Ministry of higher education
and scientific research

University of El Manar

Faculty of Sciences of Tunis

وزارة التعليم العالي و البحث العلمي

جامعة تونس المنار

كلية العلوم بتونس

AI Lab Report

Comparative Analysis of State Graph Search Algorithms

Work developed by:

AROUA Rahma

BELHADJ Ghada

Supervised by:

Ms. DOGGAZ Narjes

Academic year : 2024/2025

Contents

1 Introduction **3**
1.1 General Presentation of the Project 3
1.2 Objectives of the Lab Assignment 3

2 Problem Description **3**
2.1 Definition of the Problem 3
2.2 Representation of the Problem as a State Graph 3

3 Description of the Algorithms **3**
3.1 Theoretical Explanation of Each Search Method 3
3.2 Pseudocode 4

4 Comparative Study **6**
4.1 Results Obtained 6
4.2 Summary Table 6

5 Conclusion **6**
5.1 Synthesis of Observations 6
5.2 Suggestions for Improvement 7

1 Introduction

1.1 General Presentation of the Project

This project aims to explore and compare three state search techniques: breadth-first search (BFS), depth-first search (DFS), and the A* algorithm. These methods are used to solve complex computational problems, such as games, optimization, or navigation. The objective is to analyze their efficiency in terms of execution time and explored nodes.

1.2 Objectives of the Lab Assignment

- Implement the three algorithms on the same problem.
- Compare their performances based on quantitative criteria (number of explored nodes and execution time).
- Identify the strengths and limitations of each method to better understand their respective application domains.

2 Problem Description

2.1 Definition of the Problem

The Traveling Salesman Problem (TSP) consists of finding the shortest path that allows a salesperson to visit a given list of cities, passing through each city only once, and returning to the starting point. This problem is widely used in combinatorial optimization and has applications in fields such as logistics and planning.

2.2 Representation of the Problem as a State Graph

- Cities are represented by the nodes of the graph.
- Routes connecting the cities are the edges of the graph, with each edge having a weight corresponding to the distance or travel cost between the two cities.
- The objective is to find a path in this graph that minimizes the sum of the weights while respecting the constraint of visiting each node only once.

3 Description of the Algorithms

3.1 Theoretical Explanation of Each Search Method

- **Breadth-First Search (BFS)**: Explores nodes level by level.

- **Depth-First Search (DFS)**: Explores a path to its end before backtracking.
- **A***: A heuristic algorithm that combines actual cost and estimated cost to find the optimal path.

3.2 Pseudocode

Algorithme 1 : Breadth-First Search (BFS)

```
Input : Initial state, Goal state
Output : Path to goal and cost, or failure
Initialize an open queue with the initial state;
Initialize a closed list for already explored states;
while the open queue is not empty do
    Extract the state at the head of the queue;
    if this state matches the goal then
        return the path and the cost;
    else
        foreach successor of the current state do
            if the successor is not in the closed list then
                Add the successor to the open queue;
                Mark the successor as explored by adding it to the closed list;
return "failure" if the queue is empty and no goal is found;
```

Algorithme 2 : Depth-First Search (DFS)

```
Input : Initial state, Goal state
Output : Path to goal and cost, or failure
Initialize an open stack with the initial state;
Initialize a closed list for already explored states;
while the open stack is not empty do
    Extract the top element of the stack;
    if this state matches the goal then
        return the path and the cost;
    else
        foreach successor of the current state (in reverse order) do
            if the successor is not in the closed list then
                Add the successor to the open stack;
                Mark the successor as explored by adding it to the closed list;
return "failure" if the stack is empty and no goal is found;
```

Algorithme 3 : A* Search Algorithm

```
Input : Initial state, Goal state
Output : Path to goal and cost, or failure
Initialize an open queue with the initial state and a cost f = g + h;
Initialize a closed list for already explored states;
while the open queue is not empty do
    Extract the state with the smallest f from the queue;
    if this state matches the goal then
        return the path and the cost;
    else
        foreach successor of the current state do
            Calculate the cost g (actual) and h (heuristic);
            if a better cost is found for the successor then
                Add the successor to the open queue;
return "failure" if the queue is empty and no goal is found;
```

4 Comparative Study

4.1 Results Obtained

- **Number of Explored Nodes:**
  - DFS: Explores deeply, often traversing several paths before finding a solution. The number of explored nodes depends on the graph's depth.
  - BFS: Explores all nodes level by level until the goal is reached. The number of explored nodes is often higher for deep solutions.
  - A*: By using a heuristic, it explores fewer unnecessary nodes and focuses on promising paths.
- **Execution Time for Each Method:**
  - DFS: Average time of 0.00001500 seconds.
  - BFS: Average time of 0.00000900 seconds.
  - A*: Average time of 0.00000490 seconds.

4.2 Summary Table

Algorithm	Execution Time (s)	Total Cost	Explored Nodes
DFS	0.0000157	35	10
BFS	0.0000019	35	13
A*	0.0000049	35	9

Table 1: Performance Comparison of Algorithms

5 Conclusion

5.1 Synthesis of Observations

- **DFS:**
  - Strengths: Simple to implement; works well for graphs where the solution is deep.
  - Weaknesses: Inefficient in memory and explores unnecessary paths.
- **BFS:**
  - Strengths: Guarantees the optimal solution for uniformly weighted graphs.
  - Weaknesses: Consumes more memory to store nodes at each level.

- **A*:**
  - Strengths: Efficient due to heuristic usage; explores fewer unnecessary nodes.
  - Weaknesses: Requires a well-defined heuristic to perform optimally.

5.2 Suggestions for Improvement

- **Memory Optimization:** Use algorithms like Iterative Deepening DFS to save memory in DFS.
- **Advanced Heuristic for A*:** Adapt a heuristic closer to the actual cost to improve performance.