Genetic Algorithm: The Eight Queens Problem

Bingling Fu

Final Draft: April 14, 2018

**Problem**

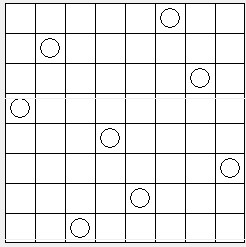
I created a genetic algorithm to find a solution to the Eight Queens Problem. The problem poses the situation: The Eight Queens question is a question on the background of chess: how can eight queens be placed on an 8×8 chess board so that no queen can directly eat other queens? In order to achieve this goal, neither of the two queens can be in the same horizontal, vertical or diagonal line.

**Design**

It can be seen as a gene. This code can naturally solve the constraint that only one queen can be placed in a row. If every element []xi of the array is not repeated, it can be regarded as an arrangement of 0 n-1. It is natural to ensure that each column has only one queen. Therefore, it is necessary to pay attention to the uniqueness of list in cross mutation and generation of individuals.

I use the code

46031752

****

*Genetic code:* Each base is an integer corresponding to a vertex. Each gene is an array of integers representing the order in which the vertexes are visited. While this may not be the most elegant way to represent the data, it simplifies mutation and crossing over and guarantees that every gene created will be a valid Hamiltonian cycle.

*Gene expression:* Each adjacent pair of bases corresponds to an edge, as well as the last and first bases. The edge weights are stored in a Bag data structure.

*Fitness function:* Since I am evaluating for the shortest path, the fitness function is the inverse of the net weight of the cycle. That is, f(g) = 1/w(g) where w(g) = Σ(edge weights). A cycle with a smaller weight will have a larger fitness value and vice versa.

*Mutation:* I am doing swap mutations of randomly selected bases. For example:

Parent: 1 | 7 | 2 | 4 | 5 | 3 | 6

swap indexes 1 and 2

Child: 7 | 1 | 2 | 4 | 5 | 3 | 6

I am also doing an insertion mutation, shifting three adjacent bases to a different location on the gene. This is done as:

Parent: 1 | 7 | 2 | 4 | 5 | 3 | 6

insert values starting at index 5 to index 2

Child: 1 | 5 | 3 | 6 | 7 | 2 | 4

Which version of mutation is performed on a child is selected at random.

*Crossing Over:* Since a pure crossing over would result in an invalid solution, I must effectively repair the gene afterward. To accomplish this, I randomly select pairs of genes, randomly select an index, and perform a swap mutation based on the value the other parent has at that index. For example:

Parent 1: 1 | 3 | 2 | 4 | 5 | 7 | 6

Parent 2: 2 | 7 | 1 | 3 | 6 | 5 | 4

swap index 3 ↑

In Parent 1 we find the index of the value that has index 3 in Parent 2 and swap that with the value at index 3. We repeat this process for Parent 2.

Child 1: 1 | 4 | 2 | 3 | 5 | 7 | 6

Child 2: 2 | 7 | 1 | 4 | 6 | 5 | 3

*Evolution:* The population is stored as a doubly linked list. The population is seeded with 4 individuals with genes that have been shuffled with the Fisher-Yates method. For each generation, all individuals sexually reproduce, doubling the population, and each child also experiences one of the two mutation methods. I then calculate the fitness function for each individual and write those values to an array. I perform an insertion sort on the array (selected since the graph sizes being evaluated are relatively small), randomly generate a culling ratio between 10% and 90% and remove that proportion of the weakest individuals from the population. The remaining individuals then reproduce, doubling the population again.

For each generation, after the culling is complete the number of survivors is logged along with the highest fitness score. The evolutionary process terminates after 100 generations or when the highest fitness score hasn't changed in 10 generations.