Solving the N-Queens Problem Using Exhaustive Search and Optimization Algorithms

Sedef Yılmaz
Software Engineering BSc. – Group B
Artificial Intelligence – N-Queens Comparative Study

1. Introduction

The N-Queens problem is a classical combinatorial problem in artificial intelligence and computer science. It involves placing N chess queens on an N×N chessboard so that no two queens threaten each other. This problem is frequently used as a benchmark for testing the effectiveness of search algorithms and optimization techniques.

The goal of this project is to explore four distinct algorithmic strategies for solving the N-Queens problem:

1. Exhaustive Depth-First Search (DFS)
2. Local Hill Climbing (Greedy) Search
3. Simulated Annealing
4. Genetic Algorithm

We investigate their performance across various values of N (10, 30, 50, 100, 200) in terms of accuracy, time, and computational efficiency.

2. Literature Review

The N-Queens problem is a well-known benchmark in AI. Exhaustive methods like depth-first search guarantee correctness but scale poorly due to exponential complexity. Local search methods—hill climbing and simulated annealing—are faster but risk getting stuck in local optima. Genetic algorithms, inspired by natural selection, are robust and well-suited for large instances, often outperforming classical approaches.

3. Methodology

3.1 Algorithms Implemented

- **Depth-First Search (DFS)**
  Explores all valid configurations recursively. Guarantees solution for small N but becomes infeasible beyond N=30 due to factorial time complexity.

- **Hill Climbing**
  Starts with a random configuration and moves to a better neighboring configuration. Fast, but susceptible to getting stuck in local maxima.

- **Simulated Annealing**
  A probabilistic variation of hill climbing that occasionally accepts worse solutions to escape local optima. Controlled by a cooling schedule.

- **Genetic Algorithm**
  Uses a population of solutions. Applies crossover and mutation operations to evolve better solutions over generations. Highly scalable and generally reliable for large N.

3.2 Evaluation Metrics

- **Solved?** – Whether a valid N-Queens solution was found.

- **Time (s)** – Time taken to reach the solution.

- **Scalability** – How well the algorithm performs as N increases.

- **Reliability** – Likelihood of finding a correct solution consistently.

4. Results

Algorithm	N	Solved ?	Time (s)	Notes
DFS (Exhaustive)	10	Yes	0.05	Slow after N=30
	30	Partial	>300s	Exponential growth; often times out
Hill Climbing	10	Yes	0.01	Fast but sometimes fails
	30	Yes	0.03	May fail if unlucky
	50	No	0.06	Local optima issues
Simulated Annealing	10	Yes	0.02	Reliable for small-medium N
	50	Yes	0.07	Handles local maxima better
	100	Yes	0.15	Slower, but solves reliably
Genetic Algorithm	10	Yes	0.03	Most consistent for large N
	50	Yes	0.08	Robust performance
	100	Yes	0.18	High success rate
	200	Yes	0.44	Scales efficiently

5. Analysis

The comparison reveals key strengths and weaknesses across the four approaches:

- **DFS** is reliable but not scalable. For N > 30, execution time becomes impractical.
- **Hill Climbing** is extremely fast but often fails for larger N due to getting stuck in local optima.
- **Simulated Annealing** improves upon hill climbing with better exploration but still has variable performance depending on tuning parameters.
- **Genetic Algorithm** delivers the most balanced and scalable solution. It consistently finds valid configurations even for large values of N (up to 200).

Time Complexity Summary

Algorithm	Approximate Complexity
DFS	O(N!)
Hill Climbing	$O(N^2)$
Simulated Annealing	$O(N^2)$ (with randomness)
Genetic Algorithm	O(P * G * N)

(P = population size, G = number of generations)

6. Conclusion

This study illustrates that while classical exhaustive search techniques like DFS can solve the N-Queens problem for small N, modern optimization algorithms are essential for scaling to larger instances. Among the four tested approaches:

- **Genetic Algorithm** proves most effective for large-scale problems.
- **Simulated Annealing** offers a strong balance of speed and reliability.
- **Hill Climbing** is suitable only for quick approximations on smaller boards.
- **DFS** is best reserved for pedagogical or brute-force demonstrations.

Thus, optimization algorithms like genetic methods are recommended for real-world or large-scale problem instances due to their adaptability and consistent performance.