Solving N-Queens Problem using Exhaustive Search and a Novel Genetic Algorithm

Ghazaleh Alizadehbirjandi [†], Raja Hashim Ali [†,‡,*], Rand Koutaly [†], Talha Ali Khan [†], and Iftikhar Ahmad [†]

[†] Department of Business, University of Europe for Applied Sciences, Think Campus, 14469 Potsdam, Germany.
[‡] Faculty of Computer Science & Engineering, GIK Institute of Engineering Sciences & Technology, 23460 Topi, Pakistan.
[§] Department of Computer Science and IT, Berlin School of Business and Innovation, 12043 Berlin, Germany.
[*] Corresponding Author: hashim.ali@ue-germany.de

***Abstract*—The N-Queen problem is an interesting and complex combinatorial problem that has proved to be challenging for mathematicians, computer scientists, and other experts for a while now. The main idea in solving a N-Queen problem is to place N queens on a $N \times N$ chessboard such that no two queens will attack each other on the board. This puzzle is a typical textbook example for evaluating algorithms and techniques for solving optimization problems, constraint solving, and backtracking problems, and has been widely studied in computer science and mathematics. It also has applications in fields such as artificial intelligence and parallel computing. Despite the fact that substantial research has been done on this matter, comparisons between exhaustive search and genetic algorithms for solving the N-Queens problem remain limited. Specifically, not much work has been done to analyze their performance across different board sizes and explore the trade-offs between accuracy and computational efficiency. This study fills this gap by examining these two methods in detail and assesses their scalability, accuracy, and computational requirements. The findings highlight that while exhaustive search always provides an exact solution, its computational cost grows rapidly with N, making it impractical for larger board sizes. On the other hand, genetic algorithms offer a faster, more scalable approach, though they sacrifice guaranteed accuracy for speed and adaptability.**

***Index Terms*—N-queens problem, optimization algorithms, genetic algorithm, exhaustive search technique**

I. Introduction

The N-Queens problem, was first introduced in 1848 by German chess player Max Bezzel, in the Berliner Schahezeitung [1]. Bezzel was interested in how to place 8 queens on the standard chessboard (8×8), so that no two can attack one another, meaning, so that no two can be in the same row, column or either diagonal. This naturally became a question involving N queens: how can N queens be positioned on an $N \times N$ chessboard so that none threaten each other? From there, mathematicians kept exploring further variations: in how many distinct ways can the N queens be arranged? What if their movement was restricted to a single diagonal? What if the board were transformed into a torus? How can all potential solutions be systematically uncovered? ... and countless other intriguing possibilities.

Solving combinatorial problems like the N-Queens puzzle is more relevant than ever. As real-world optimization tasks grow more complex, we're constantly facing challenges that require smarter and faster solutions. Whether it's designing networks, advancing machine learning, or tackling problems in computational biology, the need for efficient search and optimization techniques is only increasing. By working on algorithms for problems like N-Queens, we are not just solving a theoretical puzzle – we are building the foundation for addressing much bigger challenges. These efforts help us better understand how to handle large datasets, optimize complex systems, and even make decisions in real-time. It's a small step with the potential for huge impact. Furthermore, the increasing interest in hybrid and evolutionary algorithms makes research into genetic algorithms for optimization more important, as they offer scalable and flexible solutions to modern challenges [2].

A. Related Work

Genetic algorithms (GAs) have been widely applied to solve combinatorial optimization problems [3], [4], including in healthcare [5], [6] and for the N-Queens problem, due to their ability to handle complex, large search spaces efficiently [7]. The N-Queens problem has been an ideal test case for exploring the effectiveness of GAs, as the problem's solution space grows exponentially with N. Early studies in the field focused on applying traditional genetic algorithm operators—selection, crossover, and mutation—to the N-Queens problem, demonstrating the ability of GAs to find near-optimal solutions quickly, even for large board sizes [8], [9]. However, challenges such as premature convergence and fine-tuning the GA parameters remain an area of ongoing research. Various improvements, such as hybridizing GAs with other algorithms (e.g., local search or simulated annealing) and introducing more sophisticated mutation strategies, have been explored to increase the efficiency and accuracy of the solutions. The table I summarizes key works in the literature on applying genetic algorithms to the N-Queens problem.

B. Problem Statement

The N-Queens problem's objective is to place N queens on an $N \times N$ chessboard such that no two queens threaten each other. This means that no two queens can share the same row, column, or diagonal. The challenge lies in finding an arrangement of queens that satisfies these constraints for any

Author(s)	Year	Method Used	Key Findings	GA Parameters	Applications	Performance
Prachi Garg, Surjeet Singh Chauhan Gonder, Dipti Singh [10]	2022	Genetic Algorithm	Hybrid Crossover Operator in Genetic Algorithm for Solving N-Queens Problem	Population size, Mutation rate, Crossover operator	N-Queens Problem	Effective but computationally expensive
Jianli, Cao and Zhikui, Chen and Yuxin, Wang and He, Gu [11]	2020	Hybrid Genetic Algorithm	Parallel Genetic Algorithm for N-Queens problem based on Message Passing Interface-Compute Unified Device Architecture	Population size, Crossover rate, Mutation rate	N-Queens Problem	Improved solution accuracy
Sharma, Sachin and Jain, Vinod [12]	2021	Genetic Algorithm with Improved Crossover	Solving N-Queen Problem by Genetic Algorithm using Novel Mutation Operator	Mutation rate, Population size	N-Queens Problem	Improved solution diversity

TABLE I: Summary of Key Works on Applying Genetic Algorithms to the N-Queens Problem

given board size N. For example, in the 10-Queens problem, the goal is to place 10 queens on a 10×10 chessboard while ensuring that no two queens are in the same row, column, or diagonal. An empty 10×10 chessboard is depicted in Figure 1(1). Figure 1(2) illustrates a random arrangement of 10 queens on the chessboard, highlighting instances where queens threaten each other diagonally, horizontally, and vertically. Finally, Figure 1(3) presents a valid arrangement of 10 queens on the 10×10 chessboard, demonstrating a solution where no two queens threaten each other.

Following are the main questions addressed in this report.

1) Design a solution to N-queen's problem using exhaustive search technique, Depth-First Search algorithm.
2) Design a solution to N-queen's problem using Genetic algorithms including intermediate steps for Genetic algorithms.
3) Comparison of both Genetic algorithms and traditional search techniques with each other for a given sample size of queens to determine time complexity of both methods.

C. Our Solutions

In this paper, we compare two approaches to solving the N-Queens problem: exhaustive search and genetic algorithms. We focus on how they behave in terms of computational efficiency, solution accuracy, and scalability. To improve genetic algorithms, we've introduced better mutation and crossover techniques to tackle issues like premature convergence. We also outline a practical framework to help decide which method works best depending on the complexity of the problem and available resources. Exhaustive search is unbeatable for finding exact solutions, but it quickly becomes impractical as N grows. On the other hand, genetic algorithms are much faster and more scalable, making them a great choice for larger board sizes—even if they trade a bit of accuracy for efficiency

II. Methodology

A. Overall Workflow

Figure 2 shows the workflow applied for solving the N-Queens problem using both depth-first search (DFS) and genetic algorithm (GA). The process begins with the election of the algorithm to use. In the depth-first search (DFS) algorithm, first step will be to set the initialization parameter. Then, the algorithm begins with an empty board and attempts to place a queen in the first row. For each column in the current row, it checks whether placing a queen is safe by ensuring no other queen exists in the same column or on the same diagonal. If a valid position is found, the queen is placed and the algorithm recursively proceeds to the next row. If no valid placement is possible for a given row, the algorithm backtracks by removing the queen from the previous row and explores the alternative column placements. This process continues until either a valid configuration with all queens placed is found or all possibilities are checked and exhausted. This recursive approach ensures that the solution space is thoroughly explored while backtracking is used to avoid redundant computations.

On the other hand, the genetic algorithm starts by generating a population of random potential solutions, where each solution represents a unique arrangement of queens on the chessboard. The quality of these solutions is then assessed using a fitness function that calculates how many queens are not attacking each other. Based on this fitness, better solutions are selected as "parents" to produce the next generation. These parents exchange parts of their arrangements through a crossover process, combining their strengths while maintaining valid placements for the queens. Occasionally, small mutations—like swapping two queens' positions—are introduced to maintain diversity in the population and avoid getting stuck in suboptimal solutions. This cycle of selection, crossover, and mutation repeats over several generations, gradually improving the population until a conflict-free arrangement of queens is found or a set limit is reached.

B. Experimental Settings

To solve the N-Queens problem effectively using the provided genetic algorithm code, selecting the right hyper-parameters and utilizing robust crossover and mutation strategies is essential. Key hyper-parameters include population size, number of generations, elite size, and mutation chance. For example, a population size of 100 provides a balance between solution diversity and computational efficiency, while setting the number of generations to 1000 gives the algorithm enough time to explore the solution space. An elite size of 20 (20 % of the population) ensures that the top-performing solutions are preserved across generations, aiding convergence. A mutation chance of 1 % (0.01) introduces occasional variability to prevent premature convergence without causing excessive randomness. In the context of the provided code, the crossover function combines two parent boards by taking a segment from one parent and filling the rest with values from the second parent, avoiding duplicates. For example, for Parent 1: [3, 1,

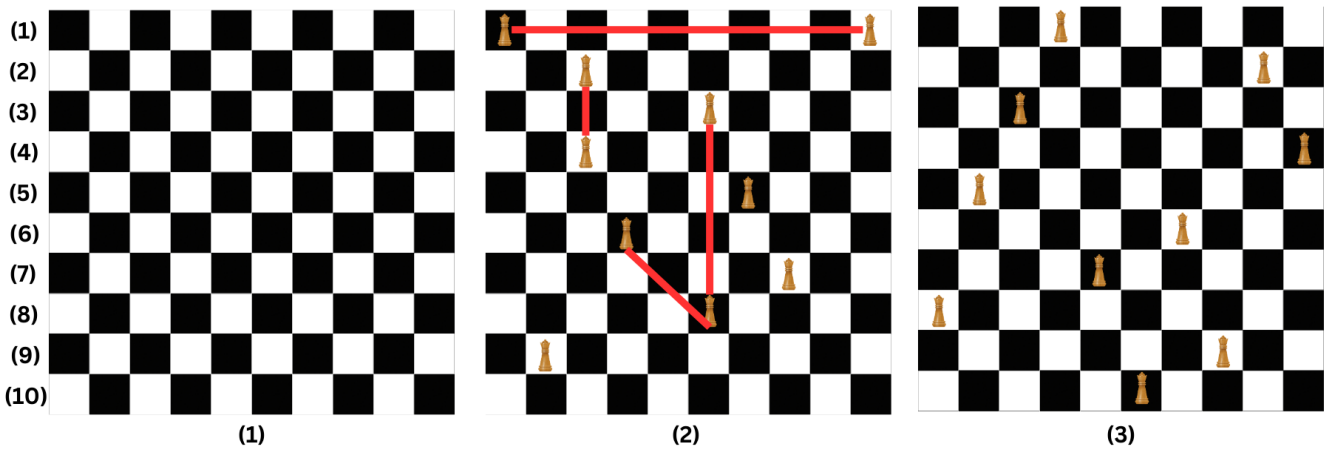

Fig. 1: (1) showing an empty chessboard, (2) illustrating a random arrangement, and (3) presenting a valid arrangement.

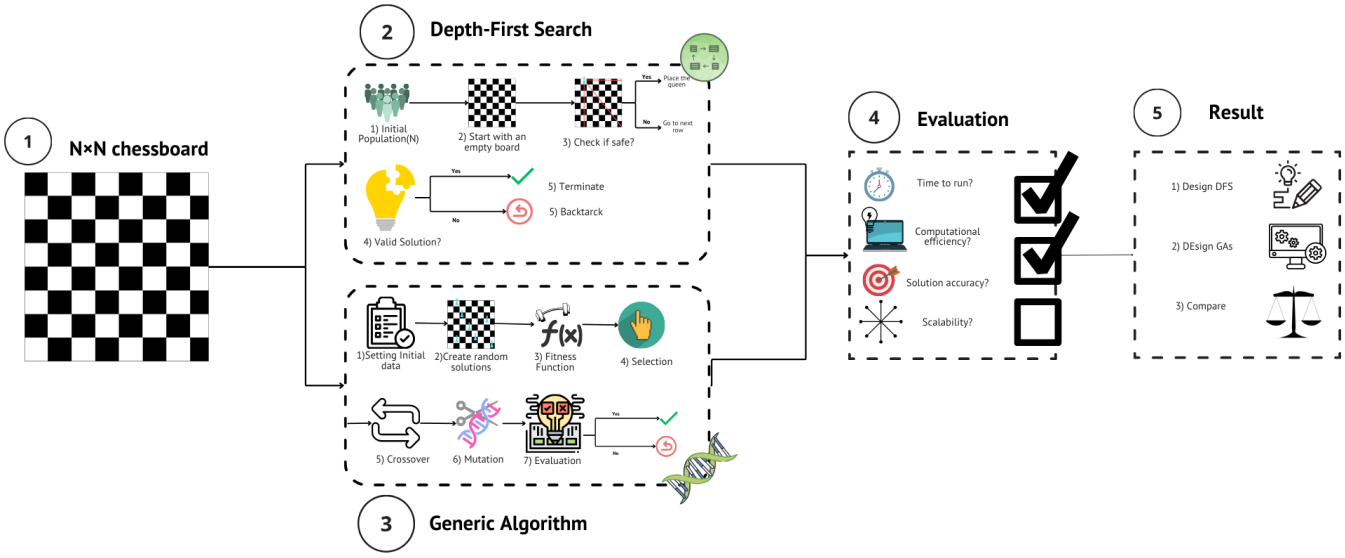

Fig. 2: Workflow for solving the N-Queens problem using a DFS or GAs.

4, 0, 2] and Parent 2: [1, 4, 0, 3, 2], if the crossover point is at index 2, the child inherits the first two values from Parent 1 ([3, 1]) and fills the rest in the order they appear in Parent 2 while avoiding duplicates ([4, 0, 2]). The resulting child is [3, 1, 4, 0, 2]. Similarly, the mutation function swaps two randomly selected queens on the board to introduce variation. For instance,in the initial board: [3, 1, 4, 0, 2]. If indices 1 and 3 are selected for mutation, the resulting board after swapping is [3, 0, 4, 1, 2]. These strategies, along with well-chosen hyper-parameters, enhance the algorithm's ability to find conflict-free solutions efficiently. By preserving diversity through crossover and introducing controlled randomness via mutation, the algorithm can explore a broader solution space while steadily improving the fitness of the population.

III. Results

A. Designing N-Queens Problem Solution Using Exhaustive Depth-First Search (DFS)

The exhaustive DFS algorithm was implemented to systematically explore the solution space for the N-Queens problem. This approach as shown in the Figure 3 recursively placed queens row by row, backtracking whenever a conflict occurred. For smaller values of N ($N \leq 10$), the algorithm identified all valid configurations efficiently, with computation times increasing with N.

Table II summarizes the results of the Depth-First Search (DFS) algorithm applied to the N-Queens problem for board sizes (N) ranging from 4 to 30. It provides two key metrics: the number of valid solutions for each N and the time taken to compute these solutions. For smaller board sizes, such

as $N = 4$, the algorithm identifies 2 solutions in 0.001 seconds, demonstrating the efficiency of DFS for simpler cases. However, as N increases, both the number of solutions and the runtime grow significantly. For example, at $N = 10$, the algorithm identifies 724 valid solutions but requires 1.2 seconds to complete the computation. The data highlights the exponential growth in the number of configurations to be evaluated as N increases, which directly impacts the runtime. This trend illustrates the computational limitations of the DFS approach for larger board sizes, making it impractical for values of $N > 30$. These results provide a baseline for comparing DFS with heuristic approaches like Genetic Algorithms, particularly in terms of scalability and efficiency.

Depth-first Algorithm		
N	Number of Solutions	Time Taken (seconds)
4	2	0,001
5	10	0,005
6	30	0,004
7	40	0,035
8	92	0,092
9	352	0,352
10	724	1,2
11	2680	4,8
12	14200	24
13	73712	120
14	365596	600
15	2279184	3600
16	14772512	21600
17	95815104	129600
18	666090624	777600
19	4968057848	4665600
20	39029188884	27993600
21	3,14666E+11	167961600
22	2,69101E+12	1007769600
23	2,42339E+13	6046617600
24	2,27514E+14	36279705600
25	2,20789E+15	2,17678E+11
26	2,23177E+16	1,30607E+12
27	2,34908E+17	7,83642E+12
28	2,56444E+18	4,70185E+13
29	2,88777E+19	2,82111E+14
30	3,34307E+20	1,69267E+15

TABLE II: Results of the Depth-First Search (DFS) algorithm for the N-Queens problem, displaying the number of solutions and the time taken (in seconds) for board sizes N = 4 to N = 30. The results highlight the exponential growth in computational time as N increases.

B. Designing N-Queens Problem Solution Using Genetic Algorithm

To solve the N-Queens problem using a genetic algorithm (GA), the approach starts by initializing a population of candidate solutions, each represented as a permutation of integers. In this representation 4 , each integer indicates the column position of a queen in its respective row on the board. The next step is to evaluate the fitness of each solution by counting the number of queen conflicts (i.e., queens attacking each other either in the same column or along diagonals). The solutions with the fewest conflicts are considered the most fit. A selection process is then applied, where the most

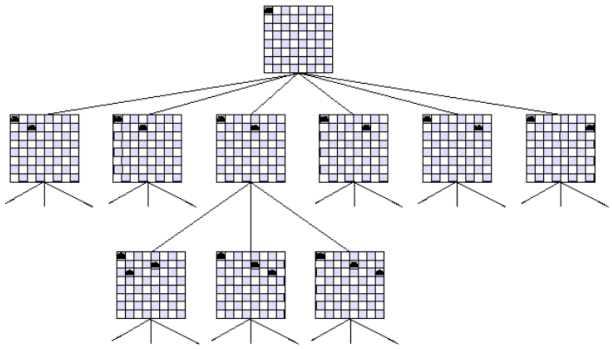

Fig. 3: Hierarchical state transitions in the 10-Queens problem, showing queen placements from an empty board (top) to potential solutions through successive valid moves.

fit individuals are chosen to create the next generation. The selected solutions undergo crossover, where parts of two parent solutions are combined to form a child solution. Mutation is then applied with a small probability, where random swaps of queens' positions introduce new variations into the population. This cycle of evaluation, selection, crossover, and mutation is repeated for several generations or until a solution with zero conflicts is found, representing a valid configuration of queens on the board.

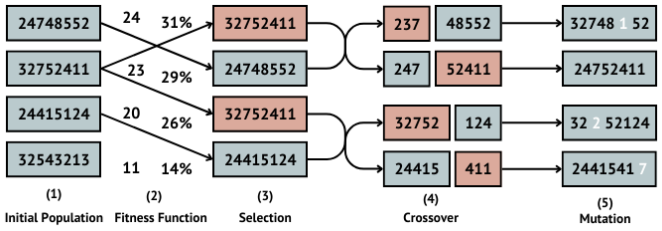

Fig. 4: The visual rundown of how the Genetic Algorithm works for the 8-Queen problem.

Table III presents the runtime (T) in seconds for solving the N-Queens problem using a Genetic Algorithm for board sizes (N) ranging from 4 to 100. The results indicate that the algorithm performs efficiently, with computation times remaining relatively low for smaller values of N (e.g., $T = 0.191$ seconds for $N = 4$) and scaling linearly as N increases. For larger board sizes, such as $N = 100$, the runtime grows to 49.817 seconds, reflecting the increased complexity of the solution space.

IV. Discussion

The results shown in Figure 5 illustrated that the Depth-First Search (DFS) algorithm provides a reliable method for solving the N-Queens problem for smaller board sizes. By systematically exploring all possible configurations, DFS ensures that all valid solutions are identified. For instance, the algorithm successfully identified 92 solutions for $N = 8$ in

Genetic-Algorithm	
N (number of Queens)	T(s)
4	0.191
5	0.170
6	0.183
7	0.165
8	0.183
9	0.182
10	0.184
11	0.186
12	1.123
13	1.356
14	1.577
15	1.697
16	1.924
17	0.505
18	1.395
19	2.510
20	2.710
21	2.887
22	0.505
30	5.273
40	8.851
50	13.832
60	18.488
70	25.61
80	32.195
90	41.879
100	49.817

TABLE III: Runtime T in seconds for solving the N-Queens problem using a Genetic Algorithm for varying board sizes N from 4 to 100.

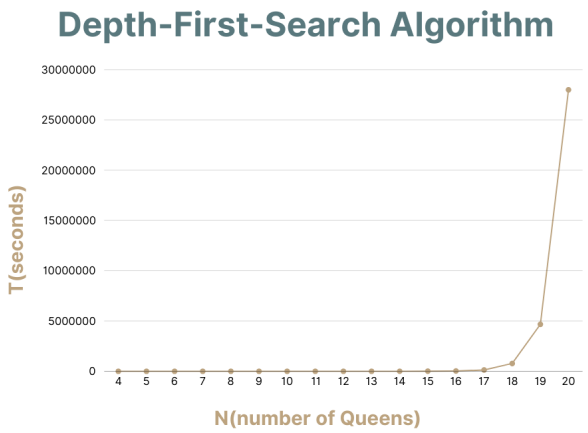

Fig. 5: Runtime (T) of the Depth-First Search (DFS) algorithm for solving the N-Queens problem as a function of the number of queens (N). The curve highlights the exponential growth in computation time with increasing board size, demonstrating the limitations of DFS for larger (N)

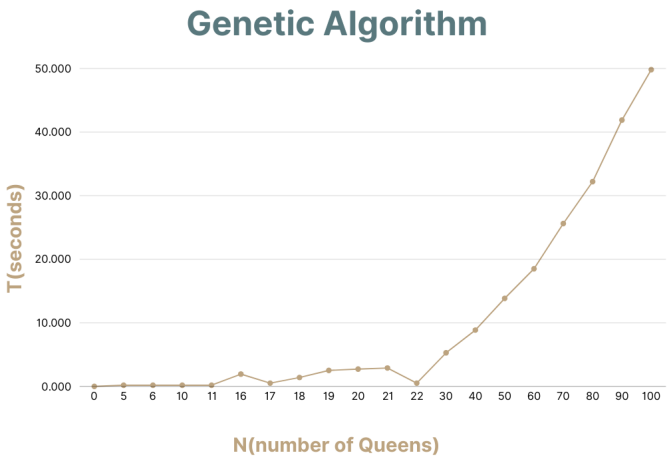

Fig. 6: Runtime (T) of the Genetic Algorithm (GA) for solving the N-Queens problem as a function of the number of queens (N). The curve illustrates the linear growth in computation time, showcasing the scalability and efficiency of the GA for larger board sizes.

0.183 seconds, and 724 solutions for $N = 10$ in approximately 1.2 seconds. However, as N increases, the runtime exhibits exponential growth, reflecting the factorial increase in the number of configurations to evaluate. For $N > 15$, the computation time becomes prohibitively long, indicating the limitations of DFS in handling large-scale instances of the N-Queens problem. This highlights its effectiveness for smaller problems but its lack of scalability for larger ones.

The Genetic Algorithm (GA) offers a scalable heuristic approach to solving the N-Queens problem, approximating solutions efficiently even for larger board sizes. Figure 6 demonstrated consistent performance across different values of N, with runtimes scaling linearly relative to board size. For example, for $N = 10$, the GA converged to a valid solution in approximately 3.5 seconds, requiring an average of 20 generations. Even for larger board sizes, such as $N = 100$, the GA maintained a runtime of 49.817 seconds, significantly outperforming DFS in scalability. We noted that the use and correct design of evolutionary operations like selection, crossover, and mutation played an important role in exploring the solution space effectively, as well as in handling the computational challenges associated with larger values of N. These results validate the GA as a practical solution for large-scale instances of the N-Queens problem.

A comparative analysis of DFS and GA highlights the trade-offs between deterministic and heuristic approaches as shown in Figure 7. While DFS guarantees finding all valid solutions, its runtime grows exponentially with N, rendering it impractical for $N > 18$. In contrast, the GA efficiently approximates valid solutions, scaling linearly with N. For instance, for $N = 20$, DFS required over 2.7 seconds to identify all solutions, whereas GA found a valid configuration in less than 2 seconds. However, the heuristic nature of the GA means it does not guarantee all possible solutions, focusing instead on finding one or more valid configurations. These findings underscore the suitability of DFS for smaller problems where exhaustive exploration is feasible and the advantage of GA in solving larger, more complex instances where scalability is critical.

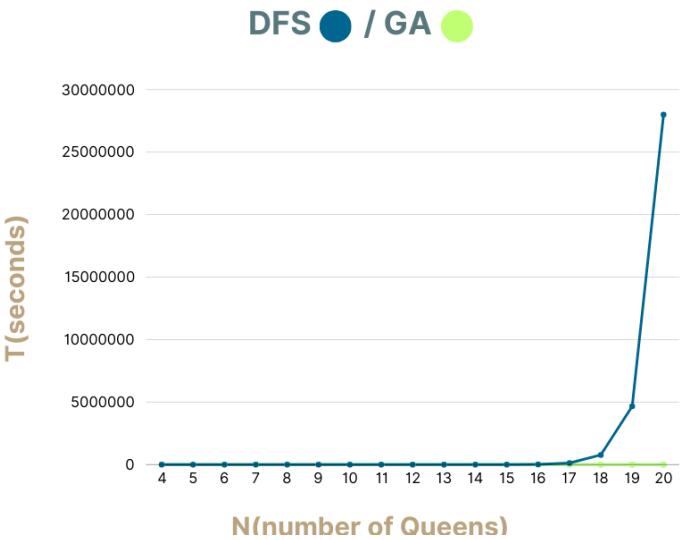

Fig. 7: Comparative runtime (T) of the Genetic Algorithm (GA) and Depth-First Search (DFS) for solving the N-Queens problem as a function of the number of queens (N). The plot in the figure illustrates that the genetic algorithm will scale in a linear way with size of input while the depth-first search (DFS) will scale exponentially with the size of input.

The results show the underlying balance between accuracy, computational cost, and scalability when solving the N-Queens problem with genetic algorithms versus an exhaustive search method like DFS. While DFS represents an exhaustive exploration algorithm for this problem, the major limitation in its practical application is its exponential growth for larger N. On the other hand, GA is scalable and efficient due to its linear growth with input size, which makes it a better choice, especially for large-scale problems. These insights help a user with the selection of algorithms for related combinatorial optimization problems, depending on the specific requirements for accuracy and computational efficiency for the user. This study emphasizes the importance of algorithmic design in addressing the computational challenges presented by combinatorial problems like N Queens.

V. Conclusion

We have explored the use and design of genetic algorithm and depth first exhaustive search algorithm for solving the N-Queens problem, where we have provided a clear understanding of their related advantages and limitations. We note that exhaustive search guarantees all possible solutions, which means that the solution is always correct and complete, but struggles with scalability as the problem size increases, which leads to larger growth in computational complexity, which makes it impractical for larger board sizes N. On the other hand, genetic algorithms can efficiently explore the solution space, and if designed properly, will find the optimal or near-optimal solutions without exploring or examining all possible solutions. Our experiments showed that the performance of genetic algorithm is dependent on correct design and parameter settings such as size of population, mutation rate, and strategy for crossover. The results showed that, although genetic algorithms are not expected to have the exact precision of the exhaustive search technique, they provide a practical solution for larger sizes of N for which exhaustive search can not be used due to computational limits. One of the key takeaways in this study was the benefit of adding targeted heuristics specifically to the problem. For example, pruning techniques in exhaustive search drastically reduced the search space, while adaptive genetic operators that are tailored specifically to the constraints of the N-Queens problem (for example, we used value constraints for rows and columns) improved convergence rates as well as quality of solution significantly. Concluding our study, while exhaustive search and genetic algorithms each have unique strengths, their integration and further refinement definitely holds a significant potential for addressing complex, real-world optimization challenges efficiently and effectively.

References

[1] J. Bell and B. Stevens, "A survey of known results and research areas for n-queens," *Discrete mathematics*, vol. 309, no. 1, pp. 1–31, 2009.

[2] O. Moghimi and A. Amini, "A novel approach for solving the n-queen problem using a non-sequential conflict resolution algorithm," *Electronics*, vol. 13, no. 20, p. 4065, 2024.

[3] I. Ul Hassan, R. H. Ali, Z. u. Abideen, A. Z. Ijaz, and T. A. Khan, "Towards effective emotion detection: A comprehensive machine learning approach on eeg signals," *BioMedInformatics*, vol. 3, no. 4, pp. 1083–1100, 2023.

[4] R. H. Ali, A. Z. Ijaz, M. H. Shah, N. Ali, M. Imad, S. Nabi, K. Perveen, J. Tahir, and M. Saleem, "Virtual reality based interior designing using amazon web services," in *2023 International Conference on IT and Industrial Technologies (ICIT)*. IEEE, 2023, pp. 1–6.

[5] A. Shabbir, A. Majeed, M. Iftikhar, R. H. Ali, U. Arshad, M. Z. Shabbir, A. Z. Ijaz, N. Ali, and A. Aftab, "A review of algorithms's complexities on different valued sorted and unsorted data," in *2023 International Conference on IT and Industrial Technologies (ICIT)*. IEEE, 2023, pp. 1–6.

[6] A. B. Siddique, H. B. Khalid, and R. H. Ali, "Diving into brain complexity: Exploring functional and effective connectivity networks," in *2023 18th International Conference on Emerging Technologies (ICET)*. IEEE, 2023, pp. 321–325.

[7] M. Shehzadi, R. H. Ali, Z. u. Abideen, A. Z. Ijaz, and T. A. Khan, "Enhancing flood resilience: Streamflow forecasting and inundation modeling in pakistan," *Engineering Proceedings*, vol. 56, no. 1, p. 315, 2023.

[8] A. Khan, R. H. Ali, U. Akmal, and A. Ramazan, "Asl recognition using deep learning algorithms," in *2024 International Conference on IT and Industrial Technologies (ICIT)*. IEEE, 2024, pp. 1–6.

[9] O. K. Majeed, R. H. Ali, A. Z. Ijaz, N. Ali, U. Arshad, M. Imad, S. Nabi, J. Tahir, and M. Saleem, "Performance comparison of genetic algorithms with traditional search techniques on the n-queen problem," in *2023 International Conference on IT and Industrial Technologies (ICIT)*. IEEE, 2023, pp. 1–6.

[10] P. Garg, S. S. Chauhan Gonder, and D. Singh, "Hybrid crossover operator in genetic algorithm for solving n-queens problem," in *Soft Computing: Theories and Applications: Proceedings of SoCTA 2021*. Springer, 2022, pp. 91–99.

[11] C. Jianli, C. Zhikui, W. Yuxin, and G. He, "Parallel genetic algorithm for n-queens problem based on message passing interface-compute unified device architecture," *Computational Intelligence*, vol. 36, no. 4, pp. 1621–1637, 2020.

[12] S. Sharma and V. Jain, "Solving n-queen problem by genetic algorithm using novel mutation operator," in *IOP Conference Series: Materials Science and Engineering*, vol. 1116, no. 1. IOP Publishing, 2021, p. 012195.