

ITCS 6150 – INTELLIGENT SYSTEMS

PROGRAMMING PROJECT 2 REPORT
SOLVING N-QUEENS PROBLEM USING HILL-CLIMBING AND ITS
VARIANTS

Project By:

Deepak Veerapandian – 801100869

Rishikumar Gnanasundaram – 801101490

PROGRAMMING PROJECT 2 REPORT

SOLVING N-QUEENS PROBLEM USING HILL-CLIMBING AND ITS VARIANTS

Introduction:

N-Queens:

The N-queens puzzle is the problem of placing N chess queens on an N×N chessboard so that no two queens threaten each other; thus, a solution requires that no two queens share the same row, column, or diagonal. The eight queens puzzle is an example of the more general n-queens problem of placing n non-attacking queens on an 8×8 chessboard, for which solutions exist for all-natural numbers n with the exception of n=2 and n=3.

Hill-Climbing:

Hill Climbing is heuristic search used for mathematical optimization problems in the field of Artificial Intelligence. Given a large set of inputs and a good heuristic function, it tries to find a sufficiently good solution to the problem. This solution may not be the global optimal maximum. The mathematical optimization problems imply that hill climbing solves the problems where we need to maximize or minimize a given real function by choosing values from the given inputs. Example- N-queen problem where we need to minimize the distance traveled by salesman. 'Heuristic search' means that this search algorithm may not find the optimal solution to the problem. However, it will give a good solution in reasonable time.

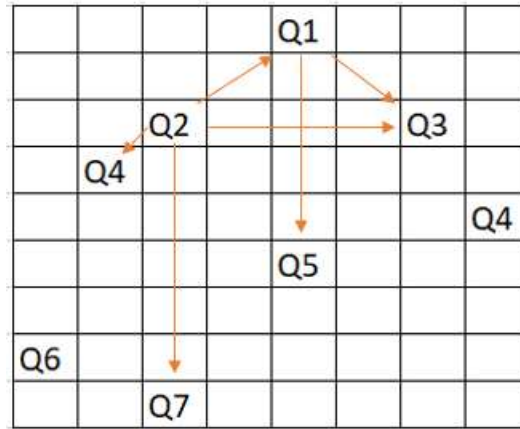
In Hill climbing, there is no need to maintain a search tree. It only records the state and the value of the objective function. At each step the current node is replaced by the best neighbor. In this case, the child with least heuristic is chosen and there is no need to store the remaining children.

					Q1		
	Q2						
						Q3	
Q4							
		Q5					
				Q6			
							Q7
			Q8				

8-QUEEN Solution

Heuristic Function $h(n)$:

The heuristic cost function h is the number of pairs of queens that are attacking each other, either directly or indirectly.



In this placements of queen, the attack for each queen is calculated. This can be done by checking the row, column and diagonal of each queen. Here the queen Q1 is attacked by Q5 and Q3. The queen Q2 is attacked by Q4, Q1, Q3 and Q7. So, the heuristic is the sum of all the attacks. Therefore, the heuristic of this combination is 6. $H(n) = 6$

N-Queen using Hill-Climbing:

Problem Formulation:

We can get the input from the user regarding the size of the chess board in which he wishes to solve the problem. when the user enters the choice of board we will now initialize the array that corresponds the board size. The array is now filled with hyphens and then a variable Q is placed in each column with a random row count and therefore a random configuration is generated. In this process we are going to restart the game for 500 times. We are checking whether the board is a goal. If it is a goal, we are returning it as a success move else we will have to generate its children which is 56 possible combination of boards. Once it is generated we will have to find the heuristics of all the 56 states. We are sorting the list based on the heuristics and then we will compare the heuristics of parent and the children in the first position of the children list. If `isSideways` is true the we check for condition `parent heuristics >= child heuristics`. If `isSideways` is false then we need check for `parent heuristics > child heuristics`. In any of the other conditions we consider it a failure move. This process will be continued until we complete 500 restarts. The other attempt is to give a random restart where we restart the board until we get a success. The random restart will be performed for both with sideways moves and without sideways moves.

There are 8 queens in the board. Each one of them is placed in each column. So the number of different combinations that can be created from the current node are 56.

$$\text{Number of Successors} : 8 * 7 = 56$$

Program structure and functions:

The code contains two classes **hillClimbing** and **Node**. hillClimbing is the main class which contains all the functions. The global variables and functions used are listed below.

- ArrayList<>
 - Output – list that contains the list of all possible children
 - tempChild – list that contains the children with same heuristic
- int REPEATS – number of allowed moves for sideways move
- int RESTARTS – total number of restarts
- int N – number of queen
- boolean isSideWaysMove – flag to enable sideways move
- boolean isRandomRestart – flag to enable random restart
- boolean isGoal – flag to decide if goal is achieved
- boolean isFailure – flag to decide if failure is achieved
- int successSteps – average number of success steps
- int failureSteps – average number of failure steps
- int averageSteps – average number of steps for random restart
- Class Node – generates memory for every child
 - Child – combination of queens
 - Heuristic – heuristic value
- Class heuristicComparator – sorts the children in ascending order of heuristic value

The functions used in the code are explained below.

main ():

We will get the size of the array(board). We will execute few statements in while loop until it reaches the restart value. we are inserting hyphens in all the positions of the array. We have to insert Q that represents queen in each column in a random row, We perform a goalCheck if it is true then we add it to success attempts else it is added to failure attempts. Total attempts will be increased that will be executed any way even if the condition fails or succeeds. Success attempts divided by total number of steps will give the total success.

goalCheck():

We will first remove the output list if the board is not a goal we proceed forward and create the children and if the heuristics of the first element in the children list is zero then it means that goal

is achieved. The board of the first child list is copied and then if sideways moves is true it succeeds for parent heuristics \geq child heuristics if the sideways moves is false it succeeds for parent heuristics $>$ child heuristics. If the sideways moves is true then when the parent and the child heuristics are equal then we will proceed for another 100 times until the parent heuristics become greater than child heuristics. If it does not become greater even after 100 moves we make that configuration a failure.

Heuristics():

The heuristics function helps us to find the heuristics of a particular board. We first find the row in which the queen is present in the column that we are searching. In the first if-condition we need to check whether any queen is present in the same row and subsequently we need to check the lower and the upper diagonal. The heuristics in the board means the total number of queens that are in attacking position.

createChildren():

This function helps in creating 56 different children for each and every board. The board is taken as the input and then the Q is placed in different row positions in all the columns. In the end the child list is now sorted based on the heuristics.

arrayCopy():

This function is used to make a copy of the board.

arrayprint():

This function is used to print the board.

OUTPUTS and OBSERVATIONS:

Steepest ascent hill climbing:

The code was run for 500 times to calculate the average success rates and average number of steps required for success in Steepest ascent hill climbing.

- 1.Success and failure rates for 500 restarts - Failure:430 times, Success:70 times, Success% : 14%, Failure% : 86%
- 2.The average number of steps when it succeeds-3 steps
- 3.The average number of steps when it failed-4 steps
- 4.The search sequences from four random initial configurations - attached in the document

Hill-climbing search with sideways move:

The code was run for 500 times to calculate the average success rates and average number of steps required for success in Hill climbing with sideways moves.

- 1.Success and failure rates for 500 restarts - Failure:31 times, Success:469 times, Success% : 93.8%, Failure% : 6.2%.
- 2.The average number of steps when it succeeds - 17 steps.
- 3.The average number of steps when it failed - 62 steps.
- 4.The search sequences from four random initial configurations - attached in the document.

Random-restart hill-climbing search:

The random generation was run for a total of 100 times to calculate the average number of restarts required and average number of steps.

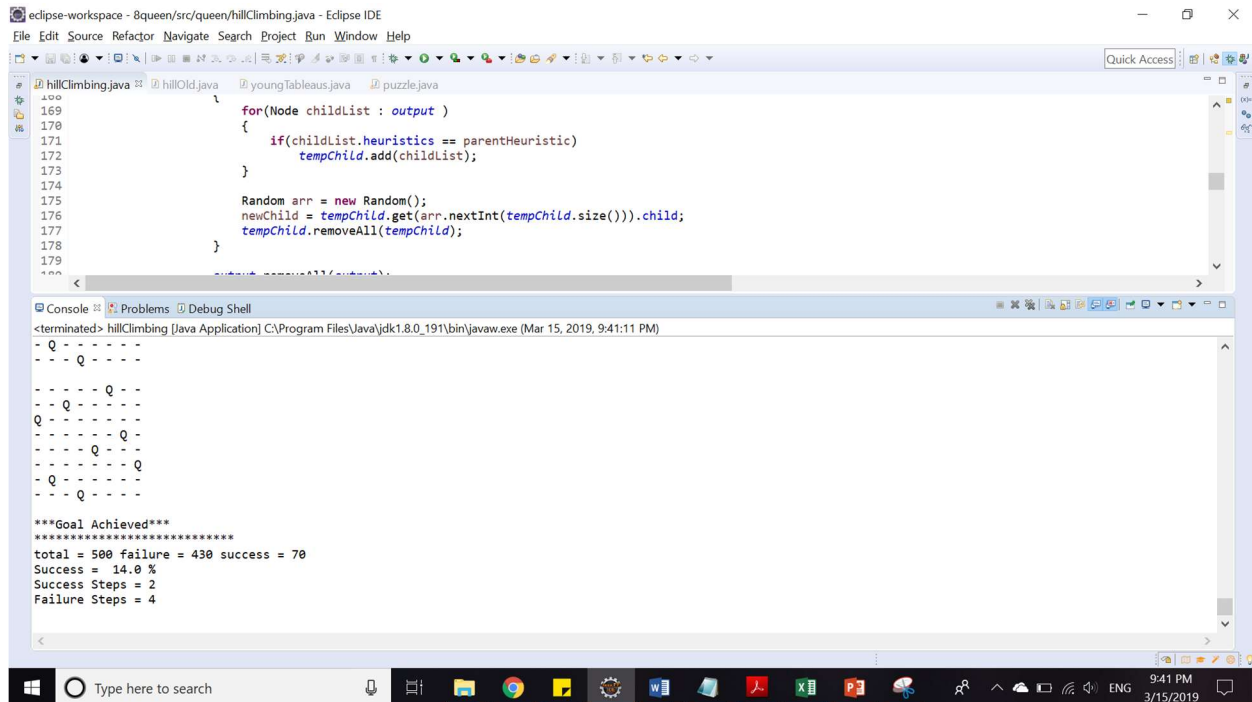
- 1.The average number of random restarts required without sideways move - 7.
- 2.The average number of steps required without sideways move - 29.
- 3.The average number of random restarts used with sideways move - 1.
- 4.The average number of steps required with sideways move - 19.

NOTE:

The outputs are attached in the text files.

SAMPLE OUTPUTS:

Steepest ascent hill climbing:



```
edclipse-workspace - 8queen/src/queen/hillClimbing.java - Eclipse IDE
File Edit Source Refactor Navigate Search Project Run Window Help

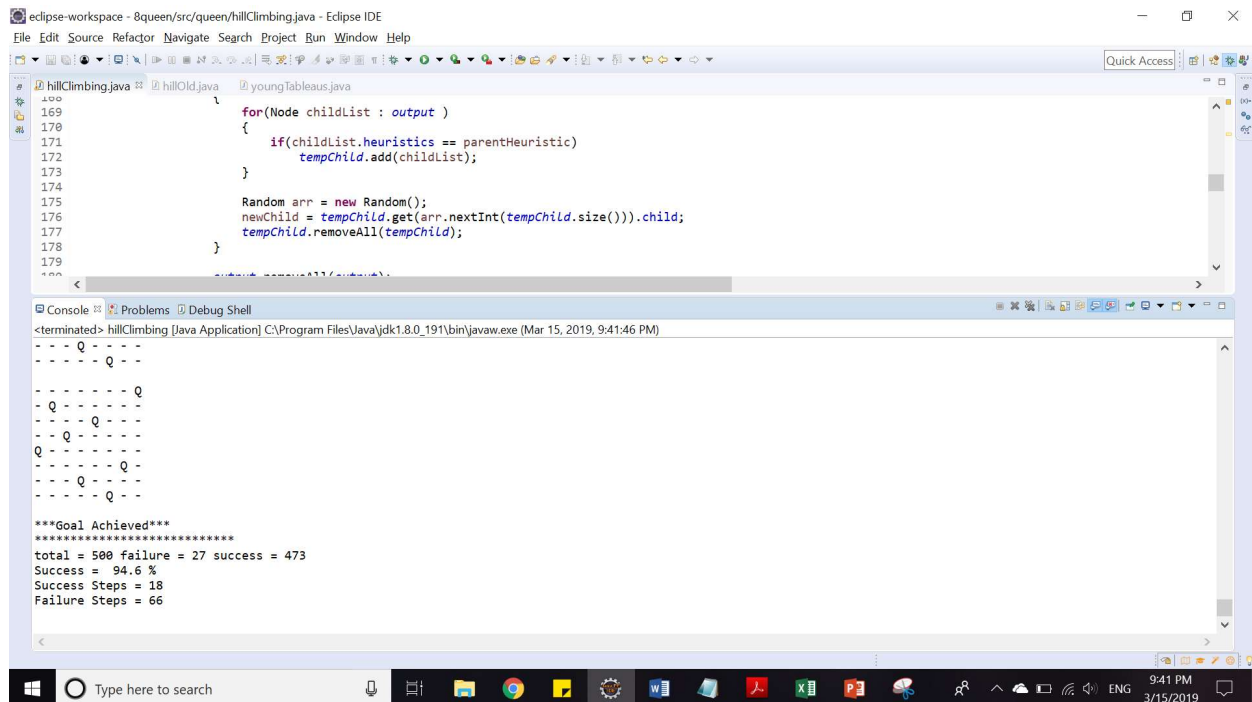
169
170
171
172
173
174
175
176
177
178
179

    for(Node childList : output )
    {
        if(childList.heuristics == parentHeuristic)
            tempChild.add(childList);
    }

    Random arr = new Random();
    newChild = tempChild.get(arr.nextInt(tempChild.size()));
    tempChild.remove(tempChild);
}

***Goal Achieved***
*****
total = 500 failure = 430 success = 70
Success = 14.0 %
Success Steps = 2
Failure Steps = 4
```

Hill-climbing search with sideways move:



```
edclipse-workspace - 8queen/src/queen/hillClimbing.java - Eclipse IDE
File Edit Source Refactor Navigate Search Project Run Window Help

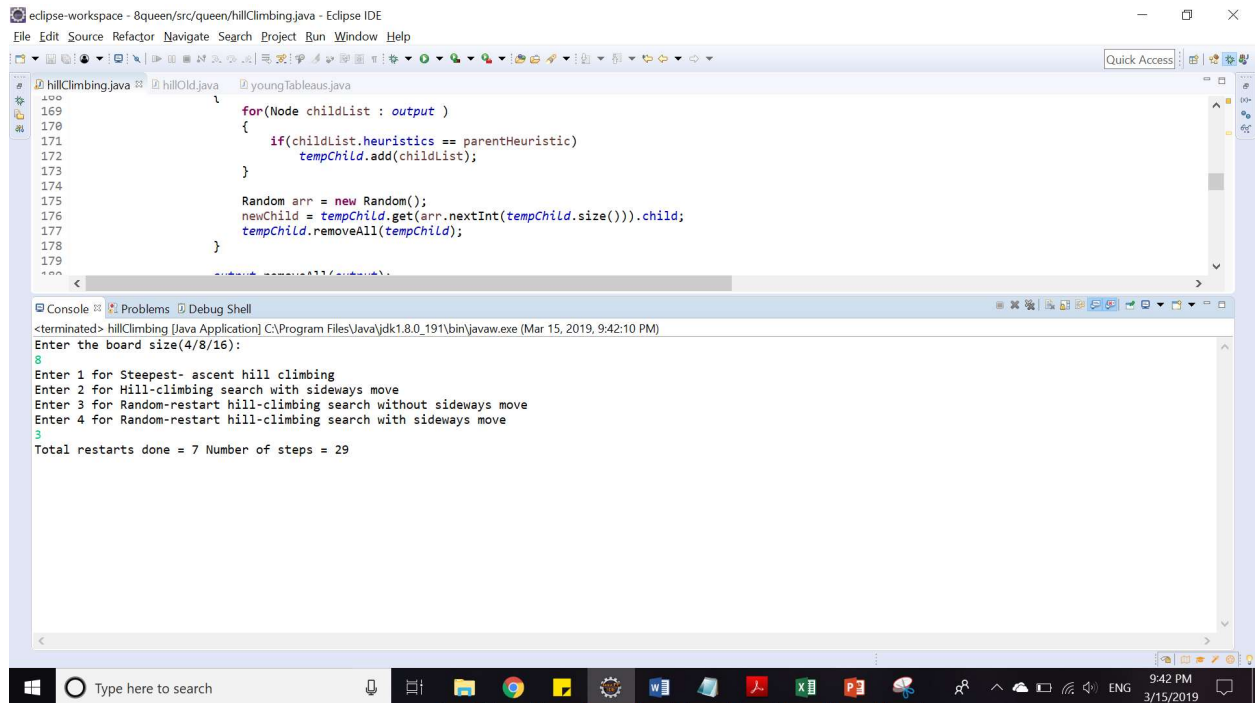
169
170
171
172
173
174
175
176
177
178
179

    for(Node childList : output )
    {
        if(childList.heuristics == parentHeuristic)
            tempChild.add(childList);
    }

    Random arr = new Random();
    newChild = tempChild.get(arr.nextInt(tempChild.size()));
    tempChild.remove(tempChild);
}

***Goal Achieved***
*****
total = 500 failure = 27 success = 473
Success = 94.6 %
Success Steps = 18
Failure Steps = 66
```

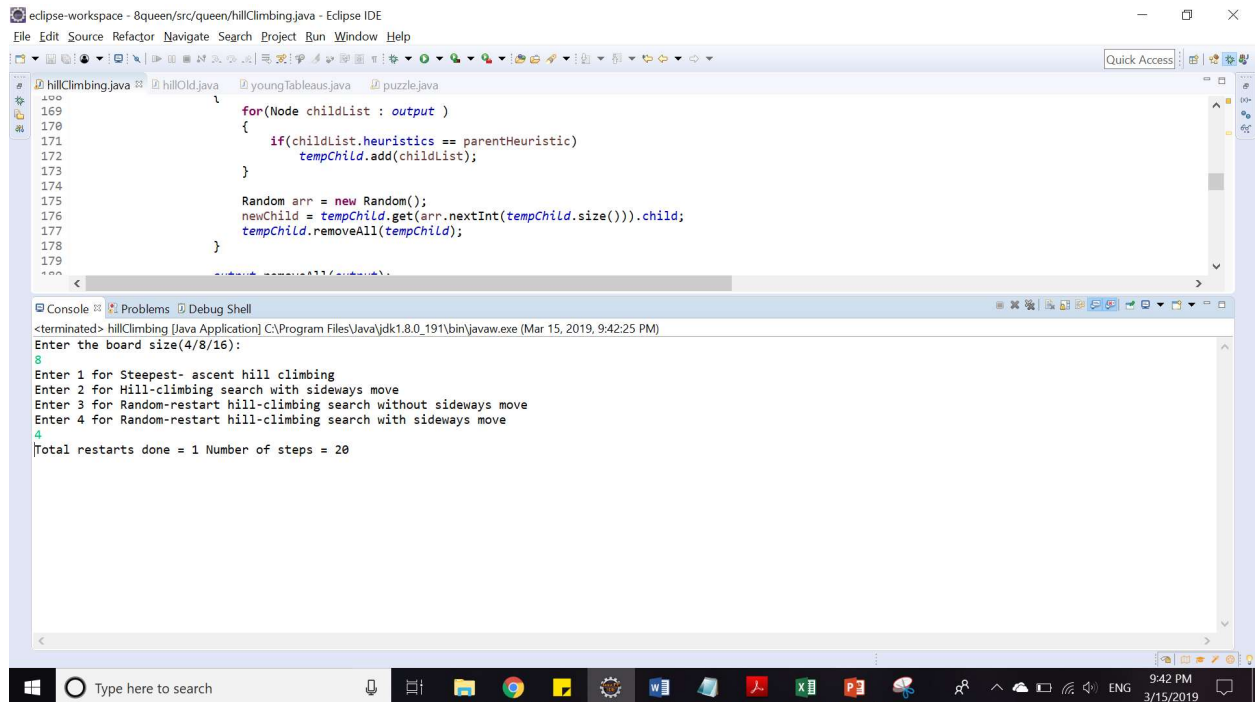
Random-restart hill-climbing search without sideways move:



The screenshot shows the Eclipse IDE with the file `hillClimbing.java` open. The code implements a random-restart hill-climbing search for the 8-queens problem. The console output shows the program execution details:

```
<terminated> hillClimbing [Java Application] C:\Program Files\Java\jdk1.8.0_191\bin\javaw.exe (Mar 15, 2019, 9:42:10 PM)
Enter the board size(4/8/16):
8
Enter 1 for Steepest- ascent hill climbing
Enter 2 for Hill-climbing search with sideways move
Enter 3 for Random-restart hill-climbing search without sideways move
Enter 4 for Random-restart hill-climbing search with sideways move
3
Total restarts done = 7 Number of steps = 29
```

Random-restart hill-climbing search with sideways move:



The screenshot shows the Eclipse IDE with the file `hillClimbing.java` open. The code implements a random-restart hill-climbing search for the 8-queens problem, including sideways moves. The console output shows the program execution details:

```
<terminated> hillClimbing [Java Application] C:\Program Files\Java\jdk1.8.0_191\bin\javaw.exe (Mar 15, 2019, 9:42:25 PM)
Enter the board size(4/8/16):
8
Enter 1 for Steepest- ascent hill climbing
Enter 2 for Hill-climbing search with sideways move
Enter 3 for Random-restart hill-climbing search without sideways move
Enter 4 for Random-restart hill-climbing search with sideways move
4
Total restarts done = 1 Number of steps = 20
```