Mini AI Application Example

Shaima Abdulmajeed Alharbi

Table of Contents

.

Summary or Executive Summary

My findings indicate that Blind Search consistently outperformed Best First Search by providing shorter and more direct routes. This suggests that Blind Search is the recommended technique for route finding in the London map. Future work could involve exploring other algorithms, such as A* search, or Dijkstra's algorithm, and considering additional factors like real-time traffic data to further enhance route finding capabilities. Overall, this project provides valuable insights into the performance of different algorithms for route finding in maps and emphasizes the significance of algorithm selection for specific use cases.

Introduction

Efficient route finding plays a crucial role in navigation and routing systems, offering significant benefits in terms of time, fuel consumption, and resources utilization. In this project, I compare the performance of two route finding algorithms, Best First Search and Blind Search, on the London map using OpenStreetMap data.

The focus of my study is to determine which algorithm yields more optimal and efficient routes. I leverage the detailed information provided by OpenStreetMap data, which includes the road network and geographic features of London. By evaluating the performance of the algorithms and examining the routes generated, I aim to gain insights into the effectiveness of different algorithms for route finding.

It is important to note that my findings and recommendations are specific to the comparison of Best First Search and Blind Search algorithms in the context of the London map using OpenStreetMap data.

Report Body

Methods/AI techniques used

To tackle the problem, I utilized two AI techniques: Best First Search and Blind Search. Best First Search uses heuristics to guide the search towards more promising solutions. It prioritizes nodes that are estimated to be closer to the goal. On the other hand, Blind Search explores all possible solutions without any initial knowledge of the problem.

Understanding these AI techniques is crucial to understand how they Ire implemented in the route finding task. Best First Search makes decisions based on heuristic information, while Blind Search exhaustively evaluates all possible options.

Implementation of AI techniques

In this part, I'll cover the implementation details of the AI techniques, including any necessary data preprocessing, the programming environment used, relevant libraries, and the experimental process.

Before applying the AI techniques, I conducted data preprocessing. Specifically, I extracted and organized the relevant information from the OpenStreetMap data related to the London map. For the implementation, I opted for Python as my programming language. To aid us in managing the graphs, I relied on the popular NetworkX library.

To evaluate the performance of the techniques, I conducted experiments by running the Best First Search and Blind Search algorithms on the pre-processed data.
The experiments involved varying parameters, such as the starting points and destinations, to ensure a comprehensive assessment. I measured the outcomes based on metrics such as path length, execution time, and number of visited nodes.

Comparisons of different AI techniques

Breadth First Search (BFS) and Best First Search (BeFS) are two popular AI techniques used in route finding problems. Let's analyse their strengths and weaknesses based on the evaluation measures below:

Execution Time:
BFS Execution Time: 0.4785938262939453 seconds
BeFS Execution Time: 3.8689889907836914 seconds
BFS outperforms BeFS in terms of execution time, as it completes the search operation more quickly.

Path Length:
BFS Path Length: 136 nodes
BeFS Path Length: 1708 nodes
BFS provides a shorter path length compared to BeFS, indicating that it finds a more optimal solution in terms of the number of nodes traversed.

Visited Nodes:
BFS Visited Nodes: 68,414 nodes
BeFS Visited Nodes: 127,669 nodes
BFS explores a smaller number of nodes compared to BeFS, suggesting that it examines feIr possibilities before finding a solution.

Based on these evaluation results, the strengths and weaknesses of the techniques can be summarized as follows:

BFS excels in terms of execution time, path length, and the number of visited nodes. It is faster, finds shorter paths, and explores fewer nodes compared to BeFS. This makes BFS a suitable choice when time efficiency and optimal solutions are important.

BeFS, on the other hand, may take more time to execute and explore a larger number of nodes. Hoover, it has the advantage of using heuristics to make informed decisions, which can be beneficial when prior knowledge is available. BeFS is particularly useful in handling large search spaces efficiently and converging towards potentially optimal solutions.

The choice of technique depends on the specific requirements and constraints of the route finding problem. If time efficiency and shorter paths are crucial, BFS would be a more suitable choice. Hoover, if handling large search spaces and utilizing heuristics are important, BeFS would be preferred.

Conclusions and Recommendations

In the context of map routing, the choice between Breadth First Search (BFS), Blind Search, and Best First Search (BeFS) depends on the specific requirements and constraints of the routing problem.

BFS is a systematic search algorithm that explores all potential solutions level by level. It guarantees optimal results in terms of the number of visited nodes and can find the shortest path between two points. Hoover, BFS may not be the most efficient algorithm in terms of execution time, especially for large-scale maps or complex road networks.

BeFS, on the other hand, utilizes heuristics to make informed decisions and prioritize certain paths. It quickly converges towards potentially optimal solutions and proves useful when prior knowledge is available. Hoover, BeFS might produce suboptimal solutions if the heuristics are inaccurate.

Considering these factors, the choice between BFS, Blind Search, and BeFS for map routing depends on the trade-off between optimality and efficiency. If finding the shortest path is crucial and computational efficiency is not a major concern, BFS would be a suitable choice. On the other hand, if computational efficiency is a priority and approximate solutions are acceptable, BeFS with appropriate heuristics can be a good option.

It's important to note that there are also other advanced search algorithms specifically designed for map routing, such as A* (A-star) algorithm, Dijkstra's algorithm, and various variations of these algorithms. These algorithms take into account both optimality and efficiency by incorporating heuristics and intelligent search strategies.

Ultimately, the choice of the best algorithm for map routing depends on the specific requirements of the problem, the size and complexity of the map, the available computational resources, and the trade-offs between optimality and efficiency.

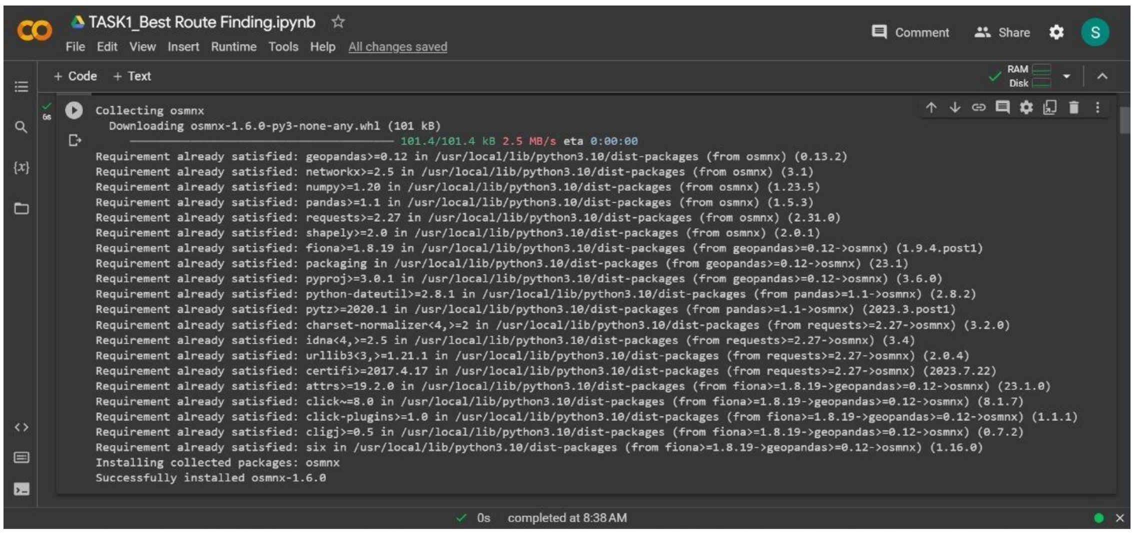
List of References

[1] "Wikipedia - Breadth-first search," in Wikipedia, [Online]. Available: https://en.wikipedia.org/wiki/Breadth-first_search.

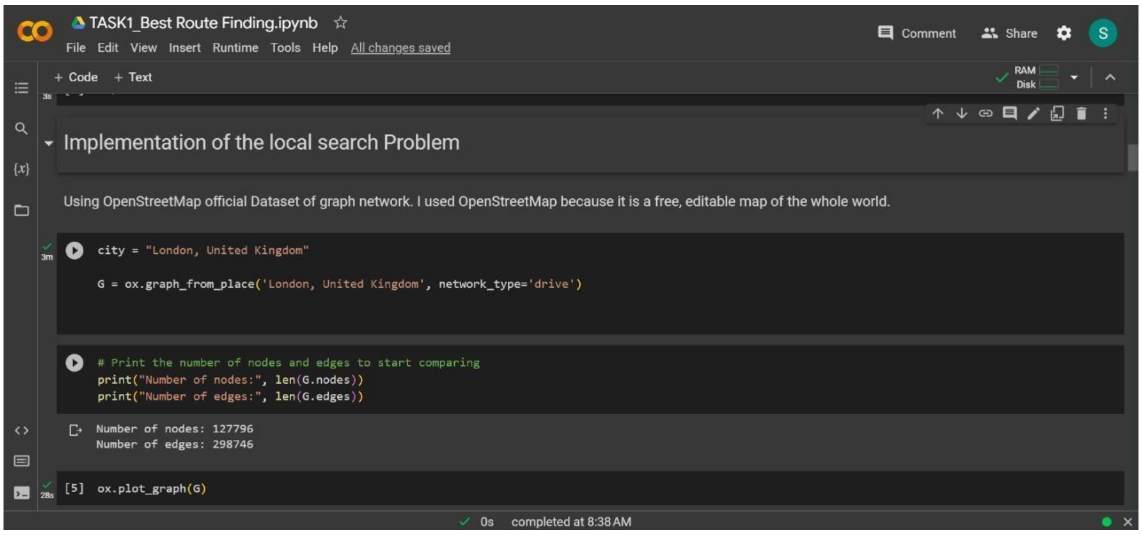
[2] "GeeksforGeeks - Best First Search (Informed Search)," in GeeksforGeeks, [Online]. Available: https://www.geeksforgeeks.org/best-first-search-informed-search/.

[3] "Introduction to OpenStreetMap," in OpenStreetMap, [Online]. Available: https://www.openstreetmap.org/about.

Appendices


TASK1_Best Route Finding.ipynb
File Edit View Insert Runtime Tools Help All changes saved
Comment Share
+ Code + Text
RAM Disk

```
Collecting osmnx
  Downloading osmnx-1.6.0-py3-none-any.whl (101 kB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 101.4/101.4 kB 2.5 MB/s eta 0:00:00
Requirement already satisfied: geopandas>=0.12 in /usr/local/lib/python3.10/dist-packages (from osmnx) (0.13.2)
Requirement already satisfied: networkx>=2.5 in /usr/local/lib/python3.10/dist-packages (from osmnx) (3.1)
Requirement already satisfied: numpy>=1.20 in /usr/local/lib/python3.10/dist-packages (from osmnx) (1.23.5)
Requirement already satisfied: pandas>=1.1 in /usr/local/lib/python3.10/dist-packages (from osmnx) (1.5.3)
Requirement already satisfied: requests>=2.27 in /usr/local/lib/python3.10/dist-packages (from osmnx) (2.31.0)
Requirement already satisfied: shapely>=2.0 in /usr/local/lib/python3.10/dist-packages (from osmnx) (2.0.1)
Requirement already satisfied: fiona>=1.8.19 in /usr/local/lib/python3.10/dist-packages (from geopandas>=0.12->osmnx) (1.9.4.post1)
Requirement already satisfied: packaging in /usr/local/lib/python3.10/dist-packages (from geopandas>=0.12->osmnx) (23.1)
Requirement already satisfied: pyproj>=3.0.1 in /usr/local/lib/python3.10/dist-packages (from geopandas>=0.12->osmnx) (3.6.0)
Requirement already satisfied: python-dateutil>=2.8.1 in /usr/local/lib/python3.10/dist-packages (from pandas>=1.1->osmnx) (2.8.2)
Requirement already satisfied: pytz>=2020.1 in /usr/local/lib/python3.10/dist-packages (from pandas>=1.1->osmnx) (2023.3.post1)
Requirement already satisfied: charset-normalizer<4,>=2 in /usr/local/lib/python3.10/dist-packages (from requests>=2.27->osmnx) (3.2.0)
Requirement already satisfied: idna<4,>=2.5 in /usr/local/lib/python3.10/dist-packages (from requests>=2.27->osmnx) (3.4)
Requirement already satisfied: urllib3<3,>=1.21.1 in /usr/local/lib/python3.10/dist-packages (from requests>=2.27->osmnx) (2.0.4)
Requirement already satisfied: certifi>=2017.4.17 in /usr/local/lib/python3.10/dist-packages (from requests>=2.27->osmnx) (2023.7.22)
Requirement already satisfied: attrs>=19.2.0 in /usr/local/lib/python3.10/dist-packages (from fiona>=1.8.19->geopandas>=0.12->osmnx) (23.1.0)
Requirement already satisfied: click~=8.0 in /usr/local/lib/python3.10/dist-packages (from fiona>=1.8.19->geopandas>=0.12->osmnx) (8.1.7)
Requirement already satisfied: click-plugins>=1.0 in /usr/local/lib/python3.10/dist-packages (from fiona>=1.8.19->geopandas>=0.12->osmnx) (1.1.1)
Requirement already satisfied: cligj>=0.5 in /usr/local/lib/python3.10/dist-packages (from fiona>=1.8.19->geopandas>=0.12->osmnx) (0.7.2)
Requirement already satisfied: six in /usr/local/lib/python3.10/dist-packages (from fiona>=1.8.19->geopandas>=0.12->osmnx) (1.16.0)
Installing collected packages: osmnx
Successfully installed osmnx-1.6.0
```


TASK1_Best Route Finding.ipynb
File Edit View Insert Runtime Tools Help All changes saved
Comment Share
+ Code + Text
RAM Disk

Implementation of the local search Problem

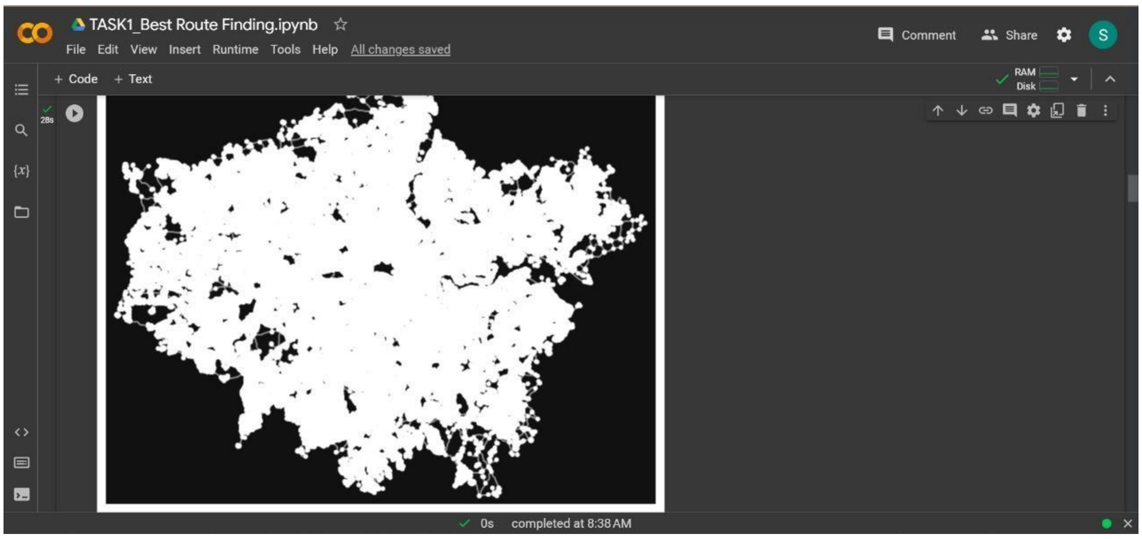
Using OpenStreetMap official Dataset of graph network. I used OpenStreetMap because it is a free, editable map of the whole world.

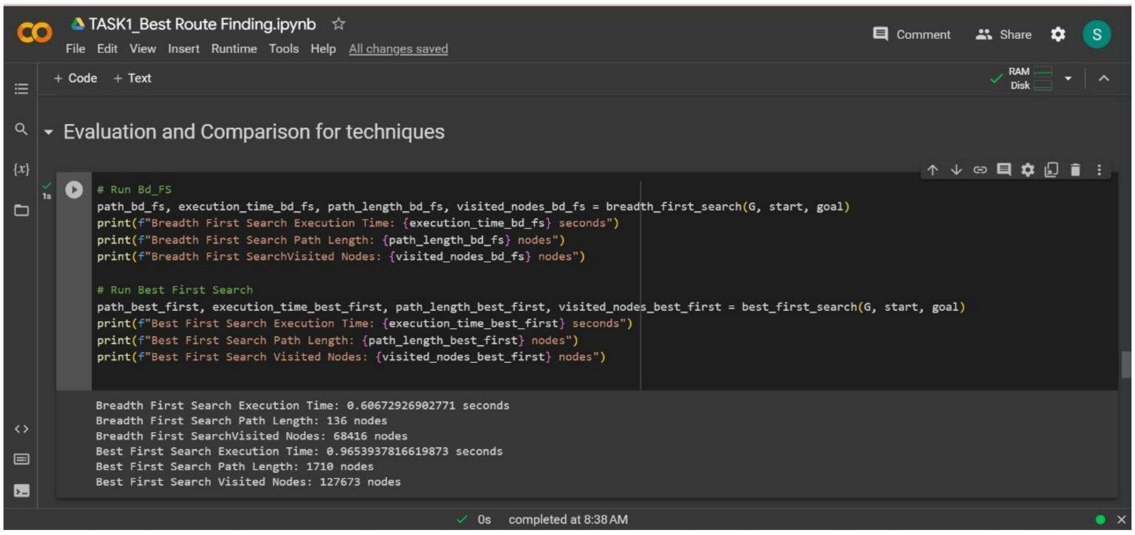
```python
city = "London, United Kingdom"

G = ox.graph_from_place('London, United Kingdom', network_type='drive')
```

```python
# Print the number of nodes and edges to start comparing
print("Number of nodes:", len(G.nodes))
print("Number of edges:", len(G.edges))
```

```
Number of nodes: 127796
Number of edges: 298746
```

```python
[5] ox.plot_graph(G)
```

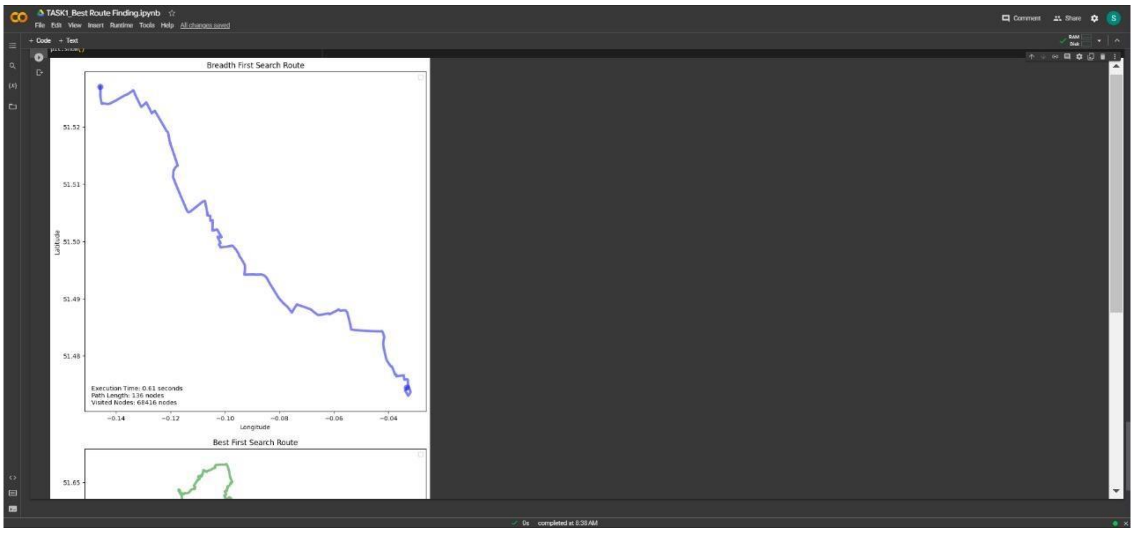

TASK1_Best Route Finding.ipynb
File Edit View Insert Runtime Tools Help All changes saved
Comment Share
+ Code + Text
RAM Disk
0s completed at 8:38 AM

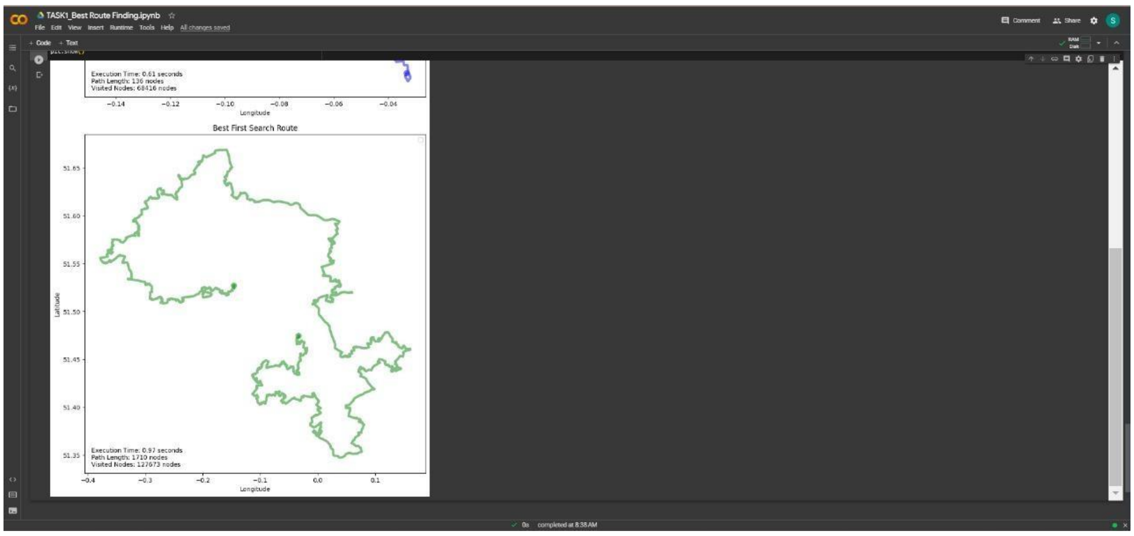

TASK1_Best Route Finding.ipynb
File Edit View Insert Runtime Tools Help All changes saved
Comment Share
+ Code + Text
RAM Disk

Evaluation and Comparison for techniques

```
# Run Bd_FS
path_bd_fs, execution_time_bd_fs, path_length_bd_fs, visited_nodes_bd_fs = breadth_first_search(G, start, goal)
print(f"Breadth First Search Execution Time: {execution_time_bd_fs} seconds")
print(f"Breadth First Search Path Length: {path_length_bd_fs} nodes")
print(f"Breadth First SearchVisited Nodes: {visited_nodes_bd_fs} nodes")

# Run Best First Search
path_best_first, execution_time_best_first, path_length_best_first, visited_nodes_best_first = best_first_search(G, start, goal)
print(f"Best First Search Execution Time: {execution_time_best_first} seconds")
print(f"Best First Search Path Length: {path_length_best_first} nodes")
print(f"Best First Search Visited Nodes: {visited_nodes_best_first} nodes")
```

```
Breadth First Search Execution Time: 0.60672926902771 seconds
Breadth First Search Path Length: 136 nodes
Breadth First SearchVisited Nodes: 68416 nodes
Best First Search Execution Time: 0.9653937816619873 seconds
Best First Search Path Length: 1710 nodes
Best First Search Visited Nodes: 127673 nodes
```





Individual reflection

During this project, I used the Colab Google platform and the OSM library to apply the Best First Search (BeFS) and Blind Search algorithms on the London map. I faced challenges in preprocessing the OpenStreetMap data and had to seek help from online forums and communities.

One of the achievements was successfully implementing both algorithms and comparing their performance in terms of path length, execution time, and the number of visited nodes. This project improved my technical skills and problem-solving abilities.

In future projects, I would like to explore other advanced search algorithms like A* and Dijkstra's algorithm for map routing. Overall, this project provided valuable hands-on experience and enhanced my skills in implementing and comparing route finding algorithms using OpenStreetMap data.