Application of Data Structures and Algorithms to a Taxi Routing System

(John Huggard: 20432474 | Luke Feeney: 24110699 | Denis Semenov: 24288890)

Problem Selection:

In this project, we developed an efficient Taxi Routing System to simulate realistic, large-scale urban transportation scenarios. Taxi routing is fundamentally a shortest-path optimization problem on a dynamic, large-scale graph network where "pick-up and drop-off points represent vertices, roads are edges, and travel times are the edge weights"[1] , a framework consistently validated by recent research (2021-2024)[1,2,3]

This is not an academic exercise created by researchers, the taxi market is a huge and rapidly expanding sector valued at $265.51 Billion and expected to grow at a CAGR (Compound Annual Growth Rate) of 9% from 2025 to 2032[4]. Efficient routing is essential for minimizing wait times, maximizing driver utilization, and preventing traffic congestion.

While modern research favours Reinforcement Learning and hybrid machine learning approaches, these methods still build on graph-theory principles such as shortest-path optimization (e.g. Dijkstra's and A* algorithms). Taxi routing remains one of the most significant real-world applications of data structures and algorithms today, in terms of scale and importance.

System Design Proposal:

The Taxi Routing System is designed to efficiently match taxi drivers with passengers and calculate optimal routes for both pickup and drop-off operations in a simulated city. The city uses A* algorithm as the primary pathfinding mechanism to identify the shortest paths while given variable road conditions in the form of edge weights. Given the complexity of this problem, we implemented a modular design leveraging the Object-Oriented Programming paradigm. The following table describes how we designed our main system components.

Component	Functionality	Key Operations
City Map	Simulate urban environment as a grid based map with obstacles representing buildings and weights representing speed limits	- Generate random map layouts with configurable dimensions and obstacle density - Build graph representation of the city - Methods to get valid positions and validate positions
Path Finder	Implements path finding algorithms to find optimal routes between points	- A* algorithm implementation with Manhattan distance heuristic - Dijktra's algorithm as an alternative pathfinding method - Path reconstruction and distance calculation
Driver	Represents the taxi driver in the system	- Track location and availability status - Maintain current assigned passenger and path information
Passenger	Represents passengers requesting taxi services	- Store pickup and drop-off locations - Track assignment status and assigned driver
Route Assigner	Handles assignment of drivers to passengers and route calculations	- Add and manage drivers and passengers - Calculate distances between drivers and passengers - Implement optimal assignment algorithms - Generate statistics about assignment Algorithms and Routes
Visualiser	Provide visualisations of the city, drivers, passengers and routes	- Render grid with obstacles - Display drivers, pickup locations, drop-off locations - Present statistical information about the system state
Taxi Routing System	Coordinates all components and provides an interface	- Initialise and configure the system - Run simulations with selected algorithms - Process results and coordinate visualisations

To allow us to test the algorithms under different conditions and simulate a real application as accurately as possible, the system has 4 inputs: (i) Grid parameters (height, width, number of obstacles) (ii) Road conditions (Edge weight ranges) (iii) Person parameters: Number of drivers and passengers (iv) Algorithm Selection.

The system also has 4 outputs: (i) Assignments (Mapping of drivers to passengers) (ii) Routes (Calculated paths for pickup and drop-off) (iii) Statistics (Performance metrics of the route operations) (iv) Visualisation (Graphical representation of the city, people and routes).

Being a simple simulation, the system naturally has a number of constraints:

Application of Data Structures and Algorithms to a Taxi Routing System

(John Huggard: 20432474 | Luke Feeney: 24110699 | Denis Semenov: 24288890)

Constraint	Description
Grid-based Structure	The grid structure is a heavily simplified visualisation of a city. A real city will have roundabouts, slip roads, cul-de-sacs, etc.
Static Environment	In reality a city is not a static environment. Traffic and speed changes, roadworks or accidents may occur, creating a dynamic graph where nodes and edges may change.
Single-assignment model	Each driver can only be assigned to one passenger and after a trip they cannot pickup a different passenger
No Time Dimension	The simulation does not allow for time-based traffic changes or new passengers/drivers randomly being initialised
Optimisation Criteria	Currently, the system is designed for pickup distance and not other factors such as total trip length, which means the system is less efficient than it could be. Furthermore, the search is done from the driver, but a bidirectional search from driver and passenger could be more efficient.

Algorithm and Data Structure Justification:

Our system relies on several data structures, carefully chosen to optimise the time and space complexity of the system Throughout this section we will identify alternative / improved methods we found from articles and the "methodology" sections of selected research papers with appropriate links in the appendix.

- Graph Representation (Adjacency List): We implemented the city map as a dictionary-based adjacency list where each node (position) maps to a list of neighbouring nodes. This structure offers O(1) lookup time for connected nodes and uses O(V+E) space, making it significantly more efficient than adjacency matrices for our sparse city grid where most positions connect to only 2-4 neighbours. However, this structure is less efficient for dense graphs[5].
- Priority Queue (Binary Heap): Both pathfinding algorithms uses Python's heapq library to implement priority queues with O(log n) operations for insertion/removal. This is important for efficiently selecting the next nodes in pathfinding algorithms. This is not as efficient as Fibonacci heaps[6] but simpler to implement.
- Hash Collections[7]: We use Python sets for tracking obstacles and visited nodes, and dictionaries for distance mapping and edge weights. These provide O(1) lookups which significantly improves performance in path verification and distance calculations. Performance however, degrades with hash collisions[8].

Our Path Finding Algorithms were identified with a similar evaluation.

- A* Search Algorithm was chosen as the primary search algorithm. The algorithm efficiently finds optimal paths by using a Manhattan distance heuristic to guide the search. This allows for improved performance over many search algorithms (e.g. Djikstra's) as it explores fewer nodes. The time complexity is O(ElogV), though worst case is $O(V^2)$[9]. This is due to the fact that we would perform a full graph exploration in the worst case. Alternatives that could be even better is bidirectional search[10] ($O(b^{d/2})$, meets in the middle), Jump Point Search[11] (O(ElogV), eliminates symmetric paths), Contraction Hierarchies[12] (O(logV) precomputes shortcuts).
- As we alluded to earlier, heuristics are not always possible, in this case, Djikstra's algorithm can be more efficient. We add Djikstra's algorithm as a secondary pathfinding method that the user can select. The algorithm guarantees optimal paths with consistent O(ElogV) time complexity.
- To match drivers and passengers we implement a Greedy Assignment Strategy. This algorithm pairs the closest available driver-passenger pair first. This gives it an O(DxPxlogV) time complexity (D: drivers, P: passengers). This algorithm, while good for real-time scenarios due to its fast computation is not globally optimal. The algorithm may produce suboptimal assignments in complex scenarios. Alternatives used in papers are the Hungarian algorithm[13] (optimal but $O(n^3)$), Auction algorithms[14] ($O(n^2 \times \text{max cost})$, and Constraint optimisation[15] ($O(2^n)$, flexible but expensive).

Our current system handles moderate-scale scenarios well but would face challenges with big increases in grid size or passengers/divers. The path calculation and assignment become increasingly expensive as the grid and actors grow. There are a number of bottlenecks we identified in our system:

Application of Data Structures and Algorithms to a Taxi Routing System

(John Huggard: 20432474 | Luke Feeney: 24110699 | Denis Semenov: 24288890)

Component	Bottleneck	Scaling Solutions	Complexity Improvement
Path Calculation	Computation cost grows with grid size	Spatial Indexing[16] (Quad Trees): O(logn) lookup instead of O(n)Path Caching: Amortised[17] O(1) for frequently searched paths. Hierarchical Pathfinding[18]: O(logVxlogV) by abstracting the graph.	Improve complexity from O(ElogV) to O(logExlogV)
Driver Assignment	Quadratic growth with number of actors	Spatial Partitioning[19]: Reduces comparisons from O(DxP) to O(D + P). Parallel Processing: Divides computations across multiple cores.	Can reduce assignment complexity fro O(DxPxlogV) to O((D+P)logV)
Graph Operations	Memory use grows quadratically with grid size	Sparse Matrix Representation: Reduces from $O(V^2)$ to O(E) On-demand Region Loading[20]: Only loads relevant subgraphs	Reduces memory requirements from O(V+E) to O(active_V + active_E)

Modelling and Implementation:

The Taxi Routing System models, as described in section I, is a city as a grid-based graph with obstacles representing impassable areas. Each cell represents a location, with adjacent cells connected by weighted edges representing travel time/distance. We implemented adjacency lists for efficient neighbour traversal and edge weight dictionaries to store “travel costs”. Driver and passenger objects contain location data, while assignments are managed through a mapping dictionary.

For pathfinding, we implemented A* search with Manhattan distance heuristic for improved efficiency as our primary algorithm. In the case where heuristics are not possible the user may select Djikstra’s as a secondary algorithm. Driver-passenger matching uses a greedy assignment algorithm that sorts pairs by distance and iteratively assigns the closest available pairs, optimising for efficiency and response time as explained above.

Key challenges included the grid size vs. resolution (how finely the environment is divided) trade-off, with our implementation using a user-configurable grid that balances information with performance. Memory complexity scale as O(V + E), with priority queue implementations ensuring O(E log V) pathfinding time complexity. The system handles edge cases including unreachable locations and resource limitations.

The system we designed is built with 6 classes: CityMap (spatial representation), PathFinder (implements Dijkstra’s and A*), Driver and Passenger (actors), RouteAssigner (driver-passenger matching), and Visualizer (graphical output). The modular design begins with map initialization, places actors, calculates distances, assigns drivers, determines routes, and visualizes results.

Potential improvement or updates include implementing the Hungarian algorithm for optimal assignment, supporting dynamic updates for real-time changes, incorporating multi-objective optimization not just distance, implementing hierarchical pathfinding for large maps, and integrating machine learning for demand forecasting and positioning.

Application of Data Structures and Algorithms to a Taxi Routing System

(John Huggard: 20432474 | Luke Feeney: 24110699 | Denis Semenov: 24288890)

Appendix - References:

1. Li, Y., Huang, Y., Liu, Z. and Zhang, B. (2024). A Distributed Scheme for the Taxi Cruising Route Recommendation Problem Using a Graph Neural Network. *Electronics*, [online] 13(3), p.574. doi:https://doi.org/10.3390/electronics13030574.

2. Kim, Y. and Yoon, Y. (2021). Zone-Agnostic Greedy Taxi Dispatch Algorithm Based on Contextual Matching Matrix for Efficient Maximization of Revenue and Profit. *Electronics*, [online] 10(21), pp.2653–2653. doi:https://doi.org/10.3390/electronics10212653.

3. Nguyen Van, S., Vu Thi Hong, N., Pham Quang, D., Nguyen Xuan, H., Babaki, B. and Dries, A. (2022). Novel online routing algorithms for smart people-parcel taxi sharing services. *ETRI Journal*, 44(2), pp.220–231. doi:https://doi.org/10.4218/etrij.2021-0406.

4. Stellar Market Research (2020). *Stellar Market Research*. [online] Stellar Market Research. Available at: https://www.stellarmr.com/report/Taxi-Market/1934.

5. Pfenning, F., Platzer, A., Simmons, R., Anderson, P. and Cervesato, I. (n.d.). *Lecture 23 Representing Graphs 15-122: Principles of Imperative Computation (Fall 2018)*. [online] Available at: https://www.cs.cmu.edu/~15122-archive/n18/lec/23-graphs.pdf [Accessed 2 May 2025].

6. Lecture slides by Kevin Wayne. (n.d.). Available at: https://www.cs.princeton.edu/~wayne/kleinberg-tardos/pdf/FibonacciHeaps.pdf.

7.Jackiewicz, R. (2024). Understanding Hashing and Collisions in Java Collections: The Role of hashCode() and equals(). *Rafal Jackiewicz's blog*. Available at: https://jackiewicz.hashnode.dev/understanding-hashing-and-collisions-in-java-collections-the-role-of-hashcode-and-equals.

8. GeeksforGeeks. (2023). *Collision Resolution Techniques*. [online] Available at: https://www.geeksforgeeks.org/collision-resolution-techniques/.

9. Belwariar, R. (2018). *A* Search Algorithm - GeeksforGeeks*. [online] GeeksforGeeks. Available at: https://www.geeksforgeeks.org/a-search-algorithm.

10.Sturtevant, N. and Felner, A. (2018). A Brief History and Recent Achievements in Bidirectional Search. *Proceedings of the AAAI Conference on Artificial Intelligence*, 32(1). doi:https://doi.org/10.1609/aaai.v32i1.12218.

11. Hu, Y., Harabor, D., Qin, L. and Yin, Q. (2021). Regarding Goal Bounding and Jump Point Search. *Journal of Artificial Intelligence Research*, 70, pp.631–681. doi:https://doi.org/10.1613/jair.1.12255.

12. Geisberger, R., Sanders, P., Schultes, D. and Vetter, C. (2012). Exact Routing in Large Road Networks Using Contraction Hierarchies. *Transportation Science*, 46(3), pp.388–404. doi:https://doi.org/10.1287/trsc.1110.0401.

13.Souza, M.P., Augusto, A., Pereira, M.A., Augusto, F., Maciel, E., Silva, E.J. and Crepalde, D.S. (2018). OPTIMIZATION OF TAXI CABS ASSIGNMENT USING A GEOGRAPHICAL LOCATION-BASED SYSTEM IN DISTINCT OFFER AND DEMAND SCENARIOS. *Revista Brasileira de Cartografia*, 68(6). doi:https://doi.org/10.14393/rbcv68n6-44490.

14. Karels, V.C.G., Veelenturf, L.P. and Van Woensel, T. (2020). An auction for collaborative vehicle routing: Models and algorithms. *EURO Journal on Transportation and Logistics*, 9(2), p.100009. doi:https://doi.org/10.1016/j.ejtl.2020.100009.

15.Lin, Y., Li, W., Qiu, F. and Xu, H. (2012). Research on Optimization of Vehicle Routing Problem for Ride-sharing Taxi. *Procedia - Social and Behavioral Sciences*, 43, pp.494–502. doi:https://doi.org/10.1016/j.sbspro.2012.04.122.

16. ELAYAROJA, G. and SANKARI, V.U. (2019). A SURVEY ON HYPERSPECTRAL IMAGE CLASSIFICATION USING ADAPTIVE SPATIAL-SPECTRAL FEATURE LEARNING. *INTERNATIONAL JOURNAL OF COMPUTER APPLICATION*, 6(9). doi:https://doi.org/10.26808/rs.ca.i9v6.01.

17. Bhadra Shetty, A. and Mogalai, B.M. (n.d.). Path Planning on Roads using Cache. *International Journal on Recent and Innovation Trends in Computing and Communication* , [online] 5(7). Available at: https://www.academia.edu/36925963/Path_Planning_on_Roads_using_Cache.

18. Li, Q., Zeng, Z., Zhang, T., Li, J. and Wu, Z. (2011). Path-finding through flexible hierarchical road networks: An experiential approach using taxi trajectory data. *International Journal of Applied Earth Observation and Geoinformation*, 13(1), pp.110–119. doi:https://doi.org/10.1016/j.jag.2010.07.003.

19.Ma, S., Zheng, Y. and Wolfson, O. (2013). *T-share: A large-scale dynamic taxi ridesharing service*. [online] IEEE Xplore. doi:https://doi.org/10.1109/ICDE.2013.6544843.

20.Ma, S., Zheng, Y. and Wolfson, O. (n.d.). *T-Share: A Large-Scale Dynamic Taxi Ridesharing Service*. [online] Available at: https://www.microsoft.com/en-us/research/wp-content/uploads/2016/02/ICDE13_conf_camera-ready_clean.pdf [Accessed 2 May 2025].