Assignment 8

Design and Analysis of Algorithm

IDAA432C

N Queens problem Using Backtracking

Submitted By:
Akshay Gupta **(IIT2017505)**
Naman Deept **(IIT2017507)**
Snigdha Dobhal **(IIT2017506)**

April 13, 2019

N Queens Problem

Akshay Gupta IIT2017505, Naman Deept IIT2017507 Snigdha Dobhal IIT2017506

April 13, 2019

1 Abstract

This paper enumerates solving the N Queens problem using the concept of backtracking and performing the complexity analysis of the problem which comes under the category of NP Hard problems. (NP stands for non-polynomial ,since the complexity in this case will not be in terms of a polynomial expression) Since the precise complexity can't be obtained for the generic problem solving, hence stochastic approaches will be considered in the calculations.
Keywords: *N Queens*, *Backtracking* ,*Recursion*,*NP Hard Problem*.

2 Introduction

N Queens Problem: The eight queens puzzle is the problem of placing eight chess queens on an 88 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 nn chessboard, for which solutions exist for all natural numbers n with the exception of n = 2 and n = 3. The n-queens problem is a classical example of a constraint-satisfaction problem. It often finds its way into AI classes as an exercise in programming a tree-search algorithm known as backtracking.

Contributions: Chess composer **Max Bezzel** published the eight queens puzzle in 1848.**Franz Nauck** published the first solutions in 1850. Nauck also extended the puzzle to the n queens problem, with n queens on a chessboard of nn squares.Since then, many mathematicians, including Carl Friedrich Gauss, have worked on both the eight queens puzzle and its generalized n-queens version. In 1874, S. Gunther proposed a method using determinants to find solutions. J.W.L. Glaisher refined Gunther's approach. In 1972, **Edsger Dijkstra** (Dutsch systems scientist, programmer and software engineer) used this problem to illustrate the power of what he called structured programming. He published a highly detailed description of a depth-first backtracking algorithm.

Finding number of possible solutions: Let's consider the 8×8 chessboard problem, if no restrictions were provided then we can simply say that having 8 queens and 64 squares ,the total number of possible ways of placing those queens will be ${}^{64}C_8$ which turns out to be $4,426,165,368$ (such a large number indeed).
But out of such a large number a very small portion of it 0.00002% only turns out to be the valid solution satisfying the condition that no queens placed in the board should attack each other. This is indeed a very small fraction, the value turns out to be 92 only out of those 4 billion possible solutions. If symmetry (about any axis) in any case is restricted ,meaning that the 2 symmetric solutions about any axis turns out to be the same, then value turns out to be even lesser 12 only indeed.For every asymmetric case ,we can have 8 possible sub-cases (4 by rotation and 4 by taking the mirror image of it across each of the edges of the chessboard).And we have 1 symmetric solutions where out of 8 sub-cases ,4 of them overlaps, hence only 4 valid sub-cases in the

case of symmetry. The only symmetric case in 8×8 chess board is 1. So the total possible ways are 8×11 + 4×1 = 92

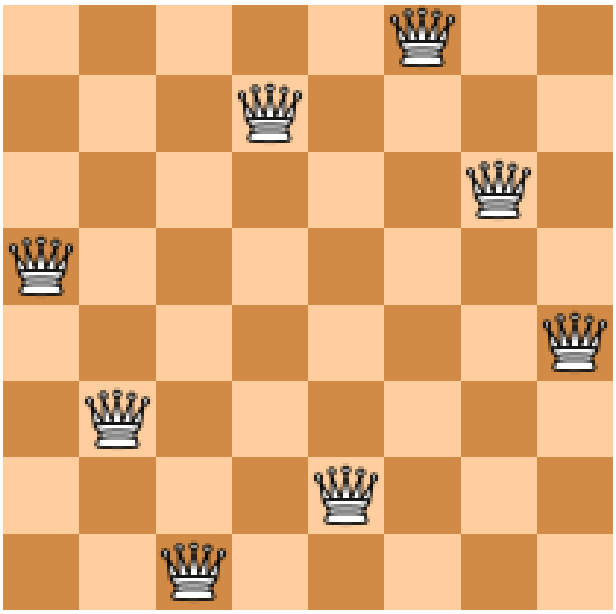

Figure 1: The only possible symmetric solution in 8 Queens Problem

If the symmetry in any of the case is considered as same solution and is not counted ,then the pattern for the N Queen problem increases in finding the number of solutions as :1, 0, 0, 1, 2, 1, 6, 12, 46, 92, 341, 1,787, 9,233, 45,752, 285,053, 1,846,955, 11,977,939, 83,263,591, 621,012,754, 4,878,666,808, 39,333,324,973, 336,376,244,042 and so on.
We can observe here that the problem is not a function of polynomial and it rises exponentially as a function of N, where N×N is the dimension of our chess board.Hence solving this problem falls under the category of NP Hard Problems

Introduction To parallel computation: Before describing the new parallel algorithm, we describe the model of a parallel computer to execute the algorithm. We use a CRCW PRAM (concurrent read, concurrent write parallel RAM) model with n numbered processors. The model supports the following operations:

1) concurrent read from any location
2) concurrent write to any location. In the case of collisions, the processor with the lowest number succeeds.
3) concurrent fetch-and-add. In the case of collisions, the processors are serialized by their number. The lowest number is the first to perform the operation, the second lowest next and so on

The fetch-and-add operation enables implementation of critical sections. It is described by $FA(l, v)$, where l is a location address and v is the value at this location. The operation returns the old value at location l and adds v to the value at location l. If several processors perform fetch-and-add to the same location in the same cycle, their operations are serialized. The final result is equivalent to the result that would be achieved if processors performed fetch-and-add in a serial order.

3 Proposed Method

3.1 Implementation:

We are not considering all the possible cases which N Queens can be placed meeting the desired condition, but instead we will begin our approach with simple backtracking algorithm and use DFS (Depth First Search) algorithm to find only a valid solution to the problem, which our machine will consider the first valid solution according to the algorithm and if it gets the possible solution ,it will halt. Hence we are training our machine to the simplest algorithm to implement the N queen problem using backtracking strategy.

3.2 Approach for this problem:

The approach for this problem is to place queens one by one in different rows, starting from the first or the

topmost row. When we place a queen in a row, we check for clashes with already placed queens. In the current row, if we find a column for which there is no clash, we mark this row and column as part of the solution. If we do not find such a column due to clashes then we backtrack and return false.The procedure will be as follows:
1) Start in the topmost row
2) If all queens are placed return true
3) Try all columns in the current row. Do following for every tried columns.

a) If the queen can be placed safely in this column then mark this [row, column] as part of the solution and recursively check if placing queen here leads to a solution.
b) If placing queen in [row, column] leads to a solution then return true.
c) If placing queen doesn't lead to a solution then set this [row, column] to 0 (Backtrack) and go to step (a) to try other columns.

4) If all columns have been tried and nothing worked, return false to trigger backtracking.

3.3 Algorithm

3.3.1 Creating our board:

We will initially create a chess board with empty cells as 0 and non empty cells as 1.
Algorithm

```
1: procedure CreateBoard(N)
2:     ▷ Creating a chess board of size N×N
3:     for i ← 0 to N − 1 do
4:         for j ← 0 to N − 1 do
5:             ChessBoard_{i,j} ← 0
6:         end for
7:     end for
8: end procedure
```

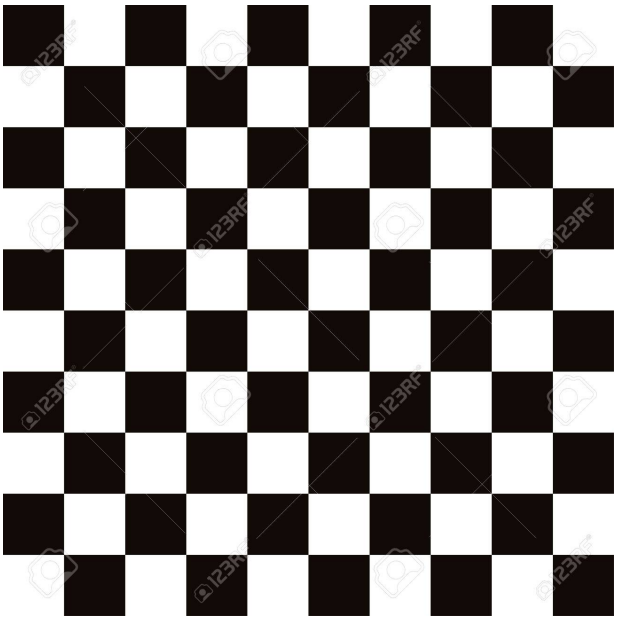

Figure 2: An empty chess board

3.3.2 Safe position:

Our very first part of the algorithm is to suggest whether the location of a given queen is safe or not.Safe means that no previous queens present in our board is able to attack the given queen.We will denote in our board the queen present as 1 and absent as 0. A queen is said to be safe if there are no queens present in the attacking position of the former queen.
The algorithm would be like a function that returns either true or false. 1)Given the position of a queen 2)If any queen is present in the attacking position, or from the figure above if any of the circular marked cells is 1, then return false ,else return true.

Input:
Coordinates of the queen formerly present say (x, y), and N is the size of the chessboard and our chessboard
Algorithm

```
1: procedure IsSafe((x, y), N, chessboard)
2:     ▷ Checking for the same row
3:     for i ← 0 to N − 1 do
4:         if chessboard_{i,y} = 1 then
5:             return false
6:         end if
7:     end for
8:     ▷ Checking for the same column
9:     for i ← 0 to N − 1 do
10:        if chessboard_{x,i} = 1 then
11:            return false
12:        end if
13:    end for
14:    ▷ Checking for the left or the right diagonal
15:    for i ← 0 to N − 1 do
16:        for j ← 0 to N − 1 do
17:            if abs(x − i) = abs(y − j) and
   chessboard_{i,j} = 1 then
18:                return false
19:            end if
20:        end for
21:    end for
22:    return true
23: end procedure
```

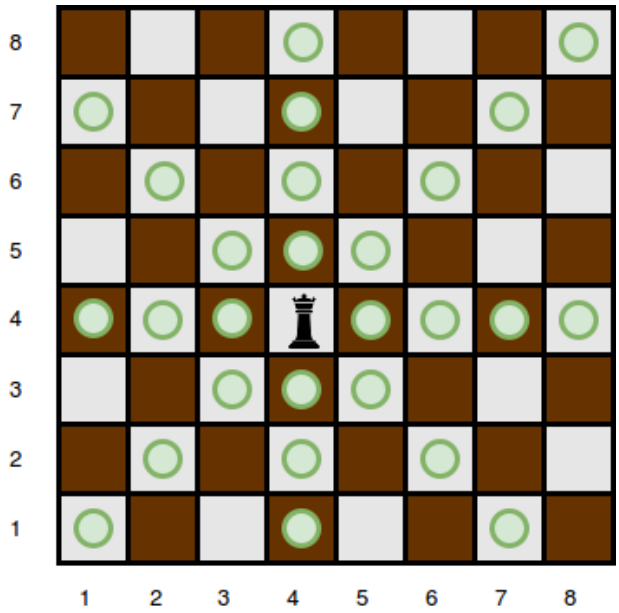

Figure 3: The attacking positions of the present Queen

3.3.3 Get Queens Position:

Here ,we will get the configuration of a queen in our chessboard, starting from the topmost row reaching to the last of the row in the chessboard, if the last row in the chessboard doesn't give any valid position for the queen at each of it's column ,then we backtrack remarking each of the updated content of the board as 0.Hence we need backtracking algorithm here to get our desired queen configurations.

Input:

row indicates that our queen is present on that row
Chessboard which is the desired matrix
N that is the size of our chess board N×N

Algorithm:

The procedure will return either the configuration is possible or not, if the configuration is possible then it will return true else false.

```
1: procedure GETCONFIG(row,ChessBoard,N)
2:     ▷ Sets the queen positions and suggests if the problem is solvable or not
3:     if row ≥ N then
4:         ▷ All the rows are satisfied
5:         return true
6:     end if
7:     for i ← 0 to N − 1 do
8:         if IsSafe((row, i), N, chessboard) then
9:             ChessBoard_{row,i} ← 1
10:        end if
11:        if GetConfig(row + 1, chessBoard, N) then
12:            return true
13:        end if
14:        ▷ Else if the configurations not possible unmark and backtrack
15:        ChessBoard_{row,i} ← 0
16:    end for
17:    ▷ No valid movements for every row and every column
18:    return false
19: end procedure
```

3.4 Parallel Implementation to the problem:

The solutions to the n-queens problem can be generated in parallel by using the master-worker technique. The manager generates the upper portion of the search tree by generating those nodes of fixed depth d, for some d. The manager dynamically passes each of these sequences to an idle worker, who in turn continues to search for sequences with n-queens property that contain the fixed sub-sequence of length d.
The master-worker technique is particularly well-suited for implementation with MPI (Message Passing Interface). This is a library package that can be used with C, C++, or FORTRAN to write parallel programs. The only message passing required for this paradigm is point-to-point communication.
The C-MPI program for generating solutions to the N queens problem, currently being written, is similar to the generation of Costas arrays, but is complicated by the existence of fixed points. The parallel generation of Costas arrays yielded almost linear speedup and we expect similar results for the parallel generation of the solutions to the n-queens problems.

3.4.1 Algorithm for parallel implementation:

The algorithm requires n processors, one processor for each queen.
The algorithm is based on the serial algorithm which is already described as a part of backtracking algorithm.
The parallelism is introduced by breaking the requirement that queens must be placed in successive columns. In the parallel algorithm, processors randomly and in parallel select columns to place queens. Conflicts that might occur between selections from different processors are resolved in parallel as well.
The parallel algorithm consists of the same two steps as the serial algorithm, the initial step and the final step.

4 Experimental Results

4.1 Plot

In our experimental results we analyzed changes in time with the changing the values of size of the chess board. The following graph was obtained from the plot of time variation with size.

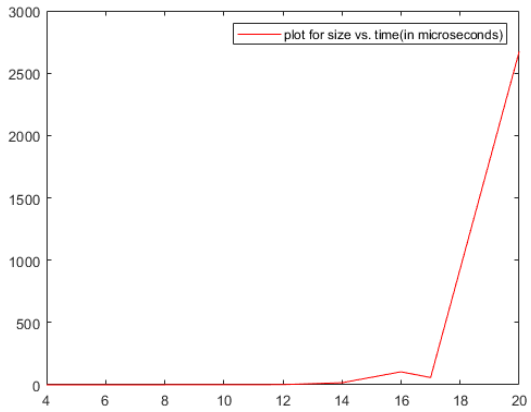

Figure 4: Graph representing the changes in time with the varying size

4.2 Table for the data

The following table shows the plot for the variation

Table 1: Size of chess board matrix vs Time

Size	**Time (in microsecs)**
4	145
5	90
6	1008
8	5422
10	6007
12	19022
14	167752
20	26739508

5 Complexity Analysis and Explanation

The complexity analysis for the desired problem is difficult to analyze since the problem falls under the category of NP Hard problems. From our approach , we can say that when on a specific row we are required to traverse through all the columns of the previous row, also the time taken for safe positions and printing of the board will be of the order of the dimension of the board in the best as well as the worst case analysis.
Thus, formulating this from our statement, we can write:

$$\mathrm{T(n)} = \mathrm{n}\times\mathrm{T(n\text{-}1)} + O(n^2)$$

which gives the average complexity as $O(n!)$, indeed a factorial complexity, as can be evident from the experimental results as well.

6 Conclusion:

From the paper we bring forth the idea of solving the N queen problem using backtracking.Other approach would be the branch and bound which is similar to the Breadth First Search, we find all the possibilities from the start and so the process requires a heavy space complexity. Backtracking is a better approach of solving the problem.

7 References

1.Introduction to Algorithms(Second Edition)
Author:Thomas H Cormen, Charles E Leiserson,
Ronald L Rivest, Clifford Stein
2.Wikipedia: https://en.wikipedia.org/wiki/Eight
queens puzzle.
3.GeeksforGeeks https://www.geeksforgeeks.org/printing
solutions n queen problem.
4. E. Orozco, On the Parallel Generation of
Costas Arrays, M.S. thesis, University of Puerto
Rico at Mayaguez, 1998.
5. https://en.wikipedia.org/wiki/Costas array