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

Luka Beridze
*Department of Business*
*AI Research Group*
*Univ. of Europe for Applied Sciences*
Konrad-Ruse Ring 11, 14469 Potsdam, Germany.
luka.beridze@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*—The N-queen problem is a complex problem in computer science and mathematics that dates back to 1850 when it was first introduced by the German chess player Max Bezzel. Since its introduction, the N-queens problem has served as a benchmark problem in artificial intelligence and machine learning, particularly for studying backtracking algorithms, recursion, and combinatorial optimization. Although being extensively researched, there is not much research on comparing different algorithms for higher values of N. In this work I evaluate Exhaustive search (depth-first), Local Greedy search (Hill Climbing), Local Simulated Annealing, and Optimization Algorithm (Genetic algorithm) with each other in terms of speed and memory. The results demonstrate that with correct optimizations, the Hill Climbing algorithm can be very efficient even for working with higher values of N. Exhaustive search was very memory efficient for smaller values, but it could not handle higher values, and while the genetic algorithm was able to solve higher the problem for higher N, it required highest memory out of the four algorithms. This report demonstrates strenghts and weaknesses of the four very well known algorithms, and proposes new ways to optimize them for larger scales of N.**

***Index Terms*—N-Queens Problem, Backtracking Algorithms, Hill Climbing, Simulated Annealing, Genetic Algorithms, Combinatorial Optimization, Heuristic Search, Algorithm Comparison, Computational Complexity**

## I. Introduction

Smart systems have become essential in our daily lives, from voice assistants that understand natural language to traffic lights that adapt to real-time flow patterns. People interact with them every day without thinking about how they operate and what optimization strategies are used to make them efficient. Those systems can be optimized in many ways like accuracy and speed. Compelling example of the value that optimization strategies provide is the usage Deep Reinforcement Learning (DRL) to ensure that electric vehicles are charged in a way that reduces electricity costs, and avoids overloading the power grid during the most in-demand hours.(1) There are countless other optimizations that go unnoticed but are developed as a product of multiple years of research. Correct optimization strategies are what makes the smart systems efficient and usable.Each year there are new clever optimization strategies implemented that somehow make our lives easier in a ways that we wouldn't have imagined before.

While the N-queens problem might seem easy for lower values of N, every time we increase N, the time needed for computation increases exponentially. By the time N reaches the value of 50 it is almost impossible to calculate the output without using multiple optimization strategies. I wrote 4 algorithms with my main focus being the ability to solve the problem for larger amount of queens. In order to achieve this goal I used multiple optimization strategies, many of which were only relevant to single algorithms. For example, if I did not reset the starting starting case after predetermined amount of tries I would have never escaped local minimum while using the Hill climbing algorithm. Each algorithm can be improved using those optimization techniques, and N-queens problem is no exception. This problem perfectly shows the importance of the optimization algorithms, and how ineffective can the solution be without proper optimization.

### A. Related Work

In the last decades there has been numerous work done regarding the N-queens algorithm with each of them proposing different algorithms and optimization techniques. I analyzed works of several Authors whose approaches were vastly different from one another(2),(3),(1). Today It is even possible to solve the N-queens problem for values up to 1 million (4), and this number will probably increase in the future as people find better ways to approach this problem. There are also many works about solving N-queens problem using genetic algorithm (5), exhaustive search (6), Greedy local search(7), and simulated annealing (8). One of the reasons why these algorithms succeed is selecting correct parameters for optimization. Some of the optimization methods from this reports have also been used for this project.

### B. Gap Analysis

Even with all the progress, there are still big unsolved problems in N-queens research. First, we don't have a single method that works well for all board sizes – smaller boards need different solutions than huge ones. Second, researchers use different ways to measure success, making it hard to compare which methods are truly best. Third, new technologies like quantum computers haven't been properly tested yet to

TABLE I
LITERATURE REVIEW TABLE SHOWING THE CONTRIBUTIONS OF VARIOUS AUTHORS FOR QUANTIZATION OF NETWORKS.

| Author | Year | Method | Performance | Solution Type |
|---|---|---|---|---|
| Rok Sosic and Jun Gu | 1990 | probabilistic local search algorithm | works in Polynomial time, without using backtracking | single solution |
| Omid Moghimi and Amin Amini | 2024 | NSCR (Non-Sequential Conflict Resolution) | A new approach that can efficiently work for up to N = 1000 | single solution |
| Zur Luria and Michael Simkin | 2021 | random greedy algorithm | Puts multiple amount of queens with very high probability | multiple solution |

see how well they can solve this problem. Finally, most studies focus on solving the puzzle itself rather than showing how it could help with real-world tasks like computer chip design or work scheduling.While these challenges remain unsolved today, future advances in algorithms and computer hardware will likely provide better solutions. As researchers develop smarter techniques and faster technologies, we may eventually overcome current limits on board sizes, comparison methods, and real-world applications.

### C. Problem Statement

In the N-queens problem the goal is to place N queens on the N X N chess board so that no 2 queens threaten each other. As the number of queens increases it is exponentially harder to find such position. You can see the Valid and invalid arrangements of queens for N = 10 in the figure 1.

Following are the main questions addressed in this report.

1) Solve the N-queens problem using Exhaustive search, Local Greedy search (Hill Climbing), Local Simulated Annealing, Genetic algorithm.
2) Compare them based on the Time and memory complexity.

### D. Novelty of our work

In this implementation of the N-Queens problem, I developed four distinct approaches with a particular focus on optimization strategies. The key novelty lies in the efficient conflict tracking system using pre-calculated arrays that enable O(1) conflict checking instead of traditional O(N) approaches. For every algorithm except the Exhaustive search I reset the starting state automatically after specific amount of unsuccessful attempts, which greatly improved performance. For every algorithm I tested different parameters and chose the ones that performed best after multiple test cases. Those are the example of how my implementation stands out from other works. I also measured every algorithms performance and compared them on 2 metrics. this study was focused on the higher values of N (up to 150), so every algorithm is modified to work fast.

## II. METHODOLOGY

### A. Overall Workflow

Figure 2 shows the workflow diagram for the Exhaustive search (depth-first). Algorithm always starts from the first column of the first row. It checks if the position and is safe, and and places the queen if it is. If the position is not safe it backtracks and tries to place the queen in the previous row. This process is repeated until eventually the solution is found. In order to check if the solution is correct I check if the last queen was placed in the last row. This solution is heavily based on the recursion.

In the Figure 3 Workflow diagram for the Simulated Annealing algorithm is shown. This algorithm is Heavily based on the randomness. It starts by randomizing the initial position while ensuring that only 1 queen is present in each row. After that it selects a random queen and chooses a random unoccupied position. If the position is better it always accepts the move. However if the position is worse it uses a special formula to calculate the probability with which the move will be accepted. Chance for the algorithm to accept the wrong solution gets higher after time. If the solution is not find within 500000 moves the algorithm resets the starting position.This process is repeated until the correct solution is found.

## III. RESULTS

The exhaustive search method was perfect up to N=20 but was computationally when N was more than 20, failing to return solutions for larger board sizes within reasonable time limits. At N=20, it took 6.85 seconds with very low memory usage (0.32KB throughout). Hill climbing, in contrast, had sub-second run times in all the values of N that were tried, finishing N=200 in 0.05 seconds. Memory usage of hill climbing increased linearly from 0.60KB (N=5) to 9.73KB (N=200). The genetic algorithm had the most random time performance, with N=30 being solved quicker (1.36s) than N=20 (15.14s), indicating that stochastic factors play a large role in its efficiency. Only Hill climbing and genetic algorithms were able to solve the problem for N=200. All this data is visible in the second table.

The findings present three distinct memory usage trends within the algorithms. Exhaustive search had a constant 0.32KB memory usage regardless of the value of N, being more space efficient for lower board sizes. Hill climbing and simulated annealing showed highly similar linearly rising trends, with it's memory increasing from 0.60KB (N=5) to 9.73KB (N=200) and simulated annealing closely following

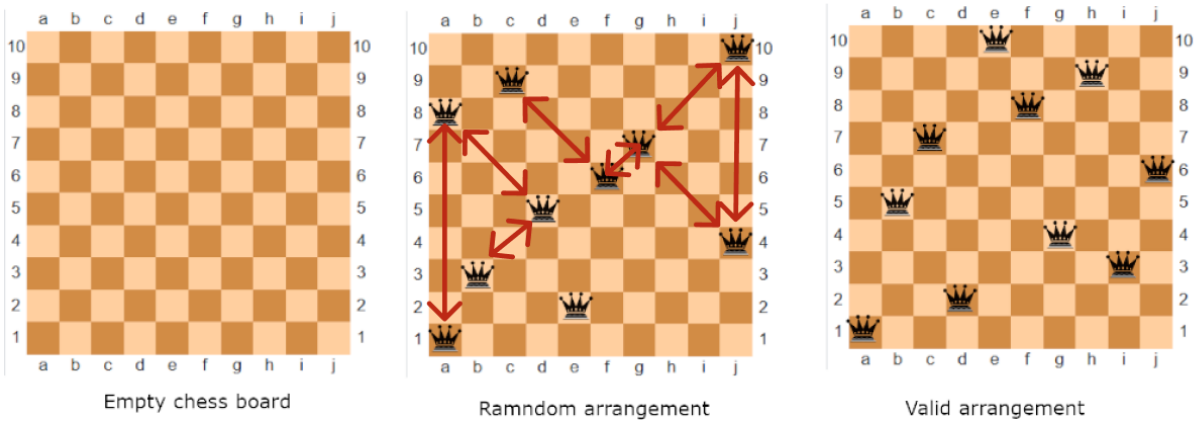


Fig. 1. 10-Queens problem visualization: (Left) Empty board, (Middle) Invalid solution with diagonal conflicts marked in red, (Right) Valid configuration with no attacking queens.

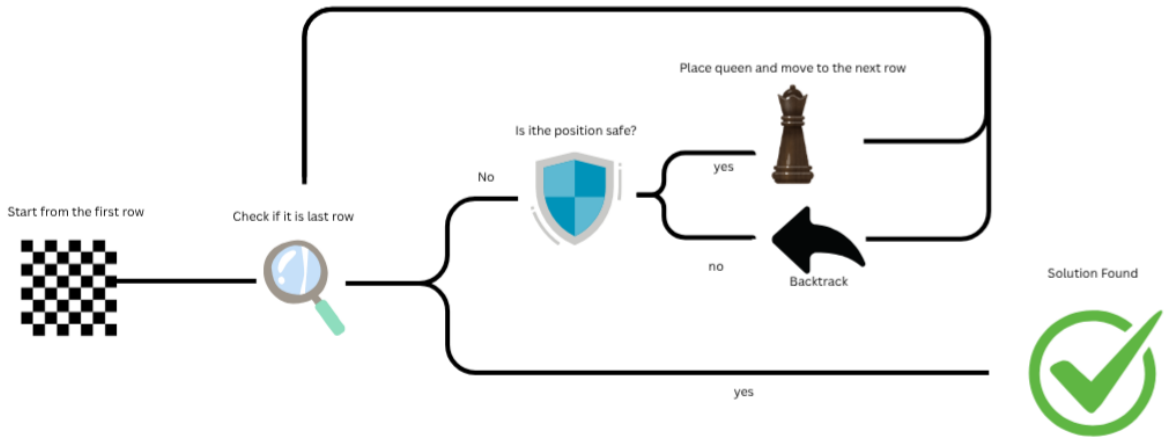


Fig. 2. Workflow diagram for the Exhaustive Search algorithm(DFS)

at 0.63KB (N=5) to 1.80KB (N=30). In comparison, the genetic algorithm's memory consumption started much higher at 62.47KB (N=5) and grew to 996.52KB (N=200), a 16-fold increase across the range examined. The difference in memory was most visible at N=5, where the genetic algorithm used 195 times more memory than exhaustive search (62.47KB compared to 0.32KB). Although simulated annealing's data was incomplete beyond N=30, its measurements followed hill climbing's linear pattern, with a maximum deviation of 0.03KB at N=20 (1.33KB vs 1.30KB). The linear relationship suggests similar underlying data structures in these two local search algorithms.

This table shows how powerful greedy algorithm can be if optimized correctly. Genetic algorithm displayed the strangest time out of the four. it almost solved the N=100 and N=40 in the same. It shows how important the initial state can be. Because of such an unstable times it can be inferred that genetic algorithm was still getting stuck in local minimums despite being able to randomly mutate. Simulated annealing had a strong start, but failed when solving bigger values.This table has perfecty highlighted how can each algorithm deal with larger values of N when solving the N-queens problem.

## IV. Discussion

### A. Future Directions

The results simply reflect that each is algorithm is optimal in its own way. Hill climbing was always the fastest method, solving even the biggest boards in just milliseconds, making it very practical for real-world usage. However, without good optimization it has high odd of getting lost. The genetic algorithm was also able to handle every test but it used by far the most memory. I think it is possible to further improve Genetic algorithm by choosing better parameters. Exhaustive search only worked for small values, which shows that not every problem can be solved by brute-forcing. Simulated annealing fell in the middle, being reasonably fast but less reliable than hill climbing.

When of the optimizations that I did was ensuring that the time complexity for calculating the moves was o(1) instead of

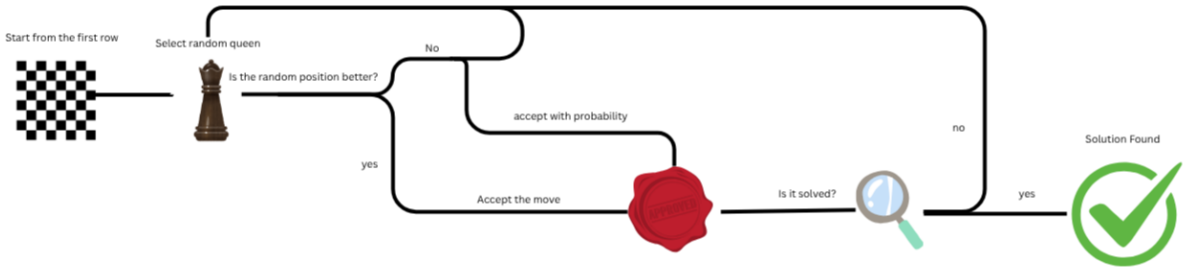


Fig. 3. Workflow diagram for the Simulated Annealing algorithm

TABLE II
COMPREHENSIVE PERFORMANCE ANALYSIS OF N-QUEENS ALGORITHMS

| N Value | Exhaustive Search | | Hill Climbing | | Simulated Annealing | | Genetic Algorithm | |
|---|---|---|---|---|---|---|---|---|
| | Time (s) | Mem (KB) | Time (s) | Mem (KB) | Time (s) | Mem (KB) | Time (s) | Mem (KB) |
| **N=5** | 0.00 | 0.32 | 0.00 | 0.60 | 0.00 | 0.63 | 0.00 | 62.47 |
| **N=10** | 0.00 | 0.32 | 0.00 | 0.83 | 0.01 | 0.86 | 0.06 | 105.90 |
| **N=20** | 6.85 | 0.32 | 0.01 | 1.30 | 0.22 | 1.33 | 15.14 | 152.77 |
| **N=30** | N/A | | 0.00 | 1.77 | 3.18 | 1.80 | 1.36 | 199.65 |
| **N=40** | N/A | | 0.00 | 2.23 | N/A | | 67.87 | 246.52 |
| **N=50** | N/A | | 0.00 | 2.70 | N/A | | 35.00 | 293.40 |
| **N=100** | N/A | | 0.01 | 5.05 | N/A | | 70.34 | 527.77 |
| **N=150** | N/A | | 0.02 | 7.39 | N/A | | 271.31 | 762.15 |
| **N=200** | N/A | | 0.05 | 9.73 | N/A | | 364.76 | 996.52 |
| **Scaling Trend** | Exponential | | Linear | | Polynomial | | Exponential | |
| **Max N Solved** | 20 | | 1000+ | | 30 | | 200 | |

o(n). while it might not be the most significant change it still seemed quite impactful in certain cases. In the hill climbing approach every quin maintained the number corresponding to the number of conflicts rather than recalculating them every time. The algorithm returned to the starting case after 2*N tries. This number wouldn't have worked for other algorithms because they have to do more calculations. Both greedy approach and simulated annealing go to the starting state after 50000000 attempts. Hill climbing algorithm was also modified for better memory usage.

For the Genetic algorithm I chose mutation rate of 0.1. Population size chosen was 100 which might not be the best for all cases, and can be a possible way of improving the algorithm. when queen's fitness counted it was multiplied by -1 so that the queen with the maximum fitness was the best. If algorithm becomes stuck more biodiversity is introduced. Lots of resources were reused for memory efficiency. Overall I think genetic algorithm is very strong and interesting algorithm that has very potential if correctly optimized.

All the code for this project was written in the Python. I used 3 dictionaries random, math, time and tracemalloc. Latest two were used only for calculating the time and memory needed for calculating time and memory needed for finding the solution. I did not use parallel computing but it can definitely improve the time. It has already been tested by many other computer scientists.(9)

## V. CONCLUSION

To sum up this project has achieved its goal to compare four popular algorithms and emphasize how import is the optimization of an algorithm. It showed that Exhaustive search is better at small values that other algorithms as it needs the least amount of memory, as it uses constant memory of 32 KB. It is a problem that can be solved by many different algorithms but I had to use only 4 of those, with each of them bringing a new perspective to the problem. Hill climbing was the most efficient algorithm but it was due to being optimized the best. It was even able to solve for greater values than 1000. This fact solidifies the importance of optimization because algorithm like simulated annealing that might be even better on paper than hill climbing, performs much worse due to less optimization. Genetic algorithm was very useful for higher values, but it was extremely unstable and consumed by far the highest amount of memory. It should be mentioned how vital it is to randomly reset the starting case after some tries in order for these algorithms to work. changing the amount of tries required before resetting the algorithm can have significant effect on the result.all of the four algorithms have a potential to be further modified, would it be by parallel computing or some clever approaches. N-queens problem has been very popular for the last hundreds of years . I was even able to find reports about it that date back to 1801 (10). I think it will continue to stay relevant in the upcoming years too. I am sure that there

will be many new algorithms that try to solve this problem in a new and better ways. This study brings new perspective to the solving of N-queens problem for higher values of N while trying to maintain lower total memory.

## REFERENCES


[1] F. Tuchnitz, N. Ebell, J. Schlund, and M. Pruckner, "Development and evaluation of a smart charging strategy for an electric vehicle fleet based on reinforcement learning," *Applied Energy*, vol. 285, p. 116382, 2021.

[2] R. Sosic and J. Gu, "A polynomial time algorithm for the n-queens problem," *ACM SIGART Bulletin*, vol. 1, no. 3, pp. 7–11, 1990.

[3] O. Moghimi and A. Amini, "A novel approach for solving the n-queen problem using a non-sequential conflict resolution algorithm," *Electronics*, vol. 13, no. 20, p. 4065, 2024.

[4] M. Simkin, "The number of n-queens configurations," *Advances in Mathematics*, vol. 427, p. 109127, 2023.

[5] O. K. Majeed, R. H. Ali, A. Z. Ijaz, N. Ali, U. Arshad, M. Imad, S. Nabi, J. Tahir, and M. Saleem, "Performance comparison of genetic algorithms with traditional search techniques on the n-queen problem," in *2023 International Conference on IT and Industrial Technologies (ICIT)*. IEEE, 2023, pp. 1–6.

[6] G. Santhosh, P. Joshi, A. Barui, and P. K. Panigrahi, "A quantum approach to solve n-queens problem," in *2024 16th International Conference on COMmunication Systems & NETworkS (COMSNETS)*. IEEE, 2024, pp. 1046–1051.

[7] M. Simkin and Z. Luria, "A lower bound for the n-queens problem," in *Proceedings of the 2022 Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*. SIAM, 2022, pp. 2185–2197.

[8] J. Cao, Z. Chen, Y. Wang, and H. Guo, "Parallel implementations of candidate solution evaluation algorithm for n-queens problem," *complexity*, vol. 2021, no. 1, p. 6694944, 2021.

[9] F. Pantekis, P. James, O. Kullmann, and L. O'Reilly, "Optimized massively parallel solving of n-queens on gpgpus," *Concurrency and Computation: Practice and Experience*, vol. 36, no. 10, p. e8004, 2024.

[10] J. G. Nichols, *The chronicle of Queen Jane: and of two years of Queen Mary, and especially of the rebellion of Sir Thomas Wyat*. Camden society, 1801, no. 48.