24CSCI05I
Logic and Artificial Intelligence

N-Queens Problem Solver

Group Number: 16

ID	Name	Countribution
236664	Ahmed Khaled	Report
229591	Hassan Ahmed	Backtracking Algorithm, Genetic Algorithm, Report, GUI
232504	Hazem Hassan	Backtracking Algorithm, Report
236322	Ahmed Sameh	Hill-Climbing Algorithm, Report, GUI
237046	Ahmed Momen	Best-First Algorithm, Report

Faculty of Informatics & Computer Science, The British University In Egypt
Cairo, Egypt

Submitted to Dr Amr S. Ghoneim

I. PROJECT OVERVIEW

I.I Summary

The N-Queens Problem is a classic and traditional combinatorial puzzle in artificial intelligence and computer science. With the challenge being finding a way “solution” to place N queens on an N * N chessboard in a way where no two queens are facing each other. In chess, the queen is capable of attacking any pieces that are on the same column, row or diagonal. A valid solution is one where no two queens occupy the same column, row or diagonal. The N-Queens Problem is NP-hard, meaning that with the increase of the board size, leads to an exponential growth in the number of possible configurations, making techniques like brute force impractical for large N. For example, for N = 8, the number of possible arrangements is 4.4 Billion, with only 92 valid solutions. The problem acts as an excellent benchmark for evaluating different search and optimization algorithms in AI [2], [4], [3], [7].

I.II Goal

We used various algorithms like Backtracking, Best-First Search, Hill-Climbing Search, and Genetic Algorithm to solve the N-Queens problem in this project. We compared the scalability and efficiency of each algorithm by testing them on different board sizes and observing their performance. Besides, we developed an interactive system that allows users to input a board size (N) and see how each algorithm approaches and solves the problem in real time. Through systematic comparisons and evaluations, we identified the most effective methods for different problem scales, providing insights into the strengths and limitations of each algorithmic approach. The goal of this project is to create an educational tool that visualizes how different AI algorithms solve complex problems and also highlights the trade-offs between speed, accuracy, and scalability when applied to real-world optimization challenges [2], [8].

I.III Importance

The N-Queens Problem is not only a theoretical puzzle, but also a puzzle with practical implications in: many real-world scheduling and optimization problems that follow similar constraints, algorithm design (helps in evaluating and understanding what balance is best between always guaranteeing a solution, and efficiency), it also has practical implications in AI and Machine learning. Techniques like genetic algorithms, heuristic searches are commonly used in optimization tasks. By comparing different methods and approaches, the project provides insights into how AI algorithms handle and tackle a combinatorial challenge like the N-Queens Problem, by comparing different approaches, in order to find the most appropriate method [7], [8].

II. MAIN FUNCTIONALITIES

Board Size Input (N): The user can input the desired board size.

Algorithm Choice: The user can select either of four different algorithms to obtain the solution, which are Backtracking Search, Best-First Search, Hill-Climbing Search, and Genetic Algorithm.

Solution Visualization: With a click of the "Solve" button, the algorithm completes the solution of the problem, and the users can have a nice-looking UI presenting the final solution board. All the algorithms offer the complete placement of queens so users can view and compare the different solving results.

Solution Navigation: with the aid of the "Previous" and "Next" buttons, the user is able to view all the correct solutions found by the backtracking algorithm itself.

Metrics Displayed: The system displays significant performance metrics such as total time and number of solutions. Users can compare objectively the efficiency of various algorithms based on such information.

System Feedback: The software provides real-time feedback in the form of pop-ups to enhance user interaction. In case of an invalid solution with the algorithm selected, the user is instantly notified. Besides, if the solving process exceeds five seconds, the system will inform the user that the calculation takes time and they will be aware that it is in progress.

III. METHODOLOGY

In this project, we implemented four different Artificial Intelligence algorithms to solve the N-Queens problem: Backtracking Search, Best-First Search, Hill Climbing, and Genetic Algorithm. Each method uses a unique strategy to explore the solution space and optimize queen placements. Below, we provide a detailed explanation of each algorithm, including theoretical background, diagrams, pseudocode, and justifications for their selection.

1. Backtracking Search Algorithm

Theoretical Background:

Backtracking is a depth-first search algorithm that builds the solution incrementally by placing queens one row at a time until it finds a solution, else it backtracks when an invalid position is encountered. It systematically explores all the possible placements of queens and backtracks whenever a conflict occurs [3], [8].

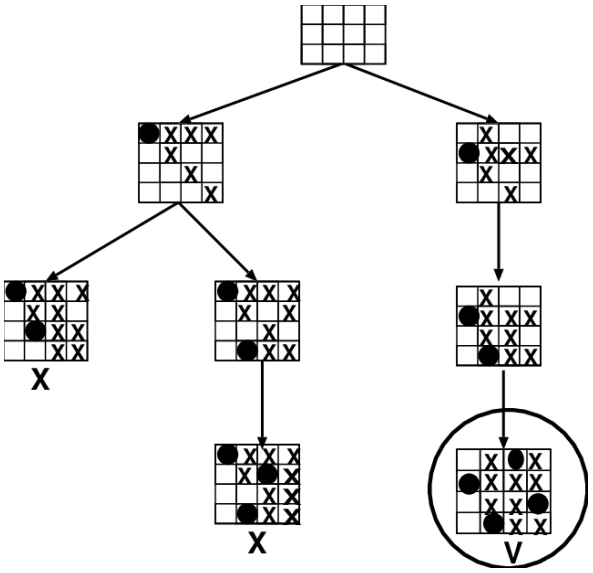

Figure 1. Backtracking Search Algorithm diagram

Pseudocode:

```
function solve(board, row):
    if row == N:
        add board to solutions
    for col from 0 to N-1:
        if is_safe(board, row, col):
            place queen at (row, col)
            solve(board, row + 1)
            remove queen at (row, col)
```

Justification for Use:

Backtracking guarantees to find all solutions if they exist, making it highly reliable for small and medium board sizes.

2. Best-First Search Algorithm

Theoretical Background:

Best-First Search is a heuristic-driven search that selects the most promising node (board configuration) based on a heuristic evaluation function. It aims to minimize the number of conflicts at each step by expanding the most promising boards first. It prioritizes states that are closer to the solution [7], [8].

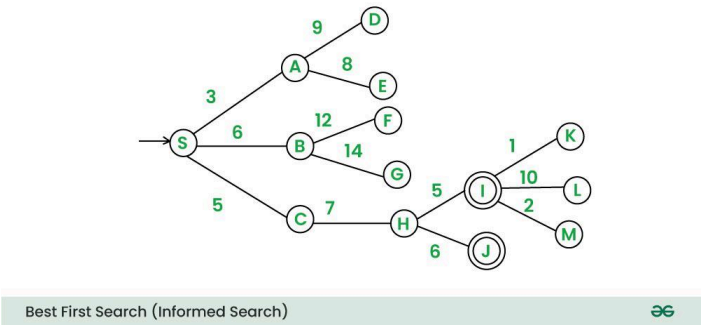

Figure 2. Best-First Search Algorithm diagram

Pseudocode:

```
initialize empty board
enqueue board with priority = 0
while queue not empty:
    dequeue board with fewest conflicts
    if board complete:
        return board
    for each column in current row:
        generate new board
         enqueue new board with heuristic
score
```

Justification for Use:

Best-First Search accelerates the search by using intelligent heuristics which reduces the unnecessary expansions compared to blind searches, especially helpful for moderately large N.

3. Hill-Climbing Search Algorithm

Theoretical Background:

Hill Climbing is a local search algorithm that iteratively moves towards the neighboring configuration with the lowest heuristic cost (fewer queen conflicts). It is simple and efficient but can possibly get stuck at local optima [4], [6].

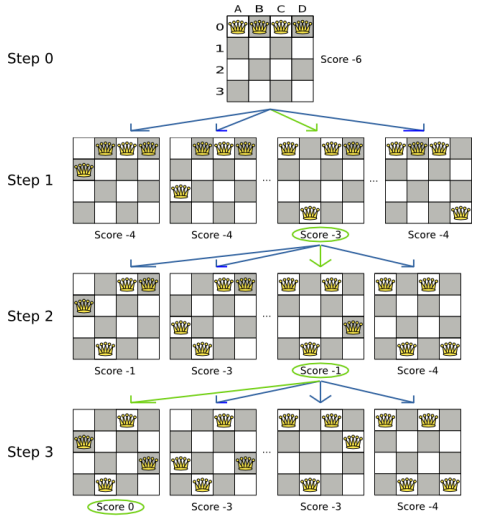

Figure 3. Hill-Climbing Search Algorithm diagram

Pseudocode:

```
initialize random board
while better move exists:
       select move that reduces conflicts
most
    make the move
if conflict count == 0:
    return solved board
else:
    return failure
```

Justification for Use:

Hill Climbing is very fast and memory-efficient, making it a good choice for small to moderate N values. It also demonstrates the concept of local search challenges such as getting stuck in local minima.

4. Genetic Algorithm

Theoretical Background:

A Genetic Algorithm (GA) is a bio-inspired optimization technique that evolves a population of candidate solutions over successive generations using selection, crossover, and mutation operations [3], [8].

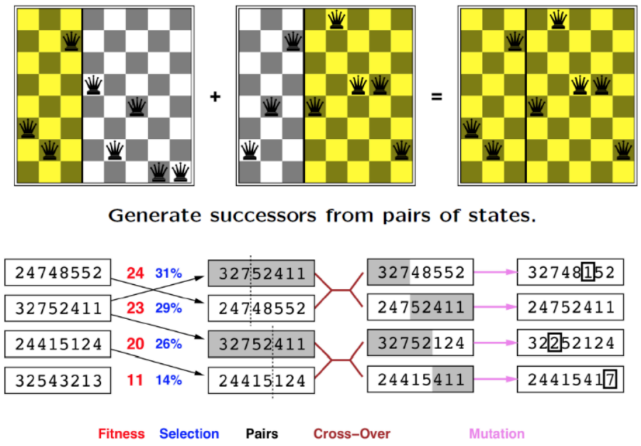

Figure 4. Backtracking Search Algorithm diagram

Pseudocode:

```
function genetic_algorithm():
    population = initialize_random()
    while not found_solution:
        evaluate_fitness(population)
                                    parents    =
select_parents(population)
        offspring = crossover(parents)
        mutate(offspring)
                                 population    =
select_new_population(offspring)
    return best_solution
```

Justification for Use:

GA is highly effective for large board sizes due to its ability to explore diverse areas of the solution space without exhaustive search. It balances exploration and exploitation, making it ideal for high-dimensional problems like N-Queens.

IV. IMPLEMENTATION DETAILS

IV.I Code Structure:

The project contains a single file "nqueens_solver.py" which has a graphical user interface (GUI) for input from the user and the four different algorithms to find a solution to the N-Queens problem that are: Backtracking Search, Best First Search, Hill Climbing, and Genetic Algorithm

The most important aspects of the code are:

Backtracking Search: is coded in the solve_n_queens_backtracking(n) function and places the queens in a systematic way row by row looking for colliding queens in columns and diagonals. It uses recursion to look for possible positions and backtracks when it can't find a safe place for a queen.

Hill-Climbing Search: is used in the solve_n_queens_hill_climbin(n) function, it starts with a randomly filled board in which each row contains one queen and then reduces the number of conflicting queens iteratively.

Best-First search: is applied within the solve_n_queens_best_first(n) function that utilizes a priority queue utilized from python's built-in function "heapq" to store board states where they are prioritized based on the number of conflicts among the queens, the less the conflicts the higher the priority.

Genetic Algorithm: is in the function solve_n_queens_genetic(n) that begins with a random initial population of board states having each board a unique queen position. The algorithm ranks every board state by its conflict count and then evolves the population with selection, crossover and mutation:

- **Selection:** chooses the best boards with the fewest conflicts
- **Crossover:** select segments of both parent boards and combines them by mixing their queen positions to create a new child board
- **Mutation:** creates random mutations in a board by moving queens to other columns that enable discovering new possibilities without becoming fixed.

Graphical User Interface: Implemented in the class "NQueensGUI" with the tkinter library it allows users to

- Input board size (n)
- Select solving algorithm
- Start the process of solving
- Explore the different solutions if possible
- Project the solution on the board in real-time
- Display the time taken by each algorithm took to solve the problem

IV.II Tools and Libraries

Development Tools:

- Visual Studio Code: Used as the primary integrated development environment (IDE) for coding and keeping track of the code.
- Python: Used as the programming language for developing the N-Queens solvers and GUI.
- GitHub: Used for version control.
- Google Docs: Used for collaboration and creation of the project report.

Python Libraries:

- tkinter and ttk: For generating the graphical user interface (GUI).
- threading: For performing background solving in order to preserve GUI responsiveness.
- copy: For deep copying of board states.
- random: For generating random boards and mutations.
- heapq: For use in creating the priority queue for the Best-First Search.
- time: For keeping track of time elapsed in finding solutions.

V. TESTING & EVALUATION

V.I Testing Process

The N-Queens algorithms were tested on various board sizes including 5×5, 10×10, and 15×15 boards. Each of the four algorithms (Backtracking, Best-First Search, Hill-Climbing Search, Genetic Algorithm) was selected through the GUI. For each algorithm, the correctness of the board, which is that no queen can attack another queen was achieved, and the solving time was measured for each solution.

V.II Evaluation Metrics

1. **Correctness:** Whether the final board had no attacking queens.
2. **Solving Time:** Measured the time taken to find the solution using a timer.
3. **Success Rate:** For probabilistic algorithms (Hill-Climbing, Genetic), measured by whether they successfully found a valid solution in a given number of attempts.
4. **Scalability:** The algorithm's ability to solve more large boards efficiently.

V.III Comparison of Performance

The Backtracking algorithm: uses an exhaustive search approach, recursively exploring all possible board configurations. While making sure of discovery of all valid solutions, it is non-scalable: it's time to solve grows exponentially with board size. For the 15×15 instance, Backtracking reaches more than two million recursive calls and consumes more than 1132 seconds, which shows how impractical it is for large boards. But regardless of the extended time it consumed from him, it's the only algorithm providing all the possible solutions.

The Best-First Search: algorithm functioned as a highly successful guided search. It assigned more significance to boards with fewer conflicts at each stage, leading to solution discovery at much faster speeds. For example, Best-First discovered a solution to a 15×15 board within 1.219 seconds, and smaller boards like 5×5 and 10×10 immediately. Its performance was uniform for different board sizes.

The Hill-Climbing Search: used a heuristic approach of iteratively improving the board state. Even though it was extremely fast (0.001 – 0.133 seconds), it sometimes got trapped in local minima and needed to be restarted quite a few times in order to find a solution - 9 times for 10×10 and 19 times for 15×15 in our tests.

The Genetic Algorithm: was a heuristic and quick but probabilistic solution. It employed natural selection to evolve better boards over generations. It typically solved in a short time, often on the first generation, but not always each time. Occasionally, in some runs, more than one generation was taken, demonstrating its probabilistic nature.

Performance Tables:

Backtracking Search Algorithm Results

Board Size	Time Taken (sec)	Solutions Found	Notes
5×5	0.000	10	Exhaustive search, found all solutions
10×10	0.125	724	Took longer but still manageable
15×15	1584.745	~2,300,000	Extremely slow, not practical

Best-First Search Algorithm Results

Board Size	Time Taken (sec)	Solutions Found	Notes
5×5	0.000	1	Found solution instantly
10×10	0.032	1	Very fast and stable
15×15	1.157	1	Solved quickly with minimal effort

Hill Climbing Search Algorithm Results

Board Size	Time Taken (sec)	Solutions Found	Notes
5×5	0.001	1 (1st try)	Very fast
10×10	0.013	1 (9th try)	Needed multiple attempts
15×15	0.123	1 (19th try)	Needed multiple attempts

Genetic Search Algorithm Results

Board Size	Time Taken (sec)	Solutions Found	Notes
5×5	0.001	1 (1st try)	Very fast
10×10	0.034	1 (1st try but not always)	Fast but sometimes takes multiple tries
15×15	0.135	1 (1st try but not always)	Fast but sometimes takes multiple tries

V.IV Summary of Evaluation Metrics

Across all experiments, the implemented algorithms consistently produced valid solutions that respected the N-Queens constraints, ensuring that no two queens can threaten each other. The Backtracking and Best-First Search algorithms demonstrated the highest reliability, consistently solving boards with 100% success. However, Backtracking became impractical for larger board sizes due to its exponential time complexity, taking over 25 minutes for a board of size 25 x 25. On the other hand, Best-First Search was both fast and stable, solving all tested board sizes quickly with minimal time, making it the most balanced algorithm in terms of speed and accuracy. Finally, Hill-Climbing Search and the Genetic Algorithm were also able to solve all board sizes, but their success was probabilistic meaning that they would require multiple attempts in some cases. This happens because these algorithms start with random guesses and make small changes to improve the solution, which sometimes leads to a dead end instead of the best answer.

In terms of scalability, heuristic-based methods like Best-First, Hill-Climbing, and Genetic Algorithm scaled far more effectively to larger board sizes than exhaustive methods. While Backtracking guarantees all possible solutions, its runtime becomes exponentially slower as N increases, making it unsuitable for N > 12 in practical use. Overall, heuristic and evolutionary strategies proved to be the most efficient for solving large-scale instances of the N-Queens problem.

V.V Screenshots

1. **Backtracking Search Algorithm:**

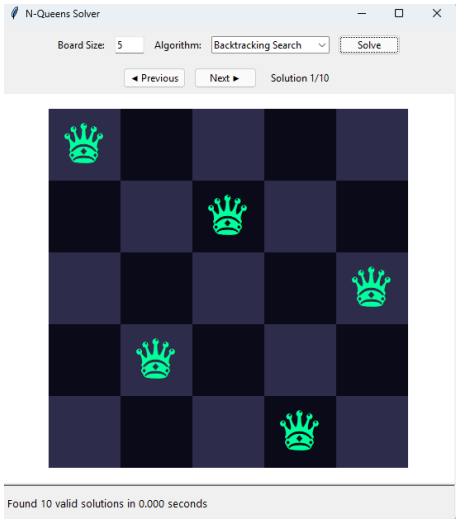

Figure 5. Backtracking Search Solution for 5x5 board

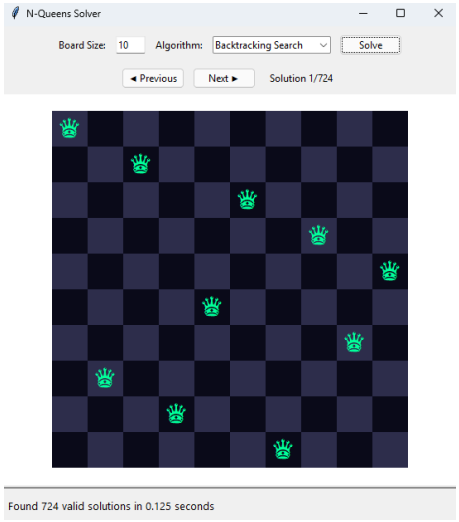

Figure 6. Backtracking Search Solution for 10x10 board

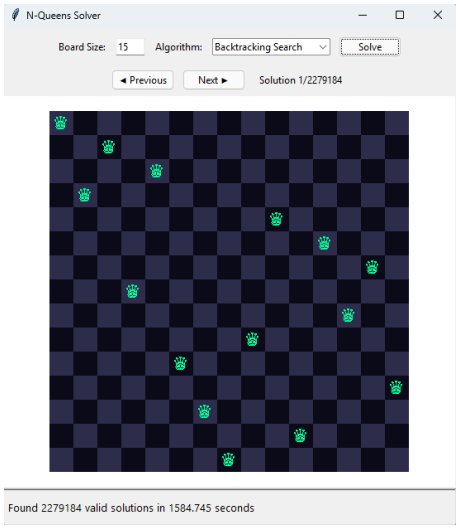

Figure 7. Backtracking Search Solution for 15x15 board

2. Best-First Search Algorithm:

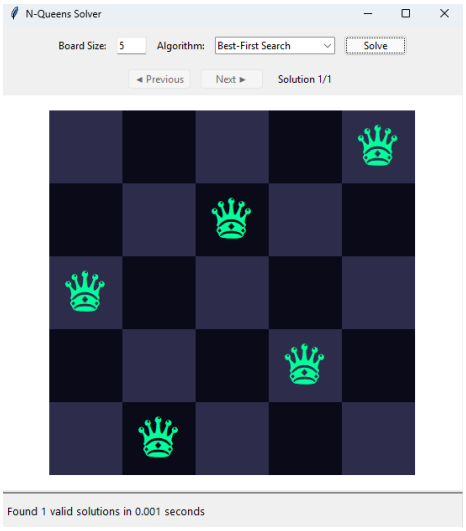

Figure 8. Best-First Search Solution for 5x5 board

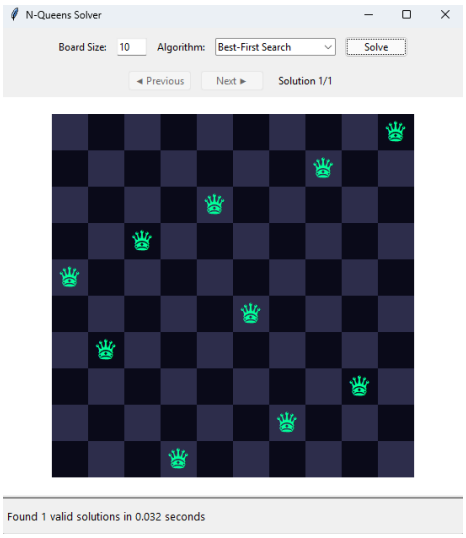

Figure 9. Best-First Search Solution for 10x10 board

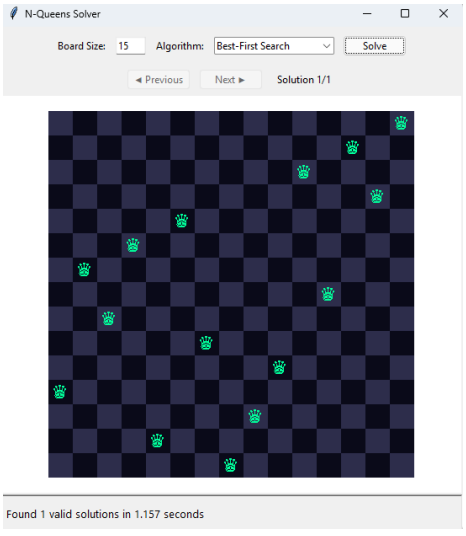

Figure 10. Best-First Search Solution for 15x15 board

3. Hill-Climbing Search Algorithm:

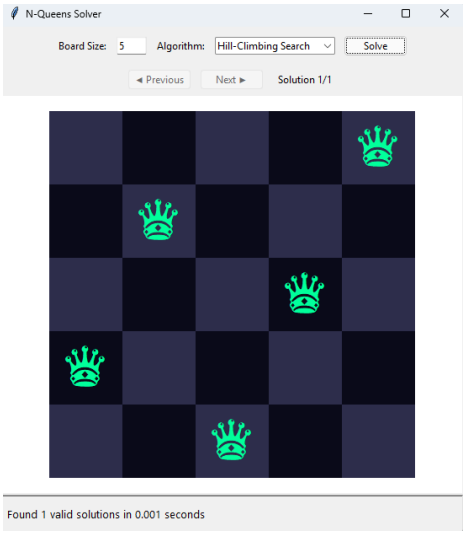

Figure 11. Hill-Climbing Search Solution for 5x5 board

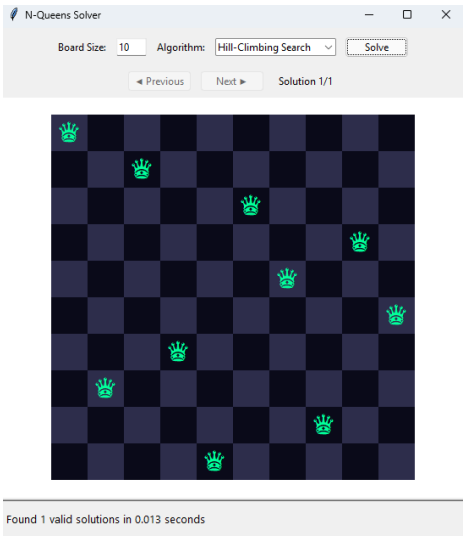

Figure 12. Hill-Climbing Search Solution for 10x10 board

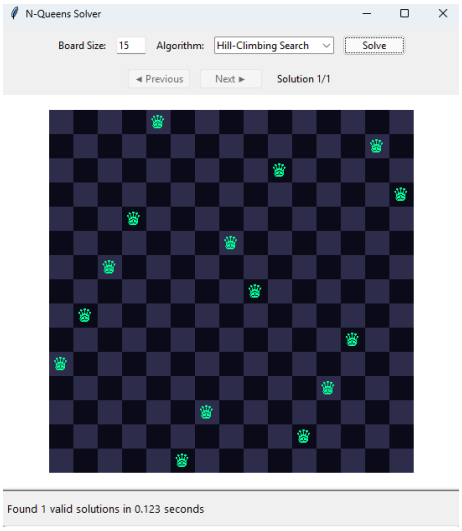

Figure 13. Hill-Climbing Search Solution for 15x15 board

4. Genetic Search Algorithm:

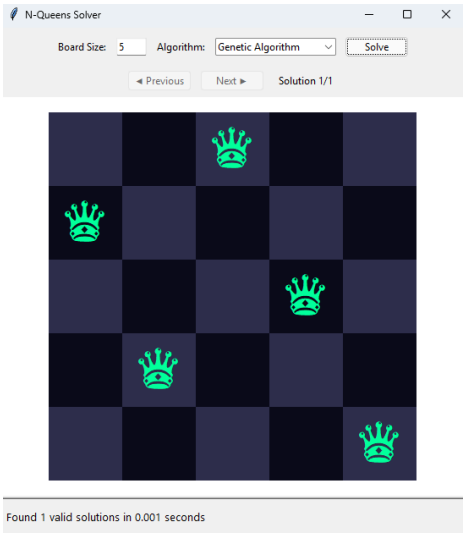

Figure 14. Genetic Search Solution for 5x5 board

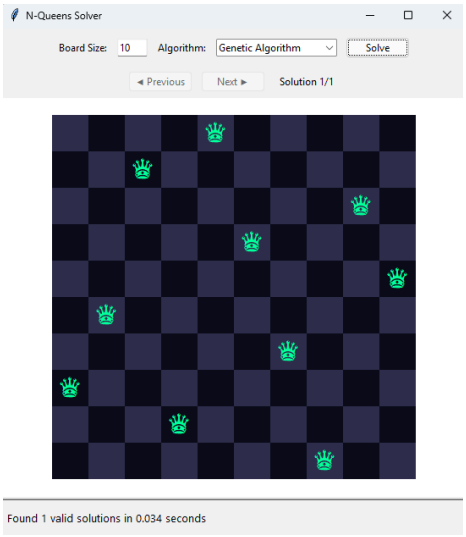

Figure 15. Genetic Search Solution for 10x10 board

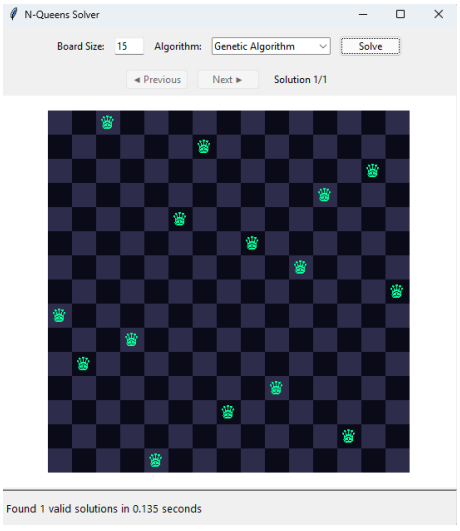

Figure 16. Genetic Search Solution for 15x15 board

VI. CHALLENGES

Handling Large Board Sizes (n > 12):

As the board size increases, especially in the Backtracking algorithm, the number of possible queen configurations grows exponentially. This significantly affects solving time and system performance. As shown, solving a 15×15 board using Backtracking took over 25 minutes making it impractical for interactive use.

Algorithm Performance:

Different algorithms showed varied reliability depending on board size. While Best-First Search consistently produced fast and stable results, heuristic-based algorithms such as Hill Climbing and Genetic Algorithm struggled to always find a solution on the try. Hill Climbing in particular often got stuck requiring multiple restarts to find a valid solution. The Genetic Algorithm was fast but its success depended on parameters like mutation rate and population size, which made consistent performance harder to guarantee.

VI.I FUTURE WORK

Progress Indicators:

Implementing a progress bar would improve user experience, especially where solving takes a long time.

Saving the Board:

Offer saving a solved board and the option to reload it later for study or display.

Expanding to Other Chess Problems:

Extending the application to solve similar problems like the Knight's Tour.

Adding log:

So users can see what happens along the algorithm.

VII. REFERENCES

[1] M. A. Khasawneh, *Hill-Climbing with Trees: A Novel Heuristic Approach for Discrete NP-Hard Combinatorial Optimization Problems*, Ph.D. dissertation, Binghamton Univ., NY, 2023.

[2] A. S. Farhan, W. Z. Tareq, and F. H. Awad, "Solving N Queen Problem using Genetic Algorithm," *Int. J. Comput. Appl.*, vol. 122, no. 12, pp. 11–12, Jul. 2015.

[3] V. Thada and S. Dhaka, "Performance Analysis of N-Queen Problem using Backtracking and Genetic Algorithm Techniques," *Int. J. Comput. Appl.*, vol. 102, no. 7, pp. 26–27, Sep. 2014.

[4] L. Andov, "Local Search Analysis N-Queens Problem," *Griffith University*, Mar. 2018.

[5] F. Soleimanian, B. Seyyedi, and G. Feyzipour, "A New Solution for N-Queens Problem using Blind Approaches: DFS and BFS Algorithms," *Int. J. Comput. Appl.*, vol. 53, no. 1, pp. 45–46, Sep. 2012.

[6] K. Mathew and M. Tabassum, "Experimental Comparison of Uninformed and Heuristic AI Algorithms for N Puzzle and 8 Queen Puzzle Solution," *Int. J. Digit. Inf. Wirel. Commun.*, vol. 4, no. 1, pp. 143–154, 2014.

[7] O. K. Majeed et al., "Performance comparison of genetic algorithms with traditional search techniques on the N-Queen Problem," in *Proc. 2023 Int. Conf. IT Ind. Technol.*, 2023, pp. 1–?.

[8] S. Mukherjee, S. Datta, P. B. Chanda, and P. Pathak, "Comparative Study of Different Algorithms to Solve N Queens Problem," *Int. J. Found. Computer Science Technol.*, vol. 5, no. 2, pp. 15–17, Mar. 2015.

[9] U. Sarkar and S. Nag, "An Adaptive Genetic Algorithm for Solving N-Queens Problem," arXiv preprint arXiv:1802.02006, 2017.