Mario Garcia
Spring Quarter 2017
14 May, 2017
Dr. Fang Tang

CS420: Project 2 Findings

For this project, we were tasked to implement two local search algorithms to solve for the N-Queen Problem. The first algorithm, Steepest Hill Climbing, is exactly as its name implies. It is intended to find the best path by using greedy search, for solutions that reach a heuristic cost of h = 0. The problem, however, is that the name itself also implies it will get stuck on any local minima, or peak, regardless of whether or not it is the best path to h = 0. This means steepest hill climbing, on its own, can have a much higher failure rate than other algorithms that are used to solve for the N-Queen Problem. Now, of course the next algorithms we use instead, as an alternate search algorithm, is Genetic Searching. Using genetic algorithms, we can find multiple solution paths and represent them as a population of chromosomes, and then through selection, we can compare each chromosome using a fitness function, to which we then crossover and mutate some of these chromosomes, or solution states, to achieve our best overall solution.

For the Steepest Hill Climbing solution, the implementation was rather straightforward. Simply calculate the highest valued successor of each state, using our constant only initial state node of our board. Although quite simple to implement, it doesn't, in fact, solve problems efficiently for the N-Queen Problem. As this algorithm is quite greedy, it tends to stick with the local minima, which is the max peak of the problem heuristic given. We end up getting a rather low success rate of solving our N-Queen board of size 21x21, as a result. Given this data, we in fact reach an average success rate at about 4%, as a result of 100 cases of 21-Queen boards. Although, we can, for the most part, improve on this algorithm by implementing stochastic choice, which will then have our algorithm decide on a random best improvement, based on the given best moves for some state board. Doing so allow us to find the solution slightly better in some cases, however the implementation itself would still yield a slightly better success rate than a simple, direct Steep Hill Climb.

For our Genetic Algorithm, the implementation was rather a troublesome one. We started with defining our Chromosome structure, which would be used to create a set that we would store inside a Population object. This Population object would then be responsible for random chromosome generation, selection of the best chromosomes, and finally crossovers and mutation chances based on an anomaly. We do not have an anomaly, so we set a constant mutation probability variable to determine when a mutation would occur. This mutation chance goes about 15%, as it seemed to do slightly well as opposed to other adjustments. After our new population was created, we would then move it over to a new Population object, and sort it out to find the best fit chromosome. The results proved rather amazing, as it was able to find the solved path at good times for a 21x21 board. However, the implementation itself could have used more optimization, since the selection algorithm proved to not be as well written as previously done. Nonetheless, this genetic algorithm is still capable of solving any of our 21-Queen cases:

```
Initial board: 8 15 9 20 5 20 18 3 1 12 5 2 12 20 8 15 17 4 18 0 5
Genetic solved board: 9 15 2 14 11 1 8 6 18 3 13 19 16 5 20 10 17 7 4 12 0
Genetic time in 22.5375 ms

Initial board: 11 20 15 5 14 12 16 3 2 17 14 2 4 3 17 12 11 4 17 19 5
Genetic solved board: 6 2 7 5 15 12 19 16 18 13 10 4 1 3 0 9 11 20 14 17 8
Genetic time in 255.244 ms

Initial board: 8 18 3 1 7 12 3 15 4 13 12 12 13 7 3 13 3 17 0 20 19
Genetic solved board: 15 8 3 14 2 9 20 13 11 19 1 16 0 5 7 10 6 17 12 18 4
Genetic time in 170.231 ms
```

Based on these three cases, our algorithm was able to solve at a decent time, an average 149.3375 milliseconds. Yet, these cases tend to fluctuate as a result of the poor optimization of our selection call. When crossover occurs, our crossover point is randomly selected, which is as should be done anyway.

These findings are rather not very astronomical, however they show that the nature of genetic algorithms can, and should, be a good carving route to true artificial intelligence, simply because random choice is implemented. Of course, randomness is a difficult concept to correctly obtain a precision for, so genetic programming is not truly genetics, but the underlying concept is accurate. Perhaps there will be a day when we can achieve true randomness, for now, this will be the best implementation we have, to improve.