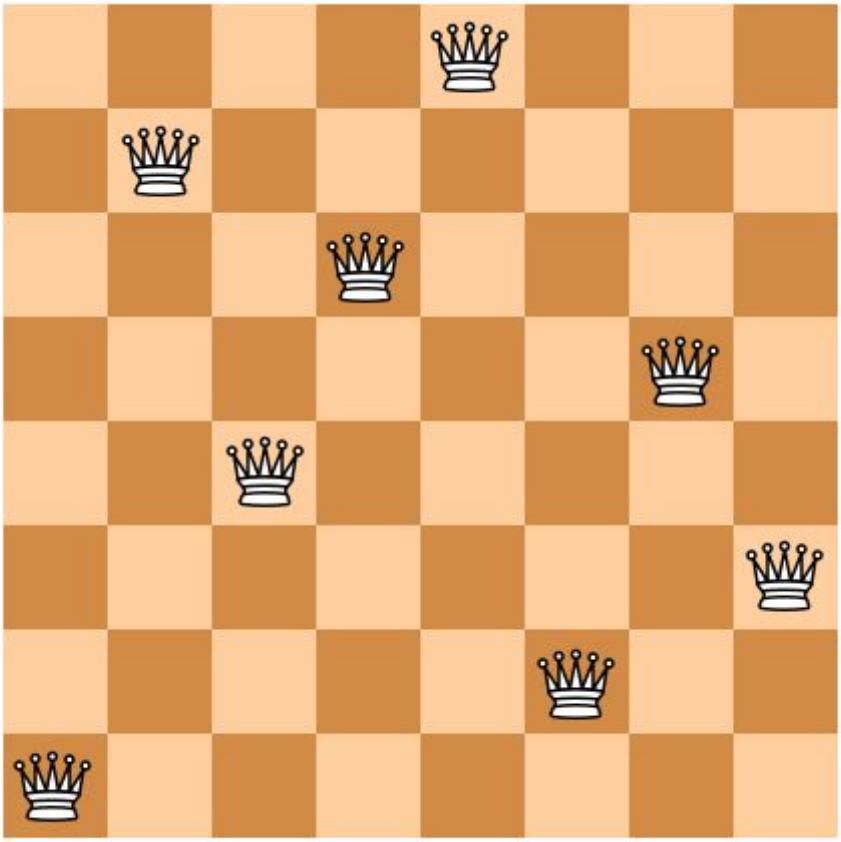
Artificial Intelligence

Assignment 2

Using Local Search and Constraint Satisfaction Algorithms to Solve 8-Queens



Team Members

Alaa Shehab (1)

Sohayla Mohammed (31)

Youmna Dwidar (73)

Introduction

Overview

The eight queens problem is the problem of placing eight queens on an 8×8 chessboard such that none of them attack one another (no two are in the same row, column, or diagonal). More generally, the n queens problem places n queens on an n×n chessboard.

Problem Statement

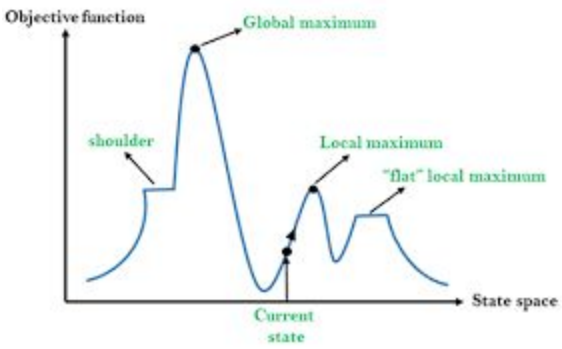
Apply search techniques to solve the 8 queens; Given an initial configuration of 8×8 chessboard, the algorithm should find any valid final configuration starting from the given initial configuration.

Approach

The following algorithms are used to solve the problem :

1. Hill Climbing Algorithm
2. K-Beam Search Algorithm.
3. Genetic Algorithm.
4. Constraint Satisfaction Problem (CSP).

Hill Climbing



Hill Climbing is a 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.

- '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.
- A heuristic function is a function that will rank all the possible alternatives at any branching step in search algorithm based on the available information. It helps the algorithm to select the best route out of possible routes.

K-beam search

Beam search is a heuristic search algorithm that explores a graph by expanding the most promising K node in a limited set. Beam search is an optimization of best-first search that reduces its memory requirements. Best-first search is a graph search which orders all partial solutions (states) according to some heuristic. But in beam search, only a predetermined number of best partial solutions (k) are kept as candidates. It is thus a greedy algorithm.

Genetic algorithm

A genetic algorithm is a search heuristic that is inspired by Charles Darwin's theory of natural evolution. This algorithm reflects the process of natural selection where the fittest individuals are selected for reproduction in order to produce offspring of the next generation.

Constraint Satisfaction Problem (CSP)

In constraint satisfaction, local search is an incomplete method for finding a solution to a problem. It is based on iteratively improving an assignment of the variables until all constraints are satisfied. In particular, local search algorithms typically modify the value of a variable in an assignment at each step. The new assignment is close to the previous one in the space of assignment, hence the name local search.

Implementation

Shared Functions

Read from file

Main functions

- getInputState2DChars
- getInputState2DBoolean
- getInputState1D

Those 3 functions use a private function called readFile which reads the input file into three variable of each type.

If the readFile is called once and any other time, the data structures are returned.

Check board correctness

- While reading the file , each line is checked to have a length of 8 .
- After reading all lines , a counter with the number of lines is checked to be 8.

Hill Climbing

Pseudo code

```
inputs: problem, a problem
local variables: current, a node
                 neighbor, a node
current <-Make_Node(Initial-State[problem])
loop
   do neighbor <- a highest_valued successor of current
      if Value[neighbor] ≤ Value[current] then
      return State[current]
      current <- neighbor

end
```

Algorithm

- Check if the initial board is the solution if yes print solution
- Get best next state that has least attacking queens
- If the best next state has less heuristic than its parent then update, else generate a new random state and do it over again.

Parameters

- The initial board as 2d array of boolean

Data Structure

- 1d array of points as 2d queen representation

Assumptions

- No assumptions

Notes

- The expanded nodes are the (8 rows * 8 columns * 16 diagonals) * 8 Queens for each step

And the steps to the final solution are the actual expanded nodes.

K-Beam

Pseudo code

```
start with k randomly generated states
loop
   generate all successors of all k states
   if any of the states = solution, then return the state
   else select the k best successors
end
```

Parameters

- K.
    - Number of states checked at each step.
- Number of iterations.
    - Maximum number of steps taken.

Data Structure

- Point[]
    - State of the board, it has size = number of queens.
- List<Point[]>
    - The pool which contains the current K states or the neighbours of the K states.

Description

- Generate K random states.

    - Either based on an initial state or not.
- Loop over the states if the solution is one of them return it, otherwise generate all possible neighbours of each of the k states.
- The neighbours are considered based on :
    - Moving each queen on board :
        - In its row.
        - In its col.
        - Diagonally to the left.
        - Diagonally to the right.
    - Random state if no viable state is found from moving a queen.
- Sort the neighbours based on the heuristic, then take the k best and loop again.

Assumptions

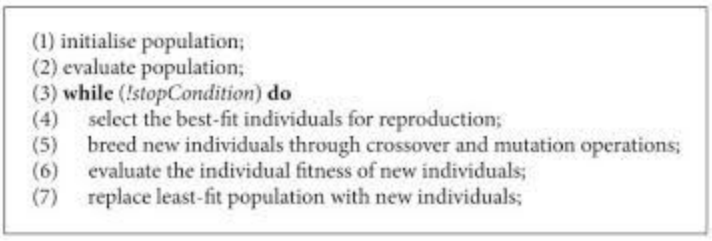
- The algorithm can start with an initial state or random states.
- If an initial state is provided the algorithm starts with getting k best neighbours of that state.
- While getting the neighbors if a queen move is not considered, a random state is considered in place of it. Thus we have k states at any given time.
- State may be visited more than once.

Notes

- Heuristic is based on number of attacking pairs on board.

Genetic Algorithm

Pseudo code



Algorithm

- Randomly generate 100 chromosomes
- Set the fitness of each chromosome
- Sort the population according to the probability of selection
- Remove the least 50 chromosomes whose probability is low
- Generate new offspring through mutation of 2 consecutive parents
- Add new generation to the population
- Loop until convergence or until reaching a max iteration

Parameters

- 1d array representing the position of each queen in its row

Data Structure

- Chromosome DS which simulate the genes, and contains 1d int array of data
- A set of String to ensure that no redundant chromosomes exists

Assumptions

- Initial state of the queens assumed to have each queen in seperate column
- Population size is set to 100

Notes

- Initial state is always added to the population
- 25 new chromosomes are generated with every run.

Constraint Satisfaction problem

Pseudo code

Backtracking :

Backtracking search

```
function BACKTRACKING_SEARCH(csp) returns a solution, or failure
    return BACKTRACK({}, csp)

function BACKTRACK(assignment, csp)
returns a solution, or failure
    if assignment is complete then return assignment
    var = SELECT_UNASSIGNED-VARIABLES(csp)
    for each value in ORDER_DOMAIN_VALUES (var, assignment, csp)
        if value is consistent with assignment then
            add {var = value} to assignment
            result = BACKTRACK(assignment, csp)
            if result ≠ failure then return result
            remove {var = value} from assignment
    return failure
```

Backtracking with forward Checking :

Same as backtracking but after each assignment , the domains of the unassigned queens is shrunk according to the constraints applied by the new assigned queen.

Parameters

Backtracking : The initial state is entered.

Backtracking with forward Checking : No parameters is inputted

Data Structure

Class SolutionPrinter that prints the solution with the same format as the Input.

- The state is represented by a 1D array.
- Each index represents a column.
- A value in the array represent the row which the queen is placed.

- In forward checking, an int 2D array in used to count the number of queens threating a certain cell.

Description

Backtracking :

- We iterate on every possible assignment for each Index.
- With each assignment , we check whether it violates any of the constraints which here are :
    - No two queens on the same row (no two elements in the array have the same value);
    - No two queens has the same difference on the row and the column side.(not on the diagonal |i-j| != |Q[i]-Q[j]|
- If the assignment is valid, we continue to the next column (queens) to assign a new variable.
- If the assignment isn't valid, we continue searching for a valid one for the current variable.
- If no valid assignment found, we backtrack to have different assignment of the previous variable.
- If solution is found , it 's printed.

ForwardChecking :

Same as backtraning but :

- After assigning a new variable, we calculate the threatened cells by the new assignment and make sure that the next variables has at least one valid assignment (there is a cell that is not threatened in each column)
- It valid continue.
- It not , we backtrack.

Assumptions

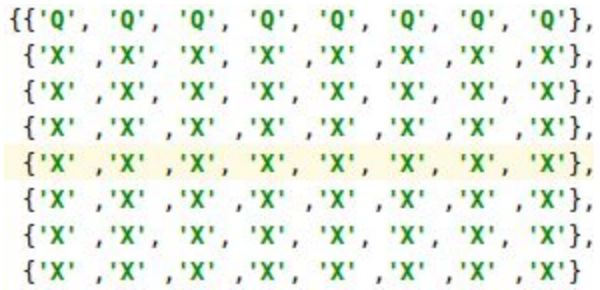
- Starting from an empty state and searching the whole search space.

- If an initial state is inputted , only in backtracking can the steps and the min cost of solution is found and printed.

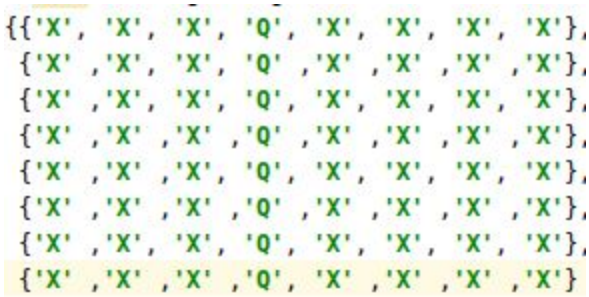
Performance

K-Beam

With initial state :

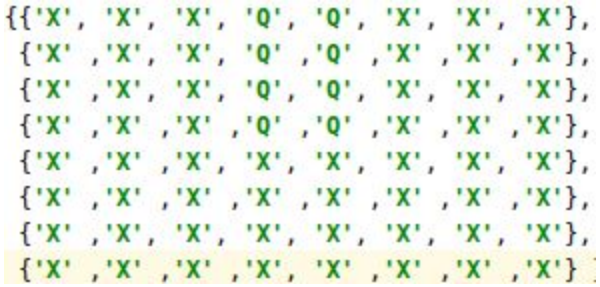


	K = 1	K = 5	K = 25	K = 125	K = 625
Running time	139.0	7.0	13.0	47.0	262.0
Number of steps	46	4	2	2	2
Number of expanded nodes	7651	1973	3180	16226	79821

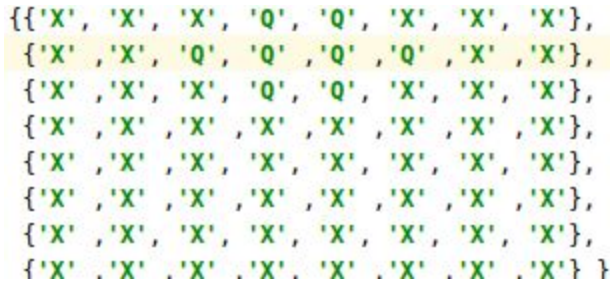


	K = 1	K = 5	K = 25	K = 125	K = 625

Running time	89.0	19.0	4.0	30.0	52.0
Number of steps	7	6	2	2	1
Number of expanded nodes	1059	3783	1632	13002	622



	K = 1	K = 5	K = 25	K = 125	K = 625
Running time	131.0	4.0	41.0	84.0	316.0
Number of steps	9	2	2	2	2
Number of expanded nodes	1328	570	3541	16895	79680



	K = 1	K = 5	K = 25	K = 125	K = 625
Running time	102.0	8.0	6.0	23.0	95.0

Number of steps	11	4	1	1	1
Number of expanded nodes	1658	2005	1188	125	625

With random initial states:

	K = 1	K = 5	K = 25	K = 125	K = 625
Running time	105.0	14.0	20.0	73.0	64.0
Number of steps	18	6	1	2	1
Number of expanded nodes	2787	4080	1741	15308	10212

	K = 1	K = 5	K = 25	K = 125	K = 625
Running time	127.0	7.0	27.0	95.0	134.0
Number of steps	10	3	2	2	1
Number of expanded nodes	1430	1480	5676	15133	16398

	K = 1	K = 5	K = 25	K = 125	K = 625
Running time	97.0	17.0	25.0	59.0	312.0
Number of steps	12	5	2	2	2
Number of expanded nodes	1802	3192	3112	15983	78144

On average :

	K = 1	K = 5	K = 25	K = 125	K = 625
Avg Running time	112.857	**10.85**	19.43	58.7	176.43
Avg Number of steps	16.14	4.28	1.714	1.85	**1.42**
Avg Number of expanded	**2350.71**	2440.43	2917.143	13238.86	37928.86

nodes					

Sorted by running time :

1. K = 5.
2. K = 25.
3. K = 125.
4. K = 1.
5. K = 625.

Sorted by Steps :

1. K = 625.
2. K = 25.
3. K = 125.
4. K = 5.
5. K = 1.

Sorted by expanded nodes :

1. K = 1.
2. K = 5.
3. K = 25.
4. K = 125.
5. K = 625.

Notes :

- When K is too small, in our test = 1.
    - Number of steps is really high, and time is thus high too.
    - Choosing K = 1 is not efficient in our algorithm since it'll not differ from hill climbing algorithm and the agent does not converge to solution faster on average.
- When K is too large, in our case = 625

    - Number of nodes visited and seen is very large although number of steps is almost one in most cases which makes sense, since we see most of the states that we might be in in the future.
    - Timing is not great on average either and most of the time is consumed during visiting states.
- When K is in between :
    - Timing on average is better than the two extremes.
    - Number of steps is almost 2 on average, which is great.
    - Number of expanded nodes is larger than the low extreme but not by much, in contrast it's significantly smaller than the higher extreme.
- We've decided to use K = 5,
    - based on our tests it achieves best running time.
    - Also an overall best average performance compared with the rest.

Constraint Satisfaction problem

It begins with the initial state as an empty state and works in two versions :

Backtracking :

Nodes Expanded	Average Running Time (5 runs)	Number of Solutions
1965	65	92

Backtracking with Forward Checking :

Propagating the current assignment to limit the next queens assignment.

Nodes Expanded	Average Running Time (5 runs)	Number of Solutions
1073	77	92

How to run

You will have a menu printed to enter an input indicating the algorithm you want to run on your input.

```
Select the Algorithm you need :
1) Hill Climbing Algorithm
2) K-Beam Algorithm
3) Genetic Algorithm
4) Constraint Satisfaction Problem (CSP) BackTracking
5) Constraint Satisfaction Problem (CSP) with Forward Checking
For exit press 0
```

Sample runs

1)

```
1
##Q####Q
#####Q##
###Q####
#Q######
#######
####Q###
######Q#
Q######
Running time : 12.0
Number of Random generated States : 0
Number of boards visited : 1
Number of Nodes Expanded : 338
##Q#####
#####Q##
###Q####
#Q######
#######Q
####Q###
######Q#
Q#######
```

2)

```
2
Running time : 110.0
Number of boards visited : 6
Number of Nodes Expanded : 590
5 2 0 7 3 1 6 4
```

3)

```
#Of steps : 2
#Of expanded : 13318
Running Time(ms) : 54.0
```

4)

```
=========================
 7  3  0  2  5  1  6  4
 #  #  Q  #  #  #  #  #
 #  #  #  #  #  Q  #  #
 #  #  #  Q  #  #  #  #
 #  Q  #  #  #  #  #  #
 #  #  #  #  #  #  #  Q
 #  #  #  #  Q  #  #  #
 #  #  #  #  #  #  Q  #
 Q  #  #  #  #  #  #  #
=========================
Total Running Time : 38
Nodes Expanded : 1965
Total Number of Solution : 92
Running time for the Initial State :0
Nodes Expanded the Initial State :0
Steps for the Initial State :1
```

5)

```
==========================
 7  2  0  5  1  4  6  3
 #  #  Q  #  #  #  #  #
 #  #  #  #  Q  #  #  #
 #  Q  #  #  #  #  #  #
 #  #  #  #  #  #  #  Q
 #  #  #  #  #  Q  #  #
 #  #  #  Q  #  #  #  #
 #  #  #  #  #  #  Q  #
 Q  #  #  #  #  #  #  #
==========================
 7  3  0  2  5  1  6  4
 #  #  Q  #  #  #  #  #
 #  #  #  #  #  Q  #  #
 #  #  #  Q  #  #  #  #
 #  Q  #  #  #  #  #  #
 #  #  #  #  #  #  #  Q
 #  #  #  #  Q  #  #  #
 #  #  #  #  #  #  Q  #
 Q  #  #  #  #  #  #  #
==========================
Total Running Time : 29
Nodes Expanded : 1073
Total Number of Solution : 92
```

O)

```
Select the Algorithm you need :
1) Hill Climbing Algorithm
2) K-Beam Algorithm
3) Genetic Algorithm
4) Constraint Satisfaction Problem (CSP) BackTracking
5) Constraint Satisfaction Problem (CSP) with Forward Checking
For exit press 0
0

Process finished with exit code 0
```