N-Queens Solving Algorithm by Sets and Backtracking

Jehanne Eliza F. Criseno*
Sean Thomas C. Vizconde*
jfcriseno@up.edu.ph
scvizconde@up.edu.ph
University of the Philippines Baguio
Baguio, Benguet, Philippines

CCS CONCEPTS

• **Theory of computation** → **Backtracking**; *Complexity classes*;
• **Mathematics of computing** → *Combinatoric problems*.

KEYWORDS

N-Queens, combinatorics, algorithm, sets, backtracking, efficiency, optimization

1 INTRODUCTION

In 1848, Max Bezzel, a German chess composer, proposed the 8-Queens Problem. Two years after, this was further extended by Franz Nauck, who was able to publish the first complete solution to the said problem. The objective is to calculate the total number of ways one can place eight number of queens in an 8×8 chessboard such that no two queens attack each other; hence, no queen must lie in the same row, column, or diagonal of any other queen as shown in Figure 1 [Campbell 1977; Erbas et al. 1992].

Furthermore, the N-Queens Problem is a generalization of the 8-Queens Problem having a similar objective, but this time, an arbitrary size n is considered [Rivin et al. 1994].

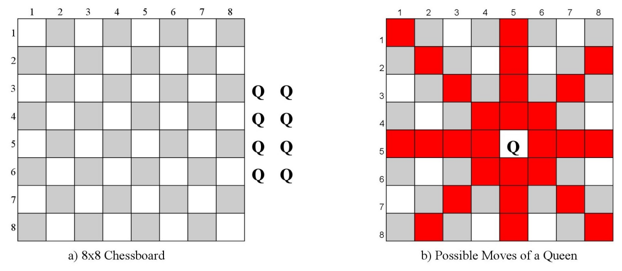


Figure 1: The 8-Queens Problem With Its Respective Chessboard and the Queen's Field of Attack.

The problem can be classified as both a chess problem, as it challenges one's tactical skills and pattern recognition; and a combinatorial problem since the solution to it is the number of ways to arrange n-queens in an $n \times n$ chessboard. Generally, when we refer to this problem, we typically are only interested in $n \geq 4$ since the cases for $n = 0, 1, 2,$ and 3 are uninteresting.

*Both authors contributed equally to this research.

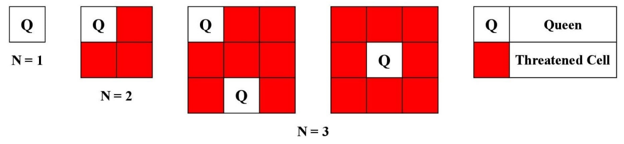


Figure 2: Trivial Cases of the N-Queens Problem.

For $n = 0$, we only have one way to arrange 0 queens in a 0×0 chessboard, that is, to not place it at all. For $n = 1$, conditions are satisfied once we place the queen in a 1×1 chessboard; hence, there is only one way for an input of 1. For $n = 2$ and 3, we can observe that there exists no possible arrangement of n-queens that satisfies the rules of the problem, which can be seen in Figure 2.

2 SOME PERSPECTIVES ON THE N-QUEENS PROBLEM

There are various perspectives on how to solve the N-Queens problem. Included here are the naive brute-force, permutation generation, and backtracking.

2.1 Brute Force Approach

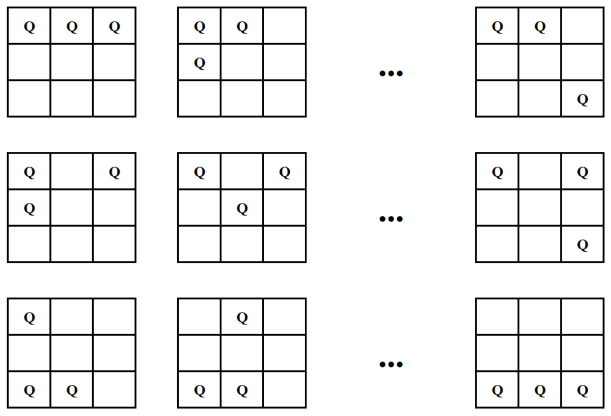


Figure 3: A Naive Approach in Solving the N-Queens Problem

A naive approach to the problem is using brute force, a trial and error method that tries all arrangements of n-queens in the $n \times n$ chessboard according to the constraints of the problem (See Figure

3) [Erbas et al. 1992]. Given the said chessboard with n-queens, there are exactly n^2Cn possible arrangements of queens.

For the 8-Queens problem, this amounts to 4426165368 different arrangements.

2.2 Permutation Generation

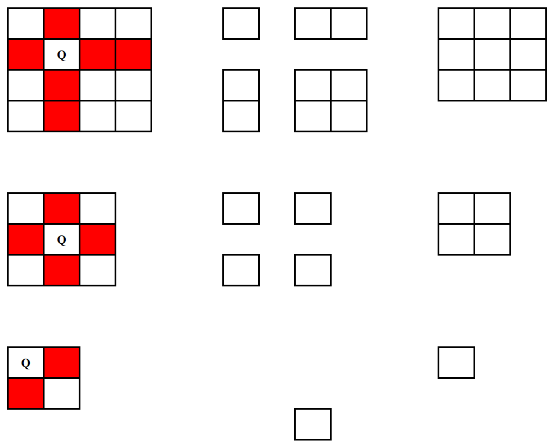


Figure 4: Placing a Queen Reduces the Dimensions of the Board by 1.

Invalid arrangements are immediately perceived when utilizing the naive approach. From the given constraints, an observation can be made that each queen must occupy a distinct:

1.) Row
2.) Column
3.) Positive diagonal
4.) Negative diagonal

However, for visualization purposes of this approach, only consider the first two restrictions. Observe that placing a queen on an $n \times n$ chessboard can remove the threatened cells, as well as the current cell of the queen, and rearrange the chessboard into an $n-1 \times n-1$ board. This observation holds until we place $n-1$ queens and are left with a 1×1 chessboard (See Figure 4).

Another way to employ this approach is by assigning a row to each queen and permutating the column number yielding an arrangement for the possible solution.

The number of possible arrangements for this improved approach is given by equation (1).

$$n \times (n-1) \times \ldots \times 1 = n! \quad (1)$$

Consequently, for the 8-Queens Problem, there is a total of 40320 different arrangements. Do note that $n!$ is the upper bound for this approach since considering the latter half restrictions will further reduce this result.

As discerned, the permutations produce collisions among queens on the diagonals. Each cell on a positive diagonal produces a fixed sum value of its respective row and column number. On the contrary, each cell on a negative diagonal produces a fixed difference value

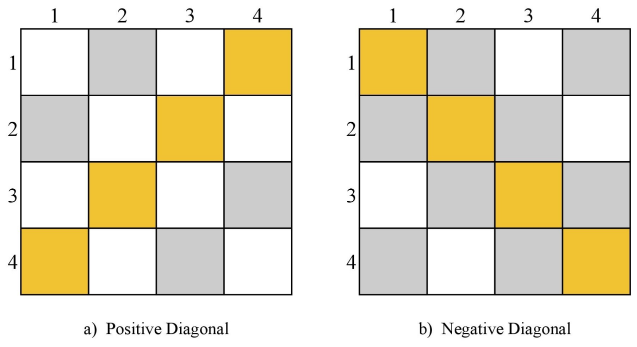
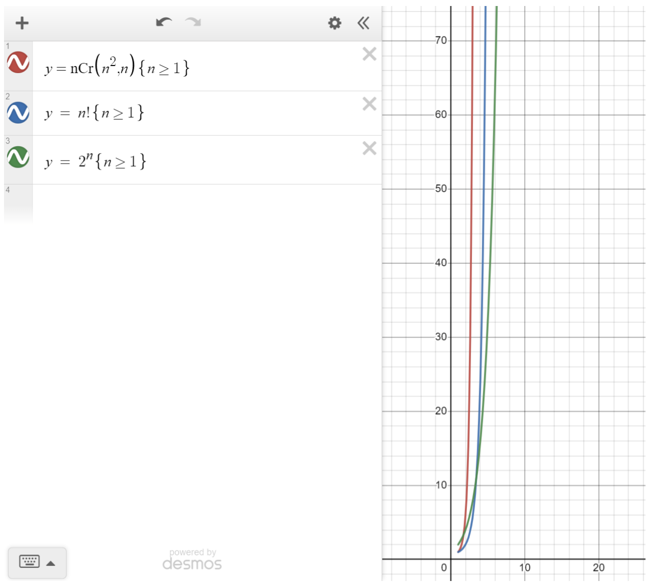


Figure 5: The Positive and Negative Diagonals of a 4x4 Board.

of its respective row and column number. Furthermore, finding a solution to the N-Queens problem reduces to finding a permutation of the column numbers, such that the values of the sum and difference of the row and column numbers are distinct for all rows [Campbell 1977; Erbas et al. 1992].


$y = \mathrm{nCr}(n^2, n)\{n \geq 1\}$

$y = n!\{n \geq 1\}$

$y = 2^n\{n \geq 1\}$

Figure 6: The Complexity Comparison of $n!$ and Brute Force Approach Using Desmos Graphing Calculator.

Observe from Figure 6 just how both approaches scale with the input n. Given that the curves for both the naive brute-force approach and the permutation generation are steeper than the curve for 2^n, a conclusion can be made that they are at least exponential in nature.

2.3 Backtracking Algorithm

The third approach one may employ is backtracking. This is a systematic way to run through all the possible configurations of a search space and tries to build a solution to a problem incrementally, testing it according to the problem criterion [Erickson 2019; Kiena

2008]. Implementing this approach is more efficient in searching for a partial solution that will eventually lead to a valid final answer.

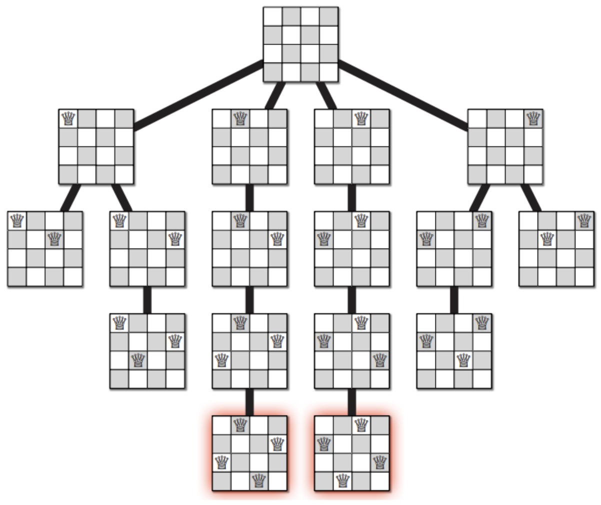


Figure 7: The Recursion Tree Representation of a Complete Solution to the 4-Queens Problem. Obtained From Erickson, J. (2019) "Backtracking," in *Algorithms*. First edition, p. 73.

Moreover, the backtracking algorithm is better represented by referencing Figure 7. The leaf nodes in the tree without a red outline represent dead ends; that is, partial states that we know for certain cannot lead to a valid answer since all states are already threatened without having placed n queens yet. Thus, in encountering such states, one may know that any states deriving from this invalid state will never produce a valid solution. In cases like these, backtracking is employed returning to the previous state and continuing to explore from there.

3 COMPUTATION COMPLEXITY THEORY

The binary alphabet is the set $\Sigma = \{0, 1\}$. The shortlex order over Σ is $(\varepsilon, 0, 1, 00, 01, 10, 11, 000, \ldots)$ [Sipser 2012].

Class **P** is defined as the class of languages that are decidable in polynomial time. More formally,

$$P = \left\{L \subseteq \{0,1\}^* : \exists A \text{ that decides } L \text{ in polynomial time } \right\}$$

[Cormen et al. 2022].

However, if we define the class **P** in terms of problems, it is then the class of problems that are solvable in polynomial time.

A verification algorithm A is a two-argument algorithm such that $A(x, y) = 1$ where x and y are binary strings and y is called the certificate [CLRS]. We then denote the language verified by A as

$$L = \left\{x \in \{0,1\}^* : \exists y \in \{0,1\}^* \text{ such that } A(x,y) = 1\right\}$$

[Cormen et al. 2022].

Finally, define the class **NP** as the class of languages that can be verified by a polynomial-time algorithm. More formally,

$$L = \left\{x \in \{0,1\}^* : \exists y \text{ with } |y| = O(|x|^c) \text{ such that } A(x,y) = 1\right\}$$

[Cormen et al. 2022].

We want the length of the certificate y, denoted as $|y|$, to be polynomial in the size of the input x and this is because on the assumption that our verification algorithm runs in polynomial time, we can then only read a polynomial amount of data. Again, if we were to define the class **NP** in terms of problems, then it is the class of problems whose solutions are checkable in polynomial time.

A language L_1 is polynomial-time reducible to a language L_2, denoted as $L_1 \leq_p L_2$ if

$$\exists f : \{0,1\}^* \to \{0,1\}^* \text{ s.t. } \forall x \in \{0,1\}^*, x \in L_1 \text{ iff } f(x) \in L_2$$

[Cormen et al. 2022].

In terms of problems, we say that a problem Q as being reducible to another problem Q' if any instance of Q can be recast as an instance of Q', and the solution to the instance of Q' provides a solution to the instance of Q [Cormen et al. 2022].

3.1 Problems in P

Some examples of problems in **P** include Sorting and Primality Testing.

In a Sorting problem, given some sequence of n numbers

$$\langle a_1, a_2, \ldots, a_n, \rangle$$

we want to return a reordering

$$\langle a_1', a_2', \ldots, a_n' \rangle$$

such that

$$a_1' \leq a_2' \leq \ldots \leq a_n'$$

[Cormen et al. 2022].

This problem belongs in the complexity class **P** because there already exists multiple polynomial run time algorithms that solve this problem such as Merge Sort and Quicksort.

For Primality Testing, the description of the problem is given by determining for some $n \in \mathbb{N}$ if it is prime [Cormen et al. 2022]. This belongs in the complexity class **P** because in the most brute force method of determining an answer to the problem, we just need to iterate over the elements

$$2, 3, \ldots, \lfloor \sqrt{n} \rfloor$$

and perform trial division. Accordingly, this means that the worst-case running time for this approach is $\Theta(\sqrt{n})$ [Cormen et al. 2022].

3.2 Problems in NP

Some examples of problems in **NP** include the Subset Sum and Hamiltonian Cycle Problem.

In the Subset Sum problem, given an array of numbers

$$w_1, w_2, \ldots, w_n \in \mathbb{N}$$

and a target value T, is there a subset S such that

$$\sum_{i \in S} w_i = T$$

[Cormen et al. 2022; Kleinberg and Tardos 2005; Skiena 2008].

In the naive approach, all 2^n possibles subsets must be checked, which makes the run time of this approach exponential. However, if there is some $U \subseteq W$ and target T that is explicitly given, we can easily check by adding all the elements of U if it indeed does sum up to T.

Next, we provide a description of the Hamiltonian Cycle Problem. Given a directed graph $G = (V, E)$ with

$$V = v_1, v_2, \ldots, v_n$$

a Hamiltonian cycle is a simple cycle that contains every vertex in V. We want to know for some given G if it contains a Hamiltonian cycle.

In the worst case, $n!$ sequences of vertices must be checked in a complete graph with n-vertices. However, if there is a given sequence

$$\left\langle v_1, v_2, v_3, \ldots, v_{|V|} \right\rangle$$

we can check in polynomial time that the sequence does indeed contain every vertex in V and each vertex appears in that sequence exactly once. Furthermore, we can also check that $(v_i, v_{i+1}) \in E$ for $i = 1, 2, 3, \ldots, |V| - 1$, that is, every subsequent pair of vertices has an edge connecting them and that the first and last vertices of the sequence also has an edge connecting them. More formally,

$$\left(v_{|V|}, v_1\right) \in E$$

[Cormen et al. 2022].

3.3 Complexity of the N-Queens Problem

Table 1 shows the number of solutions to the N-Queens problem; hence, as of now, the only known computation for the number of solutions is only until $n = 27$ [Sloane 2007].

The reason why N-Queens is an **NP-Hard** problem is because after $n = 17$, each increment in the input parameter n results in an order of magnitude greater number of solutions which means that the entire search space also increases exponentially (refer to Table 1). Moreover, if we were given some integer that corresponds to the solution of the problem for some specific n, we cannot efficiently verify this number since the only way to do so would be to compute the problem for that n and check if it matches the candidate answer that were given. Since there is no algorithm to efficiently solve this problem, then there is no way to efficiently verify it either. Hence, the problem is classified as **NP-Hard**.

4 SETS AND BACKTRACKING HYBRID ALGORITHM

The proposed algorithm takes advantage of sets to describe the availability of each cell in the board and utilizes set elimination. Here, a matrix of sets is employed, consisting of three parts:

1.) Cell number where the queen can be placed,
2.) Row number where the cell is located, and
3.) Threatened cells of the queen placed in that particular cell.

The cell number and row number are placed in the same set, while the threatened cells are placed in another set. Moreover, the matrix representation of the mentioned parts is as follows:

$$\{\,\{Cell\ Number, Row\ Number\}, \{Threatened\ Cells\}\,\}$$

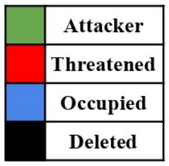
[Güldal et al. 2016]. To better visualize and follow through with the algorithm, Güldal's paper demonstrated solving the 4-Queens Problem using the Hybrid algorithm.

Table 1: The Number of Solutions for the N-Queens Problem

n	Number of Solutions
0	1
1	1
2	0
3	0
4	2
5	10
6	4
7	40
8	92
9	352
10	724
11	2680
12	14200
13	73712
14	365596
15	2279184
16	14772512
17	95815104
18	666090624
19	4968057848
20	39029188884
21	314666222712
22	2691008701644
23	24233937684440
24	227514171973736
25	2207893435808352
26	22317699616364044
27	234907967154122528

Obtained From: Sloane, N. J. (2007). The On-line encyclopedia of integer sequences. In Towards Mechanized Mathematical Assistants: 14th Symposium, Calculemus 2007, 6th International Conference, MKM 2007, Hagenberg, Austria, June 27-30, 2007. Proceedings (pp. 130-130). Springer Berlin Heidelberg.

4.1 4-Queens Problem



🟩	Attacker
🟥	Threatened
🟦	Occupied
⬛	Deleted

Figure 8: Color Code Guide

Figure 8 shows the color-guided code to be used in the demonstration. Consider the green-colored cell to be the location of the attacker, the red-colored cells to be the threatened cells of the attacker, the blue-colored cells will denote the occupancy of a queen, and the black-colored cells to be the deleted cells.

In a 4×4 board, we index every cell and generate the matrix representation of it. The cells that lie in a queen's field of attack and their corresponding row number are stored in the matrix. For

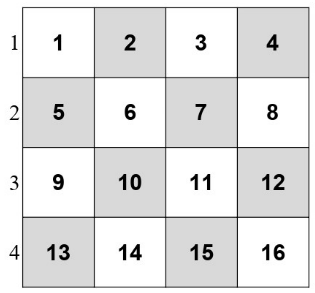


Figure 9: 4×4 **Board**

$$\begin{pmatrix}
\{1,\ 1\} & \{2,3,4,5,6,9,11,13,16\} \\
\{2,\ 1\} & \{1,3,4,5,6,7,10,12,14\} \\
\{3,\ 1\} & \{1,2,4,6,7,8,9,11\} \\
\{4,\ 1\} & \{1,2,3,7,8,10,12,13,16\} \\
\{5,\ 2\} & \{1,2,6,7,8,9,10,13,15\} \\
\{6,\ 2\} & \{1,2,3,5,7,8,9,10,11,14,16\} \\
\{7,\ 2\} & \{2,3,4,5,6,8,10,11,12,13,15\} \\
\{8,\ 2\} & \{3,4,5,6,7,11,12,14,16\} \\
\{9,\ 3\} & \{1,3,5,6,10,11,12,13,14\} \\
\{10,\ 3\} & \{2,4,5,6,7,9,11,12,13,14,15\} \\
\{11,\ 3\} & \{1,3,6,7,8,9,10,12,14,15,16\} \\
\{12,\ 3\} & \{2,4,7,8,9,10,11,15,16\} \\
\{13,\ 4\} & \{1,4,5,7,9,10,14,15,16\} \\
\{14,\ 4\} & \{2,6,8,9,10,11,13,15,16\} \\
\{15,\ 4\} & \{3,5,7,10,11,12,13,14,16\} \\
\{16,\ 4\} & \{1,4,6,8,11,12,13,14,15\}
\end{pmatrix}$$

Figure 10: Matrix Representation of the 4×4 **Board**

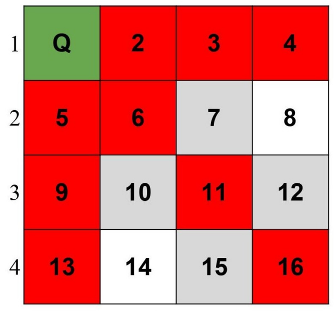


Figure 11: First Queen is Placed on Cell Number 1

instance, the threatened cells of cell number 1, which is in row 1, are given by the set $\{2, 3, 4, 5, 6, 9, 11, 13, 16\}$. Hence, this process is repeated for each cell in the board (See Figures 9 and 10).

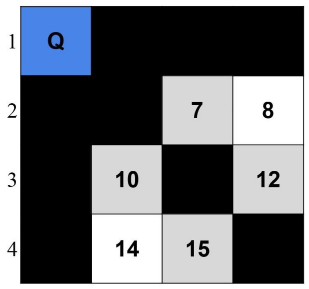


Figure 12: Updated Board After Placing the First Queen

$$\begin{pmatrix}
\{1,\ 1\} & \{2,3,4,5,6,9,11,13,16\} \\
\{7,\ 2\} & \{2,3,4,5,6,8,10,11,12,13,15\} \\
\{8,\ 2\} & \{3,4,5,6,7,11,12,14,16\} \\
\{10,\ 3\} & \{2,4,5,6,7,9,11,12,13,14,15\} \\
\{12,\ 3\} & \{2,4,7,8,9,10,11,15,16\} \\
\{14,\ 4\} & \{2,6,8,9,10,11,13,15,16\} \\
\{15,\ 4\} & \{3,5,7,10,11,12,13,14,16\}
\end{pmatrix}$$

Figure 13: Updated Matrix After Placing the First Queen

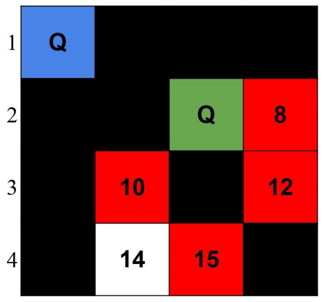


Figure 14: Second Queen is Placed on Cell Number 7

The first step in the algorithm involves placing the first queen in cell number 1; hence, the matrix representation of the board and threatened cells are shown in Figures 10 and 11, respectively. The initial matrix is then saved.

The threatened cells are then eliminated (see Figure 12), which reduces the matrix to Figure 13. Accordingly, the row entries containing the threatened cells of the first queen are removed from the matrix. Second, the next queen is placed in the next available cell, cell number 7, shown in Figure 14 and saves the generated matrix (See Figure 13) before eliminating its threatened cells (See Figure 15). The updated matrix is now represented by Figure 16 and saves this. As we can see here, only cell number 14 is left; hence, the third queen is placed here with no cells to eliminate since there are no more available cells left. The updated matrix is shown in Figure 18. To test if this is a solution, the number of queens is tested with the board size. Note that both are not equal, i.e., $3 \neq 4$, therefore, backtracking will be employed to backtrack one step and the latest matrix is restored (Figure 16). Notice here that there are no available cells left except for cell number 14, so the algorithm will backtrack once more and restore the matrix shown in Figure 13.

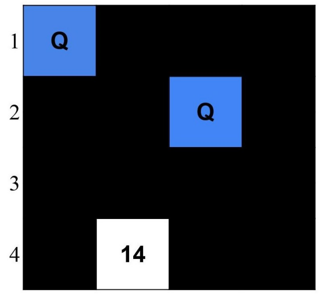


Figure 15: Updated Board After Placing the Second Queen

$$\begin{pmatrix} \{1,\ 1\} & \{2,3,4,5,6,9,11,13,16\} \\ \{7,\ 2\} & \{2,3,4,5,6,8,10,11,12,13,15\} \\ \{14,\ 4\} & \{2,6,8,9,10,11,13,15,16\} \end{pmatrix}$$

Figure 16: Updated Matrix After Placing the Second Queen

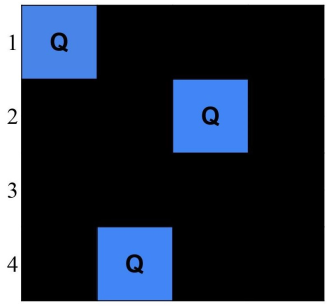


Figure 17: Updated Board After Placing the Third Queen

$$\begin{pmatrix} \{1,\ 1\} & \{2,3,4,5,6,9,11,13,16\} \\ \{7,\ 2\} & \{2,3,4,5,6,8,10,11,12,13,15\} \\ \{14,\ 4\} & \{2,6,8,9,10,11,13,15,16\} \end{pmatrix}$$

Figure 18: Updated Matrix After Placing the Third Queen

Next, the second queen will be placed on cell number 8 instead, saves the matrix (Figures 19, 20) , and eliminates the threatened cells. Consequently, continuing this path eventually leads to an unequal number of queens and board size. The algorithm will once again backtrack and restores the original board and matrix (Figures 9 and 10). Now, we place the first queen in cell number 2 and employ the same algorithm until a solution is found, that is, the number of queens and board size are equal. In this case, $4 = 4$. Furthermore, a solution to the 4-Queens Problem has a matrix representation shown in Figure 21 and is illustrated in Figure 22.

Consider Figure 19. If there is some initial configuration of queens (the queen placed on row 1) and determined that we have exhausted the possibilities for the current row of the attacker, that is, we have tried all valid and possible placements for the queen in the current row, then instead of continuing to the next available cell in any subsequent row (in this case is cell number 10) we proceed to backtrack to the previous state whose possibilities are yet to be exhausted. The reason for this is that by definition of the problem, if we want to place n-queens in an $n \times n$ chessboard, then, at the very least, each row and column must have exactly one queen. Note that in the case of a row having exhausted its possibilities, even if we find some valid placement of $n - 1$ queens after skipping the current row, then it will still never be a valid solution.

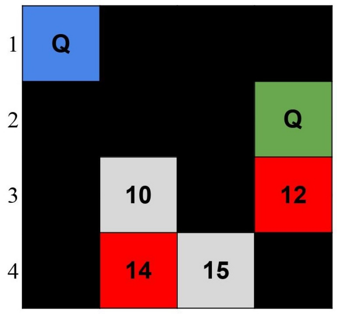


Figure 19: Threatened Cells of the Second Queen After Backtracking

$$\begin{pmatrix} \{1,\ 1\} & \{2,3,4,5,6,9,11,13,16\} \\ \{8,\ 2\} & \{3,4,5,6,7,11,12,14,16\} \\ \{10,\ 3\} & \{2,4,5,6,7,9,11,12,13,14,15\} \\ \{15,\ 4\} & \{3,5,7,10,11,12,13,14,16\} \end{pmatrix}$$

Figure 20: Matrix Representation of the Second Queen After Backtracking

$$\begin{pmatrix} \{2, 1\} & \{1, 3, 4, 5, 6, 7, 10, 12, 14\} \\ \{8, 2\} & \{3, 4, 5, 6, 7, 11, 12, 14, 16\} \\ \{9, 3\} & \{1, 3, 5, 6, 10, 11, 12, 13, 14\} \\ \{15, 4\} & \{3, 5, 7, 10, 11, 12, 13, 14, 16\} \end{pmatrix}$$

Figure 21: Matrix Representation of a 4-Queens Solution

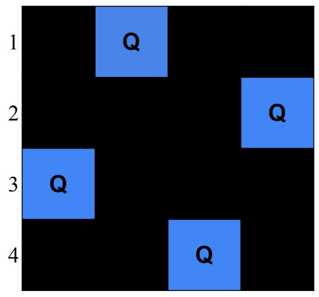


Figure 22: 4-Queens Solution

4.2 A Comparison of the Classic Backtracking and Hybrid Algorithm

To demonstrate the comparison of both algorithms, consider finding a solution for the 5-Queens Problem.

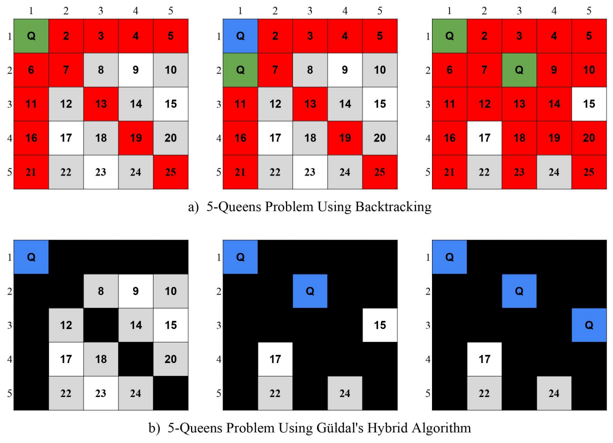

a) 5-Queens Problem Using Backtracking
b) 5-Queens Problem Using Güldal's Hybrid Algorithm

Figure 23: 5-Queens Problem using Classic Backtracking vs Hybrid Algorithm

As we can see in Figure 23, both algorithms place the first queen on cell number 1. In Classic Backtracking, placing the queen only marks the threatened cells so the algorithm has to iterate through each column of the next row of the queen's location and checks it until an unthreatened cell is found. Consequently, the second queen will be placed on cell number 8. This process is repeated until a solution is found (see Figure 24).

On the other hand, the Hybrid algorithm eliminates the threatened cells before placing the next queen, which was implemented by Güldal using matrix and sets. This reduces the number of trials to be performed as shown in Figure 23. Since threatened cells are deleted from the matrix, the placement of the next queen can immediately be visualized. This algorithm places the next queen in the first available cell after eliminating the threatened cells; hence, in this case, the second queen will be placed in cell number 8. Unlike in backtracking, the solution to this requires checking if the number of queens is equal to the size of the board. Otherwise, backtracking will be employed to go a step backward and restore the eliminated cells through the matrix. Note that when we backtrack, the eliminated cells by the invalid queen are restored, which was made possible by saving the generated matrix. The queen will then be placed in the next available cell of the same row. If there are no more available cells in the same row, then the algorithm will backtrack again until a solution is found (see Figure 24).

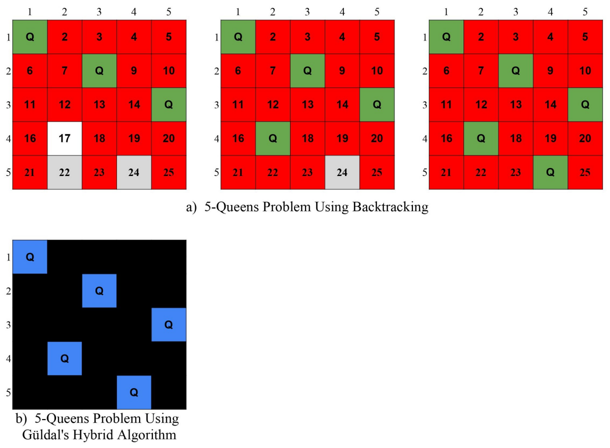

a) 5-Queens Problem Using Backtracking
b) 5-Queens Problem Using Güldal's Hybrid Algorithm

Figure 24: A Solution to the 5-Queens Problem using Classic Backtracking vs Hybrid Algorithm

Table 2 shows the difference in the number of trials between the Backtracking algorithm and the Hybrid algorithm for $n = 5$. A trial in this case is defined as the number of tries the algorithm makes before being able to place a queen on a valid cell in the board.

Table 2: Comparison of the Number of Trials in Backtracking and Hybrid Algorithm for 5-Queens

Queen No.	No. of Trials Using Backtracking Algorithm	No. of Trials Using Hybrid Algorithm
1	1	1
2	3	1
3	5	1
4	2	1
5	4	1
Total Trials	15	5

*Obtained From:*S. Güldal, V. Baugh, S. Allehaibi (2016) N-Queens Solving Algorithm by Sets and Backtracking. In SoutheastCon 2016. (p. 1).

Consider Figures 23a and 24a. In the Classic Backtracking algorithm, it needs to loop over the entire row, checking each column in that row for a valid placement of the queen. The first queen is placed in cell 1 since every cell in the first row is available. However, the second queen needs to check every column in the second row to look for the first available spot and eventually does find it in the third column of the second row. This corresponds to the number

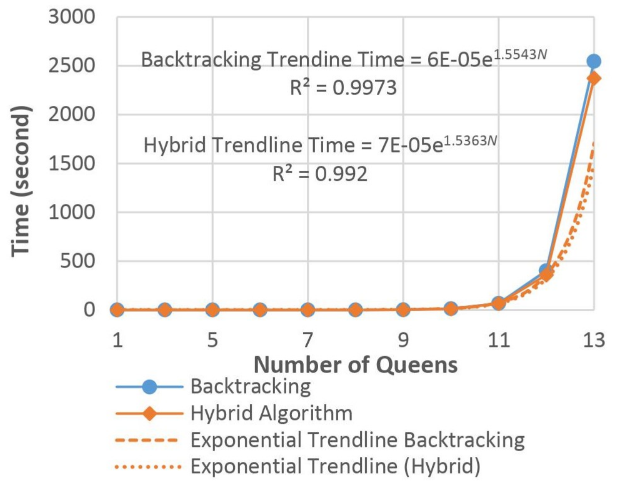


Figure 25: Comparison of Backtracking and Hybrid Algorithm. Obtained From S. Güldal, V. Baugh, S. Allehaibi (2016) N-Queens Solving Algorithm by Sets and Backtracking. In SoutheastCon 2016. (p. 3).

of trials being 3. This process continues for every queen until the algorithm finds a valid solution or determines that it cannot and backtracks.

Now, consider Figures 23b and 24b. Observe that in the Hybrid algorithm, since each placement of a queen removes all threatened cells, subsequent placements of queens then need less trials to find a valid cell, where placing a queen on cell 1 removes cells 6 and 7 on the second row. This implies that the algorithm guarantees that the first undeleted cell in the second row is available, which reduces the number of trials for this queen to 1.

The comparison of a Classic Backtracking algorithm versus the proposed Hybrid algorithm of Güldal is further illustrated in Figure 25. Even though there are slight improvements for the Hybrid Algorithm, it still follows an exponential trend line as the input n increases.

REFERENCES

Paul J. Campbell. 1977. *Gauss and the Eight Queens Problem: A Study in Miniature of the Propagation of Historical Error.* https://doi.org/10.1016/0315-0860(77)90076-3

Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein. 2022. *Introduction to Algorithms.* MIT Press.

Cengiz Erbas, Seyed Sarkeshik, and Murat M. Tanik. 1992. *Different Perspectives of the N-Queens Problem.* https://doi.org/10.1145/131214.131227

Jeff Erickson. 2019. *Algorithms.*

Serkan Güldal, Veronica Baugh, and Saleh Allehaibi. 2016. N-Queens Solving Algorithm by Sets and Backtracking. In *SoutheastCon 2016.* 1–8. https://doi.org/10.1109/SECON.2016.7506688

Steven S S. Kiena. 2008. *The Algorithm Design Manual 2nd ed.* Springer.

Jon Kleinberg and Eva Tardos. 2005. *Algorithm Design 1st Edition.* Pearson.

Igor Rivin, Ilan Vardi, and Paul Zimmerman. 1994. *The N-Queens Problem.* https://doi.org/10.2307/2974691

Michael Sipser. 2012. *Introduction to the Theory of Computation 3rd Edition.* Cengage Learning.

Steven S S. Skiena. 2008. *The Algorithm Design Manual 2nd ed.* Springer.

Neil J. Sloane. 2007. *Number of ways of placing n nonattacking queens on an n X n board.* Retrieved March 20, 2023 from https://oeis.org/A000170