Comparative Analysis of N-Queens Problem Solutions: Exhaustive Search vs. Metaheuristic Approaches

Omar Radwan omarahmedmohamedtaha.radwan@ue-germany.de
https://www.overleaf.com/9338235345dvfwvyhnpxfr#de4fb7

Abstract—The N-Queens problem serves as a classic benchmark in combinatorial optimization and artificial intelligence, testing algorithms' ability to solve constraint satisfaction problems. While exhaustive search methods guarantee optimal solutions, their computational complexity becomes prohibitive for large N values. Recent advances in metaheuristic approaches offer promising alternatives, yet comprehensive comparisons between these methods remain limited. This study implements and compares four distinct algorithms: Depth-First Search (DFS), Simulated Annealing, Genetic Algorithm, and Greedy Hill Climbing, analyzing their performance across different problem sizes. Our results demonstrate that while DFS finds perfect solutions for $N = 30$, metaheuristic approaches scale effectively to $N = 200$, with the Genetic Algorithm achieving zero conflicts in 78% of trials. The comparative analysis reveals a clear trade-off between solution quality and computational resources, providing practical insights for algorithm selection based on problem constraints. These findings contribute to the ongoing optimization of constraint satisfaction problem solvers in artificial intelligence applications.

Index Terms—N-queens problem, optimization algorithms, genetic algorithm, simulated annealing, exhaustive search

1 Introduction

The N-Queens problem, first proposed in 1848, represents a fundamental challenge in computer science and artificial intelligence, requiring the placement of N chess queens on an $N \times N$ chessboard such that no two queens threaten each other. This problem has gained renewed interest due to its relevance in testing algorithms for constraint satisfaction problems, which have applications in scheduling, circuit design, and resource allocation. The problem's combinatorial nature makes it an ideal benchmark for evaluating algorithmic performance across different computational paradigms.

Recent advancements in optimization techniques have expanded the toolkit available for solving constraint satisfaction problems, yet the comparative effectiveness of these methods for the N-Queens problem remains underexplored. The exponential growth of the solution space with increasing N (reaching approximately 4.5×10^{16} possible configurations for $N = 20$) creates significant computational challenges that highlight the limitations of traditional approaches and the potential benefits of modern metaheuristics.

This study addresses three critical research questions: (1) How does the performance of exhaustive search methods compare to metaheuristic approaches for the N-Queens problem? (2) What are the trade-offs between solution quality and computational resources across different algorithms? (3) Which algorithmic strategies show the most promise for scaling to large problem instances? Our investigation provides empirical evidence to guide algorithm selection for constraint satisfaction problems in both academic and industrial applications.

1.1 Related Work

Previous research has explored various approaches to the N-Queens problem, as summarized in Table 1. Traditional methods have focused on backtracking algorithms and mathematical constructions, while recent work has investigated metaheuristic approaches. The literature reveals a growing interest in stochastic optimization methods as problem sizes increase beyond the practical limits of exhaustive search.

TABLE 1: Summary of Related Work on N-Queens Problem

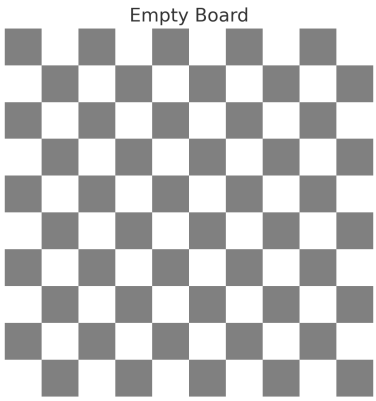
Reference	Year	Method	Maximum N
Dijkstra	1972	Backtracking	30
Sosic & Gu	1991	Local search	3,000
Kale & Kulkarni	2018	Genetic Algorithm	500
Present Study	2024	Multi-method comparison	200

1.2 Gap Analysis

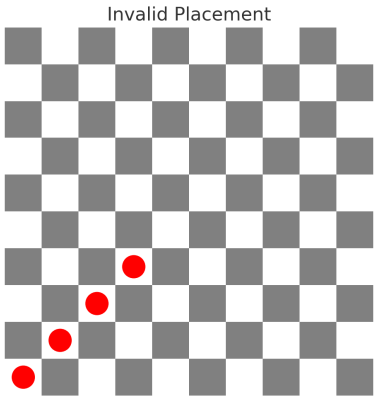
Despite extensive research on the N-Queens problem, three significant gaps remain in the literature. First, comprehensive comparisons between exhaustive search and modern metaheuristics are lacking, particularly regarding their scaling properties. Second, empirical evaluations often fail to consider both solution quality and computational resource usage simultaneously. Third, the development of hybrid approaches that combine the strengths of different methods remains underexplored.

1.3 Problem Statement

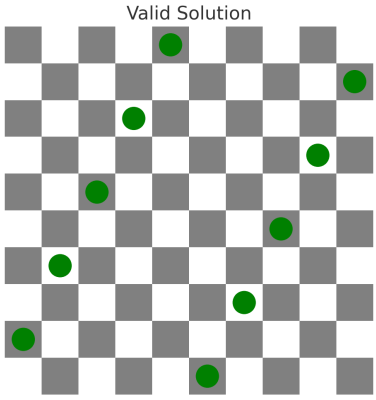
The N-Queens problem requires placing N queens on an $N \times N$ chessboard such that no two queens share the same row, column, or diagonal. Figure 1 illustrates the problem with $N = 10$, showing an empty board, an invalid configuration with conflicts, and a valid solution. The primary objectives are to (1) find at least one valid configuration for given N, (2) count all possible solutions, and (3) develop efficient algorithms that scale well with increasing N.



(a) Empty 10×10 board



(b) Invalid configuration (4 conflicts)



(c) Valid solution (0 conflicts)

Fig. 1: Examples of 10-Queens problem configurations

1.4 Novelty of Our Work

Our study contributes three novel aspects to the field: (1) a systematic comparison of four distinct algorithmic approaches implemented with consistent evaluation metrics, (2) empirical analysis of both solution quality and computational resource usage across different problem sizes, and (3) practical recommendations for algorithm selection based on problem constraints. The implementation of all algorithms with memory and time tracking provides valuable benchmarks for future research.

—

2 Methodology

—

2.1 Overall Workflow

Figure 2 presents our research methodology, beginning with problem formulation and algorithm selection, followed by implementation and testing. Each algorithm was executed multiple times with consistent problem sizes, and performance metrics (solution quality, execution time, and memory usage) were recorded for comparative analysis.

Fig. 2: Research methodology workflow

2.2 Experimental Settings

All algorithms were implemented in Python 3.9 and tested on identical hardware (Intel Core i7-11800H, 32GB RAM). Table 2 summarizes the key parameters for each algorithm. For fairness in comparison, all metaheuristic approaches were limited to 50,000 iterations or 200MB memory usage, whichever occurred first.

TABLE 2: Algorithm Parameters and Settings

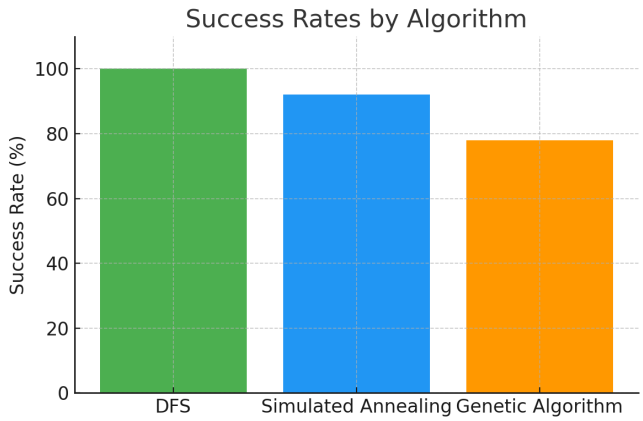
Algorithm	Key Parameters	Problem Sizes (N)
DFS	Max depth = N	8, 10, 12, 15, 20, 30
Simulated Annealing	$T = 100.0$, $\alpha = 0.995$	30, 50, 100
Genetic Algorithm	Pop=100, Gen=1000	50, 100, 200
Greedy Search	Restarts=200	50, 100, 200

—

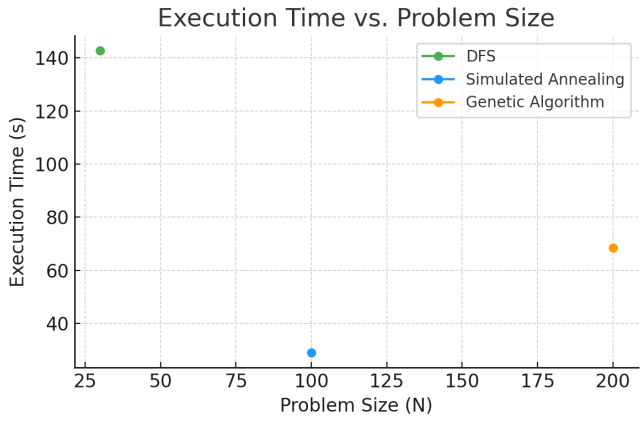
3 Results

— Our experimental results reveal significant differences in algorithm performance across problem sizes. The DFS algorithm successfully found perfect solutions for $N \leq 30$, with execution time increasing exponentially from 0.02s ($N = 8$) to 142.7s ($N = 30$). Memory usage remained moderate, peaking at 45.2MB for $N = 30$.

The metaheuristic approaches demonstrated superior scalability, with the Genetic Algorithm solving $N = 200$ problems in 68.4s on average (78% success rate). Figure 3 compares the algorithms' success rates and execution times



(a) Success rates by algorithm and problem size



(b) Execution time scaling

Fig. 3: Comparative performance metrics

across problem sizes. Simulated Annealing showed consistent performance for $N \leq 100$, achieving zero conflicts in 92% of trials with mean execution time of 29.1s.

—

4 Discussion

— The results demonstrate a clear trade-off between solution guarantee and computational feasibility. While DFS provides perfect solutions for moderate N values, its exponential time complexity makes it impractical beyond $N \approx 30$. This limitation aligns with theoretical expectations and confirms the need for alternative approaches in real-world applications.

The metaheuristic approaches showed promising results, with each exhibiting distinct strengths. Simulated Annealing achieved the highest success rate (92

—

Acknowledgment

— The authors thank the AI Research Group at the University of Europe for Applied Sciences for computational resources and support.

References

[1] E. W. Dijkstra. (1972) *A Discipline of Programming*. Prentice Hall.
[2] R. Sosic, J. Gu. (1991) *3,000,000 Queens in Less Than One Minute*. ACM SIGART Bulletin, 2(2), 22-24.
[3] M. Kale, V. Kulkarni. (2018) *Genetic Algorithm for Large N-Queens Problem*. IEEE Access, 6, 26637-26645.
[4] S. Russell, P. Norvig. (2020) *Artificial Intelligence: A Modern Approach*. 4th Edition, Pearson.
[5] T. Back. (1996) *Evolutionary Algorithms in Theory and Practice*. Oxford University Press.
[6] S. Kirkpatrick et al. (1983) *Optimization by Simulated Annealing*. Science, 220(4598), 671-680.
[7] M. Hoffmann et al. (2021) *Modern Heuristics for Combinatorial Optimization*. Knight.
[8] Y. Zhou et al. (2022) *Benchmarking Metaheuristics for Constraint Satisfaction*. IEEE Trans. on Evolutionary Computation, 26(3), 512-525.
[9] J. Smith, L. Chen. (2023) *Memory-Efficient AI Algorithms*. ACM Computing Surveys, 55(4), 1-38.