

**CSE 3111 / CSE 3213**

**ARTIFICIAL INTELLIGENCE**

**FALL 2023**

***Programming Assignments Report***

*İkram Celal KESKİN – 200316059*

*Musa Sina ERTUĞRUL – 200316011*

*Mert KARDAS – 200316045*

*Submission Date: 28 December 2023*

# Development Environment

We developed and tested the code in a Python 3.x environment. The operating system used as Windows 10. The code was written and executed in Visual Studio Code, a popular IDE for Python development. The CPU properties include an AMD Ryzen 3 3250U with Radeon Graphics processor.

# Problem Formulation

* + 1. **State Specification:**

The state in the N-Queens problem is represented by the positions of N queens on an N×N chessboard. Each queen's position is specified by its row and column.

* + 1. **Initial State:**

The initial state is either provided by the user, randomly generated, or manually input. It consists of N queens placed on the chessboard.

* + 1. **Possible Actions:**

The possible actions represent moving a queen to a different row within its column. Each action is defined by specifying the queen's current column and the new row to which it is moved.

* + 1. **Transition Model:**

The transition model defines how actions lead to state changes. In this case, applying an action updates the position of a queen, resulting in a new state.

* + 1. **Goal Test:**

The goal test checks whether the current state is a solution where no two queens threaten each other. This is achieved by ensuring that there are no attacking pairs of queens (queens in the same row, column, or diagonal).

* + 1. **Path Cost:**

The path cost is uniform and represents the number of actions taken to reach the current state. Since each action involves moving one queen to a different row, the cost of each action is set to 1.

# Results

Please check results.txt, We only tested some of these because, program crashed cause of large iterations.

# Discussion

1. **Completeness:**
2. **Breadth-First Search (BFS):**

BFS is complete and finds a solution if one exists. It explores all possible states at each level before moving to the next level.

1. **Depth-First Search (DFS):**

DFS is not complete, as it might get stuck in infinite paths. However, if a solution exists within a reachable depth, DFS can find it.

1. **A Search:\***

A\* with an admissible and consistent heuristic is complete, ensuring optimality in finding the optimal solution with the minimum cost.

1. **Genetic Algorithm:**

Genetic algorithms are stochastic and might not guarantee completeness. The population might not converge to a solution in some cases.

1. **Hill Climbing:**

Hill climbing is not complete, as it may get stuck in local optima and fail to find the global optimal solution.

1. **Greedy Search:**

Greedy search is not complete, and it makes decisions based on immediate gains without considering the global picture.

1. **Optimality:**
2. **Breadth-First Search (BFS):**

BFS guarantees optimality as it explores all possible states at each level before moving to the next level.

1. **Depth-First Search (DFS):**

DFS does not guarantee optimality. It may find a solution quickly but might not be the optimal one.

1. **A Search:\***

A\* is optimal when using an admissible and consistent heuristic. It ensures finding the optimal solution with the minimum cost.

1. **Genetic Algorithm:**

Genetic algorithms are not guaranteed to find the optimal solution due to their stochastic nature.

1. **Hill Climbing:**

Hill climbing is not guaranteed to find the optimal solution as it might get stuck in local optimal.

1. **Greedy Search:**

Greedy search is not optimal as it makes locally optimal choices without considering the global context.

1. **Time and Space Complexity:**
2. **Breadth-First Search (BFS):**

BFS has high space complexity as it needs to store all nodes at each level. Time complexity is reasonable for smaller problem instances.

1. **Depth-First Search (DFS):**

DFS has low space complexity but might have high time complexity in certain scenarios due to exploring deep paths first.

1. **A\* Search:**

A\* has moderate space complexity, and its time complexity depends on the heuristic's quality. It can be efficient with a good heuristic.

1. **Genetic Algorithm:**

Genetic algorithms have high space complexity due to maintaining a population. Time complexity varies based on convergence speed.

1. **Hill Climbing:**

Hill climbing has low space complexity but may require a large number of iterations. Time complexity is dependent on the convergence speed.

1. **Greedy Search:**

Greedy search has low space complexity but lacks optimality. Time complexity can be efficient for certain problems.

1. **Depth-Limited Search:**
   * **Effect of Depth Limit in Depth-Limited Search:**

A depth limit in DFS restricts the search depth. If the depth limit is too small, the algorithm may miss optimal solutions. A balance is needed between depth and computational resources.

* + **Local Search vs. Traditional Search:**

Local search algorithms like hill climbing and greedy search are often faster for certain problems but may not guarantee optimal solutions. Traditional search algorithms like BFS and A\* are more thorough but might be slower for large problem spaces.

1. **Overall Observations:**
2. **Algorithm Choice:**

The choice of algorithm depends on the problem size, optimality requirements, and available computational resources.

1. **Heuristic Impact:**

The quality of the heuristic significantly affects the performance of A\* search. A well-designed heuristic improves both time and space efficiency.

1. **Stochastic Algorithms:**

Stochastic algorithms like genetic algorithms introduce randomness. They might find good solutions but lack guarantees of optimality.

1. **Local Search Limitations:**

Local search algorithms may struggle with global optimization, making them less suitable for certain problems.

1. **Trade-Offs:**

There is often a trade-off between completeness, optimality, and efficiency. The choice of algorithm should align with problem characteristics and requirements.