Search Algorithm Benchmarking Project Usage

Introduction

This project contains the implementation and benchmark comparisons among five searching algorithms: Breadth-First Search, Depth First Search, Iterative Deepening Depth First Search, Best-First Search (Greedy), and A* search. The algorithms are to solve the shortest path problem on a weighted, random generated and connected graph.

Information/Prerequisites

The entire project is written in the programming language of Python, with the included libraries of matplotlib and networkx. These libraries are used for data analysis and visualization.

How to Run the Benchmark

To run/execute the benchmarks, you must run the "main.py" file as it contains the functions to output the benchmark results

The Executing Command

Must execute the command in terminal "python3 main.py" in IDE.

Expected Output: What to expect

- A single run test is fixed on seed(42) in order to ensure correctness and an unchanging visualization where the "A_star_visual_path.png" image is saved that contains the visual path.
- There will be an execution of all five algorithms across three settings of "Easy", "Medium", and "Hard".
- There will be a final table that prints onto the console, showing the mean for the runtime, the nodes expanded, peak memory, and path cost all of which serves for the final report.

The search_algorithms.py file has all the cored functions for the search methods and the Euclidean heuristic.

Benchmarking Analysis

Methodology

This project was utilized in a collaborative development with both Generative AI and references for debugging and code adjustments to not only adhere to the project requirements but as a guide in developing the project overall. The primary use of AI was heavily focused on instrumentation and refinements.

- **Benchmarking Design & Calculations**: It gave helpful assistance in structuring the "run_benchmark" function to mainly handle the multiple algorithms and calculate the needed statistical outputs for analysis.

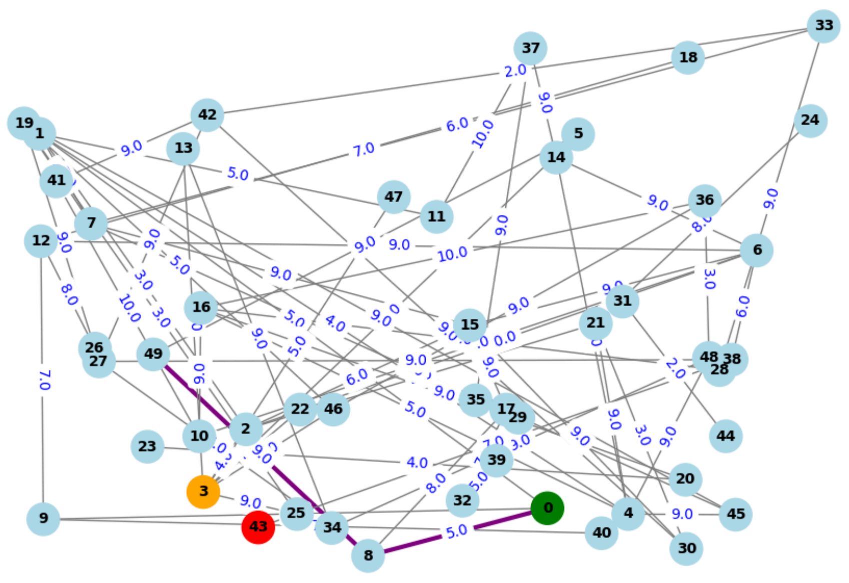
- **Debugging & Refinements**: It was greatly helpful in identifying the multitude of errors that were encountered in the execution of specifically the informed algorithms (Best-First and A* Search). An example was assisting in ensuring A* ability to maintain optimality by handling the path relaxation, and checking the Best-First correctly increments the node expansion counter.

- **Adhering for Requirements**: Incredibly helpful in describing and validating all five algorithms needed for the implementation of the right signatures and parameter requirements like passing the coordinates for heuristics and depth_limit for IDDFS.

Visualization Approach

The program was able to successfully implement an image like visualization (saved on A_star_visual_path.png) in order to confirm that the shortest path by A* was correctly constructed and displayed. However, the requirement for a frame-by-frame, interactive animation for the search process was not met due to significant complications and performance issues I've encountered while during development. There were constant delays as each frame needed large, evolving graph states and stack/queue contents which I believe created an overhead in the accuracy of the runtime metrics. With a limited amount of time and other priorities, I chose to focus on the benchmarking of runtime, memory usage, and optimality over the complications I faced with creating a working and accurate visualization animation.

A* Path Visualization (Cost: 14.00)



Setup Summary

The benchmark tests five searching algorithms across three different levels of complexity.

- Easy -> Has 20 nodes, branching factor is 2
- Medium -> Has 50 nodes, branching factor is 3
- Hard -> Has 100 nodes, branching factor is 4

Each setting runs five times (with the seeds from 100 to 104), this results to show the mean. The A* and Best-First informed searches that use the Euclidean Distance as the heuristic.

(Latest Execution of Benchmarking)

Algorithms	Runtime (ms)	Nodes Expanded	Peak Memory Usage (units)	Path Cost
Breadth First Search	0.04 ± 0.01	60.60 ± 32.24	126.20 ± 10.16	19.6

Depth First Search	0.06 ± 0.01	70.00 ± 23.42	173.20 ± 28.47	277.6
Iterative Deepening Depth First Search	0.14 ± 0.12	177.00 ± 161.96k	3.20 ± 0.84	19.6
Best-First Search	0.11 ± 0.06	50.40 ± 29.42	116.00 ± 55.86	42.8
A*	0.14 ± 0.04	68.60 ± 40.36`	93.60 ± 20.72	17.4

Analysis

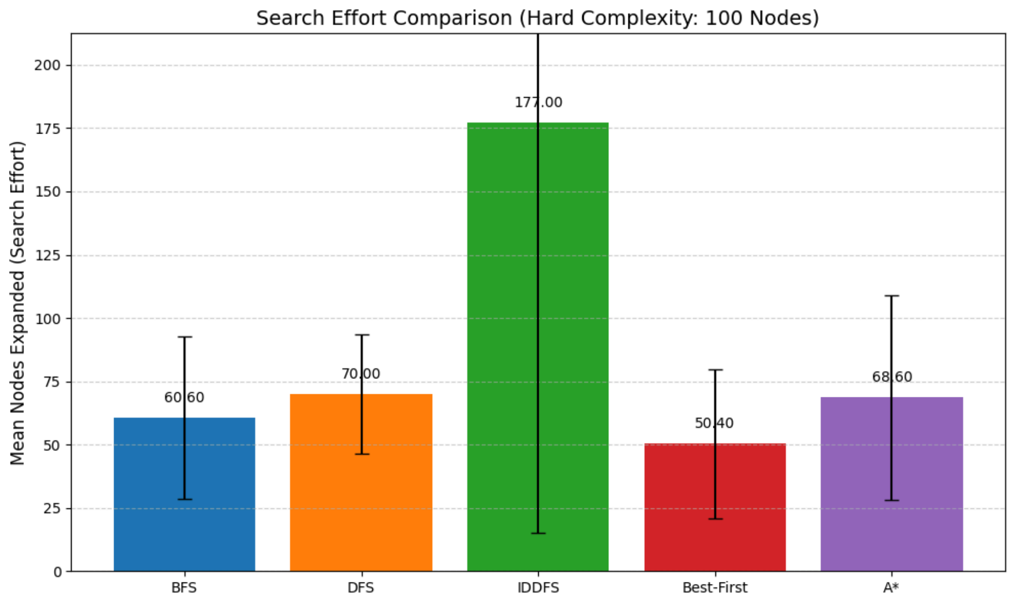
Optimality (Path Cost)

Based on which algorithm has the lowest path cost, A* is the winner for optimality with a path cost of 17.4 even when the settings are on hard. This essentially proves the theoretical approach of the A* algorithm in its ensureness of optimality where the heuristic is admissible. Algorithms that are uninformed like Breadth First Search (BFS) or Iterative Deepening Depth First Search (IDDFS) have a higher path cost than A* which certainly demonstrates A* dominance in finding the shortest-path optimality on weighted graphs.

DFS and Best-First Search are both proven to be non-optimal especially in the hard settings. With DFS having the highest path cost among the first algorithms shows the disadvantage of the algorithm that it tends to explore too deep down branches without finding the shortest path. Best-First Search, although reasonably faster in its runtime, has a higher path cost which demonstrates the tendency of algorithms like Best-First Search to be greedy.

Time & Memory Complexity

Among the five algorithms, IDDFS stands out with having the lowest memory usage being 3.20 units which means it is thoroughly memory efficient. The reason why IDDFS is particularly great in memory management is that it only stores nodes that are relevant to its search/exploration for the shortest-path. DFS has the highest memory usage with it being 173. 20 units, this could be due to its nature in using recursions that are stacked for bookkeeping in the take path it searches. Alongside DFS, BFS has the second largest memory usage of 126.20 units likely due to storing every node into the frontier at each level it explores.



Runtime and Node Expansion

By the result of the benchmark and the "Node Expansion Comparison (Hard Complexity" image, IDDFS has the highest nodes that were expanded with it having 177.00 nodes due to its effort of repetitive searches with every increase in the depth limit. Informed searching algorithms like Best-First Searches are far more efficient with its method of expanding fewer nodes due to its tendency of being greedy.

Conclusion

Achieving optimality on a weighted graph, A* reins as the superior algorithm. Though for memory usage, IDDFS stands to be an appreciated solution even though its runtime is reasonably longer among the five algorithms. For a more moderately quick and rationally better solution, Best-First Search could be the one to turn to, but it should be avoided if optimality is the requirement.

Sources:

- Best First Search : step by step approach | by shruti6 | Medium
- Hill Climbing Search vs. Best First Search | Baeldung on Computer Science
- The A* Algorithm: A Complete Guide | DataCamp
- Breadth First Search in Python (with Code) | BFS Algorithm | FavTutor
- Automate the Boring Stuff with Python

- Python Cheatsheet
- Depth Limited Search Implementation Using Python Programming - CodeSpeedy
- Iterative Deepening Search(IDS) or Iterative Deepening Depth First Search(IDDFS) - GeeksforGeeks