CS 541 Artificial Intelligence

Programming Assignment 2: 8 Queens using Genetic Algorithm

- **Deepa Varghese**

The Implemented Genetic Algorithm:

Technique:

Individual of a population: Each individual in a population is an object of the class Individual containing the below three characteristics. In the __init__ function of the class, when a new object is generated, it's fitness is immediately calculated. Thus each individual contains the board configuration and it's corresponding fitness.

1) indiv – an array equivalent to the chess board with a particular configuration of the 8 queens on the board. Eg. [5, 8, 7, 4, 3, 2, 1, 6]
   A successful configuration would be – eg. [6, 8, 2, 4, 1, 7, 5, 3]
2) fitness – an integer with maximum value 28. The fitness of an individual is the number of pairs of nonattacking queens in the indiv. Therefore, for [5, 8, 7, 4, 3, 2, 1, 6] it is 19, and for the successful configuration [6, 8, 2, 4, 1, 7, 5, 3] it is 28 (max possible)
3) si – probability of selection. This is computated as individual's fitness / sum of fitness of individuals in the population

Initial Population: The Initial population is generated by creating new objects of the class Individual with a random ordering of numbers between 1 and 8. Each new object thus created is appended to the array population[]. Then the si value of each individual of the population is calculated and noted in each object individual in array population.

Selection of Parents: 2 parents are selected in each iteration by calling the roulette() function. This function implements the "fitness proportionate selection" (sometimes called "roulette wheel selection") as explained in class.

Crossover: The crossover point for each pair of parents is generated randomly between integers 0 and 7. At the crossover point, the 2 children are generated by combining the parent arrays, as explained in the text/slides.

Mutation: each child is sent as argument in a call to the mutation() function. This function generates a random number, and if it the number is below globally set parameter MutationPct, then the mutation occurs. At a random index of child array, the integer is replaced with a randomly generated integer between 1 and 8.

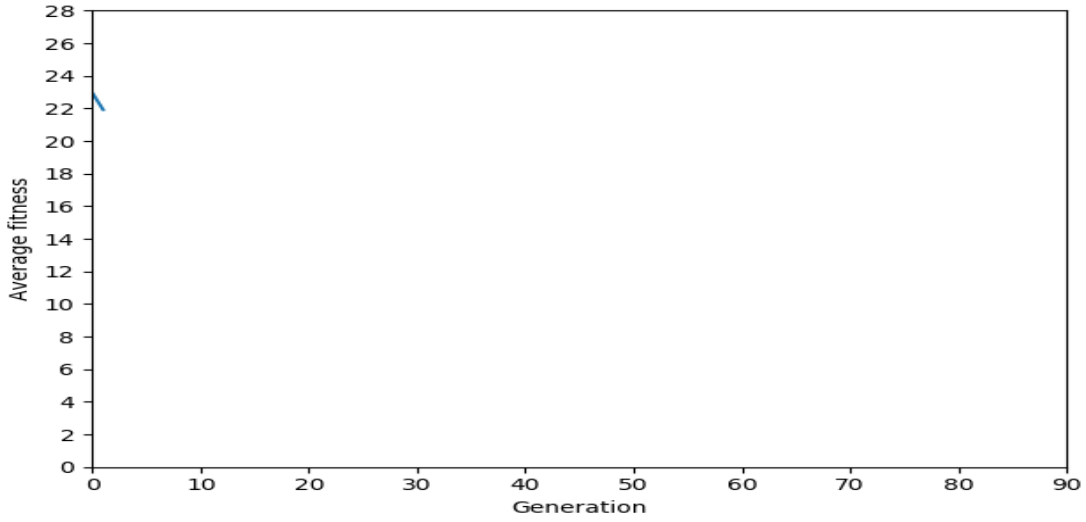
Findings/Conclusions

1) As the programs sometimes approached a local minima of 25 or 26, in the next few iterations the configuration would stir up again so that newer population's individuals' fitness was again in the range of 19-20 and such. Very low mutation probability led to programs finishing the while loop without finding a solution at all.
   Conclusion: randomly mutating the child individuals avoided local maximas, and reset the path
2) Either with a constant crossover point of 4 (midpoint) and a randomly generated the crossover point, the program performs sporadically and either method of determining the crossover point could yield similar or vastly different results.
   Conclusion: no specific option for generating the crossover point can be said to be better than the other.
3) With a population size of time, very rarely was a goal state reached. Even when the iteration number was NumIterations = 1000000 and PopulationSize = 10, the goal state could not be reached. But, with the PopulationSize = 500, the goal state has been reached (as observed) even with just 12 iterations.
   Conclusion: Higher population sizes tend to do better.

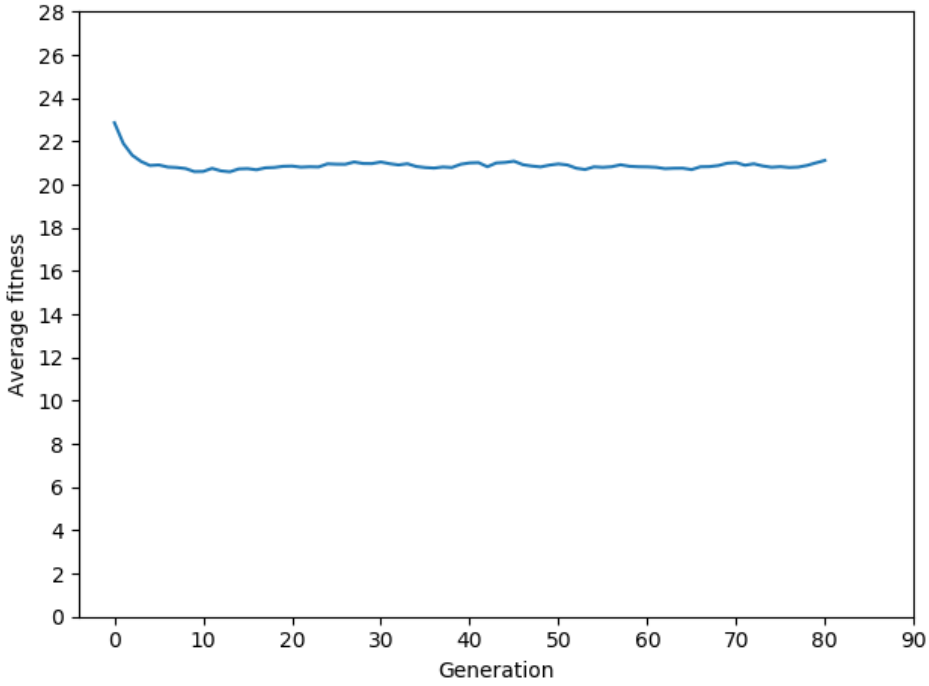
Plot:

Explicit examples whose generation's individuals are provided in the pdf are noted below. Also, for PopulationSize=10 and NumIterations=100 for which goal state was not reached before end of iterations, all generations' children are provided in pdf.

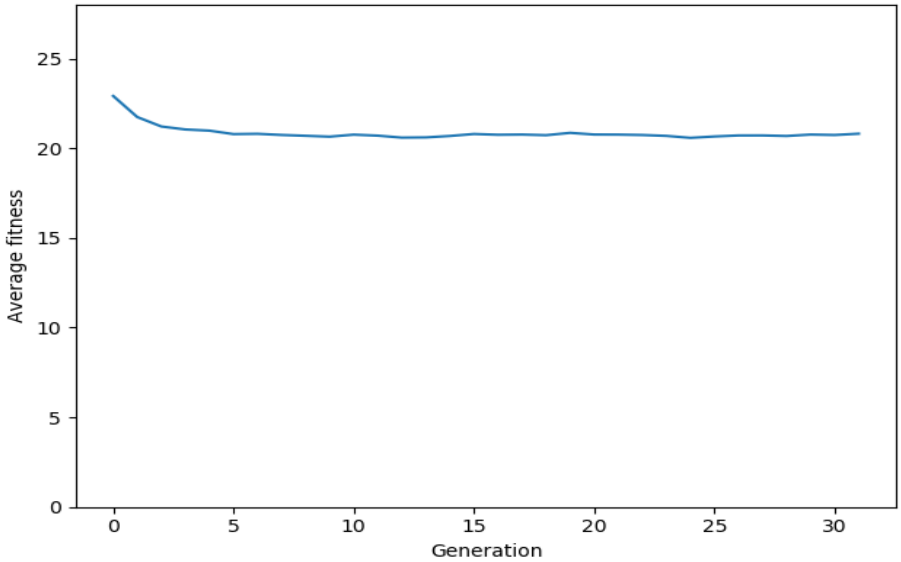
1) PopulationSize = 1000 NumIterations = 500
   Example 1:
   Solved in 1 Iteration
   Resolved: [5, 3, 1, 7, 2, 8, 6, 4]



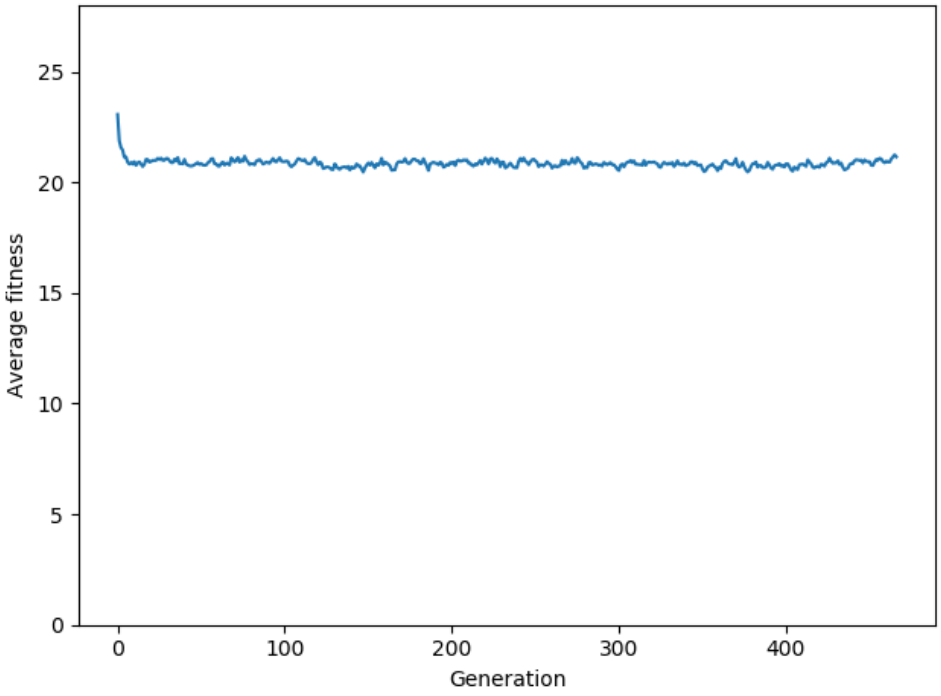
Example 2:
Solved in 80 Iterations
Resolved:
[7, 1, 3, 8, 6, 4, 2, 5]



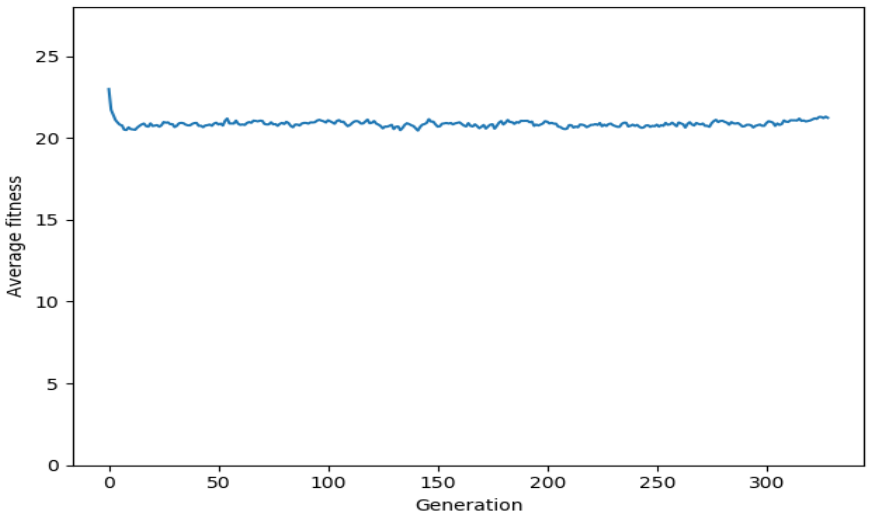
Example 3: (Initial and Final Generation individuals given in pdf)
Solved in 26 iterations
Resolved: [4, 8, 1, 3, 6, 2, 7, 5]



2) PopulationSize: 500 NumIterations: 500
Example 1: (Initial and Final Generation individuals given in pdf)
Solved in 466 Iterations
Resolved: [5, 2, 6, 1, 7, 4, 8, 3]



Example 2:
Solved in 328 iterations
Resolved: [6, 3, 1, 7, 5, 8, 2, 4]

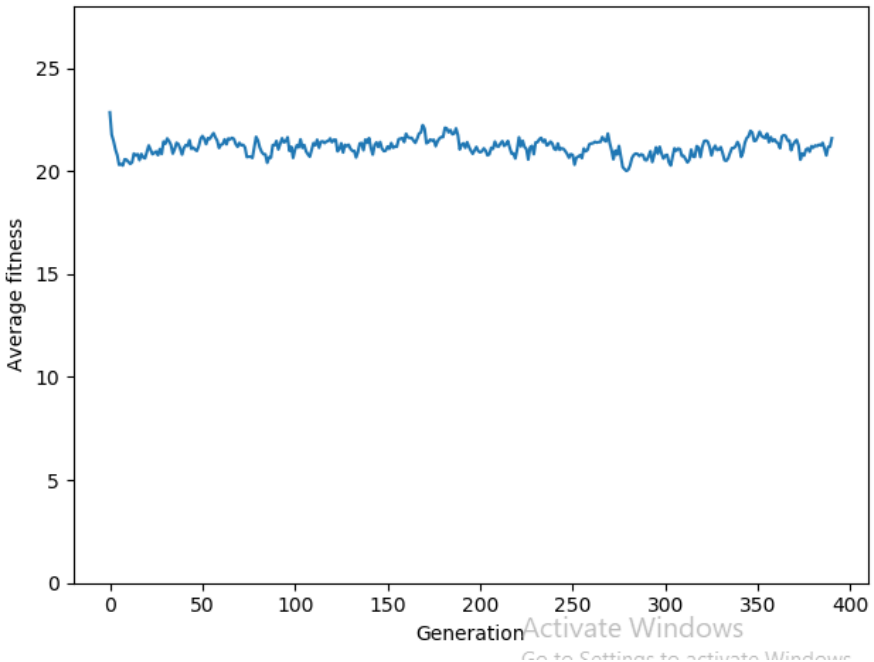


3) PopulationSize: 100 NumIterations: 500

Example 1:

Solved in 390 iterations

Resolved: [6, 2, 7, 1, 4, 8, 5, 3]



Example 2: (Initial and Final Generation individuals given in pdf)

Solved in 105 iterations

Resolved: [5, 7, 2, 6, 3, 1, 8, 4]