Build and Conquer: Solving N Queens Problem using Iterative Compression

1st Ahmed Alhassan
University of Khartoum Electric and electronic Engineering dept. (student)
Khartoum, Sudan
a.mo.a.alhassan@gmail.com

***Abstract*—The N queens problem is a very hot topic for research use. Nonetheless all previous algorithms that solved the N queens problem treat each order of it as if they are separate problems with no connection among them. The main idea of this paper is to connect the dots among various orders of the N queens problem and demonstrate that all orders of the N-queens problem are connected with each other. And then we propose an algorithm using iterative compression to solve the problem i.e. beginning from the solution of the least order of the problem and then by using the relation among different orders we add a queen in every iteration until we have the N queens residing in the NxN board.**

***Index Terms*—N–Queens Problem, iterative compression, nonattacked corners, pseudo-solution.**

I. Introduction

The N queens problem is about trying to put N chess queens in an NxN chessboard without letting any queen attack any other queen i.e. placing queens so that:

1) There are no queen sharing the same row with any other queen.
2) There are no queen sharing the same column with any other queen.
3) There are no queen sharing the same diagonal with any other queen.

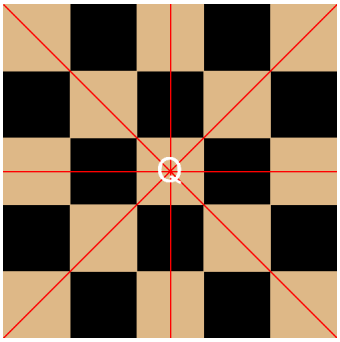


Fig. 1: How queen attacks.

This problem has at least one solution for all positive integer values of N except for $N = 2$ and $N = 3$

A. Background

The algorithms of solving N queens problem progressed from brute force algorithm which is very computationally expensive $O\binom{NxN}{N}$ to placing every queen in different row then checking the solution and moving queens until finding a solution. This algorithm has an $O(N^N)$ complexity. And the algorithm got better with using permutations and checking for diagonal attacks which made the complexity $O(N!)$. The algorithm has became even better by using backtracking but still had an exponential complexity. The complexity of finding all solutions is more complex than the #P class as stated by Hsiang, Hsu and Shieh [1]. A lot of researches used neural networks to get a solution for this problem [2]. But surprisingly only one paper has addressed the problem of finding a relation among various orders of N queens problem and show how they are related to each other. But it did not include any general relation that connects these orders. In this paper we :

- We prove that the various orders of N queens problem are indeed have a relation among them and discuss a technique to solve higher orders using lower ones.
- We discuss how some placements produced by this technique are not solutions to their respective orders but they are essential to solve higher orders.
- We propose a new iterative compression algorithm using this technique to solve the N queen problem.

B. Applications

The N queens problem is often used to test how good search algorithms can be. The same goes with genetic algorithms [3] and heuristics but is also used in various applications [4] such as :

- VLSI testing and Traffic control [5].
- Parallel memory storage schemes that was proposed by Ebras, Tanik, and Nair by using solutions of N queens problem [6] [7].
- Deadlock prevention. Tanik proves that the solution of N queens problem can be used to make a set of paths free of deadlocks [8].
- Neural network and constraint satisfaction problems [9].
- Studying the re-configurable meshes with buses (RMB) [10].
- Analyzing the statistical secondary structure of nucleic acids [11].

C. Terminology

Iterative compression : The type of algorithms that solves the problem by adding a parameter at each iteration to build

978-1-7281-1006-6/19/$31.00 ©2019 IEEE

the final solution.

Order : The number of the queens 'N' required to be on the chessboard in order to complete the solution.

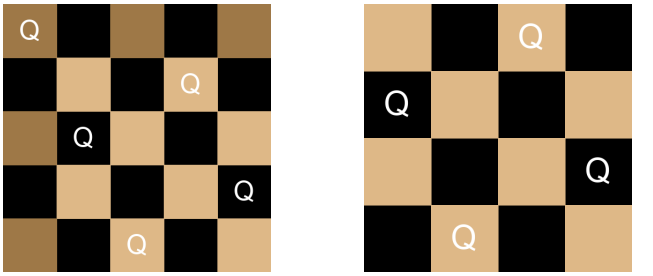
Distinct solutions: the set of all possible solution for a specific order with no solution mirrored or rotated (90, 180, or 270 degrees) from another solution.

Breadth First Search (BFS): An algorithm for searching trees that searches starting from the root then to its closest elements in the same depth then move to the element with the next level of depth and so on.

Depth First Search (DFS): An algorithm for searching trees that searches starting from the root then goes as far as possible into each branch and then backtracking when reaching the deepest level in the branch.

II. Related Work

While finding a general relation that governs the interaction between various orders of N queens problem, there are some relations easier to spot. In the paper written by Kersi and Pattnaik [12] which describes a special relation among specific solutions for N queen problem. They used that relation to solve higher orders. "Fig. 2" shows an example of using this algorithm to get one of the solutions of the 5th order using the solution of the 4th order.



(a) A solution for 5 queens problem.

(b) The only distinct solution for 4 queens problem.

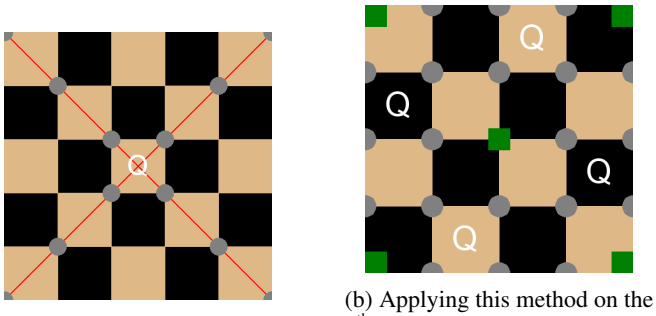
Fig. 2

The relation they noticed is that some higher order solutions are the same with lower ones with the only difference being that we add a column and a row at the sides and place a queen in the corner. This technique solves a decent number of higher orders but do not find all distinct solutions for every order. Since you have to add a column, a row, and a queen in order to get higher order solution how can we decide the places where we can add them? And how can we find all distinct solutions using only previous order solution? To answer these question we introduce a technique that helps us decide where to place them.

III. Methodology

A. Nonattacked Corners

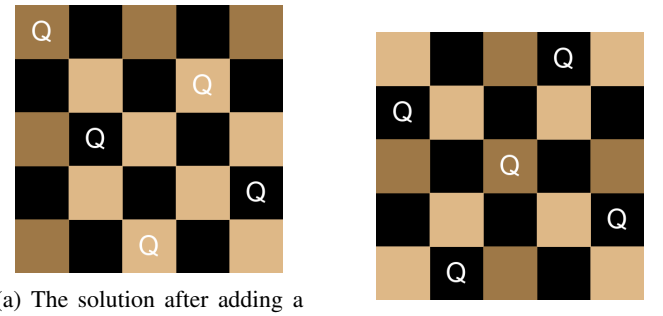
This technique is used to get the position where the new queen should be placed. As the name suggests we look for the nonattacked corners. However we can not find them directly. Instead we first mark all attacked corners by all queens then all the unmarked are the nonattacked ones. Those nonattacked corners specifies the place to add the row, the column, and the queen to get a higher order solution. This algorithm has a complexity of O(N^2). See "Fig. 3" and "Fig. 4"



(a) An example for finding attacked corners of a single queen

(b) Applying this method on the 4th order solution then marking the nonattacked with green squares

Fig. 3: Applying the corner attack technique to determine the places "green squares" to add the new queen to.



(a) The solution after adding a row, column, and queen in the place of the top left nonattacked corner in.

(b) The only distinct solution for 4 queens problem.

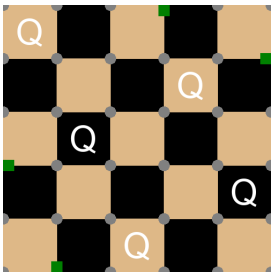
Fig. 4

B. Pseudo-solution

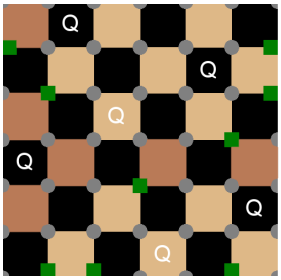
When applying the nonattacked corners method sometimes we generate a configuration which is not a valid solution for its own order for example "Fig. 5" . The fact that this configuration is not solution does not disregard its value of getting solutions of higher orders which can only be generated by pseudo-solutions.For example their role is crucial in generating several solutions from the 7th order and on which can not be generated using lower orders solutions. See "Fig. 6"

IV. Build and Conquer

As we discussed the technique to level-up from order to order the remaining questions are : how should this technique be applied to get all solutions for a certain order? and if we will use an iterative compression algorithm as an answer for the first question, where should we begin as the root of all solutions? It may be apparent that choosing the 4th order solution as the root is perfect. Because it is the only distinct solution for its order and the lower orders mainly both the

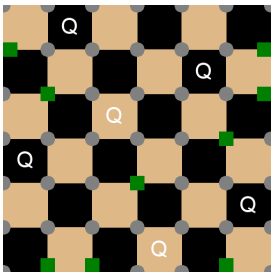


(a)

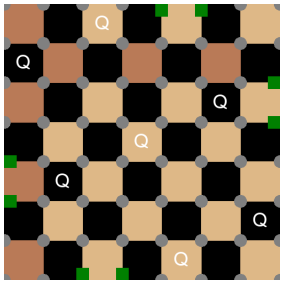


(b) One of the pseudo-solutions generated from a 5th order solution

Fig. 5: How pseudo-solution are generated



(a)



(b)

Fig. 6

2nd and the 3rd order has no solutions and the 1st order which putting the lone queen in a single cell has no nonattacked corners. However this is not the case since it is obvious that we do not get all solutions starting from the 8th order. As a result the only solution of the 4th order is not suitable to be the root chosen for an iterative compression algorithm that generates all the available solutions of the N queens problem. Instead we must use a lower order solution that produces the only distinct solution of the 4th order. But there is no lower order solution that has nonattacked corners. Instead of a solution we use a pseudo-solution.

The pseudo-solution we are looking for must be of an order lower than the 4th and must produce all of the solutions that the 4th order solution produces including itself. In order to get that pseudo-solution we take the solution of the 4th order and remove one of its queens along with the queen's column and row. And since the solution is symmetric with any kind of rotation made to it, removing any of the queens will end up with the same pseudo-solution. Then this pseudo-solution is used as the root of the iterative compression algorithm to generate the solutions.

To put the finishing touches to the work. We have to define the way which iterative compression will work on every succession. With the non-attacked corners technique to get both of the next solutions and pseudo-solutions and the root we have found previously we can define the algorithm. This algorithm can use either depth first search (DFS) or breadth first search (BFS) in order to iterate through all possible solutions. There

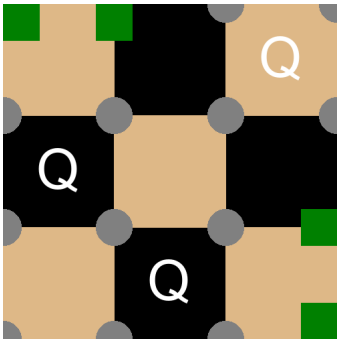


Fig. 7: The Root (pseudo-solution) made from removing one of the queens of the 4th order solution.

is a trade off between mapping the whole solution tree and using resources when executing the search using depth first or breadth first. Using depth first search is better when trying to obtain a better understanding to the origins of the solutions and producing the solution tree. But consumes more resources due to some solutions are produced by more than one lower order solution or pseudo-solution. However breadth first search is more efficient in using resources but may hide the relation among the solutions.

V. Results

Although the nonattacked corners method take a complexity of O(N^2), the total complexity of the iterative compression algorithm is pseudo-linear. Because the complexity of this algorithm also depends on the number of the configurations we search whether the configuration is a solution or pseudo-solution. Making the complexity of generating all the solutions of an order approximately O($C*(N^2)$) where C represents the number of configurations we search in order to generate the solutions.

We used the algorithm described in this paper with the breadth first search because counting the number of configurations will be easier that way. Then we cleared out all the mirrored and rotated instances of these configurations. Then classified them into solutions and pseudo-solutions. When we applied the algorithm to get the solutions of all orders up to the 10th order with using the pseudo-solution in "Fig. 7" as the root of the algorithm we got the results in "Table. I". We found all the distinct solutions of every order up to the 10th.

TABLE I: Number of distinct configurations generated for each order up to the 10th

Order(# of queens)	*# of configurations*	*# of solutions*	*# of pseudo-solutions*
4	2	1	1
5	4	2	2
6	20	1	19
7	125	6	119
8	957	12	945
9	8690	46	8644
10	90768	92	90676

VI. Further work

So far the iterative compression algorithm with both implementations of the breadth first and depth first works well in getting the solutions of the N queens problem. However there still that trade off resources and better understanding of the relations among the solutions. In order to get a better we may adjust the implementation by using a hybrid search algorithm between depth first search and breadth first search to get a better balance of this trade off which results in more efficient algorithm than breadth first that gives us better understanding to the relations among solutions than depth first search.

I only applied this work on 2D n-queens problem. It would be interesting to try and apply it to multi-dimensional n-queens problem [13]. This problem can put the nonattacked corners to a great test.

This work may contribute in reducing the complexity of the N queens completion problem which is a NP complete problem. The problem states given any initial placement of some queens where should the other queens be placed in order to complete the solution [14]. With some adjustment to the nonattacked corners technique and the use of an iterative compression algorithms it may prove to be one of the best algorithms that solves this problem.

Acknowledgment

I want to thank my family specially my mom for being patient during the time I was writing this paper. My thanks also goes to both my seniors Ahmed Osman and Ahmed Sharfi whom I learnt about the 8 queens problem from and without them this paper would not be a reality. I thank Mohammed Osman for encouraging me to put this work into a paper and Akram Izzeldin who peer reviewed this paper and giving me some constructive critics. I thank everyone who gave any kind of motivation or support.

Appendix

```
input : An array Queens of size N.
output: An array Nonattacked represents the
        nonattacked corners available where we can
        add new queens.
corners ← ZeroMatrix(N + 1, N + 1);
foreach element q of queens do
    if corner is attacked then
        corners [x][y] ←1
    end
end
nonattacked ← [ ]
for i ← 1 to N + 1 do
    for j ← 1 to N + 1 do
        if corners [i][j] == 0 then
            nonattacked.append ([i,j])
        end
    end
end
return nonattacked
```

Algorithm 1: Nonattacked corners

```
input : An array Queens of order O, the final order
        intended to get all of its solutions N.
output: An array of all Solutions to all order of
        N-queens problem from order O to N.
root ← Queens
solutions ← []
boards ← [ root ] // Contains all
    solutions and pseudo-solutions of
    the current order.

for i ← 3 to N − 1 do
    tmp ← []
    foreach element b of boards do
        newQueens ← getNonattackedCorners
         (root)
        foreach element q of newQueens do
            tmpBoard ← addQueen (q, b)
            tmp.append (tmpBoard)
            if tmpBoard is a valid solution &&
             tmpBoard is not in solutions then
                solutions.append (tmpBoard)
            end
        end
    end
    boards ← tmp
end
return solutions
```

Algorithm 2: Generating solutions using BFS

```
input : An array Queens of order O, the final order
        intended to get all of its solutions N.
output: A tree of arrays of all Solutions to all order
        of N-queens problem from order O to N.
root ← Queens
newQueens ← getNonattackedCorners (root)
if length (root) == N or newQueens is empty then
    return root
else
    foreach element q of newQueens do
        tmpBoard ← addQueen (q, b)
        root.addChild (DFS (tmpBoard, N))
    end
    return root
end
```

Algorithm 3: Generating solutions using DFS

REFERENCES

[1] J. Hsiang, D.F. Hsu, and Y.-P. Shieh. On the hardness of counting problems of complete mappings. Discrete Mathematics, 277:87–100, 2004.

[2] Mańdziuk, Jacek. (2002). Neural networks for the N-Queens Problem: a review. Control and Cybernetics. 31.

[3] S.Sathyapriya, R.Stephen, and V.S.Joe Irudayaraj. Survey on N-Queen Problem with Genetic Algorithm. International Journal of Computer Sciences and Engineering, Volume-6, Special Issue-2, March 2018.

[4] J. Bell and B. Stevens. A survey of known results and research areas for nqueens. Discrete Mathematics, 309:1–31, 2009.

[5] C. Erbas, M.M. Tanik, and Z. Aliyazicioglu. Linear congruence equations for the solutions of the n-queens problem. Information Processing Letters, 41:301–306, 1992.

[6] C. Erbas and M.M. Tanik. Storage schemes for parallel memory systems and the n-queens problem. In Proceedings of the 15th Anniversary of the ASME ETCE Conference, Computer Applications Symposium, volume 43, pages 115–120, 1992.

[7] C. Erbas, M.M. Tanik, and V.S.S. Nair. A circulant matrix based approach to storage schemes for parallel memory systems. In Proceedings of the Fifth IEEE Symposium on Parallel and Distributed Processing, pages 92–99. IEEE, 1993.

[8] M.M. Tanik. A Graph Model for Deadlock Prevention. PhD thesis, Texas A&M University, 1978.

[9] C. Erbas, S. Sarkeshik, and M. M. Tanik, Different perspectives of the n-queens problem. In CSC 92: Proceedings of the 1992 ACM Annual Conference on Communications, pages 99–108, 1992.

[10] M. Kunde and K. Gurtzig. Effcient sorting and routing on reconfigurable meshes using restricted bus length. In Proceedings of the 11th International Parallel Processing Symposium (IPPS1997), pages 713–720. IEEE Computer Society, 1997.

[11] K. Yamamoto, Y. Kitamura, and H. Yoshikura. Computation of statistical secondary structure of nucleic acids. Nucleic Acids Research, 12:335–346, 1984.

[12] Vishal Kesri, Vaibhav Kesri, and Prasant Ku. Pattnaik. An Unique Solution for N queen Problem International Journal of Computer Applications (0975 – 8887) Volume 43– No.12, April 2012.

[13] J. Barr, S. Rao, The n-queens problem in higher dimensions, Elem. Math. 61 (4) (2006) 133–137.

[14] Ian P.Gent, Christopher Jefferson, and Peter Nightingale. Complexity of n-Queens Completion. Proceedings of the Twenty-Seventh International Joint Conference on Artificial Intelligence (IJCAI-18)