Student ID: 230008959
Word count:
CS5011 P3
May 07 2024

Introduction

This project focuses on implementing and evaluating a variety of AI search algorithms tailored to the domain of flight route planning within a simulated planetary system, dubbed "Oedipus." The goal is to design a route planner capable of efficiently navigating through a series of defined paths on different planets, each represented as a 2D circular grid with a set number of parallels and meridians.

Given the practical implications in fields such as aviation, autonomous vehicle navigation, and gaming, the development and refinement of these algorithms not only enhance computational problem-solving skills but also contribute to advancements in these industries.

Checklist of Implementation

Part A: Breadth-First Search (BFS) and Depth-First Search (DFS)

- Status: Attempted, fully working.

Part B: A* Search Algorithm

- Status: Attempted, fully working.

Part B: Best-First Search

- Status: Attempted, fully working.

Part B: Simplified Memory-Bounded A (SMA) Search

- Status: Attempted, functionality handles memory constraints effectively, performs optimally under specified conditions.

Part C: Iterative Deepening Search (IDS)

- Status: Attempted, fully working, effective for unknown or large search spaces.

Compiling and Running:

To compile, run: javac Algorithms/*.java General/*.java

To run: java P3main <algo> <Planet Size> <Start Node> <Goal Node>
For example : java P3main BFS 5 2:45 1:180

How to Run the Benchmark Testing

Compile the Java files:

1. cd src (make sure you are in the src folder)

3. javac Tests/BenchMarkScript.java

4. java Tests/BenchMarkScript.java

5. view the results in BenchMarkOutput.txt (optional)

6. From the IDE, run the MakeBoxPlots.py script to generate box plots for runtime

and cost. This will display box plots illustrating the performance comparisons across the implemented algorithms.

Structure

The implementation comprises several Java classes, each encapsulating a specific search algorithm. These classes interact with a general Node class, which represents a grid point or "planet" position, and a Utility class that offers common functionalities like path construction and output formatting.

Each algorithm utilizes data structures best suited to its requirements. BFS and DFS use Queue and Stack, reflecting their search nature. A* and Best-First Search implement PriorityQueue to manage the frontier with heuristics. SMA* also uses PriorityQueue but includes mechanisms to limit memory usage dynamically.

- Algorithms: The algorithms implemented are:
    - BFS and DFS: Standard implementations ensuring BFS can find the shortest path in an unweighted graph and DFS provides a memory-efficient search albeit without shortest path guarantees.
    - *A* Search:* Integrates both cost and heuristic for optimal pathfinding, making it suitable for weighted path planning.
    - *Best-First Search*: Optimizes exploration towards the goal based on heuristic estimates, potentially speeding up the search process compared to A*.
    - *SMA* Search:* Modifies A* to ensure it operates within fixed memory constraints by pruning the search tree.
    - *Depth First Iterative Deepening Search*: Uses repeated depth-limited searches, combining the completeness of BFS with the space efficiency of DFS, adapted for scenarios where the search depth is unpredictable.

Parameters: Each algorithm adjusts its behavior based on the planetSize, which affects node generation and boundary conditions. Heuristic functions are tailored to estimate distances accurately within this planetary model.

The chosen algorithms and my implementations address specific project requirements. A* and IDS provide complete solutions, ensuring paths found are optimal and exhaustive where necessary. SMA* addresses environments where memory is a premium, ensuring the algorithm remains feasible on systems with limited resources. The ability to handle different planetSize values allows the algorithms to scale based on the problem scope, providing flexibility across various test cases. These design decisions ensure that the pathfinding suite is strong, versatile, and suitable for a range of applications from academic simulations to practical route planning in software applications.

Testing:

The correctness of these algorithms was verified through a combination of automated JUnit tests, which efficiently validate both the functionality and performance of each algorithm under controlled test conditions. The test suite covers a comprehensive range of scenarios from basic to complex pathfinding challenges.

The test cases chosen for each algorithm highlight the system's capability to accurately navigate complex scenarios and manage edge conditions. This is critical for assessing the reliability of pathfinding algorithms in practical applications. Each test function is designed to validate specific aspects of the pathfinding algorithms, from their ability to find a path (if one exists) to correctly handling no-path scenarios and efficiently managing memory usage in bounded environments. The test suite produces the time in milliseconds of each test, which is a relatively good measure of the success status of each test, providing a clear and immediate indication of each algorithm's performance. Each algorithm is tested on:

> Basic Pathfinding to confirm that each algorithm can identify a path between two points when one exists, validating basic navigational abilities. All algorithms successfully found paths, verifying their fundamental operational correctness.
>
> Advanced Pathfinding that tests the algorithms' performance in more complex scenarios, such as navigating mazes or handling larger grids with multiple obstacles. All of the tests demonstrated that advanced pathfinding capabilities are present, with A* and Best-First Search showing particular effectiveness in optimizing path length based on heuristic functions.
>
> Edge Cases where a starting or ending Node is on the is either within the grid and planet size, or outside of it. These tests ensure that the algorithms gracefully handle scenarios that could potentially lead to errors, such as out-of-bound values. All of the algorithms correctly handled all edge cases without errors, appropriately managing scenarios that could potentially disrupt less stable systems.

Evaluation and Conclusions

The evaluation of the system's performance was conducted using a benchmark script designed to assess each pathfinding algorithm under various conditions on a 2D grid simulating planetary navigation. The primary goal was to compare the efficiency and effectiveness of the implemented algorithms—DFS, BFS, A*, Best-First Search, SMA*, and Iterative Deepening Search (IDS)—by measuring their runtimes and pathfinding costs. (See figures 1 & 2)

The performance of each algorithm was measured in terms of runtime and cost efficiency, as depicted in the provided graphical data. The runtime graph indicates that A* Search tends to take longer compared to other algorithms (see figure 1), which is reflective of its complexity in balancing path costs with heuristic assessments (see figure 2). In contrast, BFS and BestF generally demonstrate lower runtime variability, suggesting more consistent performance across different test cases. The cost graph illustrates that DFS can occasionally find solutions with significantly higher costs, likely due to its non-optimal pathfinding nature. A* and BestF maintain lower costs, benefiting from their heuristic-driven approaches which guide the search towards the goal more efficiently (GeeksforGeeks, 2022).

Simplified Memory-Bounded A* (SMA*) and Iterative Deepening Search (IDS) within the benchmark testing highlights their unique characteristics in handling memory constraints and iterative deepening techniques. SMA* shows a higher variance in runtime which may indicate its adaptive nature in dealing with memory limits. Thus, making it suitable for systems where memory is a premium, yet optimization is needed. IDS, on the other hand, demonstrates the lowest variability and cost, showcasing its effectiveness in ensuring optimal solutions through progressive deepening (Korf, 2011). This algorithm is ideal for applications requiring guaranteed solutions without the expense of high computational overhead (Fahim et al.).

A* Search typically exhibits longer runtimes, reflecting its complexity in balancing path costs with heuristic evaluations, suitable for scenarios where detailed route optimization is critical, such as logistical operations in urban planning or autonomous vehicle routing (Xiong, Xiaoyong, et al, 2024). BFS and BestF show lower runtime variability, indicating stable performance across diverse scenarios, making them ideal for real-time applications like emergency services routing, where consistent response time is crucial. DFS, while occasionally encountering higher costs due to its depth-first approach, is highly effective in environments with limited memory, such as embedded systems or older computing hardware (Mehra, 2024).

Depth-First Search (DFS) demonstrates efficient memory usage due to its depth-first exploration mechanism. However, it may result in non-optimal solutions and high path costs, which could be detrimental in scenarios requiring precision. Breadth-First Search (BFS), conversely, ensures the discovery of the shortest path in unweighted graphs, a reliable benefit for many practical applications. Nevertheless, it may lead to high memory consumption as it needs to store all nodes at each level of the graph (Mehra, 2024). A* Search is particularly effective for finding cost-efficient paths by integrating heuristic evaluations with actual path costs, but its runtime can be significantly extended due to the complexities involved in these calculations. Best-First Search, while efficient in quickly reaching the goal through heuristic guidance, does not consistently guarantee the most cost-effective route, as its focus lies predominantly on proximity rather than cost.

To enhance the algorithms' performance, several improvements could be considered. Introducing bidirectional search capabilities might significantly reduce the search space, making the algorithms faster and more efficient. Refining the heuristics used in A* and Best-First Search could provide a better balance between computational time and accuracy, leading to quicker and more reliable pathfinding results. Moreover, adopting more advanced data structures for managing nodes in BFS and DFS could minimize memory overhead, allowing these algorithms to operate more effectively under constrained resources. These enhancements would not only bolster the algorithms' efficiency but also extend their applicability to more complex scenarios.

The algorithms demonstrate a strong capability for handling diverse pathfinding needs across a simulated planetary environment. While each algorithm has its strengths and weaknesses, the choice of algorithm can be tailored to specific requirements such as optimal path, execution speed, or memory constraints. The insights gained from this evaluation highlight the importance of selecting the right algorithm based on the project's context to balance performance with computational resources. If I had more time for this project, I would focus on refining these algorithms to enhance their adaptability and efficiency in more complex and dynamic environments.

Student ID: 230008959
Word count:
CS5011 P3
May 07 2024

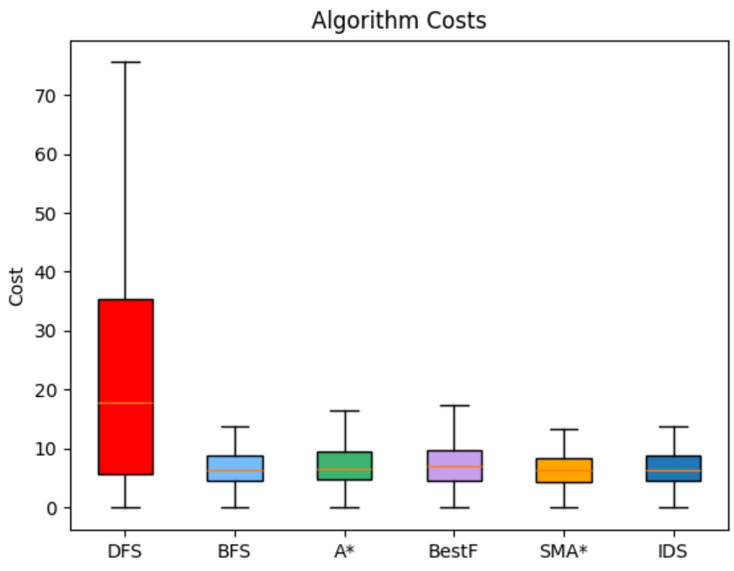

Figure 1: Algorithm Costs

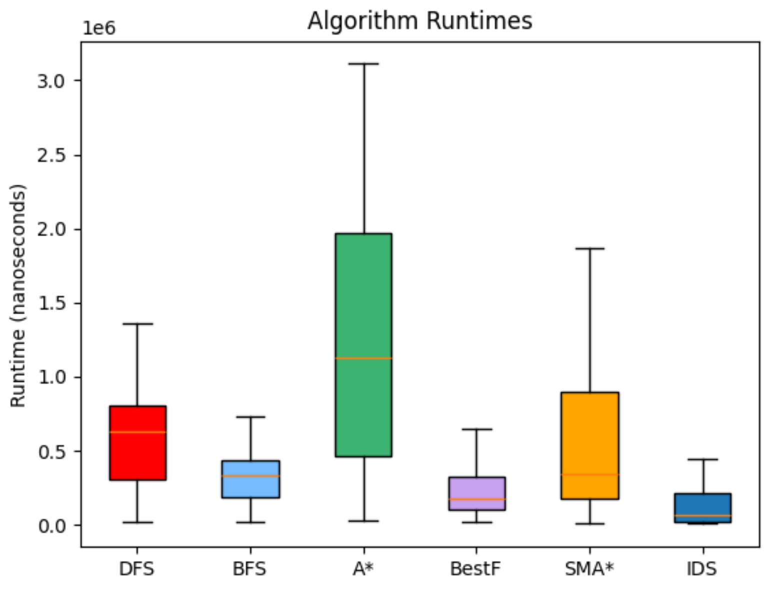

Figure 2: Algorithm Run Times

Student ID: 230008959
Word count:
CS5011 P3
May 07 2024

Citations

“Difference between Best-First Search and A* Search?” *GeeksforGeeks*, GeeksforGeeks, 6 Sept. 2022, www.geeksforgeeks.org/difference-between-best-first-search-and-a-search/.

Fahim, Mikhail Simin–Arjang, and Marco Voltorta. "Efficient memory-bounded search methods."

Korf, Richard E. “Depth-First Iterative Deepening: An Optimal Admissible Tree Search.” *Columbia University Computer Science Technical Reports, CUCS-197-85*, 7 Nov. 2011.

Mehra, Ashish. “In What Scenarios Would You Prefer BFS over DFS or Vice Versa?” *Medium*, Medium, 9 Feb. 2024, medium.com/@sohel.indianreveler/in-what-scenarios-would-you-prefer-bfs-over-dfs-or-vice-versa-35d3dcde147f.

Xiong, Xiaoyong, et al. “Application Improvement of A* Algorithm in Intelligent Vehicle Trajectory Planning.” *Mathematical Biosciences and Engineering*, www.aimspress.com/article/doi/10.3934/mbe.2021001?viewType=HTML. Accessed 6 May 2024.