An Analysis of Using a Genetic Algorithm to Solve the N-Queens Problem

Wyatt Kormick
kormi001@umn.edu

March 4, 2018

Figure

For my graph, I ran the genetic algorithm test with both a variable number of generations, and, as directed, a variable number of queens. For each, I measured the fitness of the solution provided by the algorithm to get some idea of the quality of the solution. The fitness function created a score based on whether or not a queen was placed in the same column as another queen, or diagonal from another queen. Every time a queen was placed in one of these spots a point was removed from the score. For my quality metric, I took this fitness score, divided by the max possible fitness score for the number of queens. For the number of generations, I used values of 250, 500, 750, and the default value of 1000. For the number of queens I tested with 8, 12, 16, and 20 queens. The results of the experiment can be found in Figure 1.

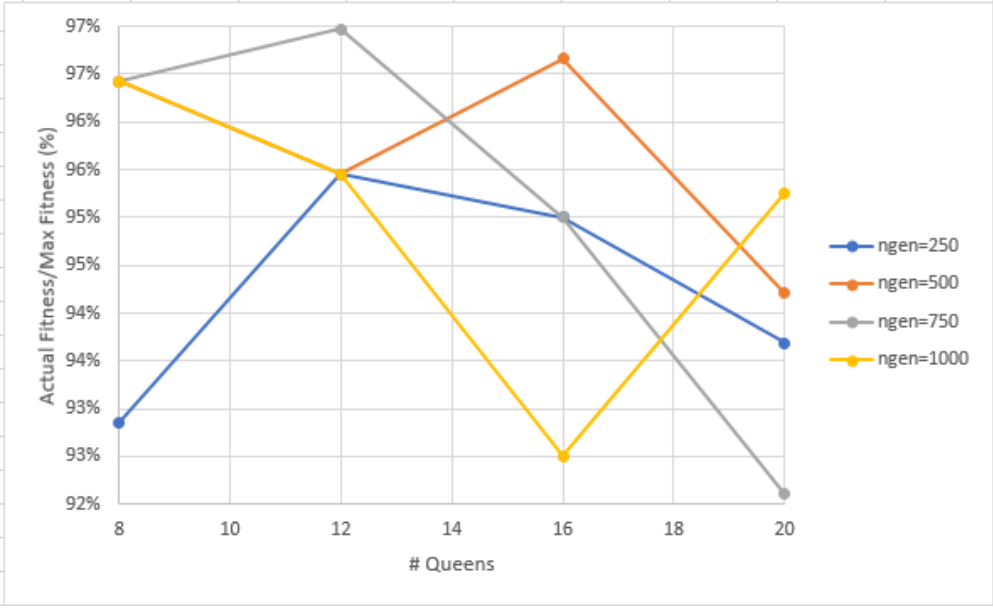

Figure 1: Graph showing the relation between the number of queens, the number of generations, and the quality of the solution.

Table 1: Run-times with Variable ngens and pop

	250	500	750	1000
33	24.13	48.06	77.86	104.48
66	95.68	204.52	312.06	416.00
99	211.97	468.79	703.96	940.61

There doesn't seem to be too much of a trend to be found in the data gathered. Maybe more trials for each group were needed to get a more accurate visualization of the relation, but I found myself limited by time. Future experiments would have more trials for each group, and possibly have a more granularity when changing the number of queens. It can maybe be said that a lower amount of queens can lead to a higher quality solution, although it isn't always the case. This might make sense as there is less to get wrong about the problem. It also seems that the lowest number of generations produces the lowest quality results. An explanation for this could be that with a lower number of generations, there isn't as much of a chance to come up with higher quality solution. The problem with using a local search such as this genetic algorithm to come up with a quality solution, is that it is based in chance. There is a very high possibility that the algorithm will only come up with a local solution. The algorithm is designed to find local maximum, and only if the right set of circumstances happen will the local maximum also be a global maximum. This could provide an explanation for the disparity in solution quality throughout.

Table

For my table I ran the genetic algorithm test with both a variable number of generations and population within each generation, and measured how long it took the algorithm to find a solution. The default values of the original algorithm had the number of generations equal to 1000, and a population size of 100. For my run-time tests, the algorithm was run with generation values of 250, 500, 750, and 1000. The population values used were 33, 66, and 99. The results of my experiments can be found in Table 1. In this table the columns represent the generation values, the rows represent the population values, and the cell values are the time it took the algorithm to find a solution in seconds.

The data gathered from the experiment would suggest that both an increasing population size, and an increasing number of generations increase the run-time of the algorithm. Within each constant population group the run-time seems to increase linearly along with the rise in the number of generations. Within the 33 population group, run-time goes up approximately 25 seconds for every 250 generations. In the 66 group, it goes up by just over 100 for every generation group. Finally, with a population size of 99, the run-time increases by about 240 seconds, every 250 generations. A possible reason for this is that the entire algorithm is held inside a for-loop, which are O(n), that runs a number of time equal to the number of generations. This is the only point where the number of generations is used so it would make sense that a linear increase in the number of generations would lead to a linear increase in run-time. For the changing population size, the rate of increase in run-time is a bit different. It is actually closer to an n^2 slope. The 250 generation group of populations closely follows the curve $y=(5n)^2$. Likewise, the 500 group follows $y=(7n)^2$, 750 follows $y=(9n)^2$, and the 1000 group follows $y=(10n)^2$.