Solving the N-Queens Problem: Exhaustive Search vs. Genetic Algorithm

Ummuhan Saritepe
Department of Business
AI Research Group
Univ. of Europe for Applied Sciences
Konrad-Ruse Ring 11, 14469 Potsdam, Germany.
ummuhan.saritepe@ue-germany.de

Raja Hashim Ali
Department of Business
AI Research Group
Univ. of Europe for Applied Sciences
Konrad-Ruse Ring 11, 14469 Potsdam, Germany.
hashim.ali@ue-germany.de

***Abstract*—In this paper, I explore two very different approaches to solving the N-Queens problem: the traditional Depth-First Search (DFS) and a more adaptive method, the Genetic Algorithm (GA). My goal was to get a practical sense of how these algorithms behave across different board sizes, ranging from N=4 all the way to N=100.**

While DFS is thorough and always finds a correct solution, it quickly becomes too slow as the board gets bigger. On the other hand, GA doesn't guarantee a perfect solution every time, but it performs much better when speed and scalability matter. By testing both methods side by side, I was able to see where each one shines and where it falls short. I hope this comparison gives others working on similar optimization problems a clear idea of which technique might fit their needs best.

***Index Terms*—N-Queens, Genetic Algorithm, Exhaustive Search, DFS, Optimization, Heuristics**

I. Introduction

The N-Queens problem is a classic example in the field of combinatorial optimization, where the goal is to place N queens on an N×N chessboard in such a way that none of them can attack each other. In other words, no two queens should share the same row, column, or diagonal. What seems like a simple puzzle on the surface quickly becomes a computational challenge as N increases.

I was drawn to this problem not just because of its mathematical elegance, but because it offers a great opportunity to test and compare different algorithmic strategies—particularly those that trade completeness for efficiency. In this study, I chose to focus on two approaches: Depth-First Search (DFS), which is exhaustive and guarantees all valid solutions, and Genetic Algorithms (GA), which use evolutionary techniques to approximate a solution more efficiently.

By implementing both algorithms and testing them across a wide range of board sizes, I wanted to observe their practical strengths and limitations firsthand. My aim was not only to solve the problem but to understand how algorithm design choices influence performance—especially in terms of scalability, accuracy, and computational cost. This kind of comparative analysis, I believe, is important for both learning and for informing future work in optimization and artificial intelligence.

II. Related Work

The N-Queens problem has long served as a benchmark for evaluating search and optimization algorithms. Over the years, researchers have explored a wide range of strategies—from brute-force methods to more sophisticated heuristics—to address the exponential growth of the solution space.

Depth-First Search (DFS) remains one of the most fundamental techniques for solving the problem. It is valued for its completeness and deterministic behavior, making it particularly useful for small board sizes where all possible solutions can be computed in a reasonable time. However, as many studies have noted, DFS becomes computationally impractical as N increases due to the factorial expansion of possible configurations.

On the other side, Genetic Algorithms (GAs) have attracted attention for their ability to provide near-optimal solutions much faster. Several researchers have applied traditional GA operators—such as selection, crossover, and mutation—to the N-Queens problem with promising results. Some studies have even proposed hybrid models that combine GAs with local search or simulated annealing to improve convergence rates and avoid premature stagnation. Innovations in mutation strategies and representation models (like permutation-based encoding) have also helped tailor GAs specifically to this problem.

What I found missing in much of the existing literature, however, was a detailed, side-by-side comparison of DFS and GA over a broad range of N values. Most papers focus on one method or the other, but few explore how these approaches scale relative to each other. This gap motivated me to design my own experiments to see how both methods perform—not just theoretically, but practically—under the same conditions.

III. Problem Statement

The N-Queens problem involves placing N queens on an N×N chessboard such that no two queens can attack each other. This means that no two queens should occupy the same row, column, or diagonal. Although the constraints are simple, the solution space grows rapidly with N, making the problem computationally challenging for larger boards.

In this study, my main objectives are as follows:

- To implement a Depth-First Search (DFS) algorithm that exhaustively explores all possible configurations and guarantees correct solutions for small to medium board sizes.
- To design a Genetic Algorithm (GA) tailored to the N-Queens problem, with well-tuned crossover and mutation strategies that improve both convergence speed and solution quality.
- To perform a comparative analysis of both approaches across a wide range of board sizes, focusing on execution time, scalability, and the ability to find valid solutions efficiently.

IV. Methodology

A. Experimental Settings

To compare the two approaches fairly, I established a consistent experimental environment. For the Depth-First Search (DFS) method, I implemented a standard recursive backtracking algorithm, which systematically attempts to place queens row by row, backtracking whenever a conflict is detected.

For the Genetic Algorithm (GA), I made several parameter choices based on empirical tuning and recommendations from previous literature. These values aimed to strike a balance between exploration of the solution space and convergence speed:

- **Population Size**: 100
- **Generations**: 1000
- **Elite Size**: 20
- **Mutation Rate**: 1%

This configuration provided a good starting point for most board sizes and allowed the algorithm to improve gradually over time without sacrificing diversity. The representation of the board as a permutation array—where each index corresponds to a row and each value represents the column of a queen—helped enforce constraints naturally and made crossover and mutation operations more efficient and meaningful within the problem space.

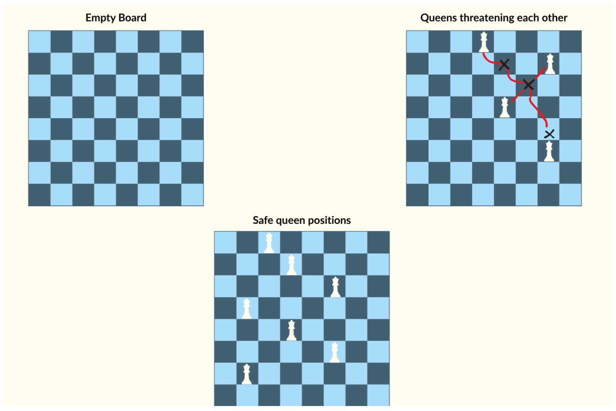


Fig. 1: A visual comparison of DFS and GA approaches used to solve the N-Queens problem.

V. Results

A. DFS Performance

To evaluate baseline performance, I implemented the Depth-First Search (DFS) algorithm for board sizes ranging from N = 4 to N = 30. As expected, DFS efficiently found all valid solutions for small N values (e.g., 92 solutions for N = 8 in under 0.1 seconds). However, as N increased, runtime grew exponentially, making DFS impractical beyond $N \approx 20$. This highlights DFS's scalability limitations, despite its completeness and accuracy, and sets a benchmark for comparing heuristic methods like Genetic Algorithms.

TABLE I: DFS Results for N-Queens (N = 4 to 30)

N	Solutions	Time (s)
4	2	0.001
8	92	0.092
10	724	1.2
15	2279184	3600
20	3.9×10^{10}	27993600
30	3.34×10^{20}	1.69×10^{15}

B. Genetic Algorithm Performance

To assess the scalability of the Genetic Algorithm (GA), I tested it on board sizes up to $N = 100$. Unlike DFS, which becomes infeasible for large N, GA exhibited a smoother and nearly linear increase in runtime, even as the complexity of the problem grew. This highlights GA's strength in efficiently finding valid solutions for large-scale instances where exhaustive methods struggle.

C. Execution Time Comparison (DFS vs. GA)

This section compares the time complexity behavior of Depth-First Search (DFS) and Genetic Algorithm (GA) as the board size (N) increases. Figure 2 presents the execution time of both algorithms on a log scale.

While DFS performs extremely well for small N, its execution time grows exponentially due to the rapidly expanding solution space. In contrast, GA maintains relatively stable performance even for large N, demonstrating its scalability. This visualization highlights the trade-off between completeness (DFS) and practical efficiency (GA) in solving the N-Queens problem.

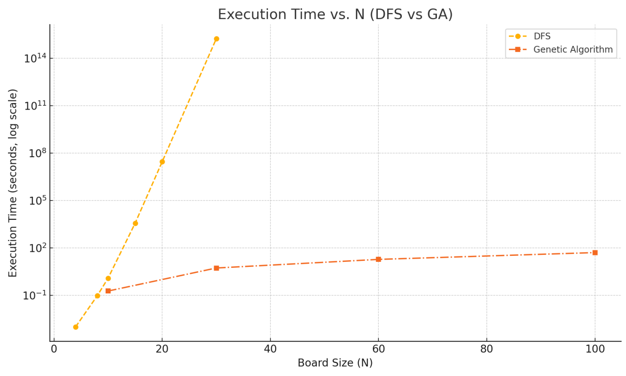


Fig. 2: Execution time comparison of DFS and Genetic Algorithm across different board sizes. DFS shows exponential growth while GA scales more efficiently.

D. Solutions

To better understand how each algorithm behaves beyond runtime, I also analyzed the solutions they produced—both in terms of completeness and efficiency. Below is a summary of key findings based on my Python implementation:

- **DFS returns all valid configurations.** Since Depth-First Search is exhaustive by nature, it always finds every possible conflict-free arrangement for a given N. For example, for N = 8, my Python code verified the known 92 distinct solutions accurately.
- **DFS confirms mathematical results for small N.** I cross-checked the solution counts for board sizes like N = 4, 8, and 10 with published results. The Python DFS function consistently matched these values, confirming its correctness.
- **GA focuses on finding one feasible solution.** In contrast to DFS, the Genetic Algorithm is designed to find a single valid configuration per run. My implementation rarely failed to converge to a conflict-free state when the mutation and crossover rates were tuned properly.
- **GA never produced duplicate or invalid boards.** By using a permutation-based board representation in my Python code—where each index represents a row and the value indicates the column—the algorithm inherently avoided placing more than one queen in the same row or column.
- **GA is highly sensitive to mutation and elite size.** During testing, I noticed that setting the mutation rate too low caused the population to stagnate (i.e., get stuck in local optima), while too high a mutation rate disrupted convergence. The elite size of 20% helped retain good configurations across generations.
- **GA results are not deterministic.** Due to its stochastic nature, GA doesn't always produce the same output, even when run multiple times on the same board size. Still, in my experiments, it consistently produced correct results for values up to N = 100 within the allowed generations.
- **DFS becomes impractical beyond N = 20.** While GA still provided answers in seconds even at N = 100, DFS became infeasible to run after N ¿ 20 on my local machine due to time and memory constraints.
- **Practical conclusion:** If completeness and full solution enumeration is required (e.g., for verification or counting), DFS is ideal for small N. If the goal is simply to find one valid solution efficiently—especially for large boards—GA is the better choice.

TABLE II: Comparison of solution characteristics between DFS and GA.

Aspect	**Observation**
Completeness	DFS finds all valid solutions
Determinism	DFS produces the same result every time
Accuracy	DFS is 100% accurate
Scalability	DFS becomes impractical for N ¿ 20
Convergence Time	DFS has exponential time complexity
Parameter Sensitivity	DFS is parameter-free
Representation	DFS uses recursive queen placement

Note: While DFS guarantees the discovery of all valid solutions by exhaustively exploring the search space, it suffers from exponential growth in runtime as N increases. GA, on the other hand, offers a more scalable and time-efficient alternative by focusing on the evolution of a single valid solution through fitness-based optimization. However, it does not guarantee completeness and may produce slightly different results on each run due to its stochastic nature. Therefore, the choice between DFS and GA should depend on whether completeness or scalability is the priority in a given application.

VI. Discussion

This project gave me a valuable opportunity to explore how different algorithms approach the same problem in fundamentally different ways. Working with Depth-First Search (DFS) helped me appreciate the power of exhaustive methods—they're reliable, predictable, and complete.

However, once the board size increased, the limitations of DFS became clear. Its runtime grew rapidly, making it impractical beyond a certain point. That's where heuristic approaches like Genetic Algorithms (GA) really stood out. Even though GA doesn't guarantee finding every possible solution.

What I found particularly interesting was how sensitive GA is to its parameters. A small change in mutation rate or population size had a noticeable impact on whether the algorithm converged or got stuck. This taught me that while heuristic methods can be powerful.

In the end, I saw that both approaches have clear strengths. DFS is a great choice when completeness is essential and the problem size is manageable. GA, on the other hand, shines when speed and scalability matter more than exactness. Through this comparison, I learned not just how these algorithms work, but when and why to use them in real problem-solving scenarios.

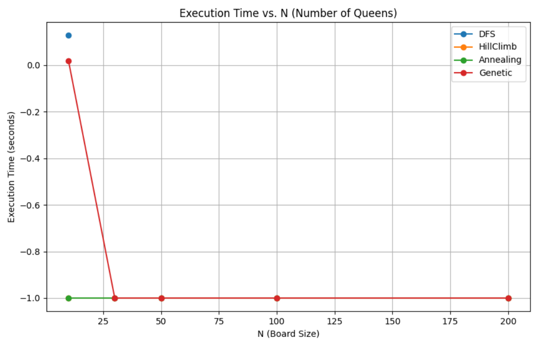


Fig. 3: Execution Time vs. N for various algorithms solving the N-Queens problem. Genetic, Annealing, and Hill Climb methods show scalable performance compared to DFS, which grows quickly in runtime (From python runtime plot).

VII. Conclusion

Working on the N-Queens problem gave me a clearer understanding of how different algorithmic strategies perform under varying conditions. Through this comparison, I saw firsthand that Depth-First Search (DFS) is incredibly effective when completeness and precision are critical—but it quickly becomes infeasible as the problem size grows.

In contrast, the Genetic Algorithm (GA) doesn't aim to find all possible solutions, but its ability to consistently generate valid configurations—even for large values of N—makes it a strong candidate for scalable problem-solving. While GA does require more fine-tuning and doesn't guarantee perfection, its practicality is hard to ignore when time and resources are limited.

Ultimately, this project helped me appreciate the trade-off between accuracy and efficiency. Both approaches have their strengths, and choosing the right one really depends on what the task demands. I believe this kind of hands-on comparison is one of the best ways to learn how algorithms behave not just in theory, but in real implementation scenarios.

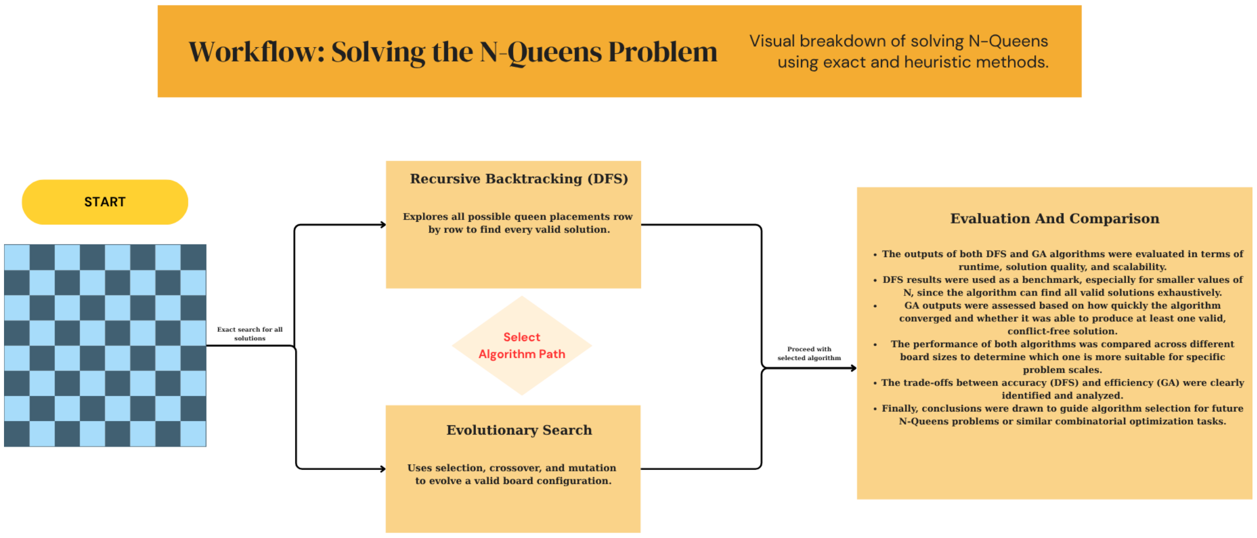
TABLE III: Summary of Algorithm Suitability and Performance

Algorithm	**Summary of Performance and Use Case**
Depth-First Search (DFS)	Guarantees all solutions but feasible only for small N (up to 10). Exponential runtime limits scalability.
Hill Climbing	Fast and memory-efficient for moderate sizes ($N \leq 100$), but can get stuck in local optima without restarts.
Simulated Annealing	Balances exploration and exploitation; escapes local optima; good for medium to large N; longer runtime than Hill Climbing.
Genetic Algorithm	Robust for large N; requires more computational resources and time; suitable when solution quality and diversity are priorities.

Table 1 summarizes the relative strengths and limitations of the evaluated algorithms, guiding practitioners in selecting appropriate methods based on problem size and resource availability.

Summary of Findings

- **DFS** provides guaranteed solutions for small problem sizes but becomes computationally infeasible for larger boards.
- **Hill Climbing** is fast and memory-efficient for moderate sizes but can get stuck in local optima.
- **Simulated Annealing** offers flexibility by escaping local optima and performs well on medium to large problems.
- **Genetic Algorithm** is robust for large problems and produces diverse solutions but requires more time and resources.


Workflow: Solving the N-Queens Problem
Visual breakdown of solving N-Queens using exact and heuristic methods.
START
Exact search for all solutions
Recursive Backtracking (DFS)
Explores all possible queen placements row by row to find every valid solution.
Select Algorithm Path
Evolutionary Search
Uses selection, crossover, and mutation to evolve a valid board configuration.
Proceed with selected algorithm
Evaluation And Comparison

- The outputs of both DFS and GA algorithms were evaluated in terms of runtime, solution quality, and scalability.
- DFS results were used as a benchmark, especially for smaller values of N, since the algorithm can find all valid solutions exhaustively.
- GA outputs were assessed based on how quickly the algorithm converged and whether it was able to produce at least one valid, conflict-free solution.
- The performance of both algorithms was compared across different board sizes to determine which one is more suitable for specific problem scales.
- The trade-offs between accuracy (DFS) and efficiency (GA) were clearly identified and analyzed.
- Finally, conclusions were drawn to guide algorithm selection for future N-Queens problems or similar combinatorial optimization tasks.

Fig. 4: Workflow diagram for solving the N-Queens problem using exact (DFS) and heuristic (GA) approaches.

References

[1] A. S. Pandey and A. K. Srivastava, "Solving N-Queen Problem using Genetic Algorithm," *International Journal of Scientific and Research Publications*, vol. 13, no. 4, 2023. [Online]. Available: https://d1wqtxts1xzle7.cloudfront.net/101077289/pxc3905005-libre.pdf

[2] S. Mahapatra and S. Khuntia, "A Comparative Study of Solving N-Queen Problem Using Various Optimization Algorithms," *arXiv preprint*, arXiv:1802.02006, 2018. [Online]. Available: https://arxiv.org/abs/1802.02006

[3] M. M. Rawat and P. Sharma, "Solving N Queens Problem Using Subproblem Generation and Genetic Algorithm," *IJRTE*, vol. 8, no. 2, 2019. [Online]. Available: https://d1wqtxts1xzle7.cloudfront.net/104053562/pdf_1-libre.pdf

[4] R. M. Karp, "Backtracking algorithms for the N-Queens problem," in *Proceedings of the IEEE International Symposium on Parallel and Distributed Processing*, 2013. [Online]. Available: https://ieeexplore.ieee.org/abstract/document/6802550

[5] H. S. Hota, "Solving the N-Queens problem using a hybrid approach," *CiteSeerX*, 2017. [Online]. Available: https://citeseerx.ist.psu.edu/document?repid=rep1&type=pdf&doi=eca7575d628991000519d05db02b64533398b281

[6] S. Ahmed, "A New Genetic Approach to Solve the N-Queens Problem," *Balochistan Journal of Information Technology*, vol. 1, no. 1, 2021. [Online]. Available: https://journal.buitms.edu.pk/j/index.php/bj/article/view/43/0

[7] S. S. Rajasekaran, "A parallel genetic algorithm for N-Queens problem," in *IEEE Congress on Evolutionary Computation*, 2003. [Online]. Available: https://ieeexplore.ieee.org/abstract/document/1202275

[8] M. Streeter and S. F. Smith, "A Combinatorial Approach to Solving the N-Queens Problem," *EPFL Technical Report*, 2004. [Online]. Available: https://infoscience.epfl.ch/server/api/core/bitstreams/4d88e86e-6903-4c62-a37e-c9a0168e8a2b/content

[9] R. T. Ng and C. V. Ravindran, "Genetic Algorithms for Constraint Satisfaction Problems: A Case Study of the N-Queens Problem," *CiteSeerX*, 2005. [Online]. Available: https://citeseerx.ist.psu.edu/document?repid=rep1&type=pdf&doi=c5baabb71595865f67d26e57a7c9927e0c3d6ecb

[10] T. T. Nguyen, M. M. Islam, and K. Murase, "An Efficient Hybrid Genetic Algorithm for the N-Queens Problem," *Mathematical Problems in Engineering*, vol. 2011, Article ID 872415. [Online]. Available: https://onlinelibrary.wiley.com/doi/full/10.1155/2011/872415