**Programming Project 2**

**Solving N-queens problem by Hill-Climbing and its Variants**

**1. Introduction**

**1.1 N-Queens puzzle**

The N queens puzzle is the problem of placing N chess queens on an N×N chessboard suc 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 8 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.

**1.2 Hill Climbing Algorithm**

Hill Climbing Algorithm is a local search algorithm which does not take the path cost to the goal in consideration. It is simply a loop that continually moves in the direction of increasing value—that is, uphill. It terminates when it reaches a “peak” where no neighbor has a higher value. The algorithm does not maintain a search tree, so the data structure for the current node need only record the state and the value of the objective function. Hill climbing does not look ahead beyond the immediate neighbors of the current state.

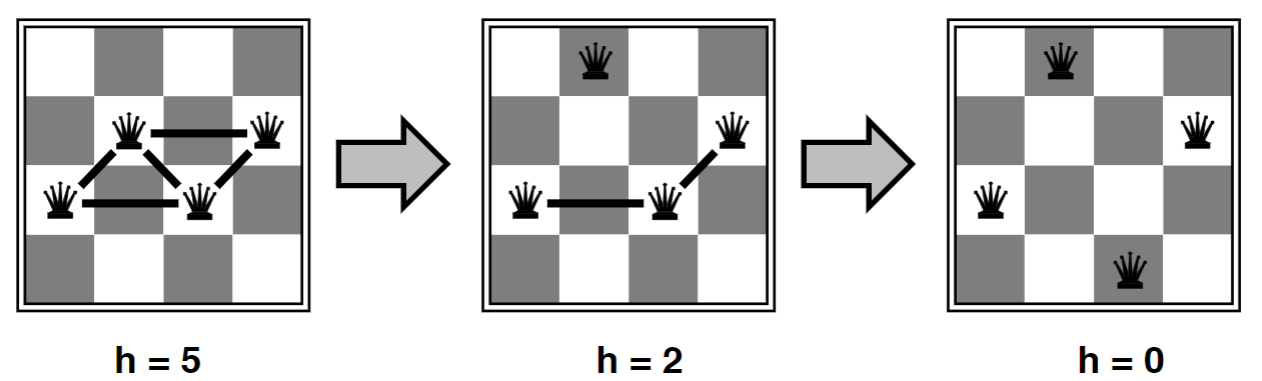


Fig 1.1 Example of 8-Queen problem

We will illustrate the following variations of Hill Climbing Algorithm using the N-Queens Puzzle.

* **Steepest-Ascent Hill Climbing:**

We calculate heuristic based on every queen moved in the column to a particular row. We choose the move that gives us least possible attacks. If there are no moves that make the situation better the program fails.

* **Hill Climbing with sideways moves**

We execute the same procedure as in Steepest Ascent but instead of declaring it as a failure when there are no moves with lesser heuristic value, we choose a heuristic with the same value and hope to make the program succeed.

* **Random Restart with and without sideways moves**

In this variation, we generate a new board configuration every time we don’t find a lesser or equal heuristic.

**2. Program Structure**

The project has been implemented using Python 3.7.0. The program consists of a class called Hill Climbing. The Queen is represented using 1 and other positions are filled with 0.

1. **Steepest Ascent Hill Climbing**

**Class Variables**

* board- stores the board as a list of list
* global\_max – stores the heuristic of the current board
* count\_i – counts the number of horizontal attacks
* col – stores the number of columns of board
* pos\_list – stores the positions of queen
* temp\_list – temporary list to stores the position of queen
* heuristic\_board – stores heuristic for every move
* new\_state – stores the configuration generated after making a move
* success – number of success
* failure – number of failures
* status – if the iteration is running, status is 1 else it is 0
* counter – keeps track of iteration number
* input\_count – number of iterations taken as input from user
* s\_count – total number of steps when it succeeds
* f\_count – total number of steps when it fails
* w\_count – number of steps in each iteration

**Functions**

1. **HC\_initial ():**

It acts as the driver function from which all other functions are called and this function keeps running till the number of iterations required are completed

1. **heuristic (board):**

It takes the board as a parameter and calculates the total number of attacks for that particular board configuration. The attacks can be horizontal or along the diagonal and direct or indirect.

* **Local Variable**

**r\_attack -** number of attacks in reverse diagonal

**f\_attack -** number of attacks in forward diagonal

**attack –** totalnumber of attacks in the board

**count\_i -** number of attacks in the same row (horizontal attacks)

1. **hb ():**

Creates a board with heuristic values in a every position that shows how many attacks are possible if the queen is moved to that position.

* **Local Variable**

**x -** gives the row number of Queen

**y –** gives the column number of Queen

1. **Selection (heuristic\_board):**

This function is used to select the move that will have least number of attacks and create a new board by making the move.

* **Local Variable**

**temp** – It is a temporary variable used to find the minimum h(n)

**min\_value** – It stores the minimum h(n) value from the heuristic\_board

1. s**ort ():**

Used to sort the list which consists positions of Queens in the board based on increasing value of column

1. **Hill Climbing with Sideways Moves**

The variables and the functions are same as in 2.A only the below parts have been changed/added

**Class Variables**

* side\_count – number of sideways moves that have been made

**Functions**

1. **selection (heuristic\_board):**

This function is used to select the move that will generate the least number of attacks and create a new board by making the move. If there are no moves that have lesser number of attacks than the current board then a move that creates a board with equal number of attacks is chosen.

* **Local Variable**

**temp** – It is a temporary variable used to find the minimum h(n)

**min\_value** – It stores the minimum h(n) value from the heuristic\_board

**min\_list** – stores the value and index of heuristic equal to heuristic of

current board

1. **Random Restart without Sideways**

The variables and the functions are same as in 2.A only the below parts have been added

**Class Variables**

* init\_count – For checking if we need to generate a new board
* r\_count – number of random restarts done
* t\_count – total number of steps in each iteration

1. **Random Restart with Sideways**

The variables and the functions are same as in 2.A only the below parts have been changed/added

**Class Variables**

* side\_count – number of sideways moves that have been made

**Functions**

1. **selection (heuristic\_board):**

This function is used to select the move that will generate the least number of attacks and create a new board by making the move. If there are no moves that have lesser number of attacks than the current board then a move that creates a board with equal number of attacks is chosen.

* **Local Variable**

**temp** – It is a temporary variable used to find the minimum h(n)

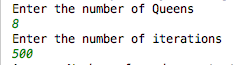
**min\_value** – It stores the minimum h(n) value from the heuristic\_board

**min\_list** – stores the value and index of heuristic equal to heuristic of

current board

**2.1 User Input**

The user has to give the size of the board and number of iterations as input.



**2.2.1 Functionality of Hill Climbing with and without Sideways move**

* Initially, the number of queens and the number of iterations is taken from the user.
* Accordingly, a new board is created which is of size Queen\*Queen. This board is created by randomly placing one queen in each column of the board. Thus, for number of queens = 8, the board size is 8\*8.
* After creation of this board, the total number of attacks i.e. the Global Maximum is calculated by calling out the heuristic ().
* It is first checked if the Global Maxima is already 0 i.e. there are no attacks and the randomly generated initial state is already the goal state. If yes, then the control goes out of the loop making status=0 and success=1 and thus new iteration is run.
* If the Global Maximum is not equal to 0, then the hb () is called out which calculates the heuristic board. Every element of the heuristic board defines the heuristic value calculated when the queen is places in location of the said element. The Global Maximum values are also placed on the heuristic board with an increment of 1 so that they are not selected again while minimum heuristic selection.
* After creation of the heuristic board, the selection () is called out which selects the best local maximum lesser than the Global Maximum.

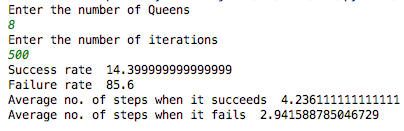
* Depending upon the selection function used, the algorithm can go either sideways or not.
* If it is not required to go sideways, then the best local maximum is selected, and a new board configuration is formed which replaces the old board (new initial state). The program continues to calculate the Global Maximum and so on. The success condition would be when local maximum is 0 due to which the Global Maximum of the newly generated board would be 0. This is counted as a success. The failure condition would be when the local maximum is lesser than the Global Maximum.
* If it is required to go sideways, then there will be a point when the local maximum will be equal to the global maximum. Thus, we can choose to select a random local maximum with the same value and continue our program execution. Just so that the program does not end up in an infinite loop, a side counter = 100 has been applied. The program will keep on finding a local maximum which has a value is lesser than the Global Maximum and thus the goal state. In case it doesn’t, and the side count value has been reached, then this counts a s a failure and a new board for new iteration is generated.

**2.2.1 Functionality of Random Restart Hill Climbing.**

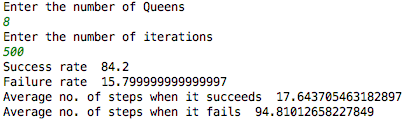
* The functionality of Random Restart differs from Steepest Hill Climbing by making sure that the Success state has been reached by restarting a board when it reaches failure. The program execution remains almost similar, but the iteration is stopped when a success state is reached. So, the probability of reaching a goal state is 1. The functionality differs that new iteration begins only after a success state has been reached.

**3. Sample Output**

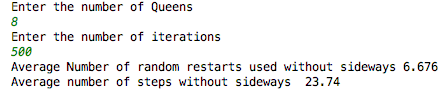
**3.1 Steepest Ascent Hill Climbing**

****

**3.2 Hill Climbing with Sideway Moves**



**3.3 Random Restart without Sideway Moves**

****

**3.4 Random Restart with Sideway Moves**

