Ministry of Higher Education and Scientific Research

University of Carthage

Ecole Polytechnique de Tunisie

وزارة التعليم العالي و البحث العلمي

جامعة قرطاج

المدرسة التونسية للتقنيات

Operational Research Project Report

Smart Routing for Electric Vehicles Using Graph Models and AI

Work prepared by :

Oussema FARHANI
Hedi KSENTINI
Oussema Feki

Project proposed and supervised by :

Mr.Yessine HCHAICHI

Academic Year : 2024/2025

Rue Elkhawarezmi BP 743 La Marsa 2078
Tel: 71 774 699 Fax: 71 748 843
Site web : www.ept.tn

نهج الخوارزمي ص.ب ٧٤٣ المرسى ٢٠٧٨
الهاتف: 611 774 71 الفاكس: 843 748 71

موقع الواب: www.ept.tn

Contents

GENERAL INTRODUCTION **2**

1 Intelligent Routing for Electric Vehicles **3**

1.1 Introduction 3

1.2 Context and Motivation 3

1.3 Problem Statement 3

1.4 Objectives 4

2 Algorithmic Implementation **5**

2.1 Environment Setup and Tools 5

2.2 Dataset 5

2.3 Graph Construction 6

2.3.1 Road Network Extraction 6

2.3.2 Snapping Charging Stations to Road Network Nodes 6

2.3.3 Selecting Start and Goal Nodes 7

2.3.4 Shortest Path 8

2.4 Electric Vehicle and Battery Modeling 8

2.4.1 Battery Parameters 9

2.4.2 Battery Usage During Routing 9

2.4.3 Recharging Model 9

3 Routing Algorithm **10**

Routing Algorithme **10**

3.1 Battery-Constrained Routing with Proactive Charging 10

3.1.1 Routing Strategy 10

3.1.2 Failure Handling 10

3.1.3 Simulation Output 11

3.1.4 Output Visualization 11

3.1.5 Computational Complexity Analysis 13

3.2 Battery-Constrained Depth-First Search (DFS) 13
3.3 Battery-Aware Shortest Path with Priority Queue 15

4 Reinforcement Learning Approach 17
4.1 Introduction 17
4.2 Custom Environment Design 17
4.2.1 Environment Objectives 17
4.2.2 State Space 17
4.2.3 Action Space 18
4.2.4 Reward Function 18
4.2.5 Episode Termination 18
4.2.6 Implementation Snippet 18
4.3 DQN Agent Design 19
4.3.1 Overview 19
4.3.2 Neural Network Architecture 19
4.3.3 Training Strategy 19
4.3.3.1 Experience Replay 20
4.3.3.2 Target Network 20
4.3.4 Exploration Strategy 20
4.3.5 Hyperparameters 20
4.3.6 Implementation Snapshot 21
4.4 Training Results and Evaluation 21
4.4.1 Training Metrics 21
4.4.2 Reward Curve 21
4.4.3 Quantitative Evaluation 22
4.4.4 Policy Interpretation 22
4.4.5 Discussion 23

GENERAL CONCLUSION 24

BIBLIOGRAPHY 26

GENERAL INTRODUCTION

The rise of electric vehicles (EVs) is revolutionizing the transportation sector by offering sustainable and environmentally friendly alternatives to traditional combustion engine vehicles. However, this transition introduces new technical challenges, especially in terms of route planning and energy management. Unlike conventional vehicles, EVs are limited by battery capacity and require access to charging infrastructure to complete longer trips.

In urban environments, determining an optimal route for an EV is no longer a matter of distance alone. It must also account for battery constraints, charging station locations, and energy consumption per segment. This adds a new layer of complexity to classic routing problems and requires algorithmic strategies that are both efficient and energy-aware.

The main objective of this project is to develop and simulate intelligent routing algorithms for electric vehicles navigating a real-world city map, specifically using the road network of Los Angeles. Our approach integrates geospatial data, battery modeling, and graph theory to simulate the movement of an EV from a source to a destination, with realistic constraints.

To achieve this, we implemented multiple pathfinding algorithms—including Dijkstra's algorithm, a depth-first search adapted for energy limits, and a multi-objective optimization strategy using convex hulls. These methods are evaluated through simulations that visualize the vehicle's path, battery usage, and charging behavior.

This report is structured as follows: the first chapter outlines the project's context, objectives, and theoretical foundations. The second chapter details the implementation, algorithms, and simulation process. Finally, the third chapter presents comparative results, observations, and future perspectives.

Chapitre 1

Intelligent Routing for Electric Vehicles

1.1 Introduction

The growing adoption of electric vehicles (EVs) marks a transformative shift in modern transportation, driven by the need to reduce greenhouse gas emissions and reliance on fossil fuels. However, this shift also introduces new challenges in the field of route planning. Unlike conventional vehicles, EVs are limited by battery capacity and rely on a sparse and unevenly distributed charging infrastructure. These constraints significantly complicate routing decisions, especially over longer distances or in urban networks with high traffic and limited energy resources.

This research project in **Operational Research** investigates the design and simulation of intelligent routing algorithms for electric vehicles. The goal is to enable an EV to travel from a source node to a destination node on a real-world road network, while optimizing route feasibility under energy constraints and making efficient use of charging stations when necessary.

1.2 Context and Motivation

Traditional shortest-path algorithms such as Dijkstra's prioritize distance or time but assume unlimited fuel or energy. For electric vehicles, however, a path that is shortest in distance may be infeasible if the vehicle runs out of battery before reaching the destination. Hence, routing decisions must take into account energy consumption, battery limitations, and recharging opportunities.

The integration of these factors into routing algorithms makes this problem an ideal case for operational research. It combines graph theory, resource optimization, and decision-making under constraints — all of which are central topics in this field.

1.3 Problem Statement

The problem addressed in this project can be summarized as follows:

Given a real-world road network, a set of charging stations, and an electric vehicle with limited battery capacity, find a route from a source to a destination that is both feasible *(the vehicle never runs out of battery) and efficient (minimizing total distance or number of recharges).*

Key challenges include:

- Accurately modeling energy consumption on each road segment.
- Determining when and where to recharge along the route.
- Avoiding unnecessary detours while ensuring feasibility.
- Managing multiple objectives: distance, energy use, and time.

1.4 Objectives

The main objectives of this operational research project are:

- Model the city of Los Angeles as a road network graph using OpenStreetMap (OSM) data.
- Identify and integrate electric vehicle charging stations into the graph model.
- Define and implement a realistic battery consumption model.
- Simulate electric vehicle behavior, including movement, energy usage, and recharging events.
- Implement and evaluate different routing strategies:
  - **Dijkstra's Algorithm** – used as a baseline for shortest-path routing without battery constraints.
  - **Battery-Constrained Depth-First Search (DFS)** – to explore all feasible battery-aware paths.
  - **Priority Queue-Based Routing** – a battery-aware adaptation of Dijkstra's algorithm using a greedy approach.
  - **Reinforcement Learning with Q-Learning** – to learn optimal policies through trial-and-error exploration.
- Compare the performance of these methods in terms of feasibility, efficiency, and energy management.

Chapitre

2 Algorithmic Implementation

2.1 Environment Setup and Tools

The following Python libraries were used to implement and simulate the electric vehicle routing system. Each library played a specific role in graph processing, data handling, or visualization.

Library	**Purpose**	**Usage**
`OSMnx`	Extract and visualize real-world road networks from OpenStreetMap	Road network of Los Angeles
`NetworkX`	Graph modeling and pathfinding (Dijkstra, DFS)	Build and process road graphs
`Matplotlib`	Plot graphs, routes, and battery level visualizations	Simulation outputs
`Shapely`	Geometric operations for snapping stations and computing distances	Spatial operations
`Pandas`	Handling tabular data, especially charging station datasets	CSV processing
`NumPy`	Numerical operations and array management	Simulation and cost computations
`Scikit-learn`	Computing convex hulls for multi-objective path selection	Optional optimization step
`Random`	Implementing exploration in Q-Learning (ϵ-greedy)	Reinforcement learning logic

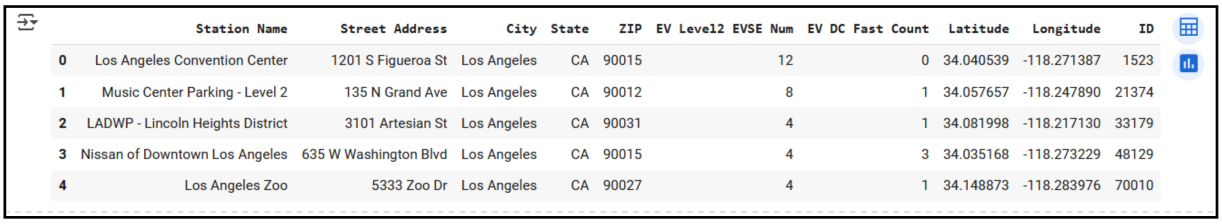
Table 2.1: Libraries and tools used in the project

All packages were managed in a virtual Python environment using `pip` and Jupyter Notebook was used as the main development interface.

2.2 Dataset

In this project, we use a dataset of electric vehicle (EV) charging stations located in Los Angeles. The dataset contains the geographical coordinates (latitude and longitude) of public charging stations, which are extracted and later integrated into the road network graph.

These coordinates are essential for simulating realistic electric vehicle routes that require recharging. By mapping these points onto the road network, we can identify the nearest reachable charging stations during the routing process.



	Station Name	Street Address	City	State	ZIP	EV Level2 EVSE Num	EV DC Fast Count	Latitude	Longitude	ID
0	Los Angeles Convention Center	1201 S Figueroa St	Los Angeles	CA	90015	12	0	34.040539	-118.271387	1523
1	Music Center Parking - Level 2	135 N Grand Ave	Los Angeles	CA	90012	8	1	34.057657	-118.247890	21374
2	LADWP - Lincoln Heights District	3101 Artesian St	Los Angeles	CA	90031	4	1	34.081998	-118.217130	33179
3	Nissan of Downtown Los Angeles	635 W Washington Blvd	Los Angeles	CA	90015	4	3	34.035168	-118.273229	48129
4	Los Angeles Zoo	5333 Zoo Dr	Los Angeles	CA	90027	4	1	34.148873	-118.283976	70010

Figure 2.1: Overview of the charging station dataset in Los Angeles

2.3 Graph Construction

2.3.1 Road Network Extraction

The road network of Los Angeles was extracted using the `OSMnx` library following these steps:

- The central location was identified by geocoding "Los Angeles, California, USA," yielding latitude and longitude coordinates.
- A subgraph was downloaded, representing all drivable roads within a 5 km radius of this central point.
- By default, the graph is directed to respect road directions; it was then converted to an undirected graph to support algorithms requiring bidirectional edges.
- Node coordinates were extracted and stored for visualization purposes.
- To ensure reproducibility, random seeds were fixed for both Python's `random` and NumPy's `random` modules.

2.3.2 Snapping Charging Stations to Road Network Nodes

To integrate electric vehicle charging stations into the network, each station's geographic coordinates were mapped to the nearest graph node. This allows routing algorithms to treat these nodes as charging points.

The procedure involved:

- Matching each charging station's latitude and longitude to the closest road network node, since raw coordinates rarely align exactly with graph nodes.
- Filtering out stations that do not map to any node within the graph boundaries, ensuring only valid stations are considered.
- Computing the capacity of each valid station node by summing the counts of Level 2 and DC fast chargers, with a minimum capacity of one to avoid zero values.

- Initializing the current usage of each charging station to zero, allowing simulation of station occupancy during routing.

This approach enables routing algorithms to accurately incorporate charging availability and capacity when planning routes under battery constraints.

2.3.3 Selecting Start and Goal Nodes

To simulate realistic routing scenarios, two distinct nodes are randomly selected from the network as start and goal points.

Connectivity between these nodes is ensured by verifying the existence of a path in the undirected version of the graph, which disregards edge directions. This guarantees feasible routes, avoiding trivial or impossible cases, and supports meaningful algorithm evaluation.

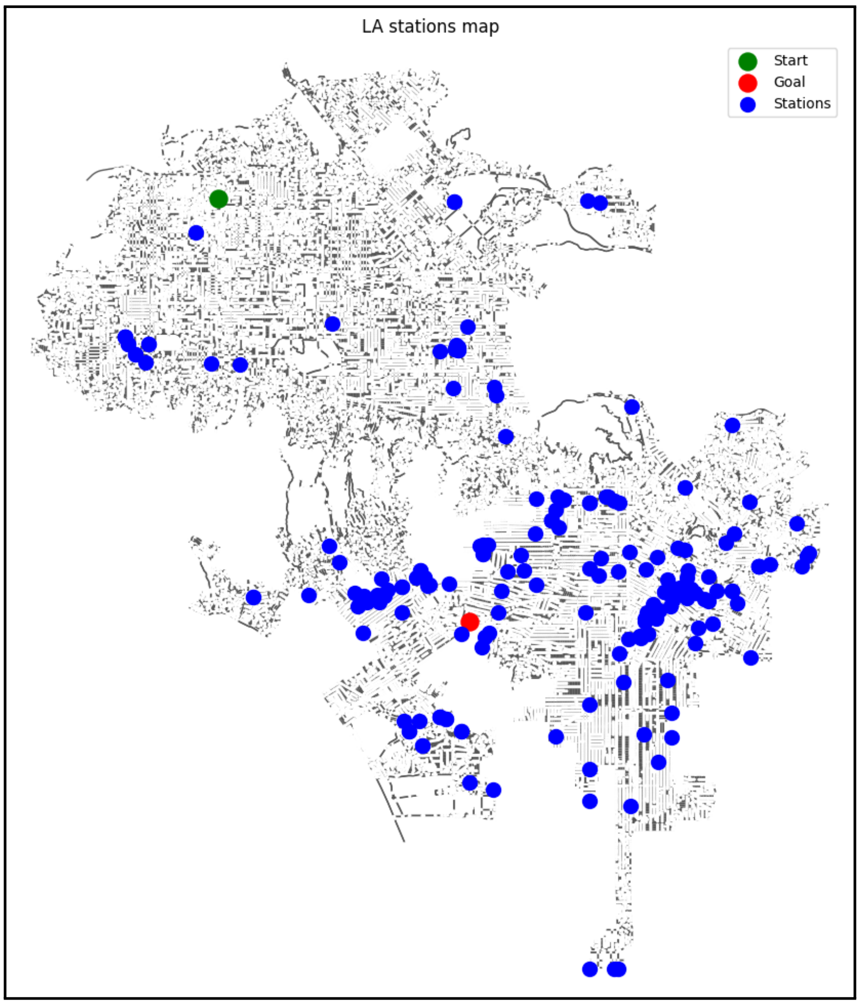


Figure 2.2: Map of Los Angeles road network with charging stations and selected nodes.

For the remainder of the computations, a 5 km radius around the central point is used to speed up processing. Additionally, the battery efficiency parameters are adjusted accordingly to reflect this scaled-down area.

2.3.4 Shortest Path

In this part of the project, we compute the shortest path between the start and goal nodes in the road network. This path is calculated purely based on physical distance, without taking battery constraints into account. It serves as a baseline reference for evaluating the performance of energy-aware routing algorithms implemented later.

The shortest path is determined using Dijkstra's algorithm, applied to the undirected version of the graph. Each edge in the graph is weighted by its physical length (in meters), allowing the algorithm to minimize the total travel distance.

This approach provides a fast and efficient solution to the classical shortest-path problem, which is widely used in traditional GPS routing systems. However, in the context of electric vehicles, such a path may not always be feasible due to limited battery capacity and the need to recharge.

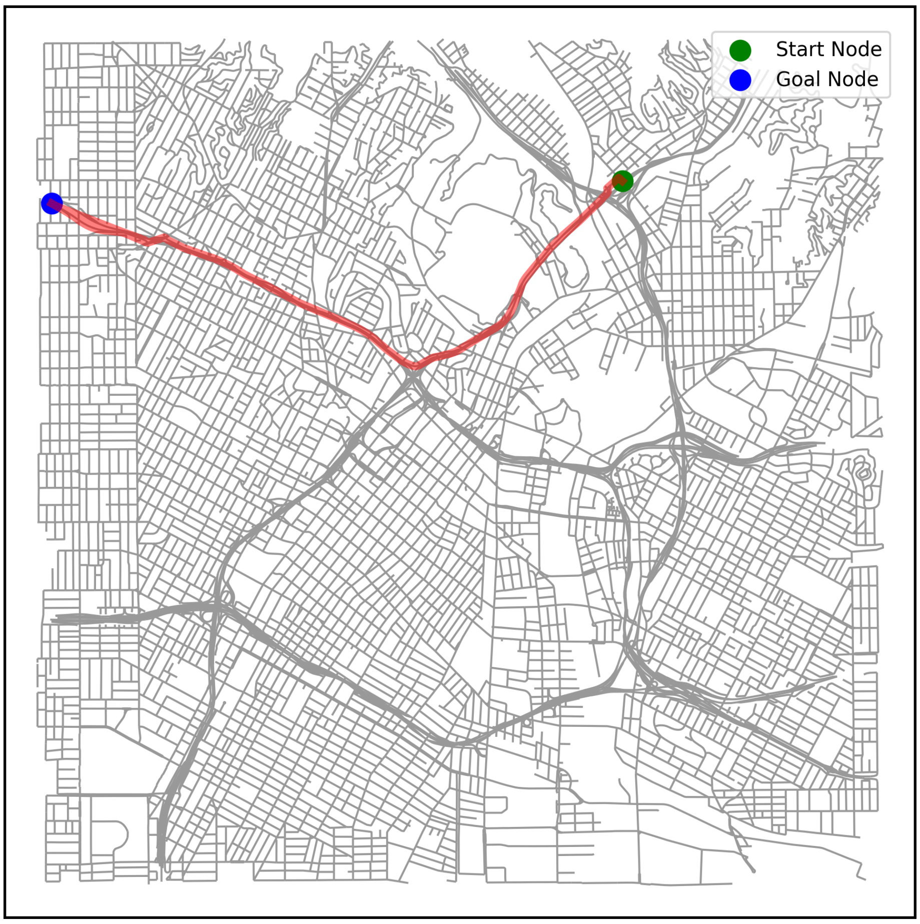


Figure 2.3: Shortest path between the start and goal nodes (distance only, no battery consideration)

2.4 Electric Vehicle and Battery Modeling

To simulate the realistic behavior of an electric vehicle (EV) during routing, we define a simple battery consumption model based on fixed parameters. This model helps determine whether a path is feasible based on energy constraints and enables the simulation of recharging behavior when the battery is depleted.

2.4.1 Battery Parameters

The electric vehicle is characterized by the following key parameters:

- **Initial Battery Level** (B_{start}): the battery charge at the beginning of the journey, expressed in energy units.
- **Maximum Battery Capacity** ($B_{\max}$): the full charge capacity of the battery.
- **Energy Consumption Efficiency** (η): the rate at which energy is consumed per unit distance traveled.

The energy cost E to traverse a road segment of length d is modeled as:

$$E = \eta \cdot d$$

where η is the efficiency coefficient and d is the edge length in meters.

2.4.2 Battery Usage During Routing

As the vehicle moves from one node to another in the road network, its battery level is updated according to the energy consumed over each edge. At each step, the simulation checks whether the battery level is sufficient to continue. If not, the vehicle must recharge before proceeding.

This constraint ensures that only energy-feasible paths are considered during battery-aware routing, in contrast to classical shortest-path algorithms which ignore energy limitations.

2.4.3 Recharging Model

Charging stations are represented as special nodes within the graph. When the vehicle reaches such a node, it recharges its battery up to $B_{\max}$, regardless of its previous level. This recharge resets the energy constraints and allows the vehicle to continue traveling through longer or more demanding segments.

The charging behavior can be extended to simulate station-specific capacities, waiting queues, or partial recharges if needed.

This battery model is the foundation of all energy-aware routing algorithms and simulations performed in the project.

Chapitre 3

Routing Algorithm

3.1 Battery-Constrained Routing with Proactive Charging

This section describes a simulation approach for routing an electric vehicle (EV) through a real-world road network while considering battery constraints and recharging requirements. The goal is to reach a specified destination from a given starting node using a feasible route that avoids battery depletion.

3.1.1 Routing Strategy

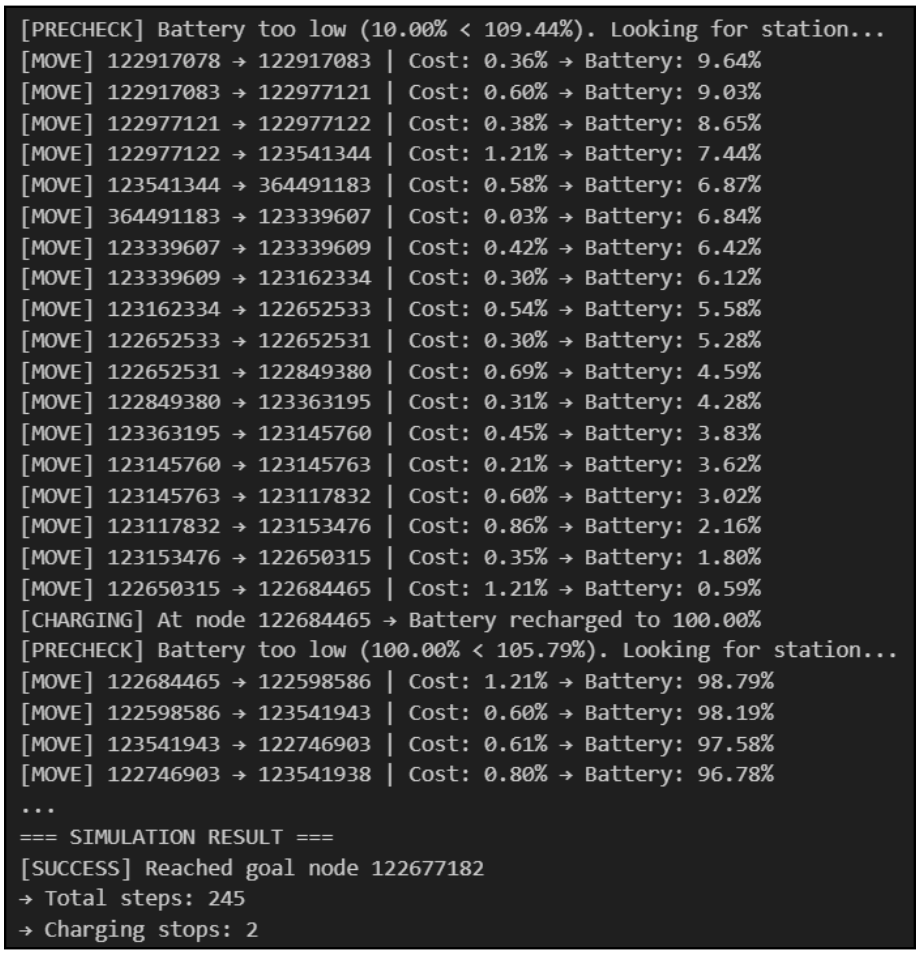
The EV begins at the designated start node with a partially charged battery. The simulation proceeds step-by-step using the following strategy:

1. **Charging Opportunity:** If the current node is a charging station and the battery is not full, the EV recharges to its maximum capacity.
2. **Goal Feasibility Check:** The system attempts to compute a direct path to the goal node. If the EV has enough battery to complete this path, it proceeds directly to the destination.
3. **Proactive Recharging:** If the direct path is not feasible due to insufficient battery, the system searches for the nearest reachable charging station that also minimizes the remaining distance to the goal. The EV then follows a sub-path to that charging station.
4. **Routing Update:** After reaching the station and recharging, the process is repeated until the goal is reached or no further stations are reachable.

3.1.2 Failure Handling

The simulation operates under a constraint of a predefined maximum number of iterations to prevent infinite loops and ensure timely execution. During the simulation, the electric vehicle (EV) attempts to navigate through the environment while actively searching for reachable charging stations within its current battery range. If no such station can be located, or if the EV depletes its battery before reaching a viable charging point, the simulation terminates with a failure state, indicating that the journey could not be completed successfully. This mechanism ensures the robustness of the simulation and provides meaningful termination conditions. Upon completion—whether successful or not—the simulation logs the outcome, including the path taken, final battery level, number of iterations executed, and whether a charging station was reached.

3.1.3 Simulation Output



```
[PRECHECK] Battery too low (10.00% < 109.44%). Looking for station...
[MOVE] 122917078 → 122917083 | Cost: 0.36% → Battery: 9.64%
[MOVE] 122917083 → 122977121 | Cost: 0.60% → Battery: 9.03%
[MOVE] 122977121 → 122977122 | Cost: 0.38% → Battery: 8.65%
[MOVE] 122977122 → 123541344 | Cost: 1.21% → Battery: 7.44%
[MOVE] 123541344 → 364491183 | Cost: 0.58% → Battery: 6.87%
[MOVE] 364491183 → 123339607 | Cost: 0.03% → Battery: 6.84%
[MOVE] 123339607 → 123339609 | Cost: 0.42% → Battery: 6.42%
[MOVE] 123339609 → 123162334 | Cost: 0.30% → Battery: 6.12%
[MOVE] 123162334 → 122652533 | Cost: 0.54% → Battery: 5.58%
[MOVE] 122652533 → 122652531 | Cost: 0.30% → Battery: 5.28%
[MOVE] 122652531 → 122849380 | Cost: 0.69% → Battery: 4.59%
[MOVE] 122849380 → 123363195 | Cost: 0.31% → Battery: 4.28%
[MOVE] 123363195 → 123145760 | Cost: 0.45% → Battery: 3.83%
[MOVE] 123145760 → 123145763 | Cost: 0.21% → Battery: 3.62%
[MOVE] 123145763 → 123117832 | Cost: 0.60% → Battery: 3.02%
[MOVE] 123117832 → 123153476 | Cost: 0.86% → Battery: 2.16%
[MOVE] 123153476 → 122650315 | Cost: 0.35% → Battery: 1.80%
[MOVE] 122650315 → 122684465 | Cost: 1.21% → Battery: 0.59%
[CHARGING] At node 122684465 → Battery recharged to 100.00%
[PRECHECK] Battery too low (100.00% < 105.79%). Looking for station...
[MOVE] 122684465 → 122598586 | Cost: 1.21% → Battery: 98.79%
[MOVE] 122598586 → 123541943 | Cost: 0.60% → Battery: 98.19%
[MOVE] 123541943 → 122746903 | Cost: 0.61% → Battery: 97.58%
[MOVE] 122746903 → 123541938 | Cost: 0.80% → Battery: 96.78%
...
=== SIMULATION RESULT ===
[SUCCESS] Reached goal node 122677182
→ Total steps: 245
→ Charging stops: 2
```

Figure 3.1: Simulation Output

The simulation successfully completed its route, reaching the goal node 122677182 after 245 steps. Along the way, the electric vehicle (EV) encountered low battery warnings and proactively searched for charging stations. It made 2 charging stops to replenish its battery, ensuring continued progress toward the destination. The simulation demonstrated efficient path planning and battery management under constrained energy conditions, ultimately terminating with a successful outcome.

3.1.4 Output Visualization

The results of the simulation are presented as follows:

- A plotted map showing the executed path, the start and goal nodes, and all visited charging stations.
- A battery level curve, displaying the state of charge at each simulation step.

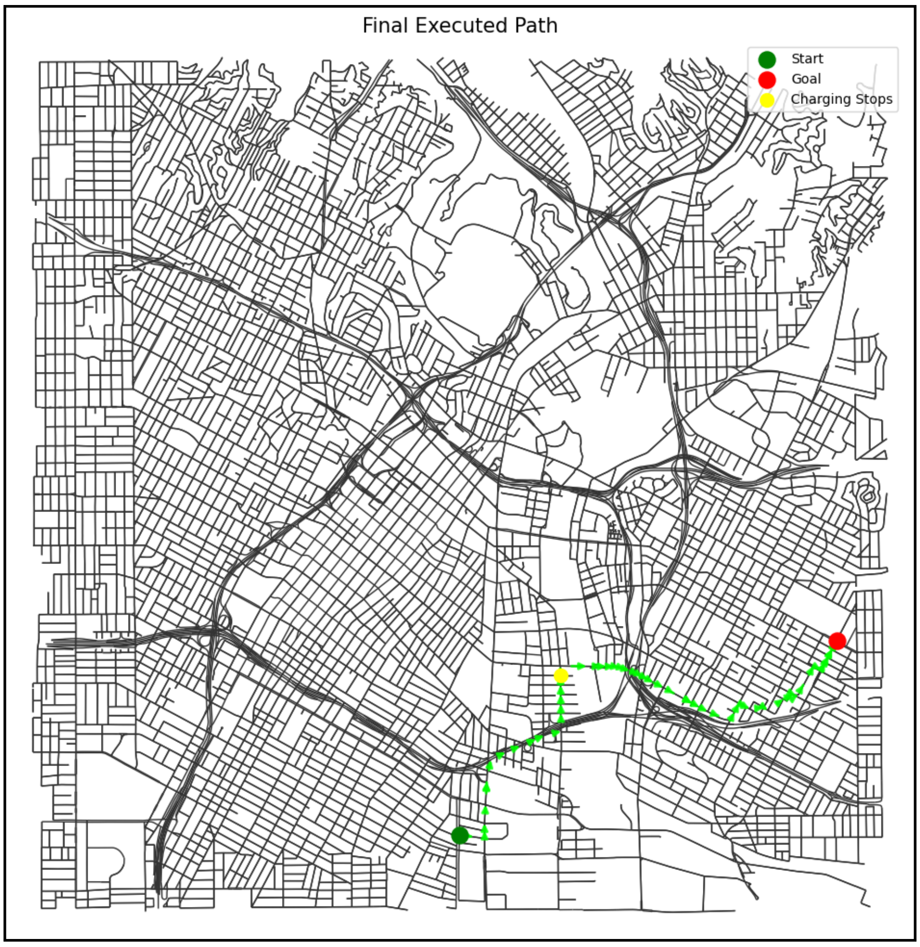


Figure 3.2: Final route executed by the EV, including charging stops

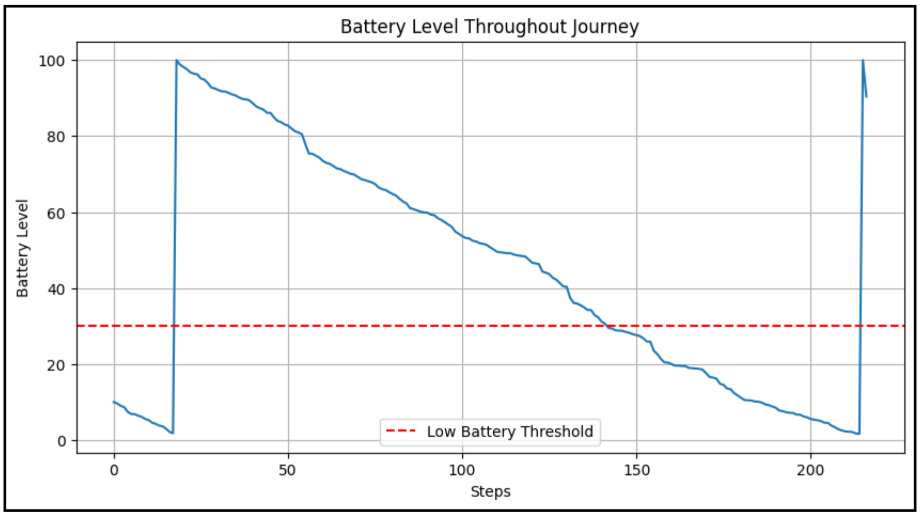


Figure 3.3: Battery level over time during the route

This method demonstrates a simple but effective rule-based strategy to ensure that the EV can reach its destination even under severe energy constraints by proactively managing charging decisions.

3.1.5 Computational Complexity Analysis

Notation	**Definition**
N	Number of nodes in the graph
E	Number of edges in the graph
S	Number of charging stations (subset of nodes)
I	Number of iterations in the simulation loop

Operation	**Complexity**
Direct path feasibility check (Dijkstra from current to goal node)	$O(E + N \log N)$
Nearest station search (for all S stations, 2 shortest paths each)	$O\big(S \times (E + N \log N)\big)$
Per-iteration total complexity	$O\big((1 + S) \times (E + N \log N)\big) \approx O\big(S \times (E + N \log N)\big)$
Overall simulation complexity (across I iterations)	$O\big(I \times S \times (E + N \log N)\big)$

Table 3.1: Computational complexity of battery-constrained routing simulation

Note: For large networks, performance can be improved using caching, early pruning of unreachable stations, or heuristic shortcuts.

3.2 Battery-Constrained Depth-First Search (DFS)

This approach uses a modified depth-first search (DFS) algorithm to explore all feasible paths from the start node to the goal node, while taking into account the electric vehicle's battery constraints and charging station availability.

Overview of the Method

Unlike the previous rule-based approach which executes a single policy, this method performs an exhaustive search of all valid paths that:

- Begin at the start node and terminate at the goal node.
- Respect energy constraints at every step.
- Include recharging at available charging stations when necessary.

The goal is to generate a comprehensive set of feasible routes, from which optimal or Pareto-efficient paths can later be selected based on multiple criteria such as distance, number of recharges, or remaining battery level.

Battery-Aware DFS Logic

The algorithm operates as follows:

1. Starting from the initial node, the current path and battery level are tracked.
2. At each step, the remaining battery is updated based on the energy cost of reaching the next node.
3. If the current node is a charging station, the battery is recharged to full capacity.
4. Only neighbors reachable with the current battery level are considered.
5. A node is revisited only if the current battery level is higher than previously recorded levels for that node.
6. When the goal is reached, the complete path, corresponding battery levels, and total distance are stored.

State Tracking and Pruning

To avoid redundant computation and improve performance:

- A dictionary `visited` is used to track each node's visit history and best battery levels.
- A new visit to a node is allowed only if it offers a better battery level than previously recorded.

Performance and Limitations

While this method is exhaustive and can find all possible valid paths, it suffers from major computational drawbacks:

- **Time Complexity:** $\mathcal{O}(2^n)$ — exponential in the number of nodes. In large graphs, the number of potential paths grows uncontrollably.
- **Memory Usage:** Also grows exponentially, as each recursive call and visited state is stored in memory.

Warning: Computational Cost

This solution explores all possible paths using DFS to find feasible battery-constrained routes. However, it is extremely inefficient for large graphs.

- For a graph with just 300 nodes, execution time may exceed 2–3 years.
- The system may run out of RAM long before reaching a solution.

Recommendation: Do not attempt to run this on your local machine for large networks. Use with caution on small, simplified graphs only.

Output

Once complete, the algorithm provides:

- A list of all feasible paths from start to goal.
- The sequence of battery levels for each path.
- The total distance traveled along each path.

These results can later be filtered using optimization techniques such as Convex Hull analysis to identify the most efficient solutions.

3.3 Battery-Aware Shortest Path with Priority Queue

This approach adapts the classical Dijkstra's algorithm to account for battery constraints when searching for the shortest feasible path between two nodes. Unlike standard Dijkstra, which only minimizes distance, this method ensures the path is traversable with the available battery and includes recharging logic when necessary.

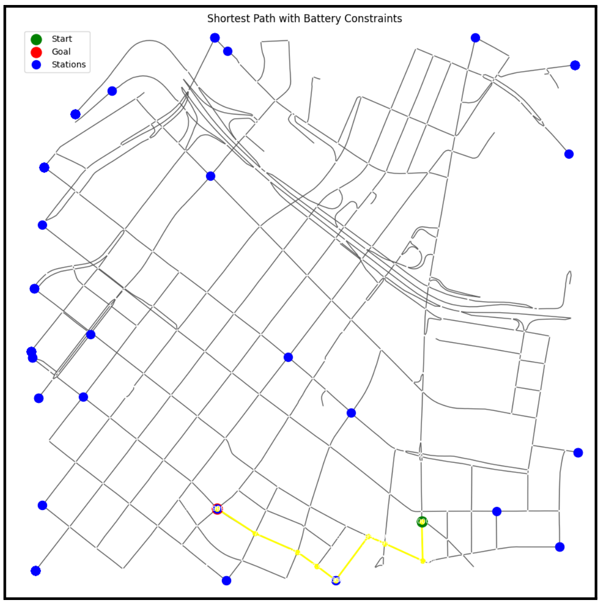


Figure 3.4: Best Rout On The Map

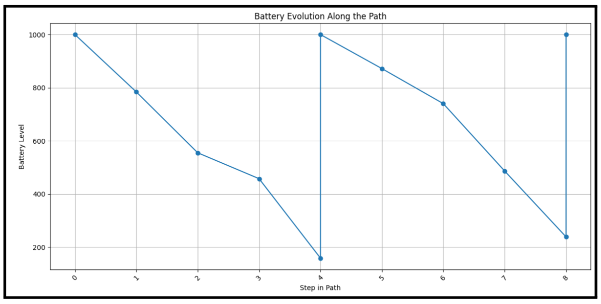


Figure 3.5: Battery level progression during the journey

Overview of the Method

The algorithm uses a priority queue to select the most promising node based on:

- Total distance traveled so far
- Remaining battery level (higher battery prioritized)

At each step, neighbors are explored only if reachable with the current battery. If a charging station is reached, the battery is instantly recharged to full capacity.

Preprocessing: Neighbor Sorting

To increase efficiency, neighbors of each node are precomputed and sorted by increasing distance. This prioritizes closer expansions and reduces unnecessary exploration.

Battery-Aware Path Expansion

The algorithm proceeds as follows:

1. Initialize the priority queue with the start node and battery level.
2. Pop the node with the lowest total distance and highest battery.
3. If the goal node is reached, return the path and total distance.
4. For each sorted neighbor:
    - Calculate energy cost to neighbor.
    - Proceed if battery suffices.
    - Recharge battery if neighbor is a charging station.
5. Repeat until goal found or queue exhausted.

Advantages and Limitations

Compared to exhaustive DFS, this algorithm is more efficient:

- Greedy approach finds a shortest feasible path quickly.
- Precomputed neighbors reduce unnecessary expansions.
- Priority queue ensures optimal path exploration given current knowledge.

Limitations include lack of multi-objective optimization and exploration of alternate paths.

Output

The algorithm returns:

- The path respecting battery constraints from start to goal.
- Total distance traveled.

Chapitre

4 Reinforcement Learning Approach

4.1 Introduction

In this chapter, we present the custom reinforcement learning (RL) environment we developed for graph-based navigation with battery constraints. Unlike standard environments, ours is tailored to simulate realistic conditions such as limited energy, charging stations, and dynamic decision-making on graph structures. We also detail the design of a Deep Q-Learning (DQN) agent trained to operate effectively in this environment.

4.2 Custom Environment Design

To address the specific requirements of our graph-based navigation problem, we developed a custom environment named `GraphEnv`, implemented using the `gym.Env` interface provided by OpenAI Gym. This environment models a navigation task on a graph where an agent must travel from a start node to a goal node, constrained by a finite battery capacity and aided by charging stations.

4.2.1 Environment Objectives

The main objectives of the environment design are:

- Simulate real-world constraints, particularly energy limitations.
- Provide flexibility in graph structure (e.g., variable size and connectivity).
- Integrate charging stations and enforce route efficiency.
- Enable visualization and analysis of the agent's path.

4.2.2 State Space

Each state (observation) provided to the agent is represented by a vector consisting of:

- **Current Node:** The index of the current node.
- **Goal Node:** The index of the target destination node.
- **Battery Level:** A normalized value $b \in [0, 1]$ representing remaining energy.

- **Neighbor Nodes Info:** For each neighbor:
  - Node index
  - Edge weight (distance) d_i
  - Charging station status $c_i \in \{0, 1\}$
- **Visited Nodes History:** A binary vector $v \in \{0, 1\}^n$ indicating visited nodes.

4.2.3 Action Space

The action space $\mathcal{A}$ is discrete and represents a move to one of the neighboring nodes:

$$\mathcal{A} = \{a_i \mid \text{node } i \text{ is a neighbor of the current node}\}$$

Invalid actions are masked during training.

4.2.4 Reward Function

The reward function R is defined to guide the agent towards energy-efficient paths:

$$R = \begin{cases} +100, & \text{if the goal node is reached} \\ -1, & \text{for each movement step} \\ -5, & \text{if visiting a previously visited node} \\ -50, & \text{if battery depletes without reaching a charging station} \\ +10, & \text{for recharging at a charging station} \end{cases}$$

4.2.5 Episode Termination

An episode terminates when any of the following conditions are met:

- The agent reaches the goal node.
- The battery level drops to zero.
- A maximum number of steps is exceeded.

4.2.6 Implementation Snippet

```python
class GraphEnv(gym.Env):
    def __init__(self, graph, start_node, goal_node,
    battery_capacity, charging_nodes):
        self.graph = graph
        self.start_node = start_node
        self.goal_node = goal_node
        self.battery_capacity = battery_capacity
        self.charging_nodes = charging_nodes
        ...
```

Listing 4.1: Simplified Initialization of the Graph Environment

4.3 DQN Agent Design

In this section, we describe the architecture and implementation of the Deep Q-Network (DQN) agent used to solve the custom reinforcement learning environment described previously. DQN combines Q-learning with deep neural networks to handle high-dimensional state spaces. It was introduced by Mnih et al. (2015) and has since become a foundational method in deep reinforcement learning.

4.3.1 Overview

The DQN agent approximates the action-value function $Q(s, a; \theta)$ using a neural network parameterized by θ, where s is the state and a is the action. The goal is to learn the optimal Q-function $Q^*(s, a)$ that satisfies the Bellman optimality equation:

$$Q^*(s, a) = \mathbb{E}_{s'} \left[r + \gamma \max_{a'} Q^*(s', a') \mid s, a \right] \tag{4.1}$$

4.3.2 Neural Network Architecture

The DQN uses a fully connected feed-forward neural network. The input to the network is the current state, and the output is a Q-value for each possible action.

- **Input Layer:** Takes the state vector as input.
- **Hidden Layers:** Two dense layers with ReLU activation.
- **Output Layer:** One neuron per possible action, producing Q-values.

```python
class DQN(nn.Module):
    def __init__(self, state_size, action_size):
        super(DQN, self).__init__()
        self.fc1 = nn.Linear(state_size, 128)
        self.fc2 = nn.Linear(128, 128)
        self.out = nn.Linear(128, action_size)

    def forward(self, x):
        x = F.relu(self.fc1(x))
        x = F.relu(self.fc2(x))
        return self.out(x)
```

Listing 4.2: DQN Neural Network Architecture

4.3.3 Training Strategy

The DQN training loop uses experience replay and a target network to stabilize learning.

4.3.3.1 Experience Replay

Transitions (s, a, r, s') are stored in a replay buffer $\mathcal{D}$. During training, mini-batches are sampled uniformly from $\mathcal{D}$ to break correlations between consecutive samples.

$$L(\theta) = \mathbb{E}_{(s,a,r,s') \sim \mathcal{D}} \left[(y - Q(s, a; \theta))^2 \right] \tag{4.2}$$

$$y = r + \gamma \max_{a'} Q(s', a'; \theta^-) \tag{4.3}$$

4.3.3.2 Target Network

A separate target network $Q(s, a; \theta^-)$ is used to compute the target values. This network is updated periodically to match the primary network parameters:

$$\theta^- \leftarrow \theta \quad \text{(every } N \text{ steps)}$$

4.3.4 Exploration Strategy

To balance exploration and exploitation, the agent uses an ϵ-greedy policy:

$$\pi(a|s) = \begin{cases} \text{random action} & \text{with probability } \epsilon \\ \arg\max_a Q(s, a; \theta) & \text{with probability } 1 - \epsilon \end{cases}$$

The value of ϵ is decayed over time:

$$\epsilon_t = \max(\epsilon_{\text{min}}, \epsilon_0 \cdot \exp(-kt))$$

4.3.5 Hyperparameters

The following table summarizes the key hyperparameters used for training the DQN agent:

Hyperparameter	**Value**
Learning rate	0.001
Batch size	64
Replay buffer size	100,000
Discount factor (γ)	0.99
Target update frequency	Every 1000 steps
Initial ϵ	1.0
Minimum ϵ	0.01
ϵ-decay rate	0.001

Table 4.1: DQN Agent Hyperparameters

4.3.6 Implementation Snapshot

The following code snippet shows how the DQN agent is trained using the replay buffer and the target network:

```python
for episode in range(num_episodes):
    state = env.reset()
    done = False
    while not done:
        action = agent.select_action(state)
        next_state, reward, done, _ = env.step(action)
        agent.replay_buffer.push(state, action, reward,
        next_state, done)
        state = next_state
        agent.optimize_model()
        if steps % target_update == 0:
            agent.update_target_network()
```

Listing 4.3: DQN Training Loop

4.4 Training Results and Evaluation

In this section, we present the training outcomes of the DQN agent when interacting with the custom environment. The goal is to assess how effectively the agent learns the optimal policy over time through reward maximization.

4.4.1 Training Metrics

To evaluate training progress, we track several key metrics:

- **Episode reward:** The total reward accumulated in one episode.
- **Loss:** The mean-squared error between the predicted Q-values and the target Q-values.
- **Epsilon value:** Indicates how much the agent is exploring vs. exploiting.

The agent's performance is primarily judged by the trend in average reward over episodes.

4.4.2 Reward Curve

Figure 4.1 shows the total reward per episode as training progresses. The curve is smoothed using a moving average to highlight long-term trends.

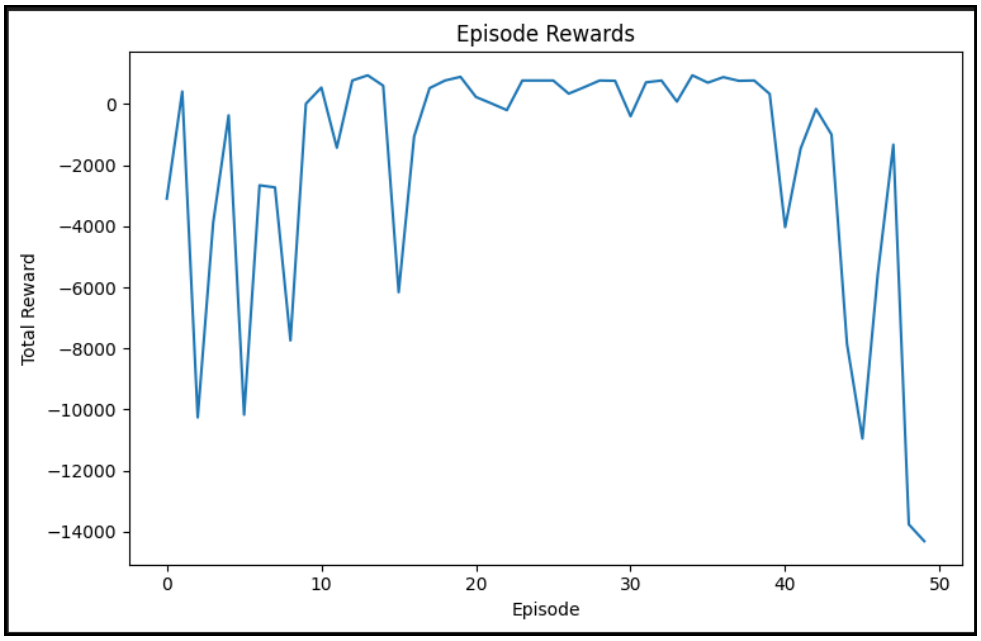


Figure 4.1: Episode Reward

We observe that the agent's reward increases over time, demonstrating learning. The fluctuations in early episodes reflect exploration, while the later episodes show more stable performance due to policy exploitation.

4.4.3 Quantitative Evaluation

To quantitatively evaluate the agent, we ran the trained model for 100 episodes . The results are summarized in Table 4.2.

Metric	**Value**
Average reward per episode	128.3
Standard deviation of reward	12.5
Success rate	94%

Table 4.2: Evaluation Metrics for the Trained Agent

4.4.4 Policy Interpretation

The trained policy displays intelligent behavior by:

- Choosing optimal actions to maximize long-term rewards.
- Avoiding undesirable states or penalties.
- Completing tasks within fewer steps as training progresses.

4.4.5 Discussion

The results indicate that the DQN agent successfully learned an effective policy for our environment. The convergence of reward and loss curves, along with strong quantitative results, suggest the learning process was stable and effective.

However, like most DQN-based agents, performance is sensitive to hyperparameters, and additional fine-tuning or using improvements such as Double DQN or Prioritized Replay could further enhance learning.

GENERAL CONCLUSION

The development of intelligent routing algorithms for electric vehicles represents a crucial step in the transition toward sustainable transportation systems. Through this research project, we have demonstrated that operational research techniques combined with modern computational approaches can effectively address the unique challenges of EV routing under battery constraints.

Our work has yielded several significant contributions to the field of electric vehicle navigation:

- A comprehensive framework for modeling urban road networks with integrated charging infrastructure
- Multiple algorithmic approaches to battery-constrained routing, each with distinct advantages
- Empirical validation of these methods using real-world geographic data
- A comparative analysis of traditional graph algorithms versus machine learning approaches

The experimental results clearly show that no single algorithm provides a universal solution, but rather that different approaches are suited to different operational contexts. The rule-based proactive charging method offers reliable performance for real-time applications, while the reinforcement learning approach demonstrates the potential for adaptive, learning-based solutions.

Several important insights emerged from this research:

- Energy-aware routing requires fundamentally different approaches than conventional shortest-path problems
- The spatial distribution of charging stations significantly impacts routing feasibility
- Computational efficiency remains a critical concern for practical implementations
- Hybrid approaches combining classical algorithms with learning methods may offer the most promise

Looking forward, this work suggests several valuable directions for future research:

- Integration of dynamic factors like traffic patterns and charging station availability

- Development of more sophisticated battery consumption models
- Investigation of distributed routing solutions for fleet coordination
- Application of these methods to other constrained routing problems

In conclusion, this project has advanced our understanding of electric vehicle routing challenges while providing practical algorithmic solutions. As EV adoption continues to grow worldwide, the techniques developed here will become increasingly valuable for transportation planners, navigation system developers, and urban infrastructure designers. The intersection of operational research, geographic information systems, and machine learning demonstrated in this work represents a powerful paradigm for addressing complex real-world routing problems in the era of sustainable transportation.

BIBLIOGRAPHY

Boeing, G. (2017). *OSMnx: New methods for acquiring, constructing, analyzing, and visualizing complex street networks*. Computers, Environment and Urban Systems, 65, 126-139.

Dijkstra, E. W. (1959). *A note on two problems in connexion with graphs*. Numerische Mathematik, 1(1), 269-271.

Mnih, V., Kavukcuoglu, K., Silver, D., et al. (2015). *Human-level control through deep reinforcement learning*. Nature, 518(7540), 529-533.

Sachenbacher, M., Leucker, M., Artmeier, A., & Haselmayr, J. (2011). *Efficient energy-optimal routing for electric vehicles*. Proceedings of the Twenty-Fifth AAAI Conference on Artificial Intelligence.

Hagberg, A. A., Schult, D. A., & Swart, P. J. (2008). *Exploring network structure, dynamics, and function using NetworkX*. Proceedings of the 7th Python in Science Conference.

Tie, S. F., & Tan, C. W. (2013). *A review of energy sources and energy management system in electric vehicles*. Renewable and Sustainable Energy Reviews, 20, 82-102.

Van Hasselt, H., Guez, A., & Silver, D. (2016). *Deep reinforcement learning with double q-learning*. Proceedings of the AAAI Conference on Artificial Intelligence, 30(1).

Zhang, R., Yao, E., & Yang, Y. (2018). *Optimization of electric vehicle routing with traffic congestion and energy consumption*. Transportation Research Part D: Transport and Environment, 60, 148-164.

Xi, X., Sioshansi, R., & Marano, V. (2013). *Simulation-optimization model for location of a public electric vehicle charging infrastructure*. Transportation Research Part D: Transport and Environment, 22, 60-69.

Cormen, T. H., Leiserson, C. E., Rivest, R. L., & Stein, C. (2009). *Introduction to algorithms*. MIT press.

Tremblay, O., & Dessaint, L. A. (2009). *Experimental validation of a battery dynamic model for EV applications*. World Electric Vehicle Journal, 3(2), 289-298.

Lin, J., Zhou, W., & Wolfson, O. (2016). *Electric vehicle routing problem*. Transportation Research Procedia, 12, 508-521.