*Exploring Heuristic Efficiency in Pathfinding Algorithms:*

*A Comparative Study of A\* and Greedy\_BFS*

**Table of Contents**

1. **Abstract**
2. **Introduction**
3. **Methodology**
   1. **Algorithm Overview**
   2. **Heuristic Functions**
   3. **Directional Weights**
   4. **Maze Design**
4. **Implementation Details**
   1. **A\* Search Algorithm**
   2. **Greedy Best-First Search Algorithm**
5. **Experimental Results**
   1. **A\* Search Results**
   2. **Greedy Best-First Search Results**
   3. **Comparative Analysis**
6. **Discussion and Insights**
7. **Future Work**
8. **Conclusion**
9. **References**

**1. Abstract**

In this project, I learn the logic behind Heuristic methods by a comparative analysis and visualization of two widely used heuristic-based pathfinding algorithms: A\* and Greedy Best-First Search (Greedy BFS). Both algorithms employ heuristics to guide their search towards a goal, but they differ significantly in their approach to evaluating nodes. in solving maze navigation problems. I study 3 heuristics methods i.e. **Manhattan, Euclidean, and Chebyshev**. They are applied to compare the algorithms’ performance in terms of path length, search length, and execution time. The report also analyses the impact of directional weights on both algorithms. My study through this project, also includes the understanding of impact of changing in weight values on path\_length and exploration\_length of each search algorithms under different heuristic methods. The project includes interactive visualizations created using Python libraries such as Pyamaze and Tkinter, alongside performance analysis using Matplotlib and Pandas to compare search lengths, path lengths, and exploration footprints. The results provide insights into the strengths and limitations of each algorithm in different search scenarios, contributing to a deeper understanding of heuristic-based search strategies

**2. Introduction**

Pathfinding algorithms are essential for navigations, robotic movements, and AI game development. A\* Search and Greedy BFS are two popular algorithms for finding the path in a maze. Both use heuristics to guide their search, but their approaches differ significantly. Where A\* algorithm focus on optimal path, on the other hand, Greedy\_BFS focus on searching the path at earliest. The differences between these two algorithms are the key reasons to choose A\* and Greedy\_BFS. This project explores these differences by implementing the algorithms with multiple heuristics and directional weights.

The objectives of this study include:

1. Comparing the performance of A\* Search and Greedy BFS using Manhattan, Euclidean, and Chebyshev heuristics.
2. Evaluating the effect of directional weights on pathfinding.
3. Visualizing exploration and pathfinding processes using the pyamaze library.

Ultimately, this work contributes to the study of heuristic-based pathfinding by providing a comprehensive visual and analytical comparison of **A**\* and **Greedy BFS**. By working on this project, I able to understand how different heuristics affect the performance of these algorithms. It also insight me about the logic behind the heuristics and weights over algorithms and some answered my many questions as well.

**3. Methodology**

**3.1. Algorithms Overview**

**Let’s me introduce with the algorithms I had used in this project.**

**3.1.1. A\* Search Algorithm***:* A\* Search is an informed search algorithm that combines the advantages of Dijkstra's Algorithm and Greedy Best-First Search, as A\* combines the cost to reach a node (“G-cost”) and the estimated cost to the goal (“H-cost”) to determine the best path. This ensures optimal and complete pathfinding.

Total cost of search in A\* search is:

T(c) = G(c) + H(c)

**3.1.2. Greedy Best-First Search Algorithm:** Greedy BFS relies solely on the heuristic (H-cost) to prioritize nodes, often leading to faster but suboptimal solutions.

Greedy Best-First Search prioritizes nodes based on their heuristic cost; it aims to reach the goal as quickly as possible. While computationally simpler, it may not always find the optimal path but find path in fast time.

Total cost of search in Greedy\_BFS is:

T(c) = H(c)

Why only these two algorithms?

These 2 algorithms are the most widely used algorithms across many domains. In addition, the heuristic approach of these algorithms are totally different which help to understand the heuristic approach better.

**3.2. Heuristic Functions**

For this project I had used 3 heuristic functions. The heuristic function helps in estimating the distance from start(current) to goal. The calculation of this distance is different in all 3 functions. Let’s discuss it little deeper and understand each of them.

* + 1. **Manhattan Distance:** Manhattan Distance measures how far the two points are when you only move along the horizontal and vertical directions (no diagonal moves).

**Formula:**

Manhattan = {x1 − x2} + {y1 − y2}

Here, (x1,y1) and (x2,y2) are the coordinates of the two points. These absolute values ensure that we don't have negative distances.

**Example:**

If Point A is at (1, 2) and Point B is at (4, 6), the Manhattan distance is:

Manhattan = {1 – 4} + {2 – 6} = 3+4 = 7

**So, the distance is 7 units.**

A black background with white text

Description automatically generated

Code 1: Code for the Manhattan

* + 1. **Euclidean Distance:** It follows the straight-line distance between two points, like the shortest path between them if you could fly or walk directly. Euclidean distance calculates the "straight-line" or "as-the-crow-flies" distance between two points.

**Formula:**

Euclidean = √{( x2 - x1 )^2 + ( y2 - y1 )^2}.

**Example:**

If Point A is at (1, 2) and Point B is at (4, 6), the Euclidean distance is:

Euclidean = √{ (4−1)^2 + (6−2)^2 }= √{ (9 + 16)} = √25

**So, the distance is 5 units.**

A black background with colorful text

Description automatically generated

Code 2: Code for the Euclidean:

* + 1. **Chebyshev Distance:** This is useful when you're allowed to move in all directions (up, down, left, right, and diagonally), and it counts how far you need to go to reach the other point, taking diagonal moves into account.

Chebyshev distance considers diagonal movement, so it measures the maximum number of steps you would need to move in any one direction to get from one point to another.

**Formula:**

Chebyshev = max{(x1−x2),(y1−y2)}

Here, it picks the larger of the horizontal or vertical distance, assuming you can move diagonally.

**Example:**

If Point A is at (1, 2) and Point B is at (4, 6), the Chebyshev distance is:

Chebyshev = max {(1−4),(2−6)} = max( 3 , 4 ) = 4 (is max)

**So, the distance is 4 units.**

**A black background with white text

Description automatically generated**

Code 3: Code for the Euclidean:

**Note:** “As-the-crow-flies” in Euclidean means, we are calculating the shortest distance between two points using Pythagorean theorem, on the other hand, Chebyshev distance gives the count of step needed to reach at goal. These two are identical in theory but different in calculation.

|  |
| --- |
| *Weight* |
| *(10, -10, -10, 10)* |
| *(-10, 10, 10, -10)* |
| *(0, 10, 10, 0)* |
| *(0, -10, -10, 0)* |
| *(0, 0, 0, 0)* |
| *(10, 10, 10, 10)* |
| *(-10, 0, 0, -10)* |
| *(10, 0, 0,10)* |

**3.3. Directional Weights**

Directional weights simulated variable terrain by assigning the costs to move in different directions (North, East, South, and West) and this cost will influencing while path selection. I had used 8 weight combinations to check and understand its impact on both algorithm while using all 3 heuristic functions.

0,0,0,0 is a neutral weight that means it have no effective at any heuristic function. 10 here is representing that if my agent moves in that direction then the cost of one step in that direction is 10., and -10 is vice versa.

In simple reinforcement term, the (10) is like a penalty for an agent to move in that direction, and (-10) is like a reward for an agent to move in that direction. [Visualization will be done in experimental setup: Section3.4.4]

**3.4. Experimental Setup**

**3.4.1. The Maze Creation:** A maze of predefined dimensions is created using the pyamaze library.

A black background with white text

Description automatically generated

Code 4: Create maze.py

In this project I had used 2 mazes.

**Maze\_1 for Heuristic\_Function.py:** This maze is 50 x 120 cells with loopPercent = 35. This maze is created to use when A\* and Greedy\_BFS algorithms are experimenting with heuristic functions i.e. Manhattan, Euclidean and Chebyshev function. To visualize the path length, it is important to have a little longer length so that it will be clearly visible the impact of heuristic functions over algorithms, therefore 50 x120 maze is required. For make it more complex, I used pattern = ‘vertical’ so that maze contain more vertical lines.

A black and white maze

Description automatically generated

**Fig: Maze\_1 50 x 120**

**Maze\_2 for Directional\_Weight.py:** On the other hand, this maze is only 20 x 80 with loopPercent = 30. To make algorithm process faster this maze is created having width only 20 and for proper visualization of path the length of the maze is 80. In this project I had shown only 8 directional weights but for my testing and learning I had done more than 200 test to see the effect of weight change over algorithms and how the path and search length count change according to the respective weight.

A black maze with white lines

Description automatically generated

**Fig: Maze\_2 20 x 80**

3.4.2. Implementation of Heuristic Functions on Search Algorithms:

So, 3 heuristic function are tested on 2 algorithms.

3.4.3 Study the impact of Directional Weight on Search Algorithms:

3.4.4 Visualization of 8 experimental Directional Weight:

Path Length: Number of cells in the shortest path.

Search Length: Number of cells explored during the search.

Execution Time: Time taken to compute the solution.

**Results**

**Table 1: A\* Search Algorithm Performance**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Heuristic** | **Goal Position** | **Path Length** | **Search Length** | **Execution Time (s)** |
| Manhattan | (1,1) | 1075 | 9077 | 0.0224 |
| Euclidean | (1,1) | 1075 | 8863 | 0.0138 |
| Chebyshev | (1,1) | 1075 | 8895 | 0.0164 |

**Table 2: Greedy BFS Algorithm Performance**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Heuristic | Goal Position | Path Length | Search Length | Execution Time (s) |
| Manhattan | (1,1) | 1837 | 5354 | 0.0019 |
| Euclidean | (1,1) | 1841 | 4846 | 0.0160 |
| Chebyshev | (1,1) | 1735 | 4377 | 0.0170 |

**Table 3: A\* Search Algorithm with Directional Weights**

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Direction | Weight | Manhattan Path Length | Manhattan Search Length | Euclidean Path Length | Euclidean Search Length | Chebyshev Path Length | Chebyshev Search Length |
| N,E,S,W | (10, -10, -10, 10) | 1187 | 8999 | 1173 | 8999 | 1173 | 8999 |
| N,E,S,W | (-10, 10, 10, -10) | 1187 | 5277 | 1795 | 5277 | 1795 | 5277 |
| N,E,S,W | (0, 10, 10, 0) | 1075 | 9090 | 1075 | 9090 | 1075 | 9090 |

**Table 4: Greedy BFS Algorithm with Directional Weights**

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Direction | Weight | Manhattan Path Length | Manhattan Search Length | Euclidean Path Length | Euclidean Search Length | Chebyshev Path Length | Chebyshev Search Length |
| N,E,S,W | (10, -10, -10, 10) | 1837 | 4377 | 1841 | 4377 | 1735 | 4377 |
| N,E,S,W | (-10, 10, 10, -10) | 1837 | 4377 | 1841 | 4377 | 1735 | 4377 |

**Discussion**

**A\* vs. Greedy BFS**

1. **Path Length:** A\* consistently finds the optimal path across all heuristics. Greedy BFS sacrifices path quality for speed, often resulting in longer paths.
2. **Search Length:** A\* explores more cells due to its consideration of G-cost, while Greedy BFS prioritizes fewer cells, focusing on the heuristic.
3. **Execution Time:** Greedy BFS is faster in execution, particularly with the Manhattan heuristic.

**Heuristic Impact**

1. **Manhattan:** Suitable for grids without diagonal movement; provides predictable results.
2. **Euclidean:** Performs well in mazes with diagonal paths.
3. **Chebyshev:** Best for scenarios allowing diagonal and straight movements.

**Directional Weights**

1. Adding weights affects the search behavior by prioritizing certain directions.
2. A\* adjusts better to directional weights, maintaining optimal paths.
3. Greedy BFS’s reliance on heuristics diminishes the impact of weights, leading to similar results across all configurations.

**Conclusion**

This project demonstrates the strengths and weaknesses of A\* and Greedy BFS in pathfinding. A\* is ideal for scenarios requiring optimal paths, while Greedy BFS excels in time-sensitive applications. The choice of heuristic significantly impacts performance, with Chebyshev being the most versatile. Directional weights provide additional flexibility in tailoring algorithms for specific environments.

Future work could explore hybrid approaches combining A\* and Greedy BFS or integrating machine learning techniques to dynamically adjust heuristics and weights based on the maze structure.

**References**

1. Russell, S., & Norvig, P. (2020). Artificial Intelligence: A Modern Approach.
2. Stentz, A. (1995). The focused D\* algorithm for real-time replanning.
3. pyamaze Documentation. Available at: <https://github.com/MAnwarHussain/python-amaze>.