University of North Carolina at Charlotte

N-Queens Problem Using Hill Climbing

PROJECT REPORT

ITCS 6150
Intelligent Systems

SUBMITTED TO:
Dewan T. Ahmed, Ph.D.

SUBMITTED BY:
Jacob Gulan

I. PROBLEM FORMULATION

1. N-Queens Problem

The N-Queens problem takes a value, N, and creates a NxN chessboard along with N queens to accompany it. The queens are set in random locations on the chessboard and the goal is to ensure that no two queens are able to attack each other. A possible solution to this problem may be seen in figure 1.

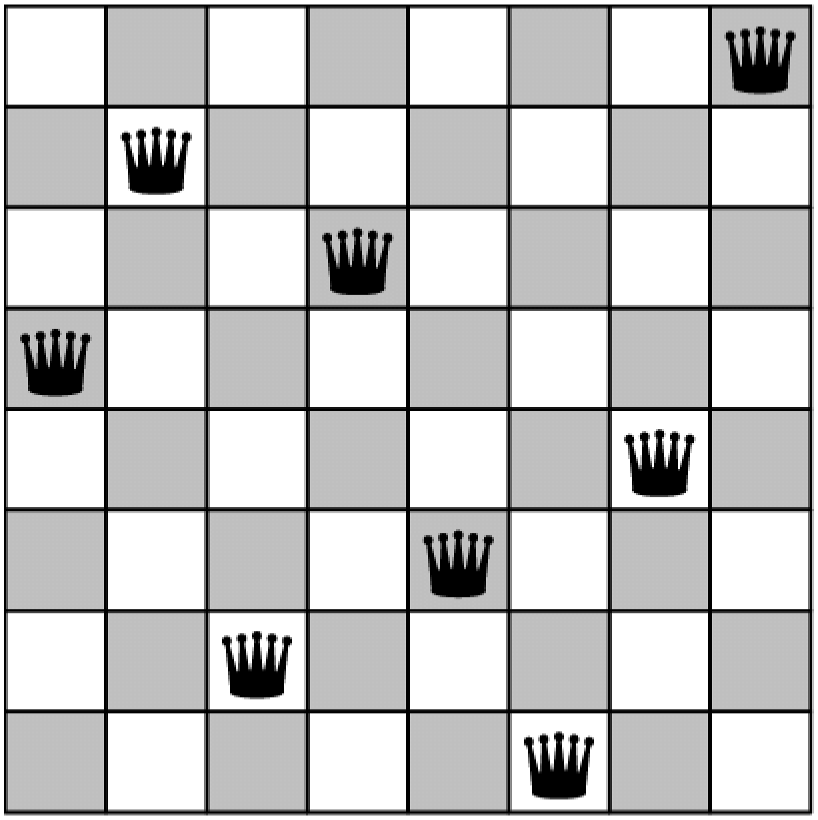

Figure 1. N-Queens Solution

2. Hill Climbing Search

An algorithm that's capable of solving the N-Queens problem is hill climbing search. In this report's adaptation steepest-ascent hill climbing will be used. Steepest-ascent hill climbing is a loop that moves in the direction of increasing value and terminates when no higher value can be found. However, steepest-ascent hill climbing in its natural form will not be sufficient enough to yield high success rates in finding solutions to the

N-Queens problem as the algorithm will often get stuck in a plateau like shoulders or flat local maximums, see figure 2.

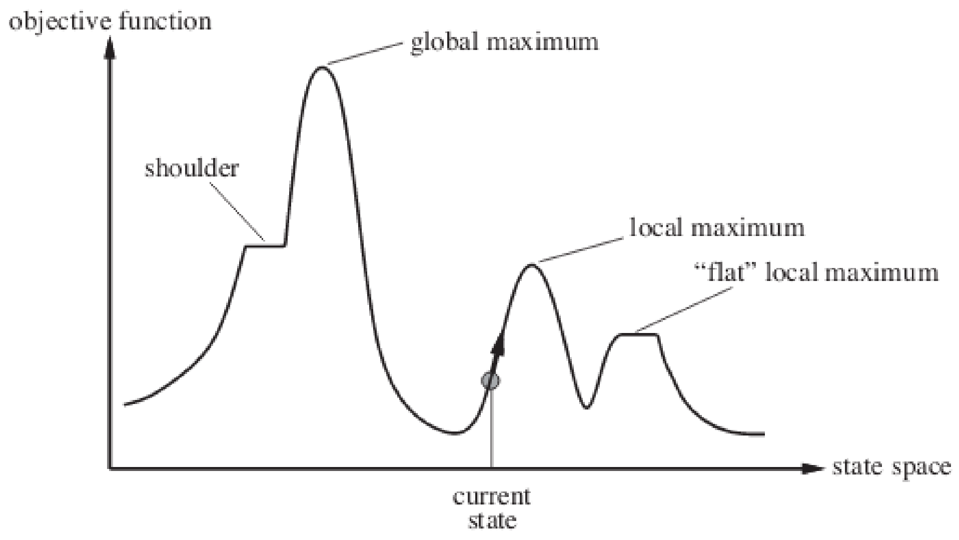

Figure 2. Hill Climbing State Space Diagram

3. Sideways Move and Random Restart Hill Climbing

To address the shortcomings of a vanilla steepest-ascent hill climbing algorithm, sideways move and random restart hill climbing may be applied to improve the probability of finding a solution. With sideways move, if the algorithm is stuck in a plateau, instead of stopping execution the algorithm will keep moving to other states with the same heuristic value in hopes that it is a shoulder and it will escape it. There is still the chance the algorithm can get stuck in a flat local maximum though and not be able to escape. Random restart hill climbing allows us to attempt searching again at a new random initial state if the algorithm gets stuck. This ensures a solution is found, but may impact runtime.

4. Implementation

The program that I've devised will use steepest ascent hill climbing with and without sideways move and random restart hill climbing with and without sideways move. Each one of these algorithms will be run 100 times and the results of which will be printed to the console. This program was created using Python 3.6.9 and the Numpy library.

II. PROGRAM STRUCTURE

1. Classes

Class	**Class description**
Queens	The Queens class is responsible for holding the locations of the queens. The heuristic cost is also calculated within this class and it also retrieves the possible future states for the HillClimber to take.
HillClimber	The HillClimber class is responsible for finding the solution to the N-Queens problem. This class contains the steepest ascent hill-climbing method with and without sideways move and the random restart hill climbing method with and without sideways move.

2. Local Variables

Queens Class

Variable	**Variable type**
n	Integer holding the amount of queens
queens	Array holding the location of the queens

HillClimber Class

Variable	**Variable type**
queens	Queens Object Instance
initialBoard	Array holding the initial state of the board
steps	Integer holding the amount of steps taken
restarts	Integer holding the amount of restarts taken
success	Boolean holding whether the algorithm was successful

3. Functions and Procedures

Queens Class

Function/Procedure	**Description**
__init__(self, n, queens)	The initializer of the Queens class sets the amount of queens to the input of n and creates a randomized location of queens that will be stored in the queens array if the location of the queens weren't passed in. Params: n - Integer (number of queens) queens - Array, default=None (holds the location of the queens)
generateQueens(self)	This function is only called during the initialization of a Queens object. This randomly generates the location for each queen. Each queen is assigned a unique row depending on its index and a random column location depending on its randomly generated value from zero to n. This array is then assigned to the self.queens array. Returns: Array (holds queen's locations)
getNeighbors(self)	The getNeighbors function returns all the possible states that are able to be taken by the queens. This function is called in the HillClimber class and is used to determine the heuristic value. Returns: Array (holds possible future states)
calcHeuristic(self)	The calcHeuristic function looks at each queen and checks to see if it's able to attack any other queen. If it is then the heuristic cost is increased by a value of one. This function is called in the HillClimber class to determine the best possible move to take in search of finding a solution. Returns: Integer (holds the heuristic cost of the board)
makeBoard(self)	The makeBoard function is used to transform the 1D array of queen locations to a 2D matrix. This function is called in the main() function and is used solely for providing a visual representation of the

	board. The value of 1 represents a queen and the value of 0 represents an empty spot. Returns: 2D array (Holds the board's current state)

HillClimber Class

Function/Procedure	**Description**
__init__(self, n)	The initializer of the HillClimber class creates an object of the Queens class to perform the hill climbing search on and stores information like the initialBoard, steps taken, restarts taken, and whether or not the search was successful or not. These variables are both used to aid in the search process and are used as a way to print results after the search has terminated. Params: n - Integer (number of queens)
minRandomNeighbor(self, neighbors)	The minRandomNeighbor function is a helper function for both the hillClimbing and randomRestart functions. This function determines what future states have the lowest heuristic cost and randomly selects one of them to be the next state. Returns: Queens Object Instance (random state with lowest heuristic cost)
hillClimbing(self, sideways)	The hillClimbing function is responsible for executing steepest-ascent hill climbing. Sideways move may be enabled by setting the sideways parameter to true. This function procedurally takes steps towards finding the solution to the N-Queens problem. If a solution is found then it is returned as successful, otherwise the final state is returned as unsuccessful. Params: sideways - Boolean, default=False (False if no sideways move, True if using sideways move) Returns: Queens Object Instance (final state before search terminates)

randomRestart(self, sideways)	The randomRestart function is responsible for executing random restart hill climbing. Sideways move may be enabled by setting the sideways parameter to true. This function procedurally takes steps towards finding the solution to the N-Queens problem. If a solution is found then it is returned as successful, otherwise if the search got stuck then a new starting board is created and the search begins again until a solution is found. Params: sideways - Boolean, default=False (False if no sideways move, True if using sideways move) Returns: Queens Object Instance (final state before search terminates)

No Class

Function/Procedure	**Description**
main()	The main() function is used for executing multiple instances of the HillClimber class and retrieving results. The user may input the amount of queens they wish in this method as well. After that, each hill climbing algorithm will execute 100 times and the results of each will be printed to the console.

III. RESULTS

1. Hill Climbing Without Sideways Move

```
Hill Climbing Without Sideways
Successes:  17
Failures:  83
Successes Average Steps:  4.0588235294117645
Failures Average Steps:  2.8433734939759034
Four Random Initial Configurations:
[[0 0 0 0 0 0 0 1]
 [0 0 0 0 0 0 0 1]
 [0 0 0 0 0 0 1 0]
 [0 0 0 1 0 0 0 0]
 [0 0 0 0 0 1 0 0]
 [0 0 0 0 0 1 0 0]
 [0 0 1 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 1 0 0]
 [0 0 1 0 0 0 0 0]
 [0 0 0 0 0 1 0 0]
 [0 0 0 0 0 1 0 0]
 [0 0 0 0 0 0 0 1]
 [0 0 0 0 1 0 0 0]
 [0 0 0 0 0 0 0 1]]

[[1 0 0 0 0 0 0 0]
 [0 1 0 0 0 0 0 0]
 [0 0 0 0 0 0 1 0]
 [0 0 0 0 0 0 0 1]
 [1 0 0 0 0 0 0 0]
 [0 0 0 0 1 0 0 0]
 [0 0 0 0 1 0 0 0]
 [0 1 0 0 0 0 0 0]]

[[0 0 1 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 0 1 0 0 0 0]
 [0 0 0 0 1 0 0 0]
 [0 0 0 1 0 0 0 0]
 [1 0 0 0 0 0 0 0]]
```

2. Hill Climbing With Sideways Move

```
Hill Climbing With Sideways
Successes:  96
Failures:  4
Successes Average Steps:  16.458333333333332
Failures Average Steps:  79.0
```

```
Four Random Initial Configurations:
[[0 0 0 0 0 0 0 1]
 [0 0 0 0 0 0 0 1]
 [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 0 0 0 0 1 0 0]
 [0 1 0 0 0 0 0 0]
 [0 1 0 0 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 0 1 0]
 [1 0 0 0 0 0 0 0]
 [0 0 0 1 0 0 0 0]
 [0 0 0 1 0 0 0 0]
 [0 0 0 1 0 0 0 0]
 [0 0 0 0 1 0 0 0]]

[[1 0 0 0 0 0 0 0]
 [0 0 0 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 1]
 [0 0 1 0 0 0 0 0]
 [0 0 0 1 0 0 0 0]
 [1 0 0 0 0 0 0 0]]

[[0 0 0 0 0 1 0 0]
 [1 0 0 0 0 0 0 0]
 [0 1 0 0 0 0 0 0]
 [1 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 0 1 0 0]
 [0 0 1 0 0 0 0 0]]
```

3. Random Restart Hill Climbing Without Sideways Move

```
Random Restart Without Sideways
Average Steps:  22.41
Average Restarts:  6.09
```

4. Random Restart Hill Climbing With Sideways Move

```
Random Restart Hill Climbing With Sideways
Average Steps:  21.41
Average Restarts:  0.05
```

IV. CONCLUSION

Looking at the results from the last section, it's evident that enabling sideways move dramatically affects the success rate of finding a solution. For steepest-ascent hill climbing without sideways move, only 17 of the 100 attempts found a solution. Whereas, with sideways move found a solution in 96 of the 100 attempts. However, although the success rate did improve, the runtime worsened by enabling sideways move and that's reflected by the drastic increase in the amount of steps taken between the two algorithms.

With random restart hill climbing, the impact of sideways move is also seen. Without sideways move the average number of restarts it took before finding a solution was 6.09 times. With sideways move the average number of restarts went down to 0.05. The difference in steps taken between these two algorithms is negligible as well with no sideways move having an average of 22.41 steps and sideways move taking an average of 21.41 steps.

Steepest-ascent hill climbing managed to solve the n-queens problem in a handful of cases, but implementing sideways move drastically improved the rate in which a solution was found. Random restart hill climbing also proved useful in finding a solution to the n-queens problem and the implementation of sideways move helped bring down the number of restarts required to find a solution and the difference between the amount of steps needed is largely negligible.

V. REFERENCES

1. https://www.researchgate.net/profile/Curtis-Bright/publication/333815714/figure/fig1/AS:770619155156992@1560741332335/A-visual-representation-of-a-solution-for-the-8-queens-problem-left-and-the-variables.png
2. https://www.geeksforgeeks.org/introduction-hill-climbing-artificial-intelligence/
3. https://www.programmersought.com/article/3622810395/
4. https://letstalkdata.com/2013/12/n-queens-part-1-steepest-hill-climbing/
5. https://github.com/AP-Atul/NQueens-Problem
6. https://github.com/AnushreeSrivastava/N-Queen-Problem
7. https://stackoverflow.com/
8. https://numpy.org/