Assignment 4: Eight Queens Problem

William Presley
CEAS
University of Cincinnati
Cincinnati, US
preslews@mail.uc.edu

***Abstract*—PRCLPS & PRCTC EVOL CMPTG Assignment 4. This report details the completion of assignment 4 through the description of the EA created and design decisions behind the created EA.**

I. Problem Description

The problem I decided to solve was the eight queens puzzle. In this problem, the number of queens that are not in check is the focus of optimization. The goal is to have all eight queens — on a chess board with dimensions of 8 by 8 — not checked by each other vertically, horizontally, or diagonally. This problem is not multi-objective or non-stationary as the only aspect to optimize is the number of queens not in check, and this objective does not change in any way over time.

II. Genome Representation Scheme

I chose my genome representation scheme to be a 16 digit array of integers to represent the 8 x and y coordinates of each queen on the board. I chose the digits to be integers because queens cannot be partially between squares on a chess board. I only gave each x and y coordinate a single integer entry in the genome (rather than several entries that sum to each coordinate value) because there are only 8 possible values for each coordinate, and having multiple chromosomes for each coordinate would be too limiting for the possible values for each chromosome. Using a single value for each coordinate rather than bit strings also saves on computation for any of the processes that index through the genome array such as the fitness function.

III. Fitness Function

The fitness function in this problem has a straightforward task of counting the number of queens not checked. My solution to creating a fitness function was to create a temporary board for each solution that is defined by a 2D array, placing each queen one by one on the board and drawing each path the queen checks, and then once each queen is placed, counting how many queens haven't been checked by the other queens. By using a temporary board and going through the placed queens iteratively, more resources are used than a non-iterative approach, however it allows for easy debugging and solution visualization as a board can be printed to the terminal to see the solution clearly. One downside that is native to the eight queens problem is the resolution of fitness solutions possible in the search space. As a result of the fitness calculation counting the number of queens unchecked, there are only 9 possible fitness values ranging from 0/8 (worst solution) to 8/8 (perfect solution). While this does limit the resolution of fitness solutions, one positive is that better solutions are clearly better and separated evenly from worse solutions.

IV. EA Description

My EA is a mix of elements from our first SGA and our ES.

A. Representation

I chose an integer array for the genome representation rather than a bitstring or a floating point value array as discussed in section 2.

B. Recombination

For recombination, I chose uniform crossover for a couple reasons. For one, it allows each queen in a solution to be crossed over with the other parent's and allow for new placements on the board as long as it is within the largest coordinate of each parent. Another reason is because I didn't see single or multi-point crossover as being beneficial with my genome representation scheme. In single and multi-point crossover, one to many of the queens from each parent would be swapped which I didn't see as beneficial as having new solutions coming from each parent like what can happen in uniform crossover.

C. Mutation

For mutation, I decided to use random resetting. When choosing between random resetting and a mutation that adds/subtracts integer values, I decided that random resetting would be best because it allows for a more aggressive disruption in solutions and because the integer equivalent of creep mutation would be limited by how much it can add or subtract within the small range of 8 values per coordinate.

D. Parent Selection

For parent selection, I went with uniform random selection as I would be relying more heavily on survivor selection. Specifically mu plus lambda so it was not critically important that the best individual solution was able to produce offspring as it would be chosen to stay in the population as long as it was still good enough relative to the new children to survive. The randomness in the parent selection allowed for the currently poor solutions in each generation to provide better solutions through crossover and mutation as a way to combat elitism that would lead the population to plateauing solutions.

E. Survivor Selection

For survivor selection, mu plus lambda was chosen as previously described. I chose this to make sure that progress was never lost as a result of poor mutation or recombination. This choice of survivor selection allowed me to make decisions that increased the aggressiveness and disruptiveness of other aspects of the EA such as mutation/recombination/parent selection to make sure as many solutions were being provided as possible before reaching the point of disruptiveness that would prevent convergence on a perfect solution. While there is still the possibility of the EA getting stuck on a solution that is not fully perfect, it can usually be overcome by increasing the safety limit of the simulation and evaluating more generations, whereas when using a survivor selection such as mu comma lambda, I would have trouble getting the population to reach a proper solution.

F. *Survival Selection*

This EA did not have any sort of specialty or strategy like an ES.

V. Grand Utility Space

A. *AES Utility Metric*

For my first measure of utility, I chose to use AES (candidate evaluations until a solution is reached). I chose this partly because I had experience from the past homework that allowed me to find good parameters to test and partly because my EA converged consistently enough that reaching the safety limit wouldn't be as much of a problem as it was previously and I wanted to see how close my chosen parameters were to the optimal parameters. For all utility runs, each parameter is run 25 times and then averaged for each recorded/plotted value.

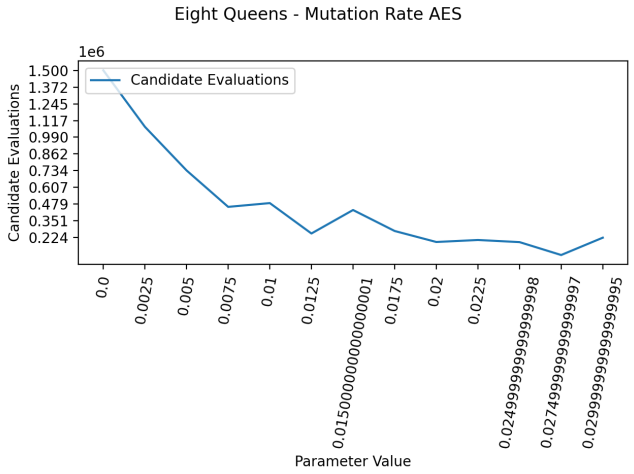

Fig. 1. Utility landscape of mutation rate using AES metric. The long x-axis values are from rounding from the 'eightqueens_WilliamPresley.java' function that increments parameter values.

For my first parameter of mutation rate, it is clear from the figure that the mutation rate increases performance of the EA (when it comes to the AES utility measurement) as mutation rate increases towards a limit of 0.03.

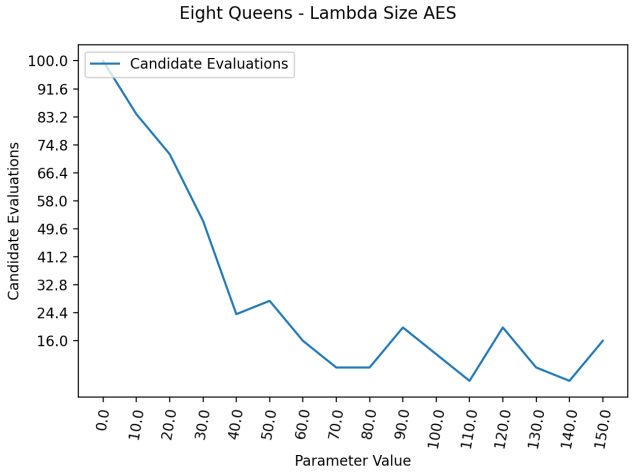

Fig. 2. Utility landscape of lambda size using AES metric. The long x-axis values are from rounding from the 'eightqueens_WilliamPresley.java' function that increments parameter values.

For my second parameter of lambda size, the performance of the EA increases as the lambda size increases to roughly 70, and then oscillates while remaining relatively low as it increases to 150.

B. *SR Utility Metric*

For my second measure of utility, I chose to use SR (success rate). I chose this because it was a good match with my first measure of utility and when both are used together, it can show when a low SR can lead to outliers that skew the AES values. I calculated SR by the number of times a parameter run converges to a perfect fitness of 1.0 divided by the number of attempts, which is 25 (because the EA is run 25 times for each parameter value).

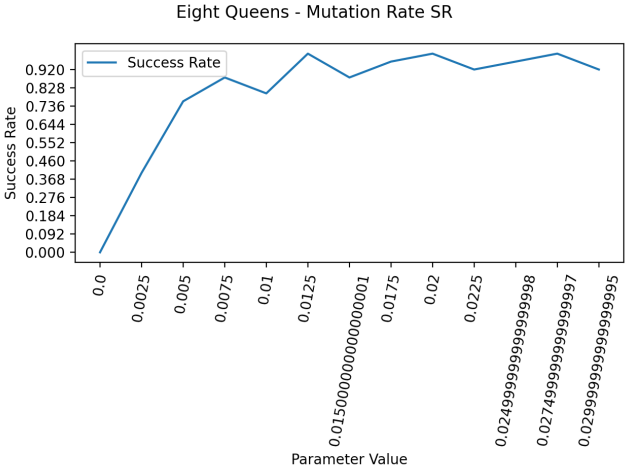

Fig. 3. Utility landscape of mutation rate using SR metric. The long x-axis values are from rounding from the 'eightqueens_WilliamPresley.java' function that increments parameter values.

For my first parameter of mutation rate, it can be seen that the success rate is roughly the inverse of the mutation rate candidate evaluations in FIGURE 1. This can be seen since the higher the success rate, the lower the amount of candidate evaluations needed for the population to converge.

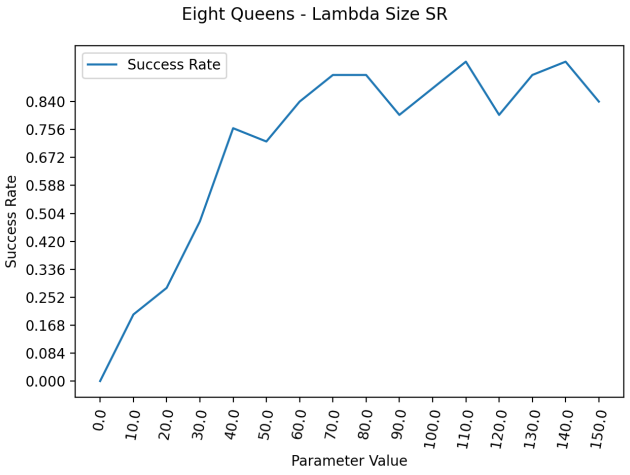

Fig. 4. Utility landscape of lambda size using SR metric. The long x-axis values are from rounding from the 'eightqueens_WilliamPresley.java' function that increments parameter values.

For my second parameter of lamba size, it can be seen that the success rate very closely matches the inverse of the lambda size candidate evaluations in FIGURE 2. This can be seen as the success rate drops, the average number of candidate evaluations increases.

VI. Final Discussion

I believe that while there might be certain aspects of EA that could be better, I made choices that behaved well together and supported a high success rate and relatively fast performing EA. My experiments in #5 support this fairly well and my original parameter values I picked (of a mutation rate of 0.01 and a lambda size of 100) were good even before having the grand utility landscape. Given more time, I might try a different setup that works well with the mu comma lambda survivor selection to see if I can get a higher success rate with a lower safety limit, aiming for the benefits provided by mu comma lambda that the population is less likely to get stuck on a dead-end solution for too long.

References

[1] A. E. Eiben and J. E. Smith, Introduction to Evolutionary Computing, 2nd ed. Erscheinungsort nicht ermittelbar: Springer, 2015.

[2] “Eight queens puzzle,” Wikipedia, 22-Apr-2021. [Online]. Available: https://en.wikipedia.org/wiki/Eight_queens_puzzle.