm has reached a solution.

**Selection Functions:**

After evaluating fitness, individuals are selected to form a new population. Selection is based on fitness, with better solutions having a higher chance of being chosen. The algorithm supports two selection methods:

* **Tournament Selection**: Two individuals are randomly chosen from the entire population and the one with the lower fitness is kept.
* **Roulette Wheel Selection**: Individuals are selected with a probably related to their fitness. So lower fitness values means that individual has a higher chance of being selected, by the same proportion of the fitness in relation to the fitness of the other individuals.

This selection function will be used in there moments: firstly, the whole population is selected on and substituted, which means that from the beginning of a generation there is a tendency so that the worst individuals are disregarded and the best individuals are kept; secondly, the selection function is used to decided which pairs of individuals are going to have a crossover applied on; lastly, the selection function will be used to select the individuals which will undergo a mutation.

**Crossover Functions:**

The individuals selected and paired and produce offspring through the crossover method, where it combines parts from both parents to create new individuals for the population. The crossovers developed are:

* **crossHalf**: takes the sorted positions of both parents and distributes them alternately between the offspings. It distributes the “genes” from both parents but if the parents have similar positions the offspring will be more equal to the parents.
* **crossSinglePoint**: selects randomly a point and then the genes from the start to this point come from a parent with the rest coming from the other parent. For the other offspring is the symmetrical.
* **crossCycle**: starting from the first position, it follows a cycle alternating between the parents by means of the output of each instance until it returns to the starting position. Then it fills the remaining positions with the parent not used as reference for that specific run.
* **crossGeometricSemantic**: this method uses random weights to combine the coordinates from both parents. Because it can produce non integers, the values are rounded to the nearest integer for the coordinate value.

**Mutation Function:**

Taking again the selection function to choose individuals to apply mutation on, the variability of the population is increased. Two possibilities exist for this step:

* **Position with conflict for random**: picks a position where there is a conflict and replaces the position of the queen by a random new position.
* **Shift Coordinate on Position with Conflict:** also targets the positions where there are conflicts. But this time it shifts just one of the coordinates (either row or column). And this times it applies this to all present conflicts.

**Elitism:**

There is the option to have elitism on or off. In the case where it is on, for every generation it saves the best 10% of individuals with the best fitness values and copies in the next generation, so that these are not lost in the crossover / mutation process. For the population to maintain the same size, it only creates new elements until that threshold is reached. So, it will create less individuals then if elitism is set to False. Alongside the different options for the functions explained above, this is also a parameter that will be optimized when running for solutions.

**Looking for the best model:**

Because there are many parameters and function options for several steps of the algorithm, we have to search independently in different sections of the possible search space. Now this is not the optimal or most accurate way to obtain the absolute best model but given the computational complexity and time we have for this project, we find this to be a good approximation. Each dot corresponding to each board size is the average time of 30 runs for that specific combination. To make the graphs possible to read we will join the points of different board sizes with lines, but this of course is not an accurate representation of the problem.

First, we look for the best combination of functions for crossover, mutation, and selection (using population size of 150). Then we optimize for the size of the initial population and if using elitism is beneficial or not. After this we have the “best model” and will run it more thoroughly to get the final results. We then analyze how the initial positions that are set randomly impact these results.

**Finding the best combination of functions:**

Because there are many different types of functions for “selection”, “mutation” and “crossover”, and some of them take a long time to run. To compare them all we will first run to a small (n) so that we get an idea of what functions are better or worse. Then for the “final” comparison we run only these best combinations to a bigger board size. The most accurate way would be to just directly compare all possible combinations, but as this is not computationally feasible, we decided to take this approach. When seeing the outputs of the first runs it was clear the crossover function “crossCycle” was performing poorly against the others, so it was removed from this analysis. In figure (a) we have the results until n=6. It is notable that the “tournament selection” selection has the lines bellow “Roulette Wheel Selection”, meaning it is performing better. We only use Tournament Selection from now onwards. For mutation, “crossGeometricSemantic” is performing the worse for both selection functions, so we will only keep the other two. Now we take this information and increase the size of the board to get a better estimate and determine what is the best combination. Looking at figure (b), we compare the combinations remaining. The best performance is obtained for “Tournament Selection”, “Position with conflict for random” and “crossSinglePoint”. We now take this combination and do tests with other parameters.

**Optimizing population size and elitism:**

We follow the same logic as before and run the best combination for different population sizes and elitism option to compare the results. When elitism is “True”, the fraction of the best population kept in order of best fitness is always 10%. Looking at the results (figure (c)), the best model changes depending on board size. Because for low sizes the solving speeds are fast but start to increase significantly for higher sizes, it is preferable to choose a model that is the best one for higher board sizes, as that is what will save the most amount of time in the long run. With that, population size of 100 and elitism being enabled seem to be the better option. Now that we have all the optimized parameters, we can run the “best” model.

**Results and associated fluctuations:**

Running until n=17, we can see that the growth in time is approximately exponential as would be expected (figure (d)). We can now do some analysis on how the initial positions, which are set randomly, affect the time it takes to solve the positions. For that we maintain all the parameters constant and obtain the distribution of the results for each run via boxplot. Using the parameters of the best model, in figure (e) we follow this logic for different board sizes. The horizontal line of 0 indicates the average for each instance. There are differences higher than 50%, which means the initial positions have a big impact on the results. The problem is seen across different board sizes. This might indicate that we should have used more than 30 runs for each point for the graphs above to get more consistent results and that the performance of the algorithm is dependent on a random factor, at least for these values of (n). This also helps explain the fact that in figure (d) from 16 to 17 there is a decrease in time, we also see an increase probably higher than expected from 15 to 16.

**Annex:**

Uma imagem com texto, captura de ecrã, file, diagrama

Descrição gerada automaticamente

Figure (a): First comparison to get the best combination of functions.

Uma imagem com texto, file, diagrama, Gráfico

Descrição gerada automaticamente

Figure (b): Finding the best combination of functions.

Uma imagem com texto, diagrama, file, Gráfico

Descrição gerada automaticamente

Figure (c): Different population sizes and elitism option.

Uma imagem com file, Gráfico, diagrama, texto

Descrição gerada automaticamente

Figure (d): Results for the best model.

Uma imagem com texto, diagrama, file, Esquema

Descrição gerada automaticamente

Figure (e): Variation of results on different runs.

*How did you design the fitness function? Did you try using different fitness functions, to see the impact on your GA?*

*Do different operators affect the convergence of your GA?*