UNIVERSITY OF SCIENCE - VNUHCM

Faculty of Information Technology

Introduction to Artificial Intelligence

REPORT

Name: Ngô Phước Tài – Student's ID: 21127419
Class: 21CLC07
Lecturers: Mrs. Nguyễn Ngọc Thảo

Mr. Nguyễn Trần Duy Minh

Mr. Lê Ngọc Thành

Mr. Nguyễn Hải Đăng

Lab01: N-queens problem

Table of Contents:

I. Overview

The N-Queens problem is a classic puzzle in computer science and mathematics. It involves placing N queens on an N×N chessboard in such a way that no two queens threaten each other. In other words, no two queens should share the same row, column, or diagonal.

The problem gets its name from the chess piece "queen," which can move horizontally, vertically, and diagonally across the board. The objective is to find a placement of queens that satisfies the given constraints.

The N-Queens problem is often used as an example in algorithm design and backtracking. Various algorithms and techniques can be applied to solve it efficiently. One common approach is to use backtracking, which involves systematically placing queens on the board and backtracking whenever a conflict arises. This allows for an exploration of all possible configurations until a valid solution is found.

This report explores the utilization of the 3 algorithms including UCS, A*, and Genetic to address the N-Queens problem, which involves placing N queens on an N×N chessboard without any queen threatening another.

In this Lab, I have finished 3 algorithms:

UCS algorithm

A* algorithm

Genetic algorithm

In the main program, the user can input the number of queens, then select the algorithm to run and check the average running time and consuming memory.

I have used these support libraries:

‘heapq’ is used for working with a priority queue.

‘random’ is used for generating random numbers and creating the initial state.

‘time’ and ‘trace malloc’ is used for calculating the function’s running time and consuming memory.

II. UCS Algorithm

The UCS algorithm is an uninformed search algorithm that explores the search space based on the cost to reach each state. It expands the nodes with the lowest cost first and guarantees to find the optimal solution when the cost function is non-decreasing.

In UCS, the cost function represents the cost to reach a particular state. In the N-Queens problem, a common approach is to assign a cost of 1 for each queen placement. The cost increases by 1 with each additional state in the Frontier

The steps involved in solving the N-Queens problem using the UCS algorithm are as follows:

Define the start state with a random position on the board.

Define the goal state where all N queens are placed without conflicts.

Initialize the Frontier as a priority queue with the initial state and its priority.

While Frontier is not empty:

> Remove the state with the lowest cost from the Frontier.
>
> Generate successor states by placing a queen in the next row of invalid positions.
>
> Calculate the priority of each successor state based on the cost to reach the current state.
>
> Add the successor states to the Frontier.
>
> If a goal state (all queens placed without conflicts) is found, terminate the algorithm and return the solution.
>
> If the Frontier becomes empty without finding a goal state, conclude that there is no solution to the N-Queens problem.

III. A* Algorithm

The A* algorithm is an informed search algorithm that combines the cost to reach a state with a heuristic function to estimate the remaining cost to reach the goal state. It first explores the states with the lowest total cost (cost to reach the current state + heuristic estimate).

The heuristic function is a crucial component of the A* algorithm. It provides an estimate of the remaining cost to reach the goal state from a given state. In the N-Queens problem, a common heuristic is the number of pairs of conflicting queens on the board. It gives a lower bound on the number of moves required to reach the goal state.

The steps involved in solving the N-Queens problem using the A* algorithm are as follows:

Define the start state with a random position on the board.

Define the goal state where all N queens are placed without conflicts.

Initialize the Frontier as a priority queue with the initial state and its priority calculate by summing its heuristic and path cost.

While the Frontier is not empty:

> Remove the state with the lowest priority (lowest cost + heuristic value) from the Frontier.
>
> Generate successor states by placing a queen in the next row of invalid positions.
>
> Calculate the priority of each successor state based on the cost to reach the current state and its heuristic value.
>
> Add the successor states to the Frontier.
>
> If a goal state (all queens placed without conflicts) is found, terminate the algorithm and return the solution.

If the Frontier becomes empty without finding a goal state, conclude that there is no solution to the N-Queens problem.

IV. Genetic Algorithm

The Genetic Algorithm is an optimization algorithm inspired by the process of natural selection and evolution. It uses the concepts of selection, crossover, and mutation to iteratively evolve a population toward a better solution.

The heuristic function evaluates the quality or suitability of a chromosome. In the N-Queens problem, the heuristic function assesses the number of conflicts between queens, where a lower number of conflicts indicates a better heuristic.

The steps involved in solving the N-Queens problem using the Genetic Algorithm are as follows:

Define the start state with a random position on the board.

Define the goal state where all N queens are placed without conflicts.

Initialize a population of chromosomes of a size 2*n randomly.

Evaluate the heuristic of each chromosome using the heuristic function.

Select parents from the population by random 1 individual has high conflict and 1 individual has low conflict.

Create children through crossover by combining genetic information from the selected parents.

Perform mutation on the offspring to introduce small random changes with a high mutate rate.

Create a new population with the same number of individuals as the original population, and these new individuals are the individuals that have a small conflict.

Repeat the process until find an individual has a heuristic number is 0.

V. Conclusion

V.1. UCS Algorithm

The UCS algorithm systematically solves the N-Queens problem by exploring states based on the cost to reach each state. By expanding nodes with the lowest cost, the algorithm can find an optimal solution when the cost function is non-decreasing.

While the UCS algorithm is effective for solving the N-Queens problem, it is important to note that the problem has an exponential number of solutions, resulting in high time complexity. Therefore, more efficient algorithms like backtracking or constraint satisfaction algorithms are typically preferred for larger values of N.

Overall, the UCS algorithm provides a valuable tool for solving the N-Queens problem and can be extended to other problems where the cost function plays a crucial role in guiding the search.

V.2. A* Algorithm

The A* algorithm provides an efficient approach to solving the N-Queens problem by considering both the cost to reach a state and a heuristic estimate of the remaining cost. By exploring states with the lowest total cost, the algorithm can find an optimal solution or determine if there is no solution. The heuristic function plays a crucial role in guiding the search toward the goal state.

While the A* algorithm is effective for solving the N-Queens problem, it is important to note that it still has exponential time complexity in the worst case. Thus, for large N, other techniques like constraint satisfaction algorithms or parallelization may be more suitable for efficient solutions.

Overall, the A* algorithm provides a powerful tool for solving the N-Queens problem and can be extended to other combinatorial problems as well.

V.3. Genetic Algorithm

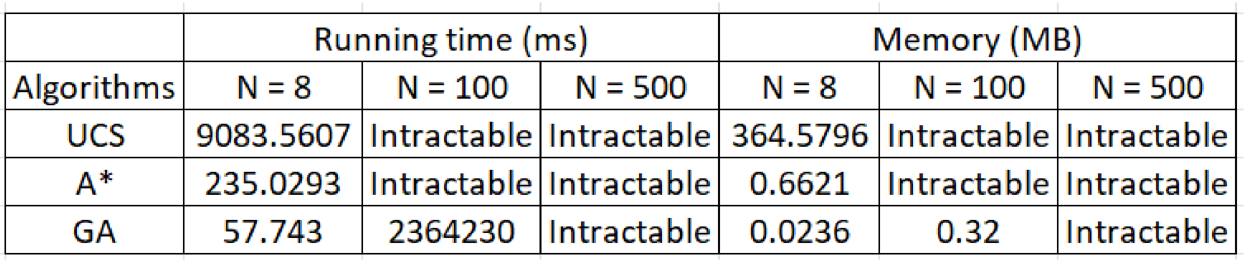
The Genetic Algorithm provides a powerful approach to solving the N-Queens problem by leveraging concepts of selection, crossover, and mutation to iteratively improve the population of solutions. It can efficiently explore the search space and find near-optimal or optimal solutions.

The Genetic Algorithm is particularly useful when dealing with large N values, where traditional search algorithms like backtracking or exhaustive search become computationally expensive. It offers a flexible and scalable approach to solving combinatorial optimization problems.

However, it is essential to consider the parameters of the Genetic Algorithm, such as population size, selection methods, crossover techniques, and mutation rate. These parameters can significantly impact the algorithm's performance and the quality of the obtained solutions.

Overall, the Genetic Algorithm provides an effective and versatile approach to solving the N-Queens problem, and its principles can be extended to other combinatorial optimization problems as well.

V.4. Comparison Table



	Running time (ms)			Memory (MB)		
Algorithms	N = 8	N = 100	N = 500	N = 8	N = 100	N = 500
UCS	9083.5607	Intractable	Intractable	364.5796	Intractable	Intractable
A*	235.0293	Intractable	Intractable	0.6621	Intractable	Intractable
GA	57.743	2364230	Intractable	0.0236	0.32	Intractable

Each algorithm, UCS, A*, and GA, offers a unique approach to solving the N-Queens problem. UCS explores the search space based on the cost to reach each state, A* combines cost and heuristic estimates to guide the search, and GA uses selection, crossover, and mutation to evolve a population of solutions.

The choice of algorithm depends on factors such as the problem size, optimality requirements, and available computational resources. UCS guarantees optimality but may be computationally expensive for larger problem instances. A* provides an informed search approach with a heuristic, while GA offers a stochastic optimization method suitable for combinatorial problems.

In conclusion, the N-Queens problem can be effectively tackled using UCS, A*, and Genetic algorithms. Understanding the characteristics and trade-offs of these algorithms helps in selecting the most appropriate approach for solving the N-Queens problem based on the specific requirements and constraints of the problem at hand.

In my assignment, UCS cannot run the N-Queens problem efficiently, it sometimes cannot find the result because of the complex start state. However, we can make this algorithm efficient by programming it like a BFS algorithm. Genetic algorithms are sometimes unstable because it has to perform the random function continuously, sometimes it will change low-conflict states leading to skipping results.

From my perspective, I think the A* algorithm should be used to solve the N - queen problem. Because A* solves the problem in a not-too-long time and doesn't take too much memory.

VI. References

Introduction to AI's course lecture

A* Search Algorithm - GeeksforGeeks

Uniform-Cost Search (Dijkstra for large Graphs) - GeeksforGeeks

What Is the Genetic Algorithm? - MATLAB & Simulink (mathworks.com)

HẾT