# CS 4200 - Project 2

Noah Reef

Winter 2023

## 1 Code Outline

For the *Uphill Ascent* approach for solving the 8-Queen Problem, I implemented an `uphill_ascent` method that takes in an initial state and attempts to solve the problem using the *Uphill Ascent* approach outlined in the course. First I generate all possible neighbors of the initial state, which is generated by the `actions` method, and evaluate the number of attacking of queens, which is implemented by the `measure_fitness` method, and choose the neighbor with the least number of attacking queens as the next choice to expand. Each node that is expanded in this process is represented as an object of the `Node` class which contains the *state,parent, and fitness.* This process repeats until it is found that the current state is the same as its parent, and return the current state as the solution.

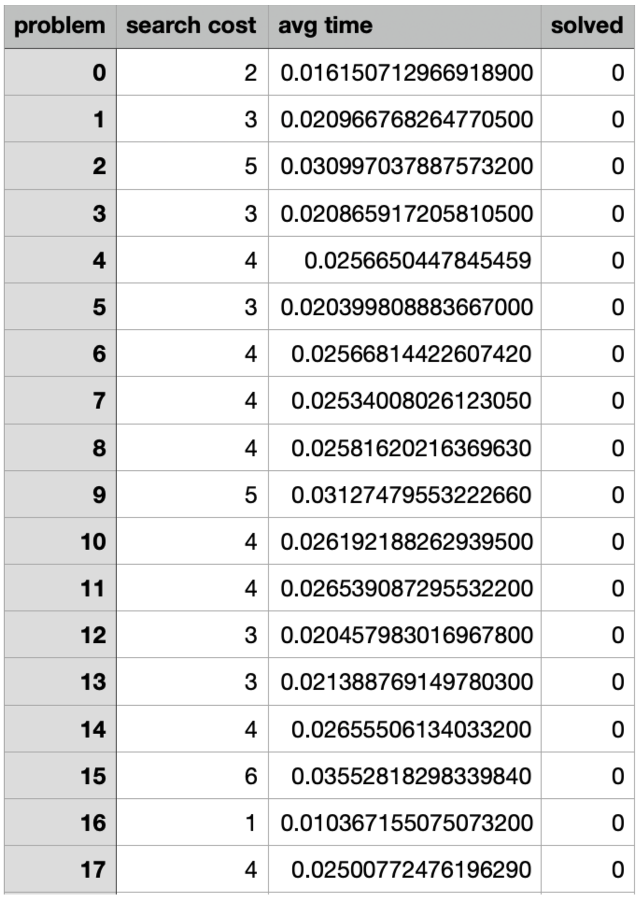
I also implemented the *Genetic Algorithm* approach as well for solving the 8-Queen problem within the `genetic_algorithm` method. First I generate 1000 random board states using the `generate_random_state` method for our initial population. Next each of the initial states are evaluated using a fitness function defined as

$$\text{fitness} = \frac{1}{e^{\#\text{ of attacking queens}}}$$

and at each iteration choose 50% of the best performing states as parents for crossover (using the `choose_parents` method, and drop the rest. I then generate children using the `crossover` method taking the top-half of the first parent and bottom-half from the second parent to make the child. I then with a 0.1 probability will mutate a random child by randomly moving one of the queens. We repeat this process until a solution is found.

## 2 Experiment Results

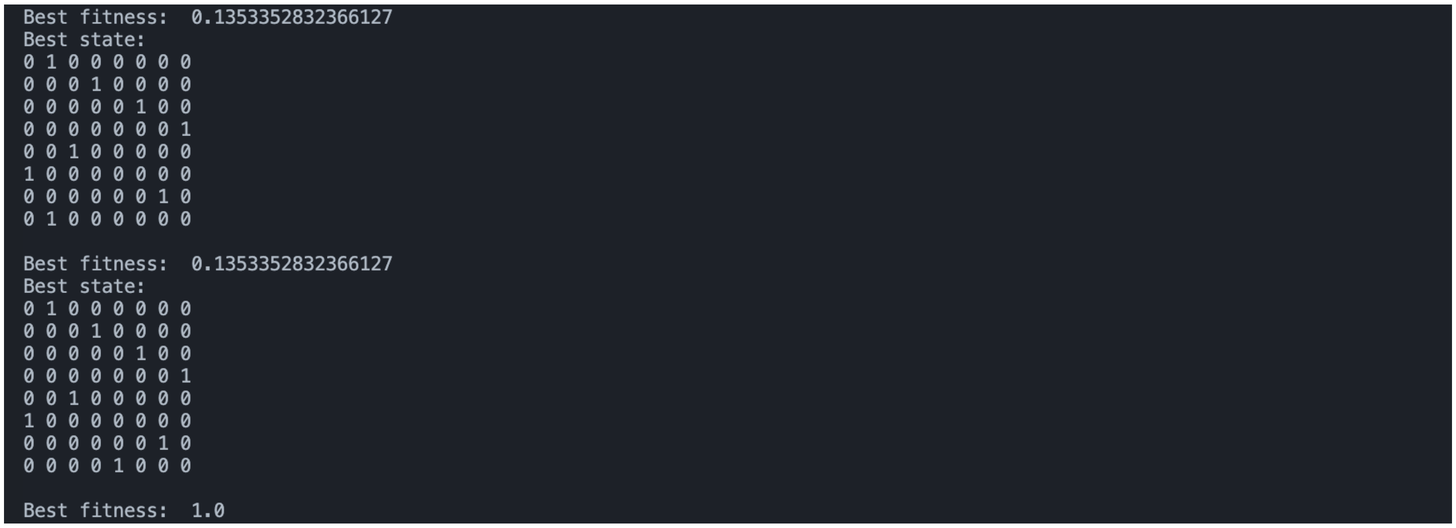
I ran an experiment with the hill climbing method over 100 randomly generated puzzle states and stored the results in `hill_climbing.csv`. Below are a subset of the results



| problem | search cost | avg time | solved |
|---|---|---|---|
| 0 | 2 | 0.01615071296691890 | 0 |
| 1 | 3 | 0.020966768264770500 | 0 |
| 2 | 5 | 0.030997037887573200 | 0 |
| 3 | 3 | 0.020865917205810500 | 0 |
| 4 | 4 | 0.0256650447845459 | 0 |
| 5 | 3 | 0.020399808883667000 | 0 |
| 6 | 4 | 0.02566814422607420 | 0 |
| 7 | 4 | 0.02534008026123050 | 0 |
| 8 | 4 | 0.02581620216369630 | 0 |
| 9 | 5 | 0.03127479553222660 | 0 |
| 10 | 4 | 0.026192188262939500 | 0 |
| 11 | 4 | 0.026539087295532200 | 0 |
| 12 | 3 | 0.020457983016967800 | 0 |
| 13 | 3 | 0.021388769149780300 | 0 |
| 14 | 4 | 0.02655506134033200 | 0 |
| 15 | 6 | 0.03552818298339840 | 0 |
| 16 | 1 | 0.010367155075073200 | 0 |
| 17 | 4 | 0.02500772476196290 | 0 |

Figure 1: First 17 Entries of hill_climbing.csv

and found that on average only 10% of the states were solved. The low percentage of solved boards can be explained by the fact that the Uphill Ascent Algorithm is a greedy algorithm and hence is prone to getting stuck at local minimums. For our genetic algorithm approach we see that it takes awhile for convergence however it does eventually converge as shown below,



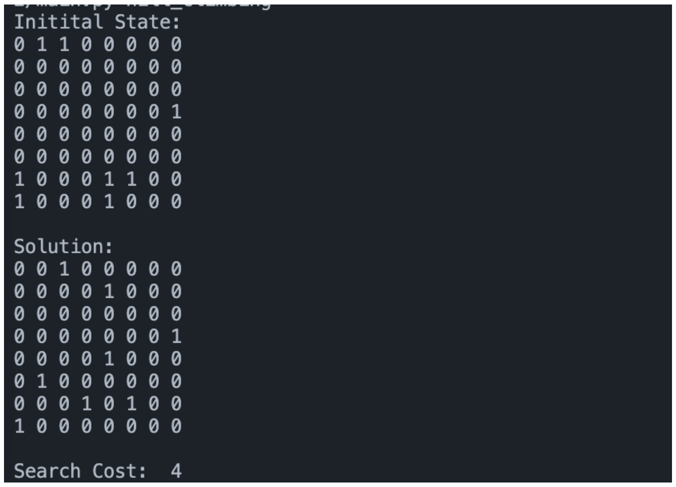
```
Best fitness:  0.1353352832366127
Best state:
0 1 0 0 0 0 0 0
0 0 0 1 0 0 0 0
0 0 0 0 0 1 0 0
0 0 0 0 0 0 0 1
0 0 1 0 0 0 0 0
1 0 0 0 0 0 0 0
0 0 0 0 0 0 1 0
0 1 0 0 0 0 0 0

Best fitness:  0.1353352832366127
Best state:
0 1 0 0 0 0 0 0
0 0 0 1 0 0 0 0
0 0 0 0 0 1 0 0
0 0 0 0 0 0 0 1
0 0 1 0 0 0 0 0
1 0 0 0 0 0 0 0
0 0 0 0 0 0 1 0
0 0 0 0 1 0 0 0

Best fitness:  1.0
```

Figure 2: Genetic Algorithm Output

## 3 Output

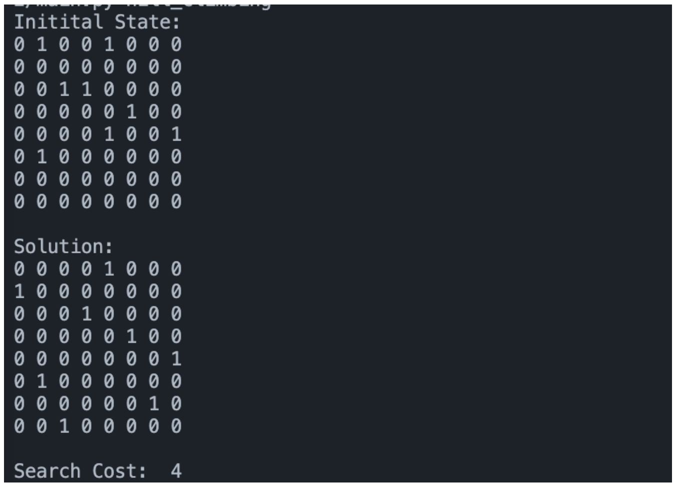


```
Initital State:
0 1 1 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 1
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
1 0 0 0 1 1 0 0
1 0 0 0 1 0 0 0

Solution:
0 0 1 0 0 0 0 0
0 0 0 0 1 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 1
0 0 0 0 1 0 0 0
0 1 0 0 0 0 0 0
0 0 0 1 0 1 0 0
1 0 0 0 0 0 0 0

Search Cost:  4
```

Figure 3: Uphill Ascent Algorithm Output 1

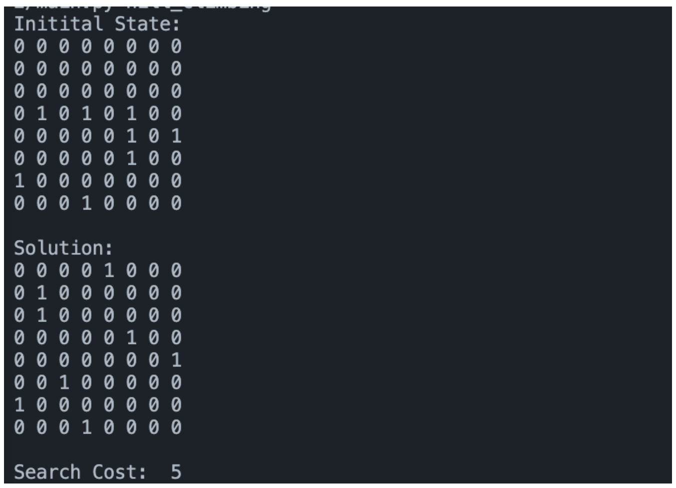


```
Initital State:
0 1 0 0 1 0 0 0
0 0 0 0 0 0 0 0
0 0 1 1 0 0 0 0
0 0 0 0 0 1 0 0
0 0 0 0 1 0 0 1
0 1 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0

Solution:
0 0 0 0 1 0 0 0
1 0 0 0 0 0 0 0
0 0 0 1 0 0 0 0
0 0 0 0 0 1 0 0
0 0 0 0 0 0 0 1
0 1 0 0 0 0 0 0
0 0 0 0 0 0 1 0
0 0 1 0 0 0 0 0

Search Cost:  4
```

Figure 4: Uphill Ascent Algorithm Output 2



```
Initital State:
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 1 0 1 0 1 0 0
0 0 0 0 0 1 0 1
0 0 0 0 0 1 0 0
1 0 0 0 0 0 0 0
0 0 0 1 0 0 0 0

Solution:
0 0 0 0 1 0 0 0
0 1 0 0 0 0 0 0
0 1 0 0 0 0 0 0
0 0 0 0 0 1 0 0
0 0 0 0 0 0 0 1
0 0 1 0 0 0 0 0
1 0 0 0 0 0 0 0
0 0 0 1 0 0 0 0

Search Cost:  5
```

Figure 5: Uphill Ascent Algorithm Output 3