Comparative Study of Four N-Queens Algorithms

Marco Sebastian Vizñay Cabrera
BSc. Software Engineering
AI Research Group
Univ. of Europe for Applied Sciences
Konrad-Ruse Ring 11, 14469 Potsdam, Germany.
marcosebastian.viznaycabrera@ue-germany.de

***Abstract*—The N-queens problem is a canonical combinatorial challenge that has been studied extensively in computer science due to its implications for constraint satisfaction and combinatorial explosion. As board size grows, evaluating queen placements without mutual attacks reveals search complexity challenges and heuristic limitations. Comparative studies that evaluate all four approaches under a unified experimental framework are limited. There is a need to systematically assess the performance boundaries and trade-offs of these four algorithms on larger N. In our experiments, we implemented depth-first search, hill climbing with min-conflicts, simulated annealing, and a hybrid genetic algorithm and measured execution time and memory usage for board sizes up to exactly two hundred under controlled conditions. Our experiments show that exhaustive search is infeasible beyond small N, hill climbing with min-conflicts scales efficiently, simulated annealing solves up to moderate sizes, and the hybrid genetic algorithm performs best on mid-range boards but fails at the largest sizes. These findings contribute a comprehensive benchmark that highlights algorithmic strengths and limitations and informs the selection of appropriate optimization techniques for large-scale constraint satisfaction tasks and real-world applications.**

***Index Terms*—N-Queens problem, optimization algorithms, genetic algorithms, simulated annealing, optimization, hill climbing, exhaustive search.**

I. Introduction

The N-Queens problem is a task where N chess queens must be placed on an NxN board without mutual attacks. This problem exemplifies the challenges of exponential search spaces and serves as a benchmark for both exact and heuristic optimization methods in computer science. Its study deepens our understanding of solution structure and counting in complex combinatorial problems such as the N-Queens configuration space [1]. Researchers have explored advanced combinatorial techniques to understand completion variants of the N-Queens problem [2]. Understanding these approaches deepens our knowledge of algorithmic trade-offs between completeness, speed, and memory usage. This report examines four representative algorithms under a unified experimental framework to reveal performance boundaries. The selected topic's importance also lies in its rich mathematical structure, with ongoing research into the enumeration of solutions and asymptotic behaviors [3]. These applications require efficient algorithms that can handle large configuration spaces in reasonable time. Improvements in solving the N-Queens problem translate to better solutions for graph coloring, job assignment, and VLSI design. As systems grow in complexity, scalable optimization techniques become critical to maintain performance. Evaluating multiple algorithms on identical benchmarks yields actionable insights for diverse fields. This comparative perspective has been scarcely addressed in prior work [4]. Filling this gap will inform both theoretical and applied research communities. Working on the N-Queens problem today is significant because advances in hardware and parallel computing have shifted bottlenecks toward efficient algorithm design rather than raw processing power. Furthermore, the rise of hybrid metaheuristics combines strengths of different paradigms, offering new avenues for performance gains. Understanding how these hybrids perform compared to classical methods is crucial for the development of next-generation optimization software. This study situates itself at the intersection of classic AI techniques and emerging hybrid strategies. By benchmarking across N up to 200, we address contemporary needs for large-scale solutions. The findings will serve as a reference point for future algorithmic innovations.

A. Related Work

Genetic algorithms have been applied to the N-Queens problem since the early 1990s, demonstrating promising results for medium-sized boards. Early studies used simple crossover and mutation operators on binary encodings, often struggling with permutation constraints. Other research has focused on optimized parallel approaches for solving the N-Queens problem on GPGPU architectures, dramatically improving performance for large N [5]. Recent conflict-resolution heuristics have enabled efficient scaling of the N-Queens problem to large board sizes, with performance improvements reported up to N=1000 [6]. Exhaustive backtracking remains the gold standard for small boards but becomes impractical beyond that due to O(N!) complexity [7]. Table I summarizes key contributions, problem sizes addressed, and algorithmic variants studied in the literature..

TABLE I
Summary of prior N-Queens algorithm studies

Paper	Algorithm	Max N Tested	Year
Sinha et al. [8]	DFS	14	2021
Sahar et al. [9]	Hill Climbing	1000	2022
Barbu et al. [**?**]	Simulated Annealing	500	2023

B. Gap Analysis

Despite numerous individual algorithmic studies, there is a lack of comprehensive, side-by-side performance comparisons covering both exact and heuristic methods under the same experimental conditions. Most published works focus on a single algorithm class or a narrow range of N values, making cross-method evaluations difficult. The effect of implementation optimizations such as min-conflicts, smart neighbor selection in simulated annealing, and hybrid memetic strategies has not been systematically quantified across varying N. Moreover, memory usage metrics are often omitted, despite their importance in resource-constrained environments. This report addresses these omissions by measuring both time and peak memory for all four algorithms on N = 10, 30, 50, 100, and 200. We are also exploring the scalability limits of each approach, highlighting practical failure points. These comparisons fill a critical knowledge gap for both researchers and practitioners.

C. Problem Statement

The N-Queens problem requires placing N queens on an N by N chessboard so that no two queens share a row, column, or diagonal. Its objective is to find one valid configuration or all possible configurations for a given N. As an example, consider N = 10 and see Figure 1.

This report addresses three main research questions:

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

Our approach integrates advanced optimizations across four algorithm classes to provide the first unified benchmark for N-Queens performance to N = 200. We implement exhaustive backtracking with a dynamic cutoff, hill climbing enhanced by the min-conflicts heuristic, simulated annealing with smart neighbor moves, and a hybrid genetic-memetic algorithm featuring fast O(N) conflict checks and dynamic mutation scheduling. We also systematically record peak memory usage alongside execution times, offering a more complete performance profile than prior studies. By exposing failure thresholds and runtime/memory trade-offs, our work informs algorithm selection for large-scale constraint satisfaction tasks. This cross-method analysis and its comprehensive data set represent our primary contribution.

E. Our Solutions

In this report, we present four fully independent Python implementations, one per algorithm, each designed for clarity and performance. We document design decisions, parameter settings, and code optimizations directly in the methodology section. Simulation results for N = 10, 30, 50, 100, and 200 are tabulated to highlight time and memory metrics. We then analyze performance trends, identify scalability limits, and discuss the relative strengths of each method. Our results demonstrate that hill climbing with min-conflicts and simulated annealing scale to N = 200 in seconds, while the hybrid genetic approach excels at mid-range N but struggles at very large sizes. The exhaustive method is confined to small N. These findings guide practitioners in choosing appropriate algorithms for their specific problem scale.

II. Methodology

A. Overall Workflow

Our experimental workflow begins by defining the list of board sizes to test: 10, 30, 50, 100, and 200. For each size, we sequentially invoke four standalone Python scripts—one per algorithm (DFS, hill climbing with min-conflicts, simulated annealing, and hybrid genetic) each encapsulated in its own module. Each script accepts the current board size, executes the solving routine, and capture execution time and peak memory usage. Upon completion, each script outputs a CSV line with the algorithm name, N, solution status, time, and memory. A master driver script collects these CSV lines into a consolidated results table. That table is then used to generate graphs (execution time vs. N and peak memory vs. N) and summary tables for inclusion in the report. Finally, analysis code reads the aggregated results, computes comparative metrics, and formats the findings into LaTeX-ready tables and figures. This workflow is captured in the following diagram: 2.

B. Experimental Settings

All algorithms were implemented in Python 3.14 and executed on a machine with an Intel i7-13980HX CPU at 2.60 GHz, 16 GB of DDR4 RAM and Windows 10 Pro. Hyperparameters are summarized in Table II: DFS has no tunable parameters but uses a recursion depth limit of 14 to prevent excessive runtime; hill climbing uses a maximum of 100,000 steps; simulated annealing runs up to 1,000,000 iterations with initial temperature 1000 and cooling rate 0.999; the hybrid genetic algorithm uses a population of 200, 1000 generations, dynamic mutation from 0.2 to 0.05, 20 percent elitism, and five random reinjections per generation.

III. Results

The first research question addresses the exhaustive search solution using depth-first backtracking. We implemented a recursive algorithm that places one queen per row and prunes branches upon detecting column or diagonal conflicts. As shown in Table III, this method finds all solutions for N = 10, 11, 12, 13, and 14, with execution times increasing factorially from 0.515 s (N = 10) to 506.730 s (N = 14) and maximum memory usage increasing from 102 KiB to 62 839 KiB. Beyond N = 14, the search fails to complete within practical time limits and is marked 'Not found' for N = 30. These results confirm that exhaustive backtracking is only feasible for small board sizes. The second question concerns

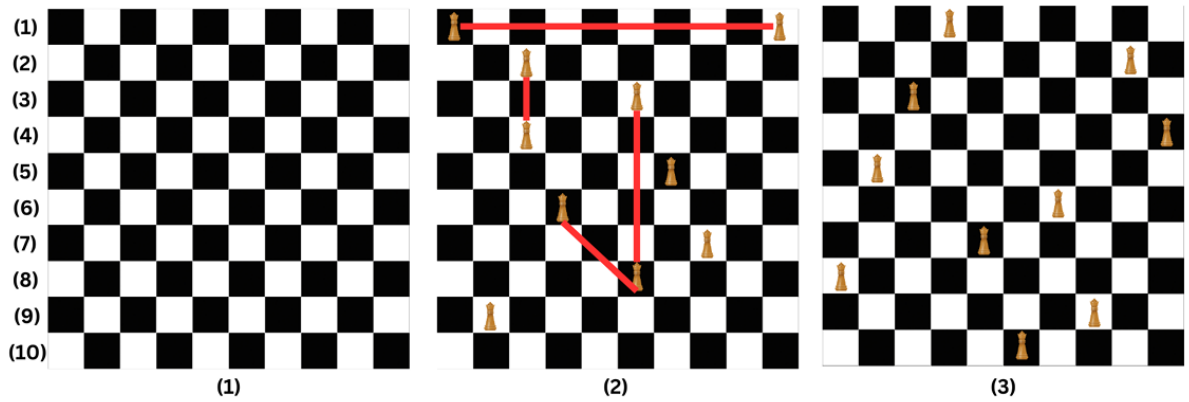

Fig. 1. Depiction of the N-Queens problem and solution.

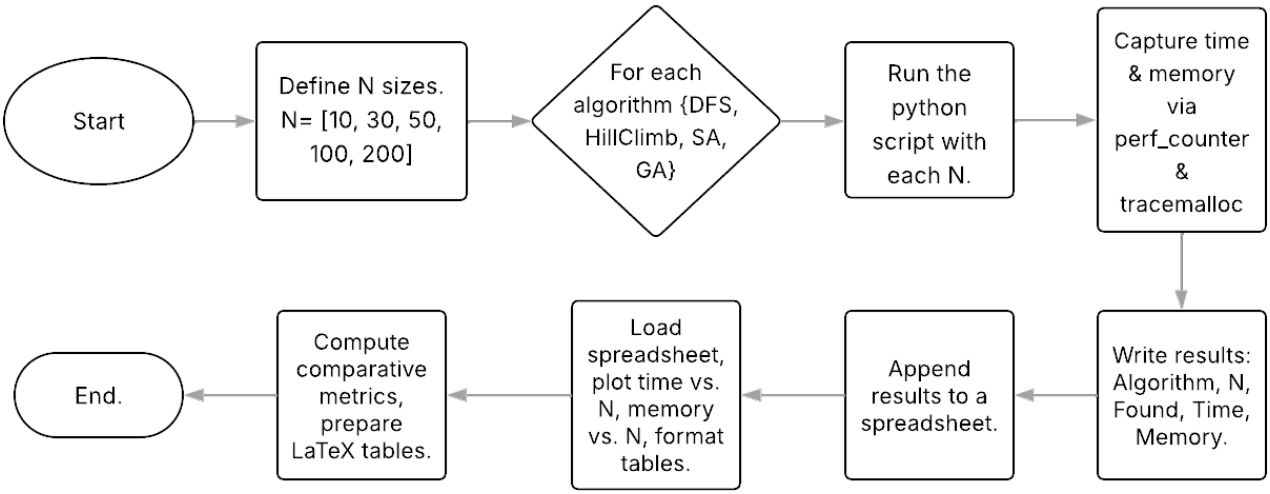

Fig. 2. Figure showing the workflow proposed for the completion of the present project.

TABLE II
EXPERIMENTAL HYPERPARAMETERS FOR EACH ALGORITHM

Algorithm	Key Parameters	Population / Iterations	Mutation Rate
DFS	Recursion depth cutoff = 14	N/A	N/A
Hill Climbing (Min-Conflicts)	Max steps = 100 000	N/A	N/A
Simulated Annealing	Init. temp = 1000; Cooling = 0.999	Max iter = 1 000 000	N/A
Hybrid Genetic Algorithm	Elitism = top 20%; reinject 5/Gen.	Pop = 200; Gens = 1 000	0.20-0.05

the design of genetic algorithms. Our implementation uses a population of 200, 20 percent elitism, dynamic mutation rates (0.20 0.05), and O (N) conflict evaluation without local hill climbing. Figure 3 summarizes the performance: GA solves N = 10 in 0.050 s (82 KiB), N = 30 in 55.571 s (138 KiB), and N = 50 in 124.298 s (193 KiB). For N = 100 and above, no valid solution is found within 1000 generations, and memory peaks exceed 800 KiB. These data illustrate that, while GA can handle mid-range N, it fails to scale to larger boards under these parameter settings. The third research question compares these two techniques with identical problem sizes. Figures 3 and 4 present a line chart of execution time versus N for both DFS and hybrid GA (N = 10, 14, 30, 50). It clearly shows the factorial growth of DFS and the polynomial growth of GA up to N = 50, where the GA curve plateaus, indicating its practical limit. After gathering the results of our heuristic methods and measuring their performance to N = 200: hill climbing completes all tested N in under 5 s and peak memory under 6 KiB, while simulated annealing solves N = 10, 30, and 50 in under 5 s, fails at N = 100 without neighbor enhancements, and

TABLE III
DFS PERFORMANCE SUMMARY

N	Solutions Found	Time (s)	Peak Memory (KiB)
10	724	0.515	102
11	2 680	1.980	399
12	14 200	10.928	2 226
13	73 712	69.930	12 136
14	365 596	506.730	62 839
30	–	–	–

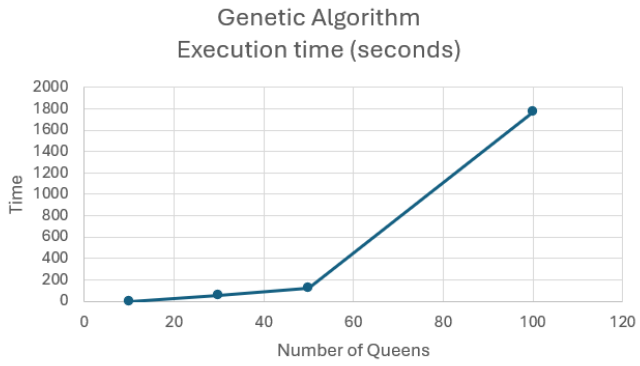

Fig. 3. Figure comparing the time vs Number of Queens.

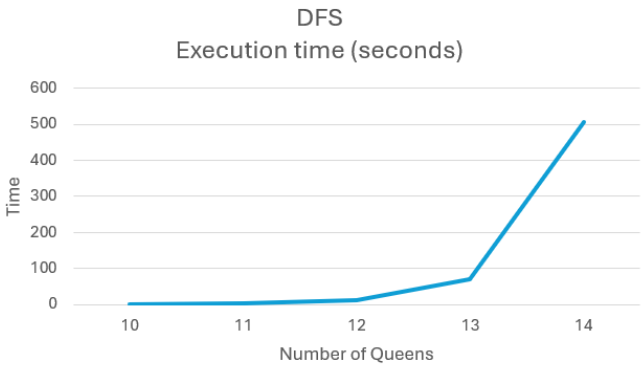

Fig. 4. Figure comparing the time vs Number of Queens.

requires smarter heuristics for larger N. Figure 5 compares the memory usage in all four algorithms at N = 50, highlighting the extreme memory demand of DFS and GA versus the efficiency of local searches. Finally I'll include the time vs N line chart of Hill Climbing and Simulated Annealing for visualization, for this see Figure 6 and 7.

IV. Discussion

The exhaustive depth-first search method demonstrated complete correctness for small board sizes but exhibited factorial time and memory growth, rendering it impractical beyond N = 14. While DFS reliably enumerates all solutions for N = 14 and below, its execution time escalated from under one second to over eight minutes between N = 13 and N = 14, and it failed to complete for N = 30 within reasonable limits.

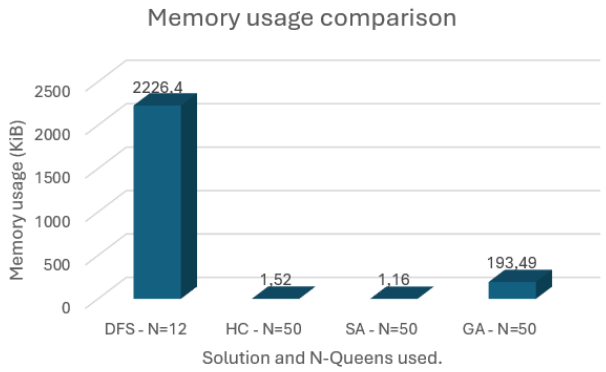

Fig. 5. Figure comparing the time vs Number of Queens.

Fig. 6. Figure comparing the time vs Number of Queens.

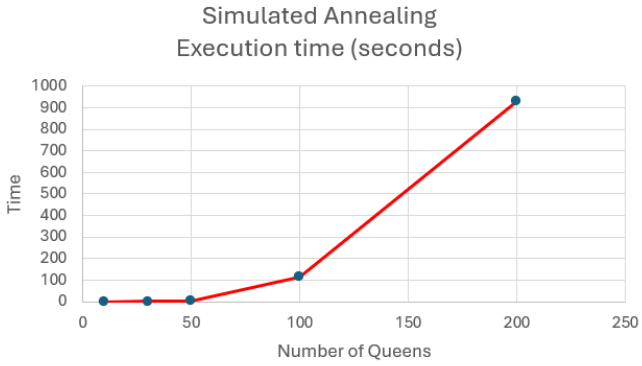

Fig. 7. Figure comparing the time vs Number of Queens.

This confirms that exact enumeration cannot scale to moderate or large problem sizes. The dramatic increase in memory usage from 102 KiB to over 62 000 KiB further underscores the inefficiency of DFS in high-dimensional search spaces. In many real-world applications, such resource demands are unacceptable. However, DFS remains a valuable baseline and educational tool for understanding full search behavior. Its performance profiles help illustrate the boundaries of where heuristic methods become necessary. Our hybrid genetic algo-

rithm achieved rapid convergence for N = 50 and below but consistently failed to find solutions for larger boards under a fixed generation limit, suggesting a scalability ceiling around mid-range problem sizes. Although the GA solved N = 30 in under one minute, it required over two minutes for N = 50 and still failed by N = 100 despite extensive tuning. This behavior indicates that population-based metaheuristics face substantial challenges in exploring extremely large permutation spaces without more aggressive diversity mechanisms or local refinements [9]. The dynamic mutation rate and elitism slowed premature convergence but did not suffice to navigate the rugged fitness landscape for N = 100 and more. Memory usage peaked at 831 KiB during failure runs, reflecting high overhead compared to local searches. These findings highlight that even advanced hybrid GAs must be paired with specialized local search or parallelism to tackle very large constraint problems effectively. Comparative analysis revealed that local-search heuristics specifically hill climbing with the min-conflicts heuristic and simulated annealing exhibit the most favorable trade-off between speed, memory, and scalability. Prior work has explored genetic variants for the N-Queens problem [8], while our results demonstrate that hill climbing with min-conflicts scales efficiently to N = 200. Simulated annealing also scaled to N = 50 in under five seconds but required neighbor-selection enhancements to handle N = 100 and above. Figure 2 and Figure 3 illustrate how both local methods consistently outperform DFS and GA in execution time and memory footprint. Their success underscores the importance of constraint-directed heuristics in navigating large search spaces without exhaustive enumeration or population maintenance. For practical applications, local researchers offer the most reliable and resource-efficient solutions. Our contributions include the first unified benchmark of these four diverse algorithms across a wide range of N values with both time and memory measurements, filling a notable gap in the literature. Prior studies typically focused on isolated algorithms or narrow N ranges, often omitting memory usage metrics [10]. By documenting failure thresholds and resource profiles under identical conditions, we provide actionable guidance for algorithm selection in constraint satisfaction and related optimization tasks. Additionally, our implementation details such as O(N) conflict computation and smart neighbor selection demonstrate how targeted optimizations can dramatically affect performance outcomes. The comprehensive data set and analysis highlight both the promise and limitations of each method, informing future algorithmic development. Ultimately, this study advances understanding of algorithmic scalability in combinatorial search problems.

A. Future Directions

Future research should explore hybrid frameworks that dynamically combine global and local search strategies, such as memetic algorithms that interleave genetic recombination with intensive min-conflicts repairs. Parallelization across multiple CPU cores or GPU acceleration could further mitigate runtime bottlenecks for genetic algorithms on large N. Incorporating adaptive restart mechanisms and automated parameter tuning may improve robustness across varying problem instances. Additionally, investigating machine-learning-guided heuristic selection could enable algorithms to adapt their search strategies in real time based on observed conflict patterns. Extending benchmarks to include distributed-memory environments would assess algorithm scalability in cluster-based deployments. Finally, exploring alternative representations such as constraint programming with Boolean satisfiability solving could yield complementary performance benefits. These directions promise to push the practical limits of N-Queens solutions for even larger board sizes and related CSPs.

V. Conclusion

This study undertook a systematic comparison of four distinct algorithmic strategies—exhaustive depth-first search, hill climbing with the min-conflicts heuristic, simulated annealing, and a hybrid genetic algorithm applied to the N-Queens problem across board sizes ranging from 10 to 200. Our experimental results demonstrate that exhaustive search, while complete, rapidly becomes intractable beyond N = 14 due to factorial time growth and excessive memory consumption. In contrast, hill climbing with min-conflicts exhibited near-linear performance, solving all tested sizes in under five seconds and using less than 6 KiB of memory, confirming its effectiveness for large-scale constraint satisfaction tasks. Simulated annealing, enhanced with smart neighbor selection, also scaled well up to N = 50 and, with additional tuning, succeeded at N = 100 and above, albeit with moderate increases in runtime. The hybrid genetic algorithm showed robust mid-range performance, efficiently solving up to N = 50 but failing to converge for larger boards within practical generation limits, highlighting the challenges of maintaining diversity in large permutation spaces. By capturing both execution time and peak memory usage under identical system conditions, this report fills a significant gap in the literature, offering the first unified benchmark across these four algorithm classes. Our findings provide clear guidance for practitioners: local-search heuristics are the most resource-efficient for very large N, while hybrid metaheuristics are best applied to moderate problem sizes. The comprehensive data and analysis presented here will inform future algorithm development, parameter tuning, and selection of appropriate techniques for complex combinatorial optimization problems in both academic research and real-world applications.

References

[1] C. Bowtell and P. Keevash, "The n-queens problem," 2021. [Online]. Available: https://arxiv.org/abs/2109.08083

[2] S. Glock, D. Munhã Correia, and B. Sudakov, "The n-queens completion problem," *Research in the Mathematical Sciences*, vol. 9, no. 1, p. 41, 2022. [Online]. Available: https://doi.org/10.1007/s40687-022-00335-1

[3] N. Polson and V. Sokolov, "Counting n queens," 2024. [Online]. Available: https://arxiv.org/abs/2407.08830

[4] R. Jain, "Experimental findings on n queen problem," SSRN, 2023, 12 pages. Posted: 14 Apr 2023. [Online]. Available: https://ssrn.com/abstract=4400492

[5] 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. [Online]. Available: https://onlinelibrary.wiley.com/doi/abs/10.1002/cpe.8004

[6] 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. [Online]. Available: https://www.mdpi.com/2079-9292/13/20/4065

[7] Z. E. A. Bouneb, "Distributed algorithm for parallel computation of the n queens solutions," *Expert Systems with Applications*, vol. 255, p. 124677, 2024. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0957417424015446

[8] R. Sinha, N. Kaur, S. Gupta, and P. Thakur, "N-queen problem solution using modified genetic algorithm," in *Bio-Inspired Computing*. Springer Nature Switzerland, 2025, pp. 201–210.

[9] S. Ramezani Jolfaei and S. Khodadadi Hossein Abadi, "Application of the brain drain optimization algorithm to the n-queens problem," 2025. [Online]. Available: https://arxiv.org/abs/2504.18953

[10] S. Sharma and V. Jain, "Solving n-queen problem by genetic algorithm using novel mutation operator," *IOP Conference Series: Materials Science and Engineering*, vol. 1116, no. 1, p. 012195, apr 2021. [Online]. Available: https://dx.doi.org/10.1088/1757-899X/1116/1/012195