EIGHT QUEENS PROBLEM

Solution with Genetic Algorithm

Introduction

The problem comprises of an 8X8 chessboard with 8 queens which should be placed on the board in such a way that no two queens attack each other. It is an example of a more general N-Queens problem that belongs to the family of NP-Complete problems.

Genetic Algorithm is employed to solve the above problem and further improved to converge to a solution in the least number of steps. This algorithm takes its inspiration from the process of natural selection as it uses a population of chromosomes (possible solution states) which are then selected, crossed over, and mutated to form a new set of chromosomes. This approach tries to converge to a state with the maximized fitness function value.

Implementation Details

For the given problem, the algorithm uses the state as a string of 8 digits in the range 1 to 8 each denoting the row number of a queen in the 8 respective columns of the board. The fitness function value is defined as 1 + sum of non-attacking queens and hence, converges to a solution if the value becomes 29. The population size is set to 20. The initial state has all the queens are placed on the same row (state - ‘44444444’) which is duplicated to fill up the entire population.

The algorithm is run for a maximum of 7000 generations or till a possible solution state (with fitness value 29) is found. In each iteration, the new population is formed by performing 20 crossovers which then replaces the old population entirely. For each crossover,

two parents are sampled from the population with probability based on the fitness value. The crossover is done by randomly selecting a position as a breakpoint and taking the first part from one parent and the second from other. A 20% probability is set for a mutation in which one digit is randomly changed to some other digit and hence, generating a new state.

Improvements

- Random Selection:-

  The major improvement in the performance came with the use of exponentiation in the probability of random selection of parents for crossover. In the textbook version, a parent is chosen with a probability of fitness(parent)/sum of fitness(parent) while in the improved version, it is chosen with the probability of **exp(**fitness(parent)**)**/sum of all **exp(**fitness(parent)**)**. This leads to a huge increases in the gap between the probability of selection for parents with close fitness value. For example, two parents with fitness value 27 and 28 will have an almost similar value of probability of selection in the textbook version while the parent with fitness value 28 will have much more probability of selection in the improved version.

  This works well for the Eight Queens problem as the state space has many possible solutions for the problem. Choosing the best parents for reproduction allows the algorithm to converge to one of those solution states very quickly in comparison to the original algorithm. However, this may sometimes lead to a problem of getting stuck in the local maximum which is further solved in another improvement listed for double mutation. I also tried various other powers of

fitness value in selection like fitness^2, fitness^4, etc. but the best results were achieved with e^fitness.

- Double Mutation:-

  With the improvement in the random selection, the problem of the algorithm getting stuck at a local maximum came up. This is solved with the help of more mutation in the states. As an improvement, the probability of mutation was increased to 80% and also a second mutation is employed with a probability of 30%.

  This improvement seems to work very well in this problem due to the change in random selection. Also, another major advantages is due to the design of the initial population. All the states in the initial population are same with all queens on the same row. Thus, crossover can not bring any change in the population and the only change possible is with the help of mutation. With much more probability and second mutation, the population gets more diverse way more quickly than the original algorithm and hence, reach one of the several solutions states in much less number of generations.

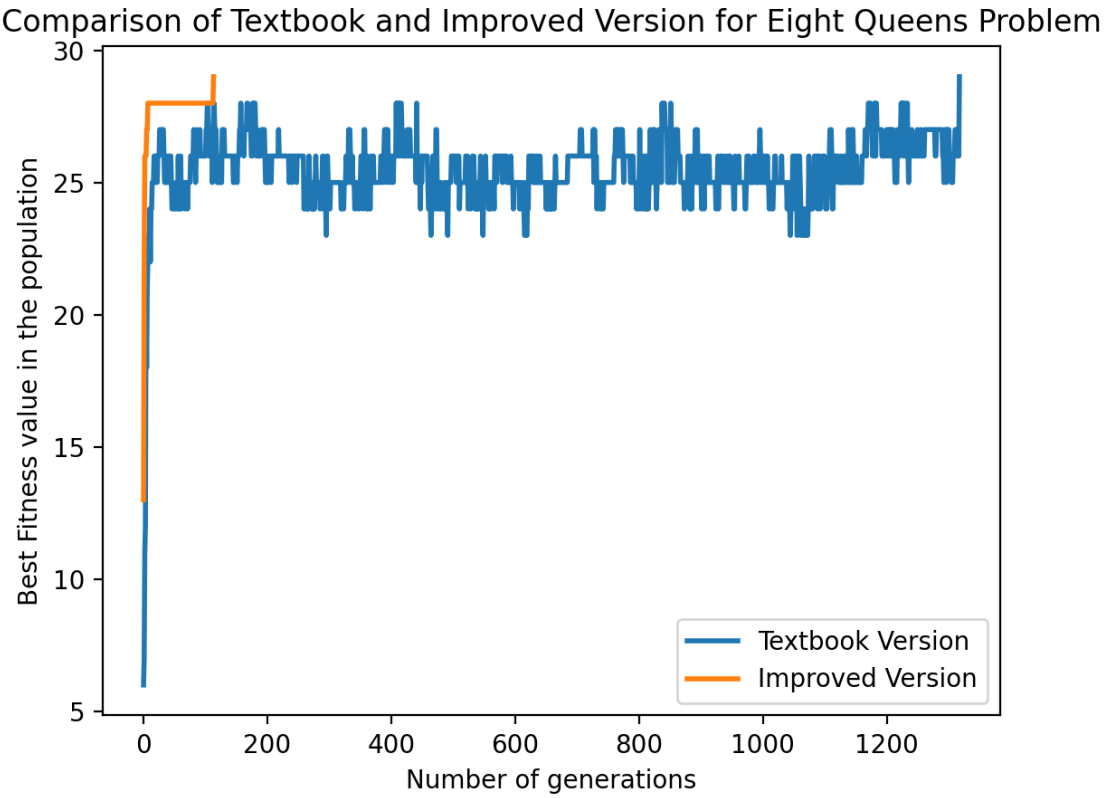
- Reproduce Function:-

  The crossover function is improved with the inspiration of steepest ascent hill climbing. Two breakpoints are selected randomly as positions for crossover in both the parents. All 6 possible combinations are formed as 6 different children and the child with best fitness is returned.

  This idea again helps in moving quickly to a state with maximized fitness value while also keeping a randomised approach for choosing the breakpoints. This does not

completely transform the algorithm to a steepest hill climb all the possible children are not evaluated to get the best best child and hence, reducing the problem of getting stuck in the local maximum.

Results




```
Final Results:-
For textbook version Generation:  1318 Max Fitness:  29.0 state with max fitness:  36824175
For improved version Generation:  114 Max Fitness:  29.0 state with max fitness:  38471625
```

The improved version reaches a final solution within 114 generation while the textbook version takes 1318 generation to get to a possible solution of the problem. An improvement factor of around 10 times is observed in this particular case. The plot above also displays the textbook version to be a lot more unstable fitness wise before terminating at a solution. The improved version tries to move quickly towards the solution taking huge jumps with fitness value almost always improving per generation.

As the algorithms involved the use of random function multiple times, the results were a bit erratic for multiple runs of the code. Hence, both the algorithms were run 10 times to get statistics about the average performance. The results are displayed in the screenshot below.

```
---------------------------------------- TEXTBOOK VERSION ----------------------------------------------------


Epoch:  0       Generation:  101        Max Fitness:  29.0 state with max fitness:  74286135
Epoch:  1       Generation:  1737       Max Fitness:  29.0 state with max fitness:  63185247
Epoch:  2       Generation:  1608       Max Fitness:  29.0 state with max fitness:  57142863
Epoch:  3       Generation:  158        Max Fitness:  29.0 state with max fitness:  38471625
Epoch:  4       Generation:  2722       Max Fitness:  29.0 state with max fitness:  72631485
Epoch:  5       Generation:  1601       Max Fitness:  29.0 state with max fitness:  51842736
Epoch:  6       Generation:  944        Max Fitness:  29.0 state with max fitness:  42736815
Epoch:  7       Generation:  354        Max Fitness:  29.0 state with max fitness:  57248136
Epoch:  8       Generation:  910        Max Fitness:  29.0 state with max fitness:  57413862
Epoch:  9       Generation:  2701       Max Fitness:  29.0 state with max fitness:  41582736
For  10  epochs:
Number of epochs without convergence:  0
Avg number of generations for convergernce:  1283.6
Min convergence generation:  101
Max convergence generation:  2722


---------------------------------------- IMPROVED VERSION ----------------------------------------------------


Epoch:  0       Generation:  128        Max Fitness:  29.0 state with max fitness:  35714286
Epoch:  1       Generation:  65 Max Fitness:  29.0 state with max fitness:  61528374
Epoch:  2       Generation:  25 Max Fitness:  29.0 state with max fitness:  42751863
Epoch:  3       Generation:  47 Max Fitness:  29.0 state with max fitness:  36271485
Epoch:  4       Generation:  40 Max Fitness:  29.0 state with max fitness:  36258174
Epoch:  5       Generation:  12 Max Fitness:  29.0 state with max fitness:  63728514
Epoch:  6       Generation:  287        Max Fitness:  29.0 state with max fitness:  63184275
Epoch:  7       Generation:  14 Max Fitness:  29.0 state with max fitness:  74286135
Epoch:  8       Generation:  11 Max Fitness:  29.0 state with max fitness:  47382516
Epoch:  9       Generation:  239        Max Fitness:  29.0 state with max fitness:  37285146
For  10  epochs:
Number of epochs without convergence:  0
Avg number of generations for convergernce:  86.8
Min convergence generation:  11
Max convergence generation:  287
```