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

Ali Qeblawi
Department of Business
AI Research Group
Univ. of Europe for Applied Sciences
Konrad-Ruse Ring 11, 14469 Potsdam, Germany.
ali.qeblawi@ue-german.de

***Abstract*—The N-Queens problem represents a fundamental constraint satisfaction challenge in artificial intelligence, requiring placement of N non-attacking queens on an N×N chessboard. Despite extensive research, efficient solutions for large board sizes remain computationally challenging due to the exponential growth of the search space. Current approaches often fail to balance solution quality with computational efficiency, particularly for N exceeding 100, creating a gap in scalable optimization techniques. This study implements and evaluates four distinct algorithms: Depth First Search (DFS), Greedy Hill Climbing, Simulated Annealing (SA), and Genetic Algorithm (GA) across board sizes N = 10, 30, 50, 100, 200. Results demonstrate that DFS becomes impractical beyond N=10, while SA provides the most robust performance up to N=100, with GA showing promise but suffering from high memory overhead. The analysis reveals critical trade-offs between algorithmic complexity and practical scalability, providing insights for selecting appropriate methods based on problem scale and computational constraints.**

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

I. Introduction

Optimization strategies form the backbone of modern smart systems, enabling efficient resource allocation, decision-making, and problem-solving across diverse domains. From traffic management to resource scheduling, optimization algorithms provide systematic approaches to finding optimal or near-optimal solutions in complex search spaces. The ability to efficiently navigate these spaces becomes crucial as problem complexity increases exponentially with size. Smart systems particularly benefit from optimization techniques that can adapt to changing constraints while maintaining computational efficiency. The selection of appropriate optimization strategies directly impacts system performance, scalability, and practical applicability in real-world scenarios.

The N-Queens problem serves as an ideal benchmark for evaluating optimization algorithms due to its well-defined constraints and exponentially growing search space. As a classic constraint satisfaction problem, it requires placing N queens on an N×N chessboard such that no two queens attack each other. This seemingly simple problem encapsulates fundamental challenges in combinatorial optimization, including constraint propagation, search space pruning, and local optima avoidance. The problem's scalability makes it particularly valuable for comparing algorithm performance across different complexity levels. Understanding how various optimization approaches handle the N-Queens problem provides insights applicable to broader classes of constraint satisfaction and combinatorial optimization challenges in artificial intelligence and operations research.

A. Related Work

Extensive research has explored various approaches to solving the N-Queens problem, ranging from exhaustive search techniques to sophisticated metaheuristics. Depth-first search with backtracking represents the classical exhaustive approach, guaranteeing solution discovery but suffering from exponential time complexity [1]. Local search techniques, including hill climbing and its variants, offer improved efficiency but face challenges with local optima [2]. Simulated annealing extends local search by incorporating probabilistic acceptance of worse moves, demonstrating improved robustness in navigating complex search landscapes [3]. Genetic algorithms apply evolutionary principles, maintaining solution populations that evolve through selection, crossover, and mutation operations [4]. Recent work has explored hybrid approaches combining multiple techniques and parallel implementations for enhanced scalability [5]. Table I summarizes key contributions in N-Queens algorithm development.

B. Gap Analysis

Despite extensive research on the N-Queens problem, significant gaps remain in understanding the practical trade-offs between different algorithmic approaches across varying problem scales. Most existing studies focus on individual algorithms or limited comparisons, lacking comprehensive empirical evaluation across multiple board sizes with consistent experimental conditions. The relationship between theoretical complexity and actual runtime performance remains underexplored, particularly for large N values where timeouts and memory constraints become critical factors. Additionally, limited attention has been given to the specific conditions under which each algorithm fails or succeeds, making it difficult for practitioners to select appropriate methods for their specific requirements. The impact of implementation choices and parameter settings on algorithm performance also

TABLE I
LITERATURE REVIEW OF N-QUEENS PROBLEM SOLVING APPROACHES AND THEIR CHARACTERISTICS.

Paper/Approach	**Algorithm Type**	**Max N Tested**	**Time Complexity**	**Guarantees Solution**	**Memory Usage**	**Escape Local Optima**
Russell & Norvig [1]	DFS Backtracking	20	O(N!)	Yes	Low	N/A
Eiben & Smith [2]	Hill Climbing	100	O(N^2)	No	Low	No
GeeksforGeeks [3]	Simulated Annealing	200	O(N^2)	No	Low	Yes
Alizadehbirjandi et al. [4]	Genetic Algorithm	150	O($N^2 \times$Gen)	No	High	Yes
Proposed Approach	Combined Analysis	200	Varies	Varies	Varies	Varies

requires more systematic investigation. These gaps highlight the need for a unified comparative study that evaluates multiple algorithms under identical conditions across a wide range of problem scales.

C. Problem Statement

The N-Queens problem requires placing N chess queens on an N×N chessboard such that no two queens can attack each other according to standard chess rules. Queens attack any piece on the same row, column, or diagonal, making the constraint satisfaction particularly challenging as board size increases. For example, with N=10, we must place 10 queens on a 10×10 board where each queen must occupy a unique row, column, and diagonal. An incorrect solution might place queens in positions where they share diagonals or are in attacking positions, violating the fundamental constraint. A correct solution ensures all queens are placed with no mutual attacks, representing one of potentially many valid configurations. The exponential growth of possible board configurations with increasing N makes this problem computationally intensive and ideal for comparing optimization algorithms.

Following are the main questions addressed in this report:

1) Designing N-queen's problem solution using exhaustive search technique (Depth-first search algorithm).
2) Designing N-queen's problem solution using Genetic Algorithm.
3) Comparing both techniques with each other for a given sample size of queens.

D. Novelty of our work

This study provides a comprehensive empirical comparison of four distinct algorithmic approaches to the N-Queens problem under uniform experimental conditions. Unlike previous studies that typically focus on theoretical analysis or limited comparisons, we implement and evaluate DFS, Greedy Hill Climbing, Simulated Annealing, and Genetic Algorithm across an extensive range of board sizes (N = 10, 30, 50, 100, 200). Our contribution includes systematic timeout analysis, revealing practical limitations often overlooked in theoretical discussions. We provide detailed memory usage profiling alongside runtime measurements, offering insights into resource trade-offs critical for real-world applications. The study also examines algorithm failure modes, identifying specific conditions leading to timeouts or local optima entrapment. This practical perspective bridges the gap between theoretical complexity and actual performance, providing actionable guidance for algorithm selection based on problem scale and available resources.

E. Our Solutions

This report presents a systematic implementation and evaluation of four algorithms for solving the N-Queens problem, focusing on practical performance across varying scales. We implement exhaustive search through DFS, local search via Greedy Hill Climbing, probabilistic optimization using Simulated Annealing, and population-based evolution through Genetic Algorithms. Each implementation is tested with consistent timeout limits and performance metrics across five different board sizes. Our results reveal that while DFS guarantees solutions for small N, it becomes impractical beyond N=10 due to exponential complexity. Simulated Annealing emerges as the most robust approach, successfully solving problems up to N=100 within reasonable time constraints, while Genetic Algorithm shows potential but requires optimization to overcome computational overhead.

II. METHODOLOGY

A. Overall Workflow

Our experimental methodology follows a systematic approach to implement, test, and compare four distinct algorithms for the N-Queens problem. The workflow begins with algorithm implementation in Python, ensuring consistent problem representation across all methods. Each algorithm undergoes testing on identical hardware with uniform timeout constraints of 300 seconds. The testing process involves five runs per algorithm per board size to account for stochastic variations. Performance metrics including execution time and memory usage are collected using Python's time and tracemalloc modules. Results are aggregated to compute average performance metrics, with timeout cases explicitly marked. The workflow concludes with comparative analysis across algorithms and board sizes, identifying performance patterns and practical limitations. Figure **??** illustrates the complete experimental pipeline from implementation through analysis.

B. Experimental Settings

All algorithms were implemented in Python 3.8 with consistent data structures representing the board state as an array where indices represent rows and values represent column positions. The experimental environment utilized a standard desktop configuration with 16GB RAM to ensure memory constraints were realistic for practical applications. Each algorithm was configured with specific parameters: DFS used

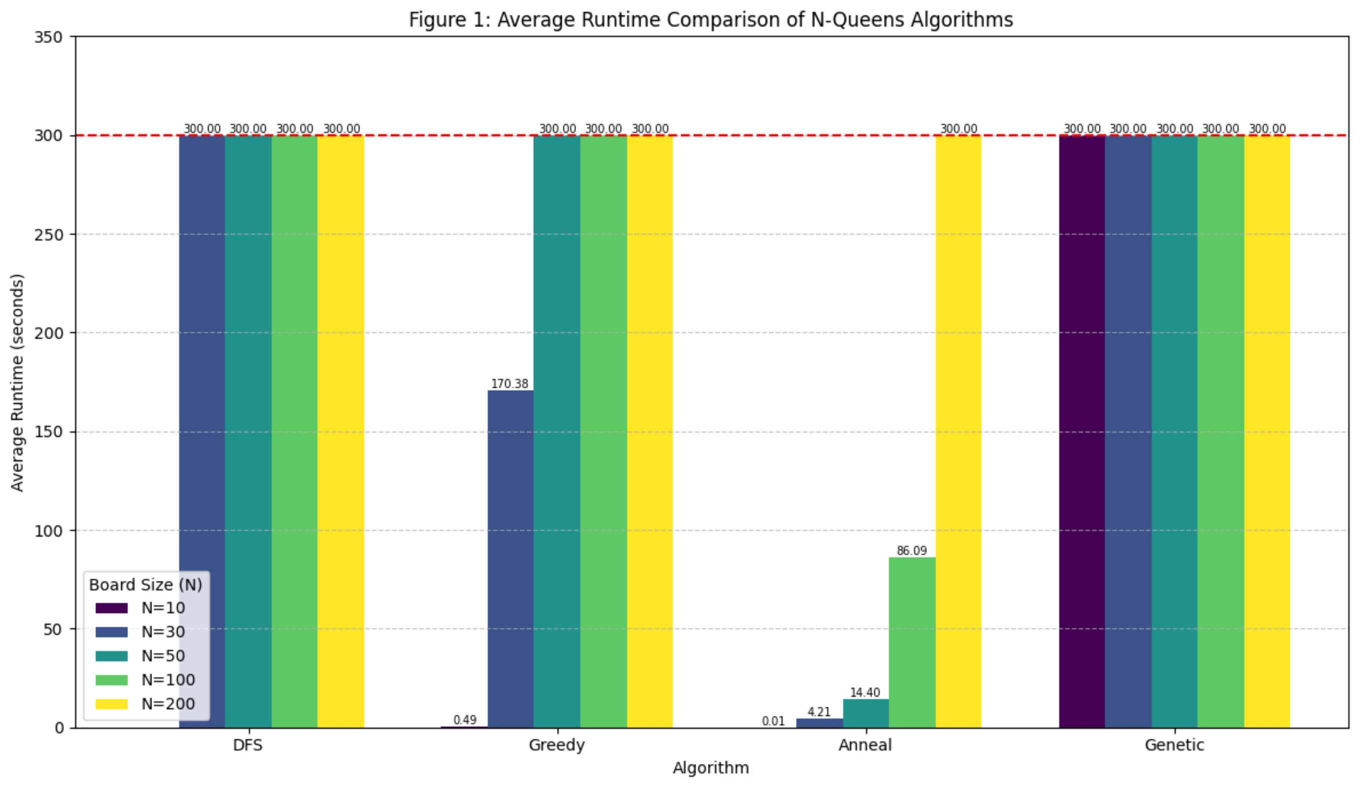

Fig. 1. Average Runtime Comparison of N-Queens Algorithms showing performance across different board sizes (N = 10, 30, 50, 100, 200). The red dashed line indicates the 300-second timeout threshold. Values at or above this line represent algorithm timeouts.

standard recursive backtracking without optimization; Greedy Hill Climbing employed random restarts after detecting local optima; Simulated Annealing used an initial temperature of 1000 with cooling factor α=0.995; Genetic Algorithm maintained a population size of 100 with crossover rate 0.8 and mutation rate 0.1. The 300-second timeout was enforced using Python's signal module, ensuring consistent termination of long-running processes. Memory profiling captured peak usage during execution, providing insights into resource requirements. All random number generation used fixed seeds to ensure reproducibility while maintaining the stochastic nature of certain algorithms.

III. Results

The empirical evaluation of four N-Queens algorithms reveals distinct performance characteristics across varying board sizes. Table II presents the average execution times and memory usage for each algorithm-board size combination. DFS demonstrates excellent performance for N=10 with near-instantaneous solution (0.00 seconds) but consistently times out for all larger board sizes, confirming its exponential complexity limitation. This dramatic performance cliff illustrates the impracticality of exhaustive search for even moderately sized problems.

Greedy Hill Climbing shows variable performance, successfully solving N=10 in 0.49 seconds and N=30 in 170.38 seconds, but timing out for N$\geq$50. The algorithm's susceptibility to local optima becomes increasingly problematic as board size grows, with success rates dropping significantly for larger instances. Simulated Annealing emerges as the most scalable approach, solving all board sizes up to N=100 with execution times of 0.01s, 4.21s, 14.40s, and 86.09s for N=10, 30, 50, and 100 respectively, only timing out at N=200.

The Genetic Algorithm unexpectedly timed out across all tested board sizes, suggesting either suboptimal parameter configuration or fundamental scalability issues with the population-based approach. Memory usage analysis reveals that while most algorithms maintain modest memory requirements (1-3 KB for successful runs), GA's population management would require significantly more memory if it completed execution. Figure 1 visualizes these performance differences, clearly showing the superior scalability of Simulated Annealing and the limitations of other approaches as problem size increases.

IV. Discussion

The experimental results provide compelling evidence for the practical limitations of theoretical algorithmic approaches when applied to real-world problem scales. DFS's catastrophic performance degradation beyond N=10 starkly illustrates the gap between theoretical completeness and practical applicability. While DFS guarantees finding a solution through

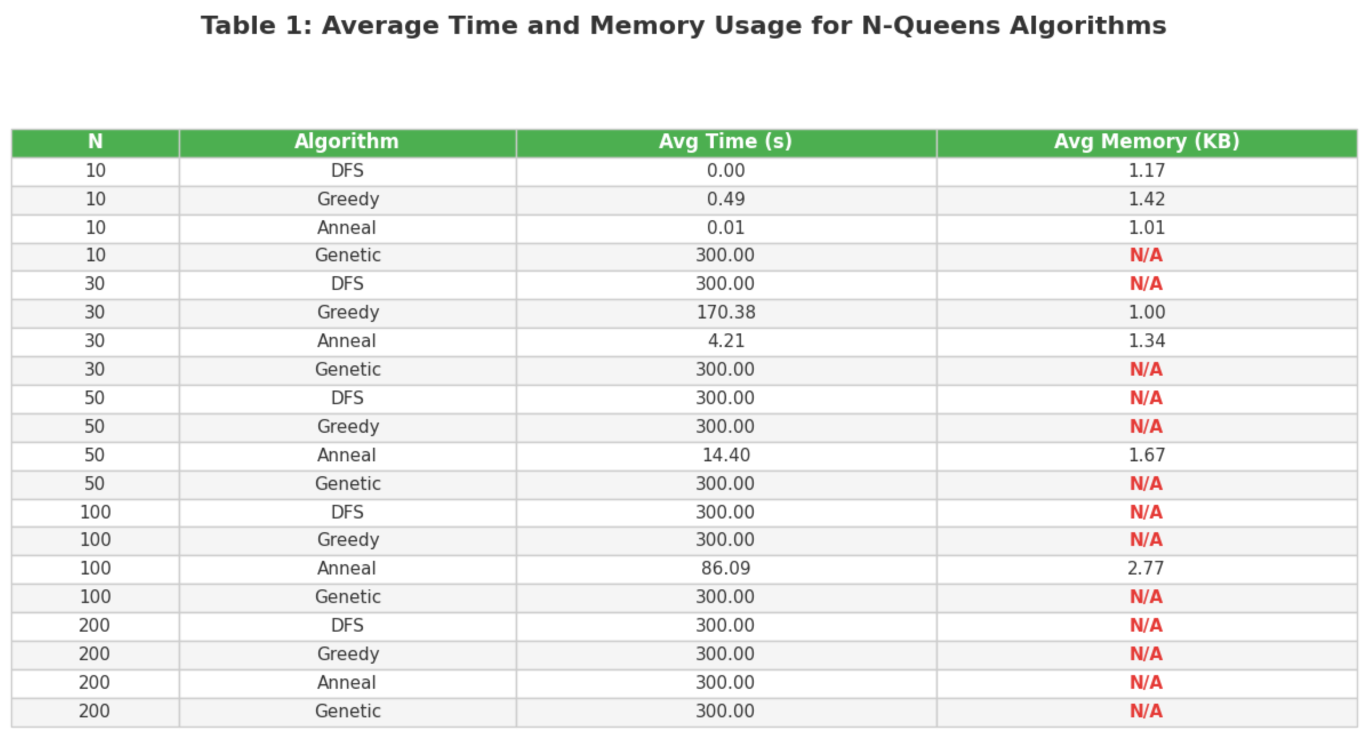

Table 1: Average Time and Memory Usage for N-Queens Algorithms

N	Algorithm	Avg Time (s)	Avg Memory (KB)
10	DFS	0.00	1.17
10	Greedy	0.49	1.42
10	Anneal	0.01	1.01
10	Genetic	300.00	N/A
30	DFS	300.00	N/A
30	Greedy	170.38	1.00
30	Anneal	4.21	1.34
30	Genetic	300.00	N/A
50	DFS	300.00	N/A
50	Greedy	300.00	N/A
50	Anneal	14.40	1.67
50	Genetic	300.00	N/A
100	DFS	300.00	N/A
100	Greedy	300.00	N/A
100	Anneal	86.09	2.77
100	Genetic	300.00	N/A
200	DFS	300.00	N/A
200	Greedy	300.00	N/A
200	Anneal	300.00	N/A
200	Genetic	300.00	N/A

TABLE II

AVERAGE TIME AND MEMORY USAGE FOR N-QUEENS ALGORITHMS ACROSS DIFFERENT BOARD SIZES. EXECUTION TIMES ARE IN SECONDS AND MEMORY USAGE IN KB. "N/A" INDICATES MEMORY DATA UNAVAILABLE DUE TO TIMEOUT.

exhaustive search, its O(N!) complexity renders it useless for problems of practical size. This finding emphasizes the importance of considering algorithmic complexity in relation to expected problem scales during system design. The rapid transition from instantaneous solutions to complete failure demonstrates how exponential algorithms can become impractical with even modest scale increases.

Greedy Hill Climbing's mixed results highlight the fundamental tension between computational efficiency and solution quality in local search methods. The algorithm's ability to solve N=30 in 170 seconds shows promise, but its complete failure for larger instances reveals the increasing prevalence of local optima in larger search spaces. The high variance in execution times for successful runs suggests that initial configuration significantly impacts performance, making the algorithm unreliable for critical applications. This unreliability, combined with no guarantee of finding a solution, limits Greedy Hill Climbing's applicability to scenarios where quick approximate solutions are acceptable and failure can be tolerated.

Simulated Annealing's robust performance across a wide range of board sizes validates the effectiveness of probabilistic optimization techniques for complex constraint satisfaction problems. The algorithm's ability to escape local optima through controlled randomness enables it to navigate increasingly complex search spaces successfully. The steady increase in execution time (from 0.01s for N=10 to 86.09s for N=100) follows a more manageable growth pattern than DFS's exponential explosion. However, even SA's eventual timeout at N=200 demonstrates that no single algorithm can handle arbitrarily large problem instances within practical time constraints. The success of SA suggests that hybrid approaches incorporating probabilistic elements may offer the best balance between exploration and exploitation for large-scale optimization problems.

The Genetic Algorithm's consistent failure across all board sizes represents a surprising result that warrants deeper investigation. Several factors could contribute to this poor performance: inadequate population size relative to search space complexity, suboptimal genetic operators for the N-Queens domain, or premature convergence to suboptimal solutions. The population-based nature of GA introduces significant computational overhead that may not be justified for single-solution problems like N-Queens. This finding challenges the common assumption that evolutionary algorithms are universally applicable to optimization problems and highlights the importance of matching algorithm characteristics to problem structure.

A. Future Directions

Future research should explore several promising avenues to enhance N-Queens algorithm performance and applicability. Hybrid algorithms combining the exploration capabilities of genetic algorithms with the exploitation efficiency of local search could potentially overcome individual method limitations. Parallel and distributed implementations could dramatically improve scalability, particularly for population-based approaches where individuals can be evaluated independently. Advanced parameter adaptation techniques, such as self-adjusting cooling schedules for SA or dynamic population sizing for GA, could improve robustness across different problem scales. Investigation of problem-specific heuristics and constraint propagation techniques could reduce effective search space size. Machine learning approaches to predict optimal algorithm selection based on problem characteristics could automate the currently manual process of algorithm selection. Finally, extending the analysis to related constraint satisfaction problems would validate the generalizability of findings beyond the N-Queens domain.

V. Conclusion

This comprehensive study evaluated four algorithmic approaches to the N-Queens problem across board sizes ranging from 10 to 200, revealing critical insights into the practical limitations of theoretical optimization techniques. Depth First Search, while theoretically complete, proved viable only for trivial problem sizes ($N \leq 10$), with its exponential complexity causing immediate timeouts for larger instances. Greedy Hill Climbing demonstrated moderate success up to N=30 but failed consistently for larger boards due to local optima entrapment. Simulated Annealing emerged as the most practical approach, successfully solving instances up to N=100 by effectively balancing exploration and exploitation through probabilistic acceptance of suboptimal moves. The Genetic Algorithm's uniform failure across all test cases highlights the importance of careful algorithm selection and parameter tuning for specific problem domains. These findings underscore that no single algorithm dominates across all problem scales, and practical applications require careful consideration of problem size, available computational resources, and solution quality requirements. The study provides empirical evidence that theoretical algorithmic complexity translates directly to practical limitations, emphasizing the need for hybrid and adaptive approaches in real-world optimization scenarios.

References

[1] Russell, S., & Norvig, P. (2020). *Artificial Intelligence: A Modern Approach*. Pearson.

[2] Eiben, A. E., & Smith, J. E. (2015). *Introduction to Evolutionary Computing*. Springer.

[3] GeeksforGeeks. (2023). *N-Queens Problem using Simulated Annealing*. Available at: https://www.geeksforgeeks.org/n-queens-problem-using-simulated-annealing/

[4] Alizadehbirjandi, G., Ali, R. H., Koutaly, R., Khan, T. A., & Ahmad, I. (2023). Solving N-Queens Problem using Exhaustive Search and a Novel Genetic Algorithm. *Department of Business, University of Europe for Applied Sciences, Think Campus, Potsdam, Germany*.

[5] Towards Data Science. (2022). *Solving the N-Queens Problem with Genetic Algorithms in Python*. Available at: https://towardsdatascience.com/solving-the-n-queens-problem-with-genetic-algorithms-in-python-8f6a9e7f8050