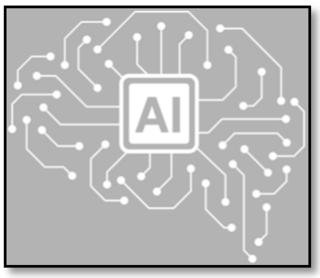
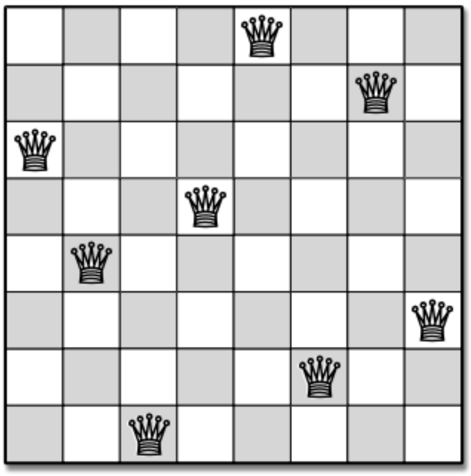
Shlomi Maalumi 212032205 || Avinoam Nukrai 206997132 || Oz Mizrahi 315178293



Introduction To Artificial Intelligence

Final Project

N-Queens



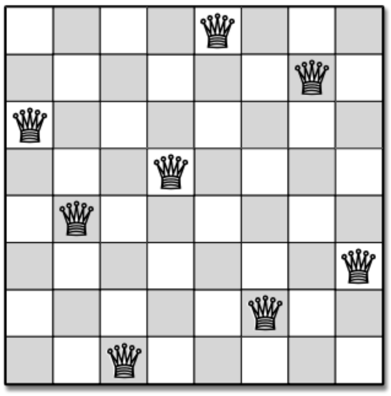


Table of Contents:

Introduction

The Problem

Overview: the N-Queens problem is a classic puzzle in Artificial Intelligence and combinatorial optimization. The problem is a generalization of the 8-queens puzzle, first proposed in the 19th century, and it has become a standard testbed for various algorithms in AI. The problem involves placing N chess queens on an N × N chessboard such that no two queens threaten each other - this means that no two queens can share the same row, column, or diagonal.

Why is this problem interesting? The N-Queens problem is particularly intriguing because it encapsulates the challenge of constraint satisfaction, a fundamental concept in AI. The constraints are clear and easy to understand, but as N increases, the number of possible configurations grows exponentially, making the problem difficult to solve. The goal is to find a valid configuration where all queens are placed without any conflicts.

Our Approaches

In our project, we chose to model, solve the N-Queens problem using two distinct algorithms: The Min-Conflict Heuristic and Genetic Algorithm.

Min-Conflict Approach: This algorithm is a heuristic search method that attempts to reduce the number of conflicts iteratively by moving queens to positions that minimize the number of conflicts. It is particularly effective for large values of N and is known for its efficiency in solving constraint satisfaction problems.

Genetic Algorithm Approach: This Algorithm is an evolutionary approach inspired by natural selection. It uses a population of potential solutions, evolves them over several generations, and applies operations like selection, crossover, and mutation to find an optimal or near-optimal solution. This approach is well-suited for exploring large search spaces and can provide diverse solutions, making it a robust choice for tackling the N-Queens problem.

Other Existing Approaches

In this paragraph we will explain why the naive approach and other approaches we saw in the course are less preferable to the proposed approaches, as follows:

Naïve Approach: This approach involves generating all possible configurations of N queens on an N×N board and checking each one for validity. This approach essentially relies on brute-force search, systematically placing queens on the board and verifying that no two queens threaten each other. This approach is highly inefficient. For example, for N = 8 the number of possible configurations is 16,777,216.

Planning Approach: GraphPlan is designed for action-based planning problems, where you need to find sequences of actions to achieve a goal. For the N-Queens Problem, GraphPlan is inefficient because it is not suited for constraint satisfaction problems like N-Queens, which involve finding configurations that meet constraints rather than action sequences. In addition, the number of potential configurations is exponential, making it challenging to define goal states for GraphPlan effectively.

Reinforcement Learning Approach: Reinforcement Learning and Q-Learning are designed for problems where learning comes from interacting with an environment and receiving feedback. They are not suitable for the N-Queens Problem because the problem is about finding exact solutions to constraints, not learning from feedback or interactions. And, in addition the N-Queens problem does not involve reward-based learning or trial-and-error, which are central to reinforcement learning.

Previous Work

The N-Queens problem has been addressed using various methods in the past, ranging from systematic searches to heuristic approaches. Historically, "Edsger Dijkstra" introduced depth first backtracking, and later solutions such as Random Restart Hill Climbing, Simulated Annealing, A*, and Best First Search (BFS) became popular.

Common Existing Algorithms

Random Restart Hill Climbing Algorithm: This local search algorithm explores neighboring states, restarting when stuck in local optima. While efficient for smaller boards, it suffers from scalability issues, requiring many restarts as the board size increases.

Simulated Annealing Algorithm: This method probabilistically avoids local optima by allowing less optimal moves, gradually reducing the probability over time. Its success depends heavily on tuning the cooling schedule and the number of iterations.

A* and BFS Algorithms: these systematic searches use heuristics to estimate the cost to reach a solution. While they ensure optimal solutions, they become computationally expensive and consume excessive memory as the board size increases.

Performance Benchmarks of Existing Algorithms

For smaller board sizes (e.g., n = 8), heuristic methods like Hill Climbing and Simulated Annealing perform well. However, as n increases, these methods require more restarts or iterations and longer runtimes. A* and BFS, although accurate, faces memory constraints and performance degradation with larger boards.

Common Assumptions of Existing Algorithms

The common assumptions between the existing algorithms are as follows:

1. **Fixed Board Size:** The number of queens (n) and the board size are predetermined, with complexity increasing as n grows.
2. **Random Initial State:** Many algorithms, like Hill Climbing and Simulated Annealing, begin with a randomly generated starting configuration of queens.
3. **Non-repeating States:** Systematic searches, such as A* and BFS, assume previously visited states won't be revisited, avoiding cycles.

4. **Heuristic Guidance:** Algorithms like A* rely on heuristics (e.g., counting attacking queens) to guide the search towards a solution.
5. **Local Optima Escape:** Techniques like Simulated Annealing assume that allowing suboptimal moves can help escape local optima and find global solutions.
6. **Equal Solution Quality:** Most methods assume that any valid configuration where no queens attack each other is equally acceptable.
7. **Incremental Changes (Not in genetic that do crossover!!!!):** Local search algorithms often assume that moving one queen at a time is sufficient to progress towards a solution.

Methodology

Min-Conflict

To solve the N-Queens problem, we implemented a hybrid approach combining the Min-Conflicts Heuristic with elements of local search. Our algorithm iteratively adjusts the positions of the queens, aiming to minimize conflicts until a valid solution is found or a preset limit of steps is reached.

Model Description

The algorithm built from the following building blocks:

1. **Initialization:** A random initial board configuration $Q = \{q_1, q_2, \dots, q_N\}$ of N queens is placed on the board such that each queen q_i placed on a random cell $(i, j) \in \{1,2,\dots,N\} x \{1,2,\dots,N\}$ on the board.
2. **Conflict Resolution Loop:**

   At each step:

   - Let $c(q_i)$ represent the number of conflicts for queen q_i
   - The algorithm selects the queen q_{max} with the highest conflict value:

   $$q_{max} = argmax_{q_i}\, c(q_i)$$

   - q_{max} is relocated to a row r' in the same column i minimizing its conflicts:

   $$r' = argmin_r\, c(q_i\, in\, row\, r)$$

   This process repeats until one of the following is occurred:

   - A solution with zero conflicts is found, i.e., $\sum^{N}{}_{i=1} c(q_i) = 0$.
   - The predefined step limit L is reached.
3. **Termination:**

   The algorithm terminates when either a conflict-free configuration is found, or the step limit L is reached. If $\sum^{N}{}_{i=1} c(q_i) = 0$, the algorithm returns a valid solution. Otherwise, it terminated.

Model Assumptions

1. **Random Initial Placement:** We assume that starting with a randomly generated board configuration gives enough diversity to explore the solution A

2. **Local Conflict Minimization:** We assume that making greedy choices (moving the most conflicted queen to a better position) can lead to a global solution or minimize conflicts.
3. **Step Limit:** The algorithm assumes that if a solution is not found within a certain number of steps (exact number will be selected after result research), the current configuration is unlikely to lead to a valid solution without starting over.

Model Success Criteria

The success of the algorithm is determined by the following:

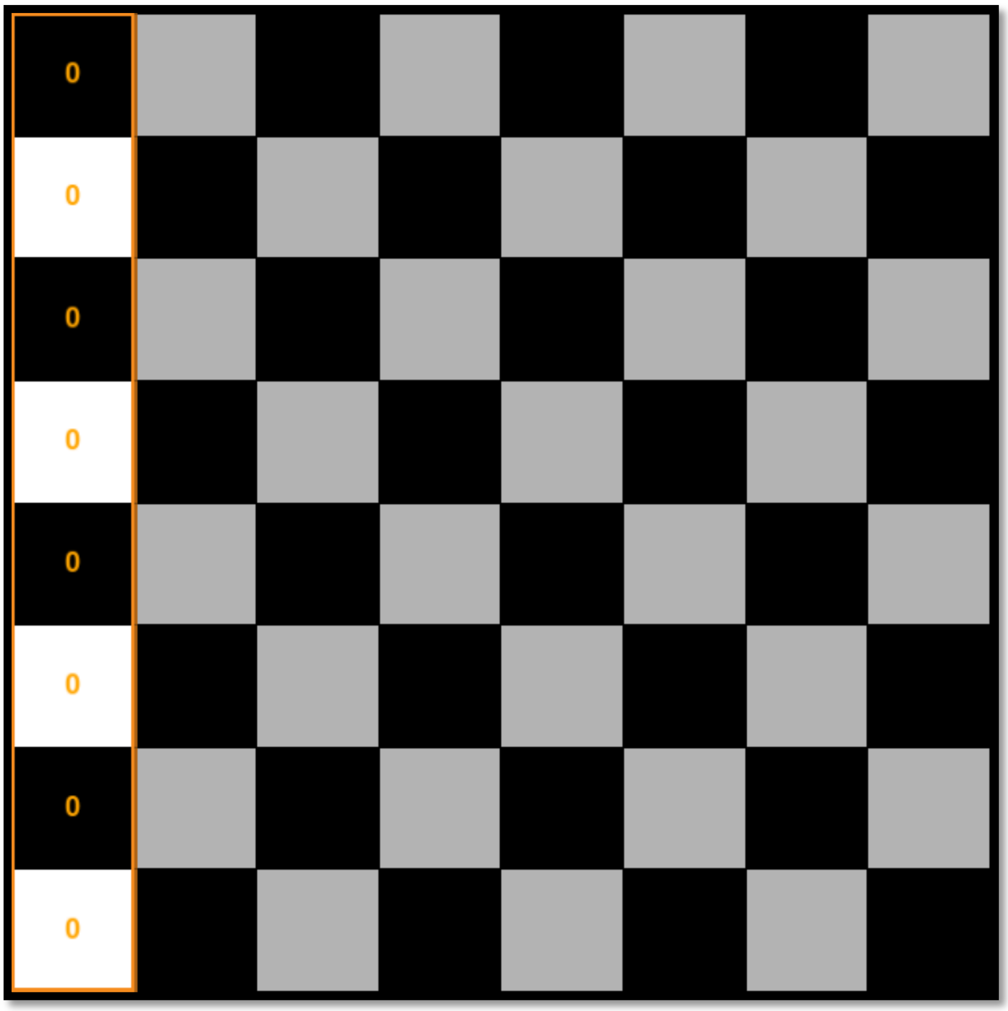
1. **Conflict Minimization:** The algorithm should consistently reduce the number of conflicts to a low level, indicating that it effectively approaches near-optimal solutions within the given step limit.
2. **Efficiency:** The algorithm should quickly converge to a configuration with minimal conflicts, demonstrating its ability to explore the solution space efficiently, even if it does not always find a conflict-free solution.

Model Evaluation

To evaluate the quality of our solution, we measure the following: The average, median number of steps required to find a valid solution. The percentage of successful runs that result in a valid solution within the step limit. Run-time efficiency as board size increases. average Number of conflicts in un-success termination.

Model Visualization

A board of size 8 is presented, in each iteration we will look for the queen with the largest number of conflicts and place her in a position with the smallest possible number of conflicts, until a conflict-free configuration is found, or the step limit L is reached as follows:



Genetic Algorithm

The Genetic Algorithm is an evolutionary approach inspired by the principles of natural selection. For solving the N-Queens problem, the algorithm operates on a population of candidate solutions. The algorithm iteratively improves this population by selecting parents, applying genetic operations (crossover\mutation), and generating new offspring. The goal is to find a valid configuration where no two queens threaten each other, meaning they cannot share the same row, column, or diagonal.

Model Description

The algorithm built from the following building blocks:

1. **Initialization:** The algorithm begins by generating an initial population of P random permutations, where each permutation represents a valid configuration of queens on the board. Formally, let $P = \{p_1, p_2, \ldots, p_k\}$ be a population of size k, where $p_i \in S_N$ is a random permutation of $\{1, 2, \ldots, N-1\}$.
2. **Fitness Function:** The fitness function evaluates how close a given solution is to being conflict-free. The objective is to minimize the number of conflicts, where a conflict occurs if two queens share the same row or diagonal. Mathematically, for each permutation p, the fitness function $F(p)$ defined as:
$$F(p) = \sum_{i=1}^{N}\sum_{j=i+1}^{N} (\delta(p[i], p[j]) + \delta(|p[i] - p[j]|, |i - j|))$$
Where:
   - $p[i]$ is the row index of the queen in column i.
   - $\delta(x, y)$ is the Kronecker delta, which is 1 if $x = y$ and 0 otherwise.

   The fitness function $F(p)$ returns the total number of attacking pairs of queens in configuration p, where a value of $F(p) = 0$ indicates a solution.
3. **Parent Selection:** Parents are selected by randomly choosing a batch from the current generation. From this batch, the two solutions with the highest fitness values are chosen to be the parents.
4. **Crossover:** Crossover combines two parent boards to create a new child board. A crossover point is randomly chosen, and segments from each parent are exchanged to produce the child board.

5. **Mutation:** Mutation introduces variability into the population. Each board may be mutated by selecting randomly column and with a probability defined by the mutation rate replace it. This process helps in exploring the solution space and avoids premature convergence to suboptimal solutions.
6. **Termination:** The algorithm continues evolving the population through generations until a conflict-free solution is found (zero fitness value) or the maximum number of generations is reached.

Model Assumptions

TO ADD

Model Success Criteria

The success of the Genetic Algorithm in solving the N-Queens problem is evaluated based on two key criteria:

1. **Solution Validity:** The primary measure of success is achieving a conflict-free configuration of queens, i.e., a fitness score of zero.
2. **Scalability:** The ability to maintain solution quality and efficiency as N increases (for larger board sizes) is critical for evaluating the robustness of the algorithm.

Model Evaluation Metrics

To assess the quality of our solution, we track:

- The average and median number of generations required to find a solution.
- The percentage of runs that successfully terminate with a conflict-free configuration.
- The running time and computational cost as the board size increases, ensuring the algorithm scales effectively.

Results

Genetic Algorithm Results

[Pure results at the end of the section]

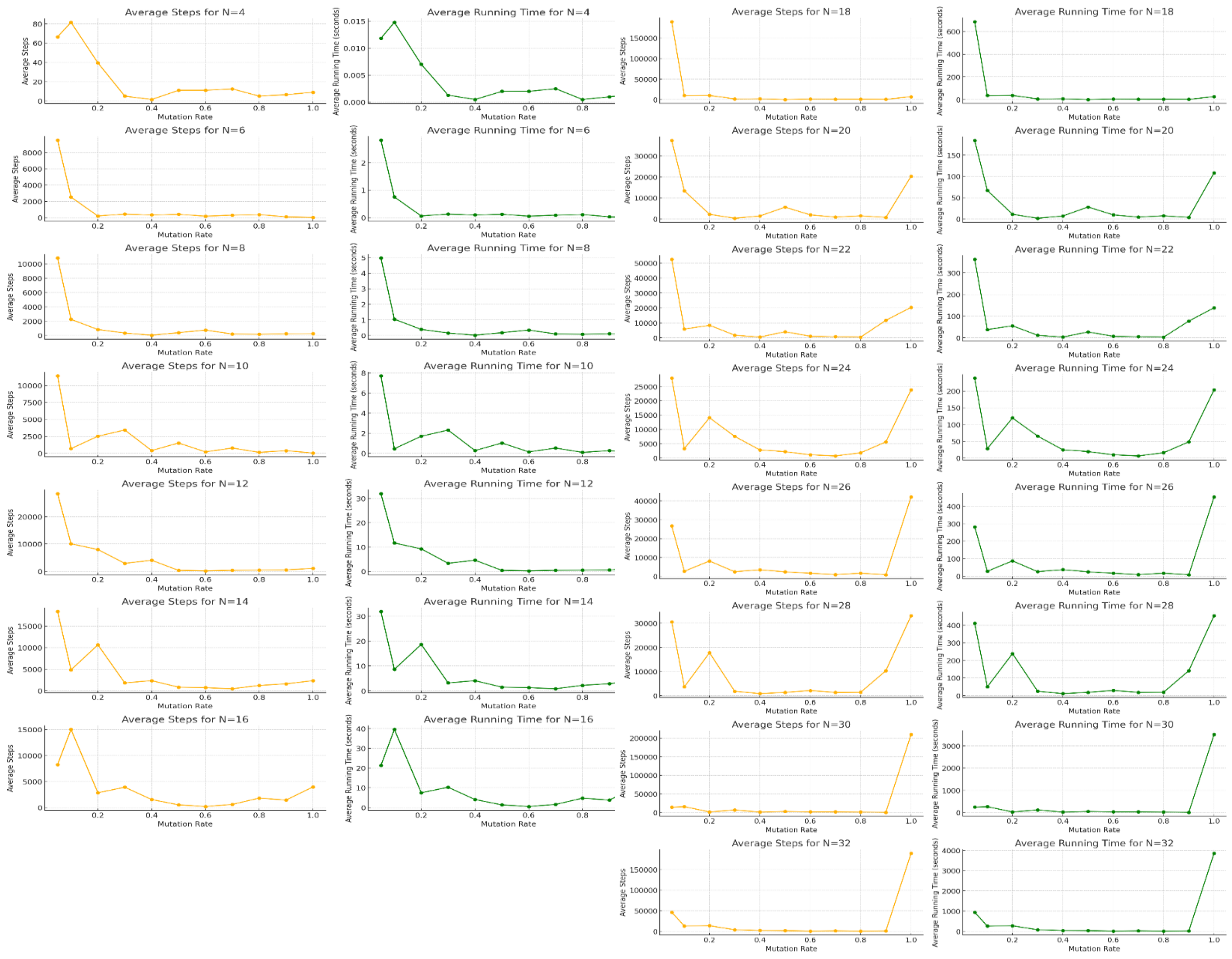
The algorithm (GA) was applied to the N-Queens problem with varying values of N (the number of queens) and key parameters such as **mutation rate** and **population size**. We let our algorithm run without limit, until a solution has been found. Below, we analyze the main findings from the experiments based on the provided plots and tables.

Mutation Rate: Running Time & Steps

The mutation rate, which determines how often random changes are introduced in the population, plays a crucial role in the performance of the GA. The optimal mutation rate was evaluated across different values of N, and several key trends were observed שגיאה! מקור ההפניה לא נמצא. -

- For small values of N (e.g., N=4), a mutation rate of 0.4 was found to be the most effective, minimizing the number of steps required to find a solution.
- As N increases, the optimal mutation rate fluctuates. For instance: c
    - For N=6, the best mutation rate is 1.0 (i.e., every individual in the population undergoes mutation), indicating that for larger problem sizes, more diversity in the population is beneficial.
    - Similarly, for N=10, a high mutation rate of 1.0 continues to provide better performance, highlighting the need for increased variability when the search space grows larger.
    - For N=8, the best mutation rate drops back to **0.4**, suggesting that moderate mutation rates are better suited for medium-sized boards.

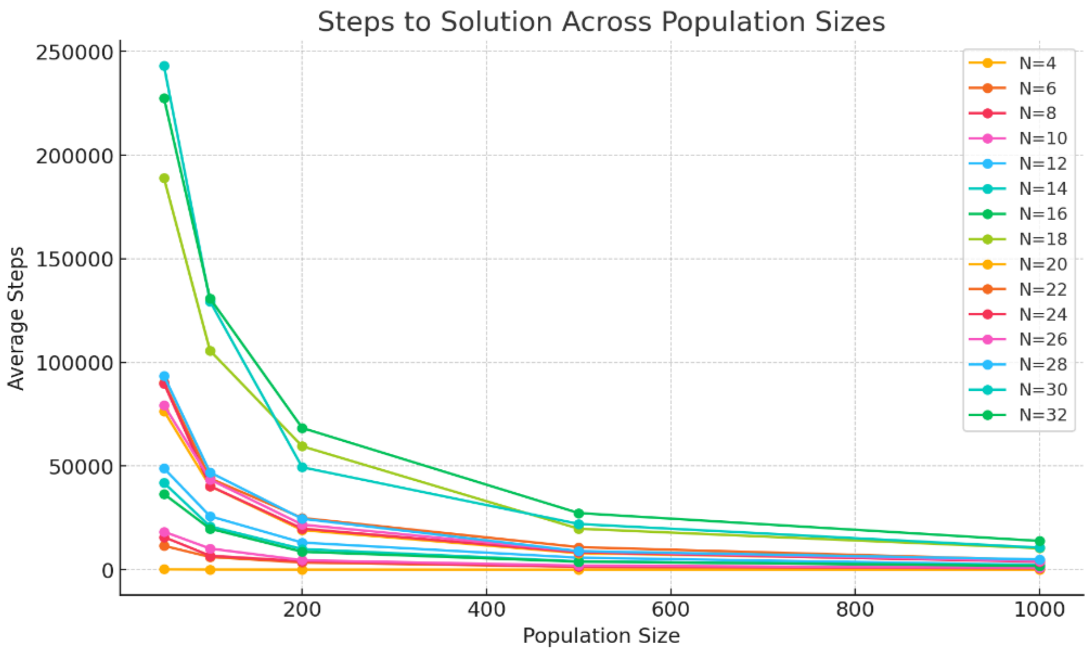
These findings indicate that **mutation rate is highly problem-size dependent**. Smaller values of N tend to benefit from moderate mutation rates (around 0.4 to 0.6), while larger problem sizes require more aggressive mutation rates (closer to 1.0) to maintain diversity and avoid premature convergence.


Average Steps for N=4
Average Running Time for N=4
Average Steps for N=18
Average Running Time for N=18
Average Steps for N=6
Average Running Time for N=6
Average Steps for N=20
Average Running Time for N=20
Average Steps for N=8
Average Running Time for N=8
Average Steps for N=22
Average Running Time for N=22
Average Steps for N=10
Average Running Time for N=10
Average Steps for N=24
Average Running Time for N=24
Average Steps for N=12
Average Running Time for N=12
Average Steps for N=26
Average Running Time for N=26
Average Steps for N=14
Average Running Time for N=14
Average Steps for N=28
Average Running Time for N=28
Average Steps for N=16
Average Running Time for N=16
Average Steps for N=30
Average Running Time for N=30
Average Steps for N=32
Average Running Time for N=32
Average Steps
Average Running Time (seconds)
Mutation Rate

Population size sensitivity:

The impact of population size on the number of steps required to find a solution is highlighted in the plot Steps to Solution Across Population Sizes. Several trends emerge from this analysis:

- For all values of N, increasing the population size reduces the number of steps to solution. For instance, for N=4, increasing the population size from 50 to 200 reduces the number of steps by more than half.
- However, the benefit of increasing population size plateaus around 400-600 individuals. Beyond this threshold, further increases in population size provide diminishing returns in terms of reducing the number of steps.
- For very large values of N, such as N=30 and N=32, larger population sizes (up to 1000) still provide some benefit, but the gains are not as substantial as they are for smaller boards.

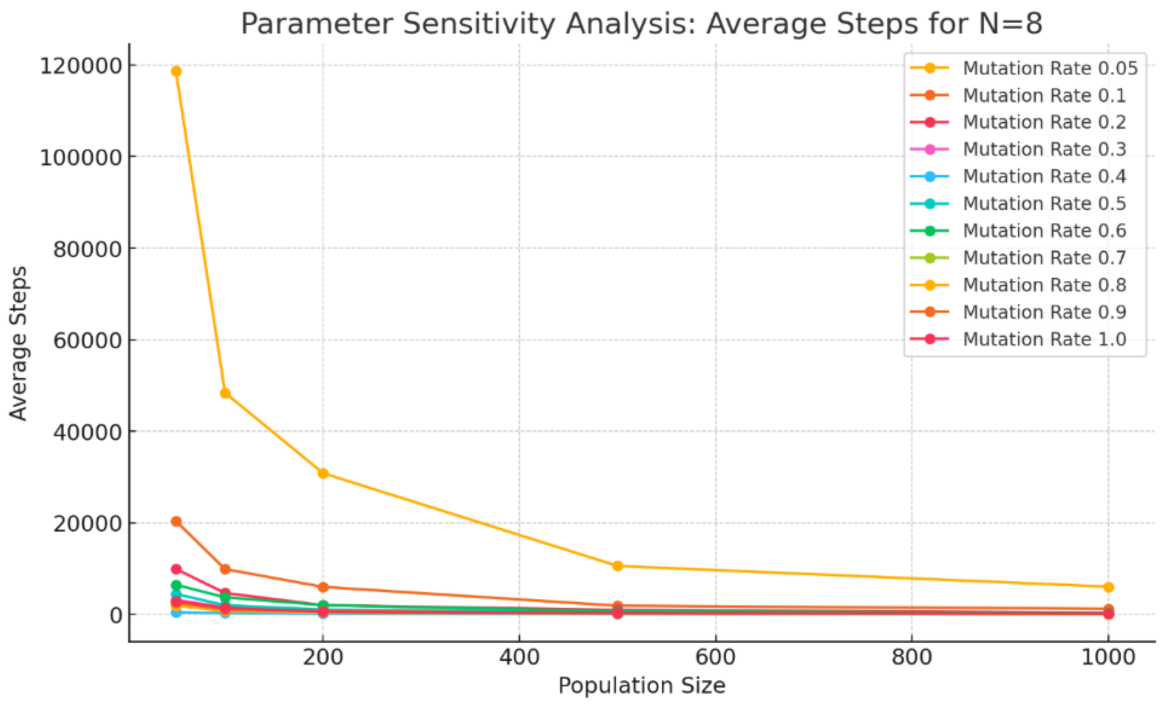


This analysis suggests that **population size is a crucial factor** in improving the efficiency of the Genetic Algorithm, but there is a saturation point beyond which additional increases provide minimal benefit. A population size between 400 and 600 is generally sufficient to maintain good performance across various board sizes.

Parameter Sensitivity Analysis

The **Parameter Sensitivity Analysis for N=8** examines the interaction between population size and mutation rate. Key findings include:

- For a mutation rate of 0.05, the number of steps required to find a solution is significantly higher, regardless of population size. This confirms that very low mutation rates lead to poor performance, as the population lacks the genetic diversity needed to explore the solution space effectively.
- Mutation rates between 0.3 and 0.6 perform best across all population sizes, with the number of steps decreasing as population size increases.
- Higher mutation rates result in fewer steps when the population size is small, but as the population size increases, the benefit of high mutation rates diminishes. This suggests that when the population size is large, the algorithm can rely more on crossover (rather than mutation) to generate diversity.



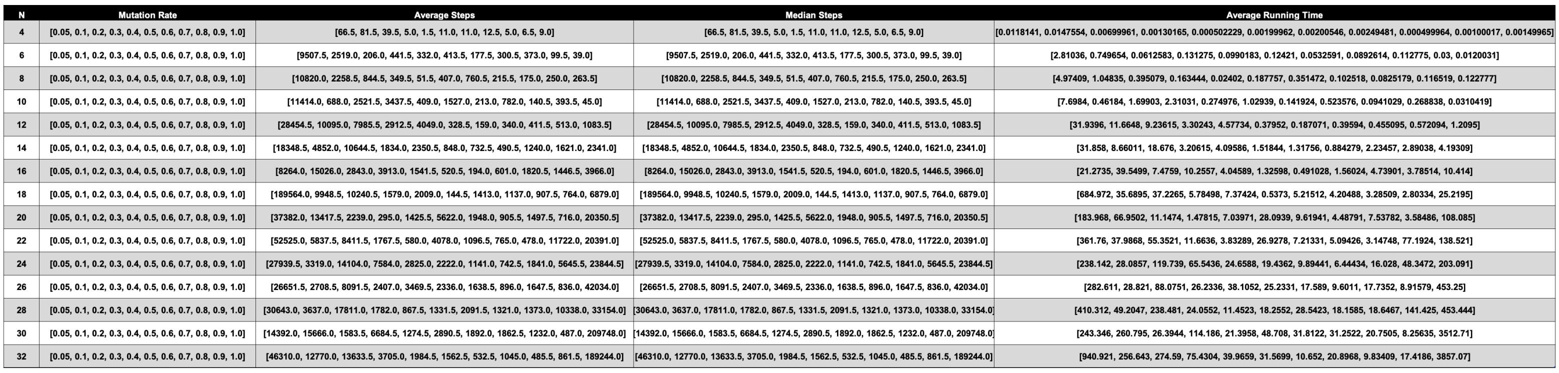
This sensitivity analysis reinforces the idea that **moderate mutation rates (0.3 to 0.6) and population sizes around 400-600 provide the best overall performance**. Higher mutation rates may be beneficial in smaller populations but are less effective when the population size is large.

Best Mutation Rate Across N Values Analysis

The table **Best Mutation Rate Summary** highlights the mutation rates that yielded the fewest steps to solution across different values of N:

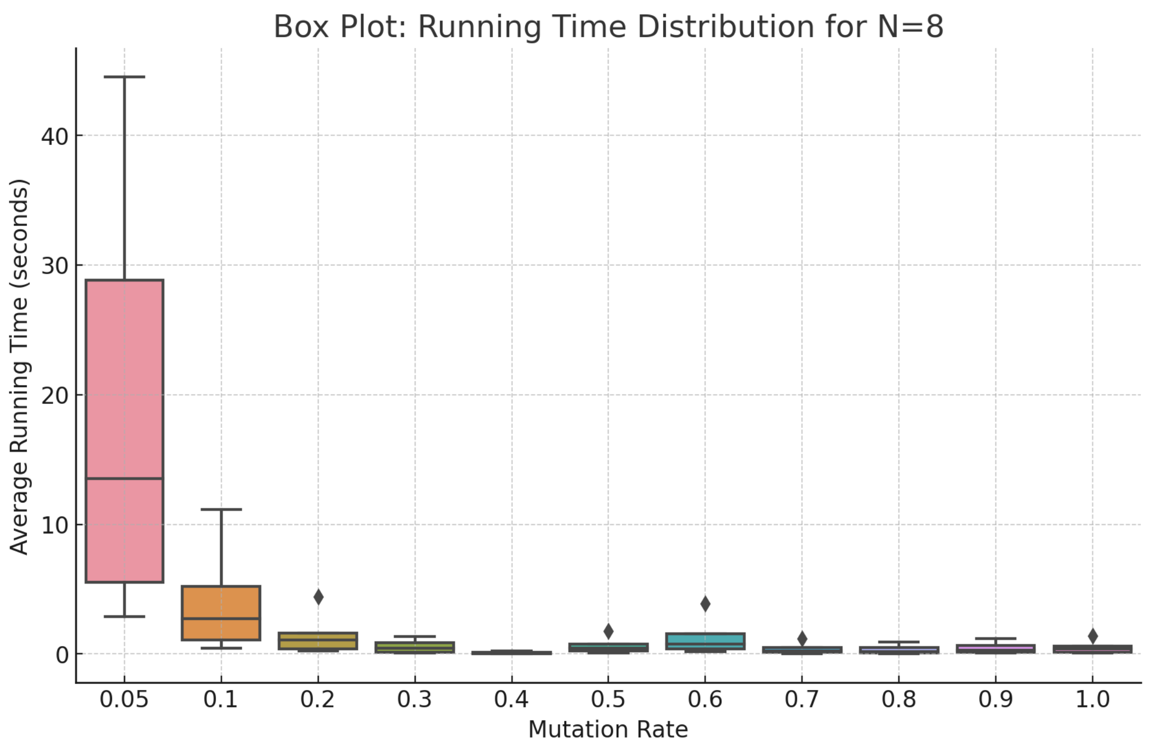
- For N=4, the best mutation rate is 0.4, with a minimum of 1.5 steps on average.
- For N=8, the best mutation rate is also 0.4, with 51.5 steps on average.
- For larger values like N=10, the best mutation rate increases to 1.0, requiring 45 steps on average.
- The mutation rate fluctuates for other values of N, such as 0.6 for N=12 and 1.0 for N=6, demonstrating that the optimal mutation rate changes based on the size of the problem.

The key takeaway from this table is that mutation rate optimization is crucial and needs to be fine-tuned for different board sizes. Smaller to medium-sized boards tend to perform better with moderate mutation rates, while larger boards often require higher rates to maintain diversity and efficiency. In order to gain the best mutation rate, we implemented a linear regression model to fit best mutation rate across different N values.



N	Mutation Rate	Average Steps	Median Steps	Average Running Time
4	[0.05, 0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1.0]	[66.5, 81.5, 39.5, 5.0, 1.5, 11.0, 11.0, 12.5, 5.0, 6.5, 9.0]	[66.5, 81.5, 39.5, 5.0, 1.5, 11.0, 11.0, 12.5, 5.0, 6.5, 9.0]	[0.0118141, 0.0147554, 0.00699961, 0.00130165, 0.000502229, 0.00199962, 0.00200546, 0.00249481, 0.000499964, 0.00100017, 0.00149965]
6	[0.05, 0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1.0]	[9507.5, 2519.0, 206.0, 441.5, 332.0, 413.5, 177.5, 300.5, 373.0, 99.5, 39.0]	[9507.5, 2519.0, 206.0, 441.5, 332.0, 413.5, 177.5, 300.5, 373.0, 99.5, 39.0]	[2.81036, 0.749654, 0.0612583, 0.131275, 0.0990183, 0.12421, 0.0532591, 0.0892614, 0.112775, 0.03, 0.0120031]
8	[0.05, 0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1.0]	[10820.0, 2258.5, 844.5, 349.5, 51.5, 407.0, 760.5, 215.5, 175.0, 250.0, 263.5]	[10820.0, 2258.5, 844.5, 349.5, 51.5, 407.0, 760.5, 215.5, 175.0, 250.0, 263.5]	[4.97409, 1.04835, 0.395079, 0.163444, 0.02402, 0.187757, 0.351472, 0.102518, 0.0825179, 0.116519, 0.122777]
10	[0.05, 0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1.0]	[11414.0, 688.0, 2521.5, 3437.5, 409.0, 1527.0, 213.0, 782.0, 140.5, 393.5, 45.0]	[11414.0, 688.0, 2521.5, 3437.5, 409.0, 1527.0, 213.0, 782.0, 140.5, 393.5, 45.0]	[7.6984, 0.46184, 1.69903, 2.31031, 0.274976, 1.02939, 0.141924, 0.523576, 0.0941029, 0.268838, 0.0310419]
12	[0.05, 0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1.0]	[28454.5, 10095.0, 7985.5, 2912.5, 4049.0, 328.5, 159.0, 340.0, 411.5, 513.0, 1083.5]	[28454.5, 10095.0, 7985.5, 2912.5, 4049.0, 328.5, 159.0, 340.0, 411.5, 513.0, 1083.5]	[31.9396, 11.6648, 9.23615, 3.30243, 4.57734, 0.37952, 0.187071, 0.39594, 0.455095, 0.572094, 1.2095]
14	[0.05, 0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1.0]	[18348.5, 4852.0, 10644.5, 1834.0, 2350.5, 848.0, 732.5, 490.5, 1240.0, 1621.0, 2341.0]	[18348.5, 4852.0, 10644.5, 1834.0, 2350.5, 848.0, 732.5, 490.5, 1240.0, 1621.0, 2341.0]	[31.858, 8.66011, 18.676, 3.20615, 4.09586, 1.51844, 1.31756, 0.884279, 2.23457, 2.89038, 4.19309]
16	[0.05, 0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1.0]	[8264.0, 15026.0, 2843.0, 3913.0, 1541.5, 520.5, 194.0, 601.0, 1820.5, 1446.5, 3966.0]	[8264.0, 15026.0, 2843.0, 3913.0, 1541.5, 520.5, 194.0, 601.0, 1820.5, 1446.5, 3966.0]	[21.2735, 39.5499, 7.4759, 10.2557, 4.04589, 1.32598, 0.491028, 1.56024, 4.73901, 3.78514, 10.414]
18	[0.05, 0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1.0]	[189564.0, 9948.5, 10240.5, 1579.0, 2009.0, 144.5, 1413.0, 1137.0, 907.5, 764.0, 6879.0]	[189564.0, 9948.5, 10240.5, 1579.0, 2009.0, 144.5, 1413.0, 1137.0, 907.5, 764.0, 6879.0]	[684.972, 35.6895, 37.2265, 5.78498, 7.37424, 0.5373, 5.21512, 4.20488, 3.28509, 2.80334, 25.2195]
20	[0.05, 0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1.0]	[37382.0, 13417.5, 2239.0, 295.0, 1425.5, 5622.0, 1948.0, 905.5, 1497.5, 716.0, 20350.5]	[37382.0, 13417.5, 2239.0, 295.0, 1425.5, 5622.0, 1948.0, 905.5, 1497.5, 716.0, 20350.5]	[183.968, 66.9502, 11.1474, 1.47815, 7.03971, 28.0939, 9.61941, 4.48791, 7.53782, 3.58486, 108.085]
22	[0.05, 0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1.0]	[52525.0, 5837.5, 8411.5, 1767.5, 580.0, 4078.0, 1096.5, 765.0, 478.0, 11722.0, 20391.0]	[52525.0, 5837.5, 8411.5, 1767.5, 580.0, 4078.0, 1096.5, 765.0, 478.0, 11722.0, 20391.0]	[361.76, 37.9868, 55.3521, 11.6636, 3.83289, 26.9278, 7.21331, 5.09426, 3.14748, 77.1924, 138.521]
24	[0.05, 0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1.0]	[27939.5, 3319.0, 14104.0, 7584.0, 2825.0, 2222.0, 1141.0, 742.5, 1841.0, 5645.5, 23844.5]	[27939.5, 3319.0, 14104.0, 7584.0, 2825.0, 2222.0, 1141.0, 742.5, 1841.0, 5645.5, 23844.5]	[238.142, 28.0857, 119.739, 65.5436, 24.6588, 19.4362, 9.89441, 6.44434, 16.028, 48.3472, 203.091]
26	[0.05, 0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1.0]	[26651.5, 2708.5, 8091.5, 2407.0, 3469.5, 2336.0, 1638.5, 896.0, 1647.5, 836.0, 42034.0]	[26651.5, 2708.5, 8091.5, 2407.0, 3469.5, 2336.0, 1638.5, 896.0, 1647.5, 836.0, 42034.0]	[282.611, 28.821, 88.0751, 26.2336, 38.1052, 25.2331, 17.589, 9.6011, 17.7352, 8.91579, 453.25]
28	[0.05, 0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1.0]	[30643.0, 3637.0, 17811.0, 1782.0, 867.5, 1331.5, 2091.5, 1321.0, 1373.0, 10338.0, 33154.0]	[30643.0, 3637.0, 17811.0, 1782.0, 867.5, 1331.5, 2091.5, 1321.0, 1373.0, 10338.0, 33154.0]	[410.312, 49.2047, 238.481, 24.0552, 11.4523, 18.2552, 28.5423, 18.1585, 18.6467, 141.425, 453.444]
30	[0.05, 0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1.0]	[14392.0, 15666.0, 1583.5, 6684.5, 1274.5, 2890.5, 1892.0, 1862.5, 1232.0, 487.0, 209748.0]	[14392.0, 15666.0, 1583.5, 6684.5, 1274.5, 2890.5, 1892.0, 1862.5, 1232.0, 487.0, 209748.0]	[243.346, 260.795, 26.3944, 114.186, 21.3958, 48.708, 31.8122, 31.2522, 20.7505, 8.25635, 3512.71]
32	[0.05, 0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1.0]	[46310.0, 12770.0, 13633.5, 3705.0, 1984.5, 1562.5, 532.5, 1045.0, 485.5, 861.5, 189244.0]	[46310.0, 12770.0, 13633.5, 3705.0, 1984.5, 1562.5, 532.5, 1045.0, 485.5, 861.5, 189244.0]	[940.921, 256.643, 274.59, 75.4304, 39.9659, 31.5699, 10.652, 20.8968, 9.83409, 17.4186, 3857.07]

איור 1


Box Plot: Running Time Distribution for N=8
Average Running Time (seconds)
0
10
20
30
40
0.05
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
0.9
1.0
Mutation Rate

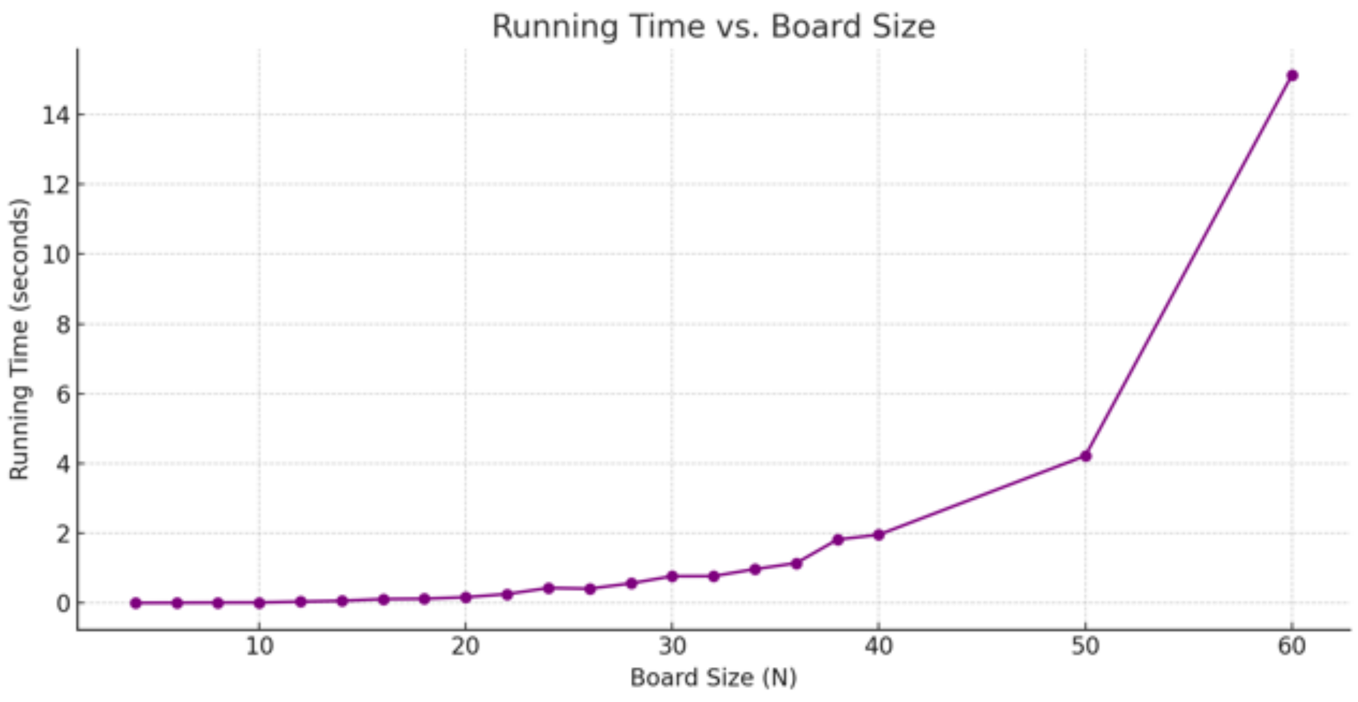
Min-Conflict Results

Overview

The Min-Conflict algorithm, applied to the N-Queens problem, exhibits distinctive trends in performance metrics as board size increases. This analysis focuses on four key metrics: running time, success rate, average and median steps, and average conflicts upon failure, to evaluate the algorithm's efficacy and efficiency across different board sizes. The findings draw on plots and data extrapolated from the experimental outcomes.

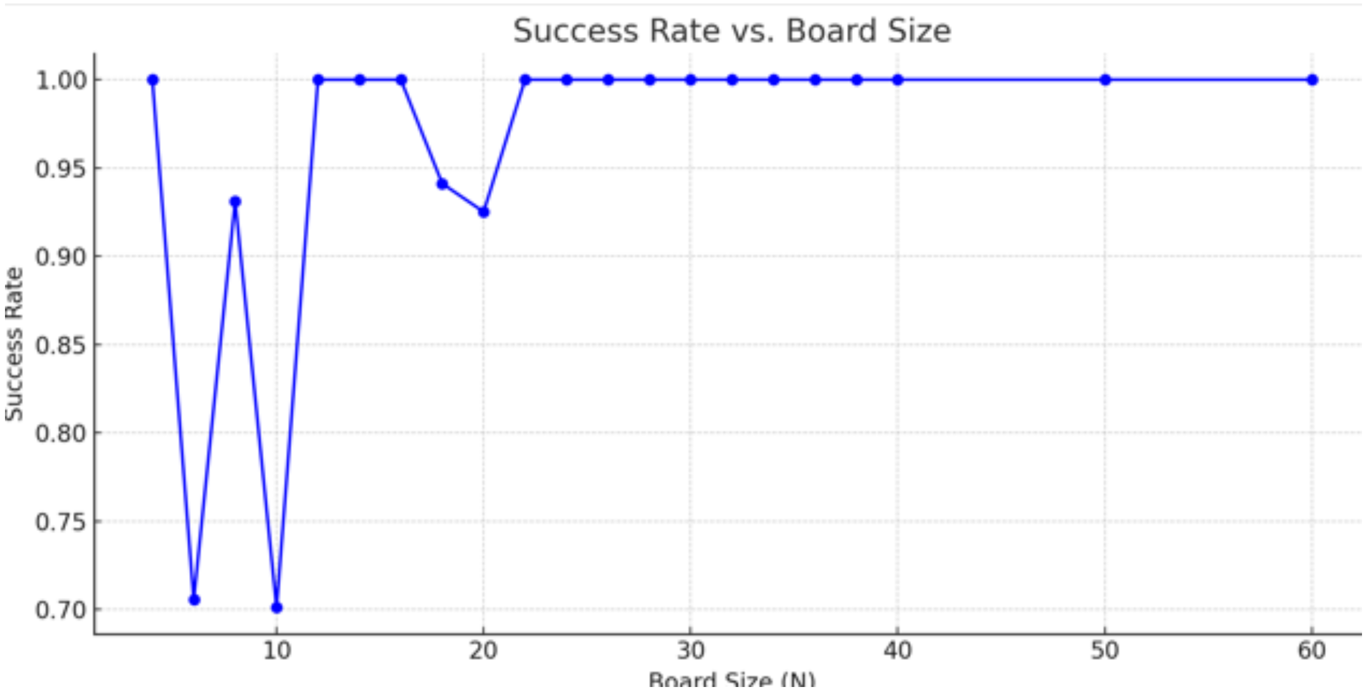
Running Time vs. Board Size

The first plot, "Running Time vs. Board Size," shows a sharp increase in running time as the board size approaches 60. Initially, for smaller board sizes (up to 30), the running time increases gradually, suggesting that the algorithm efficiently handles smaller instances of the problem with minor increases in computation time. However, the exponential spike from board size 50 to 60 indicates a scalability issue where the algorithm's performance deteriorates dramatically as the problem complexity (board size) increases. This pattern suggests that the algorithm's local search mechanism struggles to converge efficiently to a solution for larger boards due to the increased possibilities of conflicts.



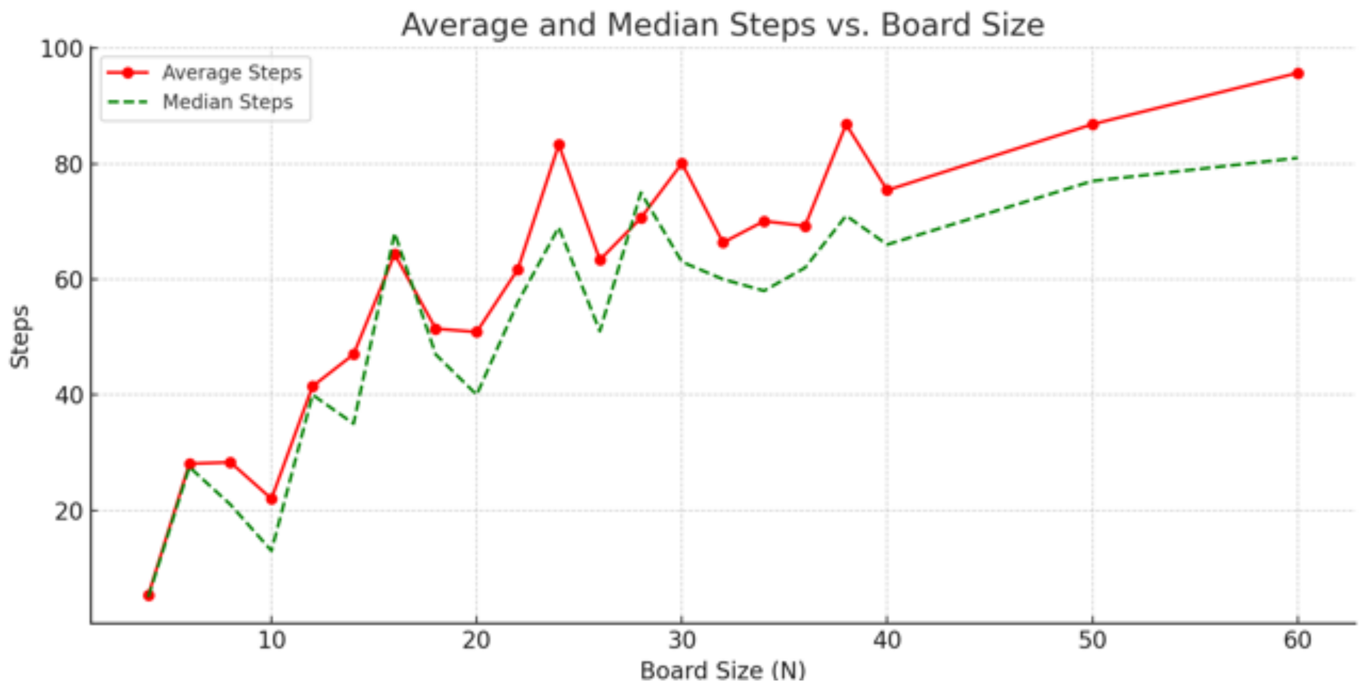
Success Rate vs. Board Size

The "Success Rate vs. Board Size" plot below indicates that the algorithm maintains a high success rate (>90%) across most board sizes, with noticeable dips around board sizes of 10 and 30. The drop in success rate at these specific points could indicate that certain board configurations are particularly challenging for the algorithm's conflict resolution strategy. However, the general stability in success rates above 90% demonstrates that the Min-Conflict algorithm reliably finds solutions without conflicts in most cases.



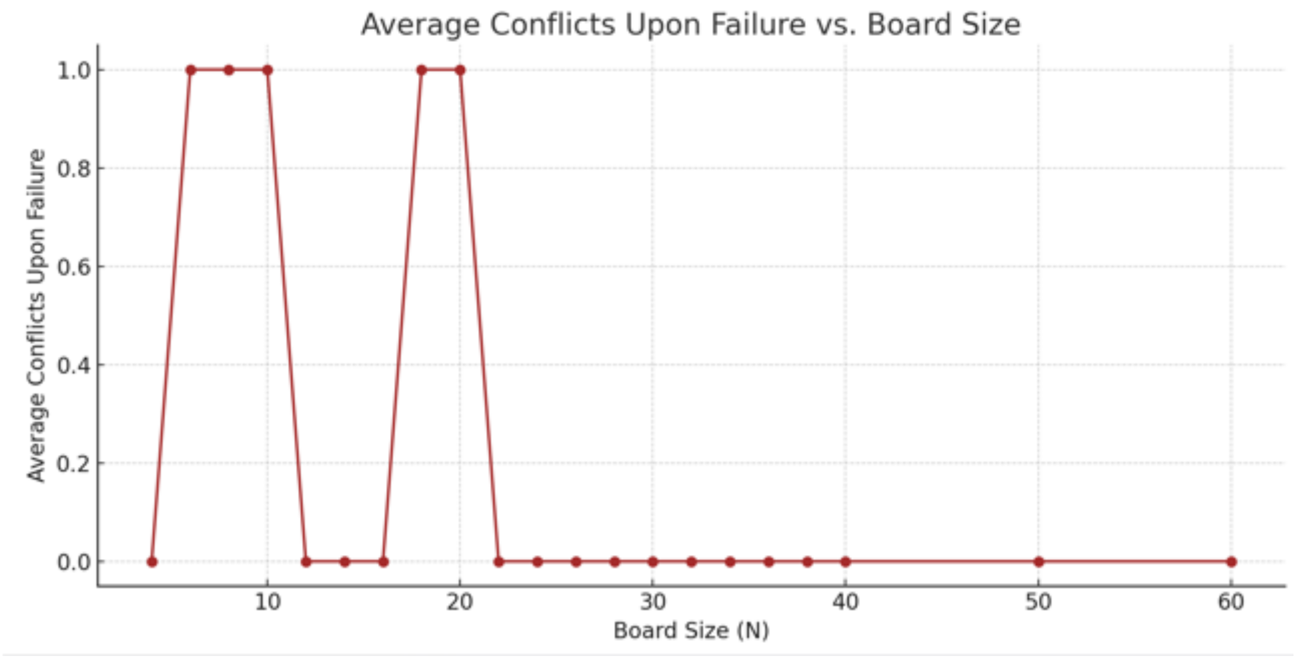
Average and Median Steps vs. Board Size

The trends in "Average and Median Steps vs. Board Size" plot down below provide insights into the algorithm's operational dynamics. Both average and median steps show increasing trends as board size grows, which is expected given the increase in potential conflicts. Notably, the median steps remain consistently lower than the average, suggesting that while the algorithm may occasionally take significantly longer to find a solution (influencing the average), it generally converges in fewer steps. This spread between the average and median indicates variability in the algorithm's performance, which could be attributed to the initial random placements of queens affecting the difficulty of resolving conflicts.



Average Conflicts Upon Failure vs. Board Size

The "Average Conflicts Upon Failure vs. Board Size" plot down below is particularly revealing. For most board sizes, the average conflicts upon failure are zero, indicating that when the algorithm fails to find a solution within the step limit, it generally leaves behind minimal conflicts. However, spikes in conflicts at board sizes 10 and 20 suggest that these configurations may present unique challenges that hinder the algorithm's ability to efficiently resolve conflicts. These peaks may align with the observed dips in the success rate, further emphasizing the difficulties at these sizes.



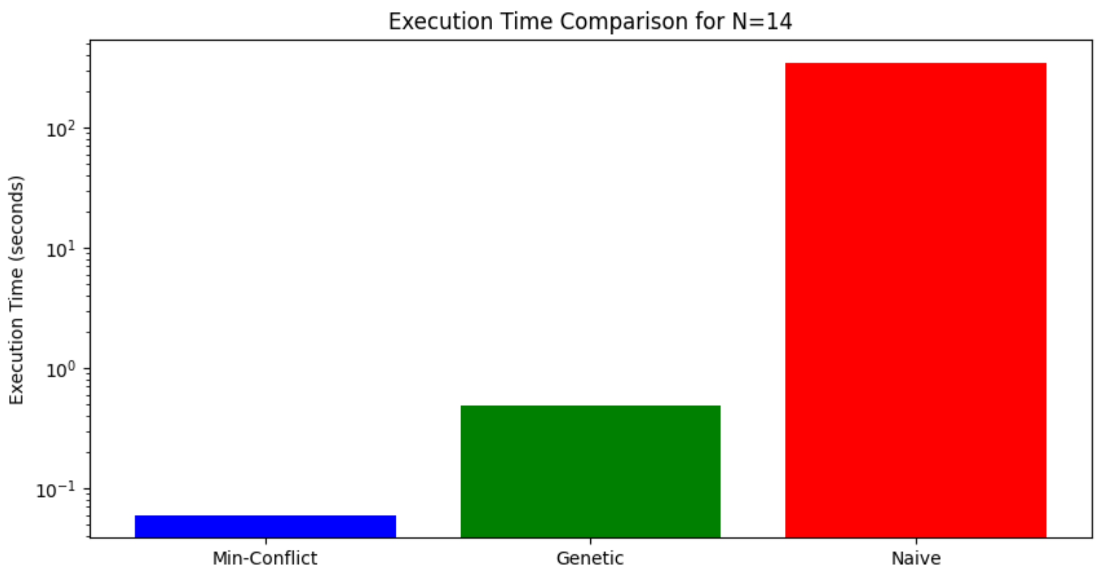
N	Success Rate	Average Steps	Median Steps	Running Time	Average Conflicts Upon Failure	Average Steps to Best Upon Failure
4	1	5.352941176	5	0.000330673	0	0
6	0.705882353	28.08333333	27.5	0.004087249	1	3.2
8	0.9311135	28.3125	21	0.00916785	1	15
10	0.701326	22	13	0.012696107	1	8.6
12	1	41.47058824	40	0.037736584	0	0
14	1	47	35	0.060876159	0	0
16	1	64.35294118	68	0.111026834	0	0
18	0.941176471	51.4375	47	0.119218633	1	52
20	0.925176	50.875	40	0.164763421	1	40
22	1	61.70588235	56	0.25479126	0	0
24	1	83.29411765	69	0.429395143	0	0
26	1	63.35294118	51	0.409130896	0	0
28	1	70.58823529	75	0.562379627	0	0
30	1	80.05882353	63	0.767745817	0	0
32	1	66.35294118	60	0.769950937	0	0
34	1	70.05882353	58	0.969338922	0	0
36	1	69.23529412	62	1.141772873	0	0
38	1	86.82352941	71	1.818229227	0	0
40	1	75.41176471	66	1.955538848	0	0
50	1	86.82352941	77	4.218598562	0	0
60	1	95.70588235	81	15.12624738	0	0

Result Conclusions

From the analysis, several conclusions can be drawn:

1. **Scalability:** The Min-Conflict algorithm demonstrates good scalability up to a certain threshold (board size of around 50), after which its performance drops significantly.
2. **Efficiency:** For most board sizes, the algorithm efficiently reaches solutions with high success rates, as evidenced by the low number of steps and minimal conflicts upon failure.
3. **Reliability:** The algorithm is generally reliable, with success rates maintaining above 90% across a broad range of board sizes, despite occasional failures at specific points.
4. **Challenges at Specific Sizes:** Board sizes around 10 and 20 appear particularly problematic, suggesting that certain configurations may inherently complicate the conflict resolution process.

Overall, the Min-Conflict algorithm proves to be a robust method for solving the N-Queens problem up to a moderate scale. Future work could explore modifications to initial queen placements or conflict resolution strategies to enhance performance at larger scales and address specific difficulties at challenging board sizes.



Summary

The N-Queens problem, a classic example in combinatorial optimization and artificial intelligence, challenges us to place N queens on an N×N chessboard such that no two queens threaten each other. The complexity of the problem increases exponentially with N, making it a significant testbed for various AI algorithms focused on constraint satisfaction.

In our project, we explored two primary computational approaches to solve the N-Queens problem: the Min-Conflict Heuristic and the Genetic Algorithm. These were selected over other methods like naive brute force, planning, and reinforcement learning approaches due to their suitability for handling the problem's constraints efficiently.

Min-Conflict Heuristic

The Min-Conflict algorithm is a heuristic method that iteratively minimizes conflicts by moving a queen to a position where it has fewest conflicts until no conflicts remain or a preset number of steps is reached. Our implementation showed promising results, particularly efficient for large N values, demonstrating the practical utility of heuristic search methods in solving constraint satisfaction problems.

Genetic Algorithm

The Genetic Algorithm, inspired by principles of evolution and natural selection, uses a population of solutions to evolve towards an optimal arrangement. By applying genetic operations such as fitness-based parent selection, crossover, and mutation, this algorithm effectively explores the solution space. It particularly excels in scalability and maintaining solution quality with increasing N, highlighting its quality in exploring large search spaces.

Results and Evaluation

Our results indicate that both algorithms are highly effective but exhibit different strengths depending on the scenario. The Min-Conflict Heuristic excels in rapid convergence to a solution with fewer computational resources for smaller to moderate-sized boards. In contrast, the Genetic Algorithm stands out in its ability to handle larger problem sizes due to its evolutionary search mechanisms that effectively explore more extensive search spaces.

Critique and Potential Improvements

While our approaches are effective, they are not without limitations. The Min-Conflict Heuristic, for example, can struggle with certain board configurations that may lead to local minima. The Genetic Algorithm, while powerful, requires careful tuning of parameters like mutation rate and population size to prevent premature convergence or excessive computational overhead.

Moreover, relaxing some of the inherent assumptions such as fixed board size or incremental changes in queen position could open up new avenues for algorithmic enhancements. Exploring adaptive heuristic or meta-heuristic approaches could also provide more dynamic solutions to the N-Queens problem across varied problem sizes and configurations.

In conclusion, our project demonstrates that while the Min-Conflict and Genetic algorithms are potent tools for tackling the N-Queens problem, there remains significant scope for enhancing these approaches. Future work will aim to refine these algorithms, explore alternative methods, and potentially develop hybrid approaches that combine the strengths of multiple strategies to address the complexities of the N-Queens problem more comprehensively.