N-Queen Constraint Satisfaction Problem
CS7IS2 Project (2021-2022)

Garvit Vijai(20303251), Rohan Girotra(20304712), Saubhagya Sharma(21350083), Manu Prasannakumar(20305536)

vijaig@tcd.ie, girotrar@tcd.ie, ssharma1@tcd.ie, prasannm@tcd.ie

Abstract. We focussed on Constraint Satisfaction Problems(CSPs) and various algorithms to solve them. We used three different artificial intelligence algorithms namely: backtracking algorithm, hill-climbing algorithm, and genetic algorithm to solve the N-Queens problem, which is a type of constraint satisfaction problem. Each algorithm goes through examining various arrangements of the queens until the solution is found. The performance of all three algorithms is measured against a baseline algorithm(a naive approach). We study the performance, speed, and nature of problem-solving of each of these algorithms.

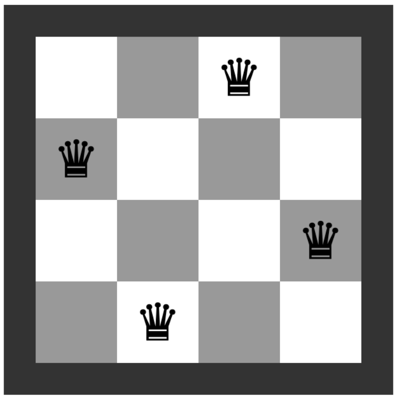
1 Introduction

We addressed Constraint Satisfaction Problems(CSPs) in our project. Constraint satisfaction is the process of finding a solution through a set of constraints that impose conditions that the variables must satisfy. The technique used in solving CSP depends on the kind of constraints being considered.

A CSP consists of a set of variables X, a set of domain D, and a set of constraints C. The Values assigned to variables in X must be from Domain in D. A constraint restricts the possible values of a subset of variables from X. The objective is to assign a value to each variable such that all constraints are satisfied.

We have chosen to implement the N-Queens CSP using three different algorithms. The goal of N-Queens is to arrange N-queens on a N x N chessboard so that no queen attacks another by being in the same column, row, or diagonal. In modern times, this problem is often used in the explanation of multiple programming techniques.

The figure, Fig.1 shows the positions of 4 queens (4-Queens) on a 4x4 chessboard.



We have implemented the following three algorithms for solving the N-Queens problem:

1. **Backtracking** - A backtracking algorithm is a problem-solving algorithm that finds all possible solutions to the problem. It incrementally builds candidates to the solutions and abandons a candidate as soon as it determines that the candidate will not lead to a valid solution and then moves on to build the next solution.

2. **Hill Climbing** - Hill Climbing is a heuristic local search algorithm. It is dependent on the heuristic function which ranks all possible alternatives at a particular branching step based on the available information. It helps the algorithm select the best route out of possible routes.

3. **Genetic Algorithm** - Genetic algorithm is a heuristic search algorithm. This algorithm reflects the process of natural selection where the fittest ones are selected for reproduction. The genetic algorithm gives optimal solutions based on the nature of the fitness function and the structure of the algorithm used.

The recording of the presentation can be viewed at:
https://drive.google.com/file/d/10PC6cbv6Tmqa8s7mdFxPemgHd-rAkoS3/view

2 Related Work

Number of works have been done in developing different approaches to solving the N-Queen problems and works are still being carried out. Researchers have not only found interesting solutions but also made comparisons and evaluated the performances of different types of problem-solving algorithms in this context.

In 2004, Adrijan Bozinovski and StevoBozinovski wrote an article that gave an insight into the space complexity of a backtracking algorithm for N-queens pattern generation. The paper presents a method that facilitates the observation of space constraints of backtracking algorithms while carrying out partial solutions. An example is provided for establishing the method with the well-known N-queens problem. [1] Another such research was conducted by Ahmed S. Farhan, Wadah Z. Tareq and Fouad H. Awad. [2] The authors suggested the use of a genetic algorithm for solving the N-Queen problem. They found a total of over 92 solutions by applying genetic algorithms for 8-queen problems. This representation had a dimension of 8, which reduced the empty cells, which was actually late comparatively. One array held 8 queens for one solution, and accordingly, the algorithm was applied, and a fitness calculation was accomplished. In Genetic algorithm, each solution is presented as a chromosome in an individual. They are estimated with the use of a function known as a fitness function which represents the accuracy of each solution in the individual. It is based on Darwin's theory of the 'survival of the fittest'. Prior to that, The Genetic Algorithm was applied by K. D. Crawford to solve n-Queen solutions [3].

In 2012, Vishal Kesri, Vaibhav Kesari, and Prashant Ku. Pattnik set out for a unique way to solve the n-queens problem. They discovered a pattern that provided a solution sans iteration. Their solution to the N-queen was found by carrying out rotation on the chessboard. They performed 90, 180, 270-degree clockwise rotations or anti-clockwise rotations, and 180-degree backward rotations than 90, 180, 270-degree clockwise rotations or anti-clockwise rotations. [4]

In 2009, Salabat Khan and his team proposed a solution for the n-Queen problem based on Ant Colony Optimization (ACO). It is a meta-heuristic based on the intelligent behaviour of ants. Ant behaviour acts as an allegory for the comportment of an agent within ACO. Dorigo developed the first ant algorithm. Ants prefer paths with high concentrations of pheromones to reach a food source. This solution involves altering the basic ACO model, adding some constraints, and changing some equations described above. The cores of this solution consist of "creation of search space" & "computation of heuristic value". They concluded that the ACO can provide better solutions to combinatorial optimization problems in short amounts of time, with a note that they would make an effort to utilize it for some other problems in the time to come. [5]

Two years before the research by Khan and the team, Ivica Martinjak and Marin Golub made comparisons of the heuristic algorithms for the N-Queen problem. Their study presents metaheuristics for algorithm simulated tempering, tabu search, and genetic algorithms to solve the n-queen problem. Test results are produced, and the upper bound complexity is established. The efficiency of algorithms is compared, and their achievements are reviewed. By decreasing the complexity of fitness functions to O(1), problems with large dimensions were resolved. Heuristic algorithms are capable to find multiple solutions for a given number of queens (only in the case of genetic algorithms, n*10). Furthermore, genetic algorithms perform better than simulated annealing algorithms. [6]

In 2014 Vikas Thada and Shivali Dhaka analyzed the performance of the N-Queen problem using Backtracking and genetic algorithms. Fundamentally, both approaches are used to solve the N-Queen problem. The general method takes days, months, and years to solve as N increases. The two methodologies are compared in the paper for a particular value of N. One the account, that the genetic algorithm is random, instead of measuring the time taken in obtaining all possible solutions for a given number of N, the time taken in procurement, for instance, count number of solutions is found out first using genetic algorithm, then the time taken in finding the same number of solutions that is count using backtracking is measured. It was found that backtracking techniques result in a faster execution time when N is set at an early value for a fixed number of solutions. Considering the genetic algorithm's random nature, not all solutions are possible with it, which provides only fairly accurate solutions. However, for the greater values of N, it was found that it provided improved values. [7] Another paper by M. Bozikovic and M. Golub in 2002 showed that though they were envisioned as heuristic methods for solving problems with "more value" and "lesser value" solutions, genetic algorithms showed to be adept at solving combinatorial problems with simple "yes" or "no" answers, as in the binary results. Moreover, experiments showed AI could find different solutions for a certain queen's count. [8]

3 Problem Definition and Algorithm

3.1 Problem Definition

The N-Queen problem is about placing N queens on an NxN board such that no two queens threaten each other. Two queens are said to threaten each other if they are in the same row, column or diagonals. Fig. 1 shows the solution of N-Queen CSP when N=4. None of the 4 queens is threatening another queen.

3.2 Algorithms

This section provides a detailed explanation of the three algorithms used to solve the N-Queen constraint satisfaction problem. Each algorithm ensures that no two queens are in the same row, column, or diagonal. Each algorithm has its own pros and cons and finding the balance between trade-offs and the optimal solution is the key to finding the best solution for the problem statement.

3.2.1 Backtracking

Backtracking is an algorithm technique wherein we find all possible solutions to a given problem. It incrementally builds each of the possible solutions until it determines that a prospective solution will not lead to a valid solution, in which case it abandons that solution and moves on to find the next solution.

The backtracking algorithm solves the problem in the following steps:

1. Start with the leftmost column on the board.
2. Visit all rows in the current column one by one.
   - 2.1. If a queen can be placed in a particular row, then mark the particular square(row, column) as part of the solution and recursively check if placing the queen in the square leads to a solution.
   - 2.2. If placing the queen in the square (row, column) leads to the solution, return True else unmark this square(row, column) and go to the next row and try step 3.1
3. If all rows are visited, print the board and return.
4. Move to the next column and again follow step 2.

3.2.2 Hill Climbing

The Hill Climbing algorithm is a local search algorithm that continuously moves in the direction of increasing value to find the best solution to the problem. It terminates when no neighbour has a higher heuristic value. Hill Climbing algorithm follows a greedy approach as at every step, it looks at a good immediate neighbour state and not beyond that.

The hill-climbing algorithm solves the problem in the following steps:

1. Initialise the chessboard with a random configuration.
2. Scan through all possible neighbours of the current state and move to the neighbour with the lowest heuristic value. If no such neighbour exists but there exists a neighbour with equal heuristic value then move to any random neighbour. A neighbour of the state is defined as the board configuration that differs from the current board configuration with respect to the position of only a single queen.
3. The state found after the local search is either the local optimum or the global optimum.
4. Output the state and return.

3.2.3 Genetic Algorithm

The Genetic Algorithm is a heuristic search algorithm and is based on the process of natural selection where the fittest ones are selected for reproduction.

The genetic algorithm solves the problem in the following steps:

1. Create a chromosome representation of the problem by representing solutions to the problem as a list of length n where each index in the list represents the row of a queen in a column.
2. The set of solutions is initialised as a set of randomly generated chromosomes (solutions).
3. A fitness function (pair of non-attacking queens) can calculate the fitness score of an arrangement. The fitness score will be the total number of non-attacking queen pairs in an arrangement.
4. The maximum number of non-attacking queens (maximum fitness score) is calculated using the formula n*(n-1)/2 by applying the combinations' formula for n queens.
5. The fitness function is used to calculate the fitness score of all solutions (chromosomes) in the solution set and compared to check if there are any solutions with the maximum fitness score (maximum number of non-attacking queens).
6. If there are no solutions in the solution set with maximum fitness score, a new set of solutions is generated using a generate function.
7. The generate function calculates the probability of all solutions as the fitness score divided by the maximum fitness score. Then, two parents with high probability values are selected randomly from the solution set and a child is created using the crossover. Crossover merges two parents at a random index to create a new solution. The child is appended to the new solution set. The same process is repeated for the length of the solution set to generate a new solution set. Based on alter probability, the one value of the child is altered(mutated) randomly.

8. New solution sets are generated until the solution set has a solution with the maximum fitness score and the algorithm returns that solution.

3.2.4 Baseline Algorithm

A brute force approach is used as a baseline. The idea is to find all possible configurations of the `NxN` chessboard with `N` queens and check which configuration satisfies the constraint. The algorithm works as follows:

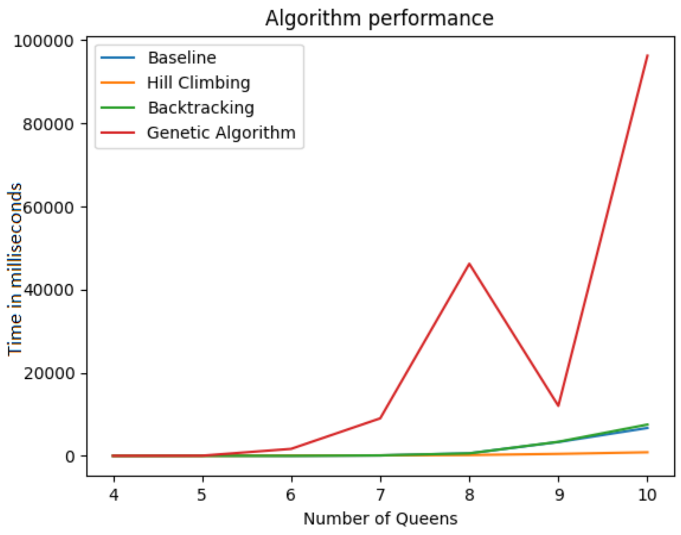
1. Generate a configuration with `N` queens.
2. If the queens don't attack each other in this configuration then print the configuration.
3. Repeat from step 1 until all configurations are explored.

4 Experimental Results

4.1 Methodology

This solution performance is measured through the time complexity of each algorithm i.e. the time in milliseconds for the execution of the respective algorithm. We have taken seven values and compared all model execution times in milliseconds. This was calculated using the *datetime* library of python where we recorded a start time before the execution of the algorithm, and an end time just after the execution and calculated the difference of both times in milliseconds.

4.2 Results



The result is presented through a line chart where the y-axis represents the time taken in milliseconds and the x-axis represents the number of queens on a chessboard. While different colours represent the different algorithms as shown in the above plot. The below table shows the execution time of different algorithms in milliseconds for the number of queens ranging from 4 to 10.

Number of Queens	Baseline (ms)	Hill Climbing(ms)	Backtracking(ms)	Genetic(ms)
4	11.983	15.999	8.012	24.009
5	40.006	20.005	35.985	52.008
6	24.024	79.994	20.001	1692
7	105.001	119.977	118.995	9036.503
8	611.005	205.003	609.018	46236.587
9	3333.916	484.001	3396.001	12032.808
10	6715.003	868.999	7540.859	96300.6

4.3 Discussion

The above plot and table are taken from the same data, they have data for queens 4 to 10. For queens greater than 7 Hill climbing is the best algorithm while for queens between 4 to 7 backtracking is performing best. While baseline is also performing fairly well from the backtracking.
The genetic algorithm is the worst performing algorithm because, with the increase in the number of queens, the number of random mutations and cross breedings of high probability parents necessary for an optimal solution increases exponentially, and hence the algorithm might have to take a large number of generations of cross-breeding to arrive at the solution with the best fitness score. The genetic algorithm might perform better in some runs even with a large number of queens because of the randomness involved in mutations and cross-breeding, but the probability of that decreases with an increase in the number of queens.

5 Conclusions

A comparative analysis based on the performance with respect to the time taken was conducted on three different algorithmic approaches to the N queen problem. The

algorithms used for this study are Backtracking, Hill Climbing, and Genetic algorithm.
A brute force approach was implemented to set a baseline for the study. The analysis revealed that Hill Climbing has the best performance among the four approaches. Backtracking has comparable performance to that of the baseline brute force algorithm, whereas the Genetic algorithm has the worst performance and is performing worse than the baseline algorithm.
The performance of the genetic algorithm degrades almost exponentially with the increase in the number of queens in the problem. The genetic algorithm might be useful for other optimization problems but fails to give better results than the brute force approach for the n queen problem. So, the best approach in terms of performance for solving the N queen problem is the Hill Climbing algorithm.
Since there was a limitation of the hardware, it was not possible to investigate the higher value of the input, N. Since the baseline is a brute force approach, it may perform worse than the other three algorithms for large values of N.

References

1. A. Bozinovski and S. Bozinovski, "N-queens pattern generation: an insight into space complexity of a backtracking algorithm. ISICT.," *ISICT,* 2004.
2. A. T. W. &. A. F. Farhan, "Solving N Queen Problem using Genetic Algorithm," *International Journal of Computer Applications,* vol. 122, pp. 11-14, 2015.
3. K. D. Crawford and R. L. Wainwr, "Applying Genetic Algorithms to Outlier Detection," in *Symposium on Applied Computing 1995*, New York, 1995.
4. V. Kesri, V. Kesari and P. Pattnaik, "International Journal of Computer Applications," *An Unique Solution for N queen Problem,* vol. 43, no. 12, pp. 1-6, 2012.
5. S. Khan, M. Bilal, M. Sharif, M. Sajid and R. Baig, "Solution of n-Queen problem using ACO," in *IEEE 13th International Multitopic Conference*, Islamabad, 2009.
6. I. Martinjak and M. Golub, "Comparison of Heuristic Algorithms for the N-Queen Problem," *29th International Conference on Information Technology Interfaces,* pp. 759-764, 2007
7. V. Thada and S. Dhaka, "Performance Analysis of N-Queen Problem using Backtracking and Genetic Algorithm Techniques," *International Journal of Computer Applications,* vol. 102, no. 7, pp. 26-29, 2014.
8. M. Bozikovic, M. Golub and L. Budin, "Solving n-Queen problem using global parallel genetic algorithm," *The IEEE Region 8 EUROCON 2003,* vol. 2, no. 1, pp. 104-107, 2003.
9. Jacobson, Sheldon & Yücesan, Enver. (2004). Analyzing the Performance of Generalized Hill Climbing Algorithms. J Heuristics. 10. 387-405