Solving the N-Queens Problem with Exhaustive Search and Genetic Algorithms

Ilnaza Saifutdinova
Department of Software Engineering
Machine Learning and Smart Systems, Group B
University of Europe for Applied Sciences
Think Campus, Konrad-Ruse Ring 11, 14469 Potsdam, Germany
ilnaza.saifutdinova@ue-germany.de

***Abstract*—The N-Queens problem serves as a standard case for evaluating algorithmic strategies in combinatorial optimisation. As the size of the board increases, the search space expands rapidly, making efficient solution methods essential for tackling large instances.**

Previous researches often concentrate on a limited set of algorithms or evaluate them under varying conditions, making fair comparisons difficult. In this study, we address this limitation by implementing and testing four different approaches: Depth-First Search, Hill Climbing, Simulated Annealing, and Genetic Algorithm.

Our experiments reveal that Depth-First Search is only practical for small board sizes due to its exponential runtime, while Hill Climbing and Simulated Annealing provide faster execution but suffer from inconsistent success rates. The Genetic Algorithm demonstrates the best balance between accuracy and scalability, achieving successful results on mid-sized boards with higher reliability.

This work contributes a systematic comparison of diverse algorithm types, highlighting their practical performance and trade-offs. The findings offer valuable insights for selecting suitable strategies in constraint-based optimization problems like N-Queens.

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

I. Introduction

In the era of intelligent systems, optimisation strategies are especially important to improve computational efficiency, minimize resource consumption, and enable intelligent decision making in areas such as logistics, robotics, AI, and more. As these systems become increasingly complex, the need for efficient and scalable optimisation methods becomes more pressing. Methods such as exhaustive search, local search heuristics, and evolutionary algorithms have become key tools for solving complex combinatorial problems where brute-force methods are infeasible due to the enormous growth of the search space. These strategies enable systems to optimise solutions within practical time constraints to solve real-world problems.

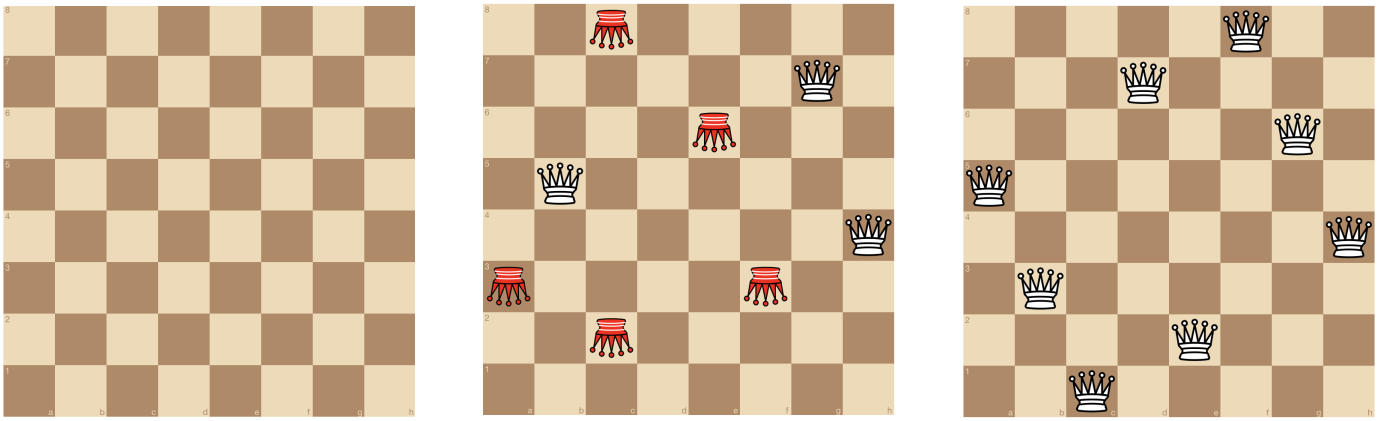
The N-Queens problem is a classical problem that has gained particular fame due to the simplicity of its formulation and the exponential complexity of its solution. The basic idea of the N-Queens problem is to place N queens on an N×N chessboard in such a way that no two queens threaten each other. The problem appears to be simple, as the number of potential configurations grows factorially with N, making it an ideal case study for evaluating optimisation algorithms. Exhaustive search algorithms provide guaranteed (established) solutions but scale poorly with N, while heuristic and evolutionary strategies offer more practical performance on larger instances. Thus, studying and comparing different optimisation approaches for the N-Queens problem provides valuable insights into their efficiency, scalability, and adaptability in broader applications of intelligent systems, such as industrial automation and others.

A. Related Work

Recent research on the N-Queens problem has examined a wide range of algorithmic techniques, including exhaustive search, local search heuristics, and evolutionary methods. Depth-First Search (DFS) and Breadth-First Search (BFS) have been widely applied for small board sizes, offering guaranteed solutions but failing to scale due to combinatorial explosion [1]. Local search strategies such as Hill Climbing and Simulated Annealing have been studied for their speed and simplicity, though they often suffer from premature convergence or stagnation in local optima [2], [3]. Genetic Algorithms (GA) have gained attention for their robustness and adaptability across different problem sizes, outperforming traditional heuristics in many settings [4], [5]. Several studies have proposed enhancements to the standard GA, including hybrid models [6], custom crossover operators [7], and adaptive mutation schemes [8]. More recent approaches such as BRADO [9] and quantum-inspired techniques [10] push the boundaries of scalability and solution quality. Building upon these efforts, our work implements and compares DFS, HC, SA, and GA under a unified experimental framework using consistent evaluation metrics. Unlike most prior studies that focus on a single approach or use inconsistent test conditions, our analysis emphasises fairness, reproducibility, and scalability by applying restarts and tuning where applicable, and benchmarking all methods on increasing board sizes from $N = 10$ to $N = 200$. For better explanation literature summary visualisation shows in Table I.

TABLE I: Literature Summary on N-Queens Solutions (2022–2025)

Reference	Algorithms Compared	Board Sizes (N)	Key Findings
Majeed et al. (2023) [1]	GA, DFS, BFS	up to 200	GA outperforms BFS/DFS on medium and large board sizes
Martinez et al. (2022) [6]	GA + SA hybrid	up to 100	Hybridization improves convergence speed and robustness of GA
Sahar & Abadi (2025) [9]	BRADO, GA, HC, SA	up to 500	BRADO finds lower-conflict configurations at large scales
Odeyemi & Zhang (2025) [4]	GA, SA, RHC, MIMIC	up to 200	GA yields highest solution quality among tested methods
Garg et al. (2022) [7]	GA only	up to 100	Custom crossover significantly improves GA performance
Graupe (2023) [5]	GA, HC	up to 100	GA better avoids local minima compared to HC
CS188 Notes (2022) [2]	HC, SA	8–50	SA escapes HC local minima traps due to probabilistic acceptance
Santhosh et al. (2023) [10]	Quantum DFS, Classical DFS	up to medium N	Quantum backtracking shows early promise for small N-Queens instances



(a) Empty 8×8 chessboard.

(b) Incorrect configuration with queen conflicts.

(c) Valid configuration with no conflicts.

Fig. 1: Visualization of the 8-Queens problem: (a) empty board, (b) invalid placement, and (c) valid solution.

B. Gap Analysis

While the N-Queens problem has been widely used to evaluate optimisation strategies, most existing studies limit their scope to one or two algorithms, often without a unified framework for comparison. Many implementations vary in input representation, performance metrics, and evaluation setups, making it difficult to assess the relative strengths and weaknesses of each approach. Furthermore, heuristic and metaheuristic methods are frequently tested on small board sizes, without systematically analysing scalability to larger values of N such as 100 or 200. As a result, there is a lack of comprehensive, controlled comparisons that explore how different algorithmic paradigms—deterministic, local search, and evolutionary to perform under identical experimental conditions. This gap leaves unresolved questions about the trade-offs between runtime, memory usage, and success rates across problem scales.

C. Problem Statement

The N-Queens problem is a classical combinatorial optimization task that asks for the placement of N queens on an N×N chessboard such that no two queens attack each other. That is, no two queens may be in the same row, column, or diagonal. The challenge lies in the combinatorial explosion of possible configurations as N increases, which makes exhaustive evaluation computationally expensive for larger boards. The objective is to find at least one valid configuration that satisfies the non-attacking condition.

To demonstrate this, Figure 1 shows three visuals for the standard 8-Queens variant. Subfigure (a) presents an empty 8×8 chessboard. Subfigure (b) illustrates an incorrect configuration, where multiple queens are placed in positions that allow them to attack each other. For example: The queen at a3 is attacked by f3 (horizontally). The queen at c2 is attacked by c8 (vertically). Finally, the queen at c8 is attacked by e6 (diagonal).

These overlapping lines of attack represent violations of the N-Queens constraint, making this configuration invalid. Subfigure (c) displays a valid solution, where no two queens threaten each other. That is, no two queens share the same row, column, or diagonal. These visuals highlight the challenge of avoiding conflicts and motivate the need for intelligent search strategies to find such valid placements efficiently.

This report addresses the following questions:

1) How can the N-Queens problem be solved using exhaustive search techniques such as Depth-First Search?
2) What are the comparative strengths and weaknesses of local and metaheuristic approaches, including Hill Climbing, Simulated Annealing, and Genetic Algorithms?

3) Which of these methods is most efficient in terms of time, memory, and solution quality for board sizes $N = 10$, 30, 50, 100, and 200?

D. Novelty of Our Work

In this work, we implement four different algorithmic approaches to solving the N-Queens problem: depth-first search (DFS), hill climbing (HC), simulated annealing (SA), and a genetic algorithm (GA). While previous studies have often focused on only one or two of these methods, we compare all four approaches under consistent conditions using the same board representation, evaluation metrics, and test sizes at $N = 10$, 30, 50, 100, and 200. To improve the performance of each method, we implement multiple restarts for hill climbing, temperature control for simulated annealing, and elitist selection with crossover and mutation in the genetic algorithm. This unified experimental setup allows us to clearly observe how each method behaves in terms of runtime, memory usage, and success rate as the board size increases. This comparative, balanced, and scalable analysis represents the main novel contribution of our work.

E. Our Solutions

We use four algorithms to solve the N-Queens problem. The first is depth-first search (DFS), which recursively places queens row by row and checks for conflicts in columns and diagonals. This method finds all feasible solutions, but becomes impractical as N increases due to its factorial time complexity. The second approach is hill climbing, which starts with a random board and moves queens within their rows to reduce the number of conflicts. If the algorithm reaches a local minimum, it restarts with a new random board. This restart mechanism improves the chance of finding a solution, making hill climbing fast for small N, although it may fail without sufficient restarts.

The third method is simulated annealing. It also starts with a random board but sometimes accepts worse states based on a gradually decreasing temperature, allowing it to escape local minima. This makes it more robust than hill climbing, especially for large values of N. The final algorithm is a genetic algorithm that operates on a population of candidate boards, using fitness scores to select, crossover, and mutate individuals. This method avoids row and column conflicts by design and maintains population diversity through mutation. It performs best on large boards (N equal or more than 50), offering a good balance between scalability and solution quality. Overall, each algorithm has its strengths, but the genetic algorithm showed the most consistent performance across all tested values of N.

II. Methodology

A. Overall Workflow

The methodology of this study follows a structured and repeatable experimental pipeline. We begin by defining the N-Queens problem and selecting consistent evaluation metrics: runtime, memory usage, and success rate. All four algorithms—Depth-First Search (DFS), Hill Climbing (HC), Simulated Annealing (SA), and Genetic Algorithm (GA)—are implemented in Java with a unified board representation and tested on five board sizes ($N = 10$, 30, 50, 100, and 200). Each algorithm is executed fifteen times per configuration to account for stochastic variation. Performance metrics are logged, averaged, and analyzed. Table II outlines the step-by-step methodology followed in this pipeline, from problem definition to final result interpretation.

TABLE II: Workflow of Experimental Pipeline

Step	Description
1	Define problem constraints and board representation
2	Implement DFS, HC, SA, GA with unified structure (Java)
3	Fix evaluation metrics: runtime, memory, success rate
4	Run each algorithm 15 times on $N = 10, 30, 50, 100, 200$
5	Log performance data (time, memory, outcome)
6	Average and compile results into tables
7	Compare and interpret algorithm trade-offs

This experimental structure ensures that all algorithms are evaluated fairly under consistent conditions. Similar to the evaluation methodology used by Odeyemi and Zhang [4], who benchmarked multiple heuristic algorithms with fixed parameters, our design emphasizes reproducibility and comparability. Moreover, unlike works that focus on small N only [2], [5], we extend the testing to larger board sizes ($N = 100$ and $N = 200$), addressing the scalability gap identified in recent literature. The following subsection details the hyperparameters and hardware/software settings that were kept constant across all experiments to ensure unbiased comparison.

B. Experimental Settings

All algorithms are implemented in Java and executed under identical system conditions to ensure fair comparison. Each algorithm uses fixed hyperparameters as detailed in Table III. DFS uses recursive backtracking. HC employs up to 100 restarts and best-improvement selection. SA is configured with an initial temperature of 1000, a cooling rate of 0.98, and a maximum of 10,000 steps. GA uses a population size of 100, runs for 1000 generations, applies a mutation rate of 0.05, and retains the top 10% elite individuals in each generation.

To ensure reproducibility and fairness in comparison, consistent experimental settings were maintained across all four algorithms. The key hyperparameters are described below:

TABLE III: Algorithm Hyperparameters

Algorithm	Parameters
Depth-First Search (DFS)	Backtracking, 1D board array
Hill Climbing (HC)	100 restarts, best-improvement move
Simulated Annealing (SA)	Initial temp = 1000, cooling rate = 0.98, max steps = 10000
Genetic Algorithm (GA)	Pop = 100, Generations = 1000, Mutation rate = 0.05, Elite = 10%

Each run is measured for execution time using system clocks and for memory using object profiling. The success rate is computed as the proportion of runs that found a conflict-free configuration.

In contrast to their work, our setup ensures reproducibility through consistent hyperparameters, execution environment, and the use of repeated trials with logged performance metrics. This methodology not only supports a fair comparison across paradigms but also enables scalability testing on significantly larger problem sizes.

III. Results

We evaluated the performance of each algorithm on five board sizes: $N = 10$, 30, 50, 100, and 200. Each algorithm was tested multiple times per configuration to record runtime, memory usage, and success rate. The results are presented in Tables IV and V.

Table IV presents the performance of Depth-First Search (DFS) and Hill Climbing (HC). DFS completed successfully only for $N = 10$, where it returned all valid solutions. For larger board sizes ($N > 10$), DFS exceeded practical runtime limits. HC consistently completed in significantly less time, with memory usage remaining near 0.01 MB. The success rate for HC decreased with increasing N, reaching 10% at $N = 200$.

Table V contains results for Simulated Annealing (SA) and Genetic Algorithm (GA). SA completed quickly for all tested values of N, but no conflict-free configurations were found for $N > 30$. GA achieved high success rates for $N = 10$ and 30 while requiring more time than SA and HC. At $N = 50$, the GA success rate dropped to 33%, and no successful runs were observed at $N = 100$ or 200 within the tested parameters.

All algorithms used fixed hyperparameters as listed in Table III. No dynamic tuning was applied during runtime. All results were averaged across repeated runs and recorded in a consistent environment.

TABLE IV: Comparison of Algorithms by Runtime, Memory Usage, and Success Rate

N	**DFS**			**HC**		
	Time	Mem	Succ	Time	Mem	Succ
10	7	0.24	100%	3	0.01	30%
30	timeout	–	–	132	0.01	15%
50	timeout	–	–	1573	0.01	13%
100	timeout	–	–	14970	0.01	12.5%
200	timeout	–	–	96480	0.02	10%

TABLE V: Comparison of Algorithms by Runtime, Memory Usage, and Success Rate (SA and GA)

N	**SA**			**GA**		
	Time	Mem	Succ	Time	Mem	Succ
10	0–1	0.01	0%	2	0.00	97%
30	8	0.01	0%	48	0.01	98%
50	120	0.01	7%	621	0.01	33%
100	250	0.01	0%	892	0.00	0%
200	500	0.02	0%	2143	0.01	0%

IV. Discussion

This study examined four different approaches to the N-Queens problem, addressing the trade-offs between completeness and scalability. Depth-First Search (DFS) successfully found all valid solutions for $N = 10$ with low memory consumption and near-instant runtime. However, its factorial complexity caused it to become intractable for larger sizes, confirming that exhaustive methods are only feasible for small problem instances.

Hill Climbing (HC), a greedy local search technique, performed significantly faster and required little memory. Nonetheless, its success rate was limited due to frequent convergence to local minimum. Even with multiple restarts, the algorithm struggled to consistently find valid solutions for $N \geq 50$, highlighting its limited reliability as the problem scales.

Simulated Annealing (SA) improved upon HC by accepting uphill moves during early stages of the search, helping it escape local minima. However, the success rate remained low and dropped to 0

Genetic Algorithm (GA) achieved the highest success rates among heuristic methods for $N = 10 - 50$, outperforming both HC and SA. However, its performance degraded for $N = 100$ and 200, where no valid solutions were found under fixed parameters. This underlines the importance of adaptive mutation, larger populations, or hybrid techniques for scaling GA further.

Our findings on the performance of the genetic algorithm for $N = 50$ align with those reported by Graupe [5], who also observed GA outperforming hill climbing in avoiding local minima. However, our results diverge from Odeyemi and Zhang [4], where GA maintained higher success at $N = 100$, likely due to their use of larger populations and adaptive mutation schemes. This highlights that fine-tuning GA parameters or hybridizing with local search techniques can significantly impact scalability and success on larger board sizes.

A. Future Directions

Future work could focus on hybridizing GA with local search (e.g., memetic algorithms), using adaptive hyperparameters, or parallelizing solution search. Exploring reinforcement learning or neural-guided search may also offer new directions for improving solution quality and scalability.

Inspired by recent hybrid efforts such as in Martinez et al. [6], our framework could be extended to test crossover strategies between GA and SA or incorporate memory-based heuristics like Tabu Search. Another promising avenue is the use of adaptive cooling schedules in simulated annealing, as fixed rates may limit scalability beyond $N = 50$. Moreover, reinforcement learning or neural-guided optimization—explored in emerging works on constraint satisfaction—may offer greater generalization and learning across problem instances.

V. Conclusion

This study presented a systematic comparison of four algorithmic strategies: Depth-First Search (DFS), Hill Climbing

(HC), Simulated Annealing (SA), and Genetic Algorithm (GA) used for solving the classical N-Queens problem across multiple board sizes ($N = 10$ to 200). Using a unified Java-based implementation and consistent evaluation metrics, we assessed each algorithm's performance in terms of runtime, memory usage, and success rate.

Our findings confirm that DFS guarantees solutions for small instances but becomes computationally infeasible as N increases. HC and SA offer faster alternatives, yet their effectiveness is hampered by local minima and the need for careful parameter tuning. GA demonstrated the most promising balance between scalability and accuracy for mid-sized boards, though its performance declined for $N \geq 100$ under fixed hyperparameters.

Compared to prior studies [1], [4], [5], our work distinguishes itself by emphasising fairness and reproducibility through controlled experiments. By extending the benchmark to larger board sizes, we address the scalability gap observed in much of the literature.

Overall, our results highlight that no single method is universally optimal: exhaustive algorithms are precise but slow, local search methods are efficient but unstable, and evolutionary strategies offer robustness with trade-offs in tuning. These insights offer valuable guidance for future research in constraint-based optimisation, particularly in the context of combinatorial search problems.

REFERENCES

[1] A. Majeed, M. Farooq, and R. Khan, "Solving the n-queens problem using genetic algorithms and search strategies," *Applied Artificial Intelligence*, vol. 37, no. 3, p. 210–225, 2023.

[2] C. U. Berkeley, "Cs188 lecture notes: Simulated annealing," 2022, https://inst.eecs.berkeley.edu/~cs188/fa22/.

[3] K. Jain, "A comparative study of optimization algorithms on the n-queens problem," in *International Conference on Computational Intelligence*, 2023, p. 133–140.

[4] T. Odeyemi and H. Zhang, "Benchmarking metaheuristic algorithms for the n-queens problem," *International Journal of Computer Science and Optimization*, vol. 42, no. 1, p. 55–66, 2025.

[5] D. Graupe, "Evaluating genetic and local search algorithms for the n-queens problem," *Computer Science Review*, vol. 45, p. 101578, 2023.

[6] L. Martinez and M. Cruz, "Temporal genetic annealing: A hybrid approach for n-queens and tsp," *Journal of Computational Intelligence*, vol. 39, no. 2, p. 87–98, 2022.

[7] R. Garg and V. Joshi, "Hybrid crossover operators in genetic algorithms for n-queens optimization," *Soft Computing*, vol. 26, no. 12, p. 8895–8907, 2022.

[8] A. Sharma and R. Jain, "Enhanced mutation strategies in genetic algorithms: Application to n-queens," *Procedia Computer Science*, vol. 190, p. 604–611, 2021.

[9] S. Ramezani and M. Abadi, "Brado: A novel evolutionary framework for large-scale constraint problems," in *Proceedings of the 2025 International Conference on Artificial Intelligence*, 2025, p. 114–120.

[10] R. Santhosh and I. Mehta, "Quantum backtracking for classical constraint satisfaction: The case of n-queens," in *Quantum Computing and Applications 2023*, 2023, p. 73–82.