**Analysis of Graph Search Algorithms**

**Depth-First Search, Breadth-First Search, Greedy\_BSF and A\* Search Algorithms in Maze Solving**

**Krishna Gopal Sharma**

**Student Number: B1055988**

**Artificial Intelligence ICA S3454618**

**Total Words:**

**1. Abstract**

This project investigates the performance of four search algorithms, the uninformed search algorithms i.e. {Breadth-First Search (BFS), Depth-First Search (DFS)} and informed searched algorithms i.e. {A\* Algorithm, and Greedy Best-First Search (Greedy BFS)} in solving a maze. The algorithms were tested under three scenarios, with goal positions at (1, 1), (29, 4), and (2, 45). I am evaluating the effectiveness of these search algorithms based on path length, exploration order, and computational efficiency. The findings provide insight into the advantages and disadvantages of each algorithm under different goal positions.

**2. Introduction**

Pathfinding and search algorithms are fundamental to Artificial Intelligence (AI), with applications spanning robotics, gaming, and optimization problems. This project focuses on implementing and comparing four search algorithms:

1. **Breadth-First Search (BFS):** An uninformed search algorithm exploring all neighbours at the current depth level before moving deeper.
2. **Depth-First Search (DFS):** A search technique exploring as far as possible along each branch before backtracking.
3. *A\* Algorithm:* An informed search combining cost from start to node (g(n)) and a heuristic estimate to the goal (h(n)).
4. **Greedy Best-First Search (Greedy BFS):** An informed search prioritizing exploration based solely on heuristic estimates (h(n)).

This report documents the methodology, implementation, results, and analysis of these algorithm’s performance across three different goal positions.

**2. Methodology**

**2.1 Problem Setup**

* The maze dimensions, structure and obstacles were predefined and loaded from a CSV file.
  + A table with numbers and letters

    Description automatically generatedDimensions: 30 x 50.
  + Structure: Cell (x, y)

Direction: E, W, N, S

Fig 1: Maze Structure

* The start position for each algorithm was the **bottom-right corner (30, 50)** of the maze.
* Three scenarios were defined with varying goal positions: **(1, 1), (29, 4), and (2, 45).**

**2.2 Implementation**

Each algorithm was implemented in Python using the pyamaze library. Each algorithm was implemented in Python, leveraging the pyamaze library for maze creation, visualization, and pathfinding. The library provides a flexible framework for defining maze dimensions, walls, and goal positions, making it suitable for comparative studies of search algorithms. The details of each implementation are outlined below: The uploaded files include the following implementations:

* **1.1 the\_BSF.py:** BFS implementation​.

A black background with green and blue text

Description automatically generated

A screen shot of a computer program

Description automatically generated

A computer screen shot of a program code

Description automatically generated

A screen shot of a computer code

Description automatically generated

A screen shot of a computer code

Description automatically generated

A screenshot of a computer program

Description automatically generated

A black background with green text

Description automatically generated

* **1.2 the\_DFS.py:** DFS implementation​.
* **1.3 the\_Astar.py:** In A\* there is heuristic values which inform them about the direction of the goal from starting position, each cell has same heuristic value i.e. 1 and the below code is representing to implement the heuristic (Manhattan distance) from current cell to goal position.

A screen shot of a computer

Description automatically generated

This is the only difference while writing the code just need to add this heuristic data in the A\* to improve it.

A computer screen shot of text

Description automatically generated

In this code for each neighbour, I need to calculate total cost f(n) which is equal to the sum of current cost g(n) and heuristic cost h(n).

**f(n) = g(n) + h(n)**

If use is heuristic value to help in calculating the search efficiency when we move towards the goal. For eg: if the goal is at left direction w.r.t the current position then heuristic value give point to move at left side towards goal at current position. Same with right, up or down direction. The heuristic value will tell the A\* where it should move to get the goal ASAP. A\* does not have exact direction of a goal but have estimated it and this estimation help to move the algorithm towards goal path.

**1.4 the\_Greedy\_BFS.py:** *Same code as A\* just greedy used only heuristic value to find path towards goal.*

A computer screen shot of text

Description automatically generated

Here, **f(n) = h(n)**

This is a basic reason why the greedy\_BFS is not an optimal algorithm but faster. As compare to A\* the greedy\_BFS is only focus on moving forward

**2.3 Performance Metrics**

The algorithms were evaluated based on:

1. **Path Length:** The number of steps from the start to the goal.
2. **Exploration Length:** The number of nodes explored before reaching the goal.
3. **Execution Time:** The time taken to compute the path (measured programmatically).

**2.4 Experimental Design**

Each algorithm was executed in three scenarios. The results were recorded for all metrics and analyzed to determine the algorithm best suited for dynamic goal conditions.

**3. Results and Discussion**

**3.1 Scenario 1: Goal at (1, 11)**

*(Put the detailed results, including path length, exploration length, and execution time for BFS, DFS, A*, and Greedy BFS.)\*

**3.2 Scenario 2: Goal at (45, 3)**

*(Put the detailed results, including path length, exploration length, and execution time for BFS, DFS, A*, and Greedy BFS.)\*

**3.3 Scenario 3: Goal at (29, 3)**

*(Put the detailed results, including path length, exploration length, and execution time for BFS, DFS, A*, and Greedy BFS.)\*

**3.4 Algorithm Comparison**

* **Breadth-First Search (BFS):**
  + Performed optimally in finding the shortest path due to its exhaustive exploration.
  + High computational cost as all nodes at a given depth are explored.
  + *(Put the BFS-specific comparisons here.)*
* **Depth-First Search (DFS):**
  + Depth-oriented exploration occasionally led to suboptimal paths.
  + Lower memory usage compared to BFS but higher path variance.
  + *(Put the DFS-specific comparisons here.)*
* *A Algorithm:*\*
  + Consistently achieved optimal paths with efficient exploration.
  + Performance depended on the heuristic accuracy (Manhattan distance used).
  + *(Put the A* comparisons here.)\*
* **Greedy Best-First Search (Greedy BFS):**
  + Faster than A\* but prone to getting stuck in local minima.
  + Explored fewer nodes but sometimes resulted in longer paths.
  + *(Put the Greedy BFS comparisons here.)*

**4. Achievements and Insights**

* **Adaptability:** A\* emerged as the most robust algorithm across all scenarios due to its balanced consideration of path cost and heuristic guidance.
* **Trade-offs:** BFS guarantees shortest paths but at a higher computational cost. DFS offers memory efficiency but risks suboptimal paths.
* **Dynamic Goals:** *(Put insights about how goal position affects each algorithm's performance.)*

**5. Conclusion and Future Work**

This study highlights the varying strengths and weaknesses of BFS, DFS, A\*, and Greedy BFS in dynamic goal-based scenarios. While A\* provided a balanced performance, each algorithm's suitability depends on the application's requirements.

**Future Work**

1. **Dynamic Obstacles:** Explore algorithm behavior when obstacles appear or change dynamically.
2. **3D Mazes:** Extend comparisons to three-dimensional mazes.
3. **Hybrid Approaches:** Investigate combinations of algorithms to leverage their strengths.

**6. Appendix**

* **Code Files:** The complete implementation is included in the files attached to this report.
* **Execution Environment:** *(Put the details of your software, libraries, and hardware here.)*
* **Data Sources:** *(Put information about the CSV files or maze generation details here.)*