A Project Report

On

Electric Vehicle Routing Problem (EVRP)

BY

RISHIKESH REDDY YEDULLA

SE20UCSE234

VIVEK KANJARLA

SE20UARI076

Under the supervision of

Dr.Sreedhar Madichetty

Dr.Abdelkader El Kamel

SUBMITTED IN PARTIAL FULLFILLMENT OF THE REQUIREMENTS OF

PR 402: PROJECT TYPE COURSE

ECOLE CENTRALE SCHOOL OF ENGINEERING

HYDERABAD

(JAN 2024)

CONTENTS

ACKNOWLEDGMENTS

I extend my heartfelt gratitude to my esteemed professors, **Dr.Sreedhar Madichetty, Dr.Abdelkader El Kamel,** whose unwavering support and invaluable guidance have been the cornerstone of this research endeavor. Their wealth of knowledge and deep insights have shaped my understanding of the subject matter, providing the necessary foundation for the successful completion of this project.

I am indebted to the faculty and staff of Mahindra University for fostering an environment conducive to intellectual growth and learning. Their commitment to academic excellence has played a pivotal role in shaping my academic journey, and I am grateful for the resources and opportunities they have provided.

I would like to express my appreciation to my peers and fellow students who have been a source of inspiration and encouragement throughout this project. The collaborative spirit and shared passion for advancing knowledge have enriched my experience and contributed to the quality of the research.

Lastly, I express my gratitude to all those unnamed individuals who, directly or indirectly, have played a role in the successful completion of this project. Every interaction and experience has contributed to my growth as a researcher and has left an indelible mark on my academic journey.

In conclusion, this project's completion would not have been possible without the support and guidance of individuals and institutions. Their collective contributions have enriched my academic experience, and for that, I am sincerely thankful.

Ecole Centrale School of Engineering

Hyderabad

Certificate

This is to certify that the project report entitled "Electric Vehicle Routing Problem (EVRP)" submitted by Rishikesh Reddy Yedulla (HT No.SE20UCSE234), Vivek Kanjarla(HT No.SE20UARI076) in partial fulfillment of the requirements of the course PR 402, Project Course, embodies the work done by him/her under my supervision and guidance.

(Dr.Sreedhar Madichetty & Signature)

Ecole Centrale School of Engineering, Hyderabad.

Date: 2-Jan-2024

ABSTRACT

The pursuit of sustainable and intelligent transportation solutions has become paramount in addressing the challenges posed by urbanization and environmental concerns. This research delves into the domain of Electric Vehicle Routing Problem (EVRP), exploring innovative algorithms to navigate a fleet of electric vehicles between cities while considering parameters such as charging station availability, vehicle battery levels, distances, and time constraints.

The project introduces a meticulously designed algorithm that leverages modified Dijkstra's Algorithm and heuristics to optimize the route planning for a fleet of intelligent electric vehicles. The algorithm dynamically adjusts charging strategies, considering each vehicle's unique characteristics, such as initial battery levels, charging rates, discharge rates, and maximum battery capacities. Additionally, the algorithm incorporates average speeds and charging times to generate a heuristic underestimate, facilitating efficient and realistic route planning.

Dijkstra's Algorithm is employed to calculate the shortest paths between cities, incorporating a MinHeap data structure for optimized time complexity. The resulting paths are then used to estimate the minimum road time and charging time required for each vehicle, contributing to the overall heuristic estimate. The algorithm iteratively explores potential routes, considering charging and waiting periods, to determine the optimal schedule for each vehicle while minimizing the total time for the entire fleet to reach their respective destinations.

The research methodology involves extensive testing and validation using real-world data, ensuring the algorithm's effectiveness in diverse scenarios. The algorithm's adaptability to varying city layouts, road networks, and charging infrastructure is a testament to its robustness and practical applicability.

This research's outcomes not only contribute to eco-driving and intelligent transportation but also hold promise for addressing the growing need for sustainable urban mobility solutions. The findings offer insights into efficient route planning for electric vehicle fleets, balancing energy consumption, and optimizing travel times, thereby paving the way for a greener and more intelligent future in transportation.

INTRODUCTION

- Background and MotivationI

n the rapidly evolving landscape of transportation, the rise of electric vehicles (EVs) has emerged as a promising solution to address environmental concerns and reduce dependency on traditional fossil fuels. Electric vehicles, powered by rechargeable batteries, have gained traction as sustainable alternatives that contribute to a cleaner and greener future for urban mobility. However, the widespread adoption of EVs introduces new challenges, particularly in optimizing their usage for efficient city-to-city travel.

The transition to electric vehicles necessitates an intelligent and eco-friendly approach to their navigation, taking into account factors such as battery life, charging infrastructure, and overall energy efficiency. Conventional routing algorithms may not suffice in managing electric vehicle fleets effectively, considering the nuanced requirements of these vehicles compared to their internal combustion counterparts. This gap in efficient route planning forms the crux of our research

Motivated by the need to usher in a new era of intelligent, eco-conscious driving, our project seeks to design and implement an AI-based control system for navigating a fleet of electric vehicles between cities. The objective is to optimize routes, minimizing both travel time and the environmental impact associated with energy consumption. By integrating artificial intelligence with charging station logistics and vehicle parameters, we aim to enhance the overall eco-driving experience, promoting sustainability and reducing the carbon footprint of urban transportation.

Through this research, we strive to contribute to the ongoing discourse on smart and sustainable urban mobility, offering a novel perspective on the integration of AI in electric vehicle navigation. Our work not only aligns with the broader global shift towards clean energy but also addresses a specific niche within the realm of intelligent transportation systems. The outcome of this project holds the potential to influence urban planning, shape policies, and inspire further innovations in the burgeoning field of electric vehicle technology.

- Problem Statement

The problem addressed in this project is the efficient navigation and control of a fleet of electric vehicles between cities while considering various parameters such as charging station availability, vehicle battery levels, distance, and time constraints. Traditional navigation systems may not be optimized for eco-friendly driving practices and fail to account for the unique challenges posed by electric vehicles. This research aims to develop an AI-based control algorithm that optimizes the routes of intelligent vehicles, minimizing energy consumption and ensuring timely arrival. The goal is to enhance eco-driving practices for electric vehicles, contributing to sustainable and energy-efficient urban transportation systems.

- Objectives of the Study

The primary objective of this study is to develop and implement an AI-based control system for optimizing the navigation of a fleet of electric vehicles within urban environments. The aim is to design an intelligent routing algorithm that considers various factors such as charging station availability, vehicle battery levels, distance, and time constraints. By leveraging artificial intelligence, the study seeks to enhance the efficiency and sustainability of urban transportation, with a specific focus on eco-driving practices.

The research aims to address the challenges associated with electric vehicle navigation, including the need for strategic charging station placement, optimal charging schedules, and route planning to minimize energy consumption. Through the application of advanced algorithms, the study endeavors to achieve eco-friendly vehicle movements, reducing carbon emissions and promoting environmental sustainability. The project will also explore the impact of the proposed AI-based system on overall traffic management, considering factors like congestion and transportation network optimization.

Furthermore, the study aims to assess the feasibility and practicality of the developed algorithm in real-world scenarios, taking into account the dynamic nature of urban traffic patterns. By evaluating the algorithm's performance against established benchmarks, the research seeks to validate its efficacy and identify areas for refinement. Ultimately, the study aspires to contribute to the evolving field of intelligent transportation systems, providing insights into the integration of AI technologies for eco-driving and sustainable urban mobility.

- Scope and Limitations

The scope of this project encompasses the development and implementation of an AI-based control algorithm designed to navigate a fleet of electric vehicles efficiently and sustainably. The algorithm considers crucial factors such as charging station availability, vehicle battery status, distance between nodes, and time constraints. By leveraging artificial intelligence, the system aims to optimize routing decisions, promoting eco-friendly driving practices and minimizing overall travel time.

Furthermore, the project extends its scope to include the evaluation of the algorithm's performance in a real-world scenario, simulating city-to-city transportation with varying road networks and traffic conditions. The applicability of the algorithm in a practical setting ensures its relevance to emerging intelligent transportation systems.

Despite the ambitious goals, this project has certain limitations that warrant consideration. Firstly, the effectiveness of the algorithm may be influenced by real-time factors, such as unpredictable traffic patterns and road closures, which are challenging to predict accurately. Additionally, the

algorithm assumes a static charging infrastructure, and its adaptability to dynamic changes in charging station availability is a point of limitation.

Moreover, the research is conducted within the constraints of current technological capabilities and data accuracy. Variability in electric vehicle models and charging standards may impact the algorithm's universality. Finally, the project acknowledges that the proposed solution may require further refinement to address evolving challenges in the field of intelligent transportation.

Electric Vehicle Routing Problem (EVRP)

Literature Review:

Overview of Existing Solutions:

Existing solutions in the field of intelligent vehicle navigation predominantly focus on optimizing routes, considering factors like traffic conditions, shortest paths, and time efficiency. While some algorithms address electric vehicles, a holistic approach integrating charging station considerations, battery management, and eco-driving principles is often lacking. Commonly, these solutions rely on traditional routing algorithms without deeply considering the unique characteristics of electric vehicles and their impact on energy consumption.

Relevance to the Current Project:

The current project addresses the limitations of existing solutions by introducing an AI-based control system tailored for eco-driving intelligent vehicles. It not only considers traditional factors such as distance and time but also incorporates charging station availability, vehicle battery conditions, and energy-efficient driving strategies. By leveraging artificial intelligence, the proposed algorithm aims to optimize routes while ensuring efficient energy management, contributing to the sustainability and eco-friendliness of urban transportation.

Gaps and Opportunities:

Several gaps exist in the current landscape, including the need for more comprehensive algorithms that account for real-time battery status, dynamic charging infrastructure, and individual vehicle characteristics. Opportunities lie in integrating machine learning techniques to adaptively learn and optimize routes based on historical and real-time data, further enhancing the intelligence and adaptability of the navigation system. Exploring partnerships with smart city initiatives and advancing vehicle-to-infrastructure communication can unlock possibilities for seamless integration and improved efficiency in the broader urban transportation ecosystem.

Methodology

In designing an AI-based control system for eco-driving intelligent vehicles, the methodology revolves around the implementation of a heuristic-driven algorithm to optimize the navigation of a fleet of electric vehicles. The goal is to achieve energy-efficient routes while considering various parameters such as charging station availability, vehicle battery levels, distances, and time constraints. The methodology can be outlined in three key aspects: the description of the implemented algorithm, identification of key parameters and considerations, and the justification for the chosen algorithm**.**

Description of the Implemented Algorithm:

The implemented algorithm is based on a combination of Dijkstra's Algorithm and a heuristic-driven approach. Dijkstra's Algorithm, a classical graph traversal algorithm, is adapted to find the shortest paths between nodes in the road network. This provides the foundation for route planning based on distance and time considerations. However, to address the specific challenges of electric vehicles, a heuristic-driven approach is introduced. This approach incorporates factors such as charging station locations, vehicle charging rates, and battery discharge rates to optimize energy consumption during the journey.

The algorithm maintains a dynamic state for each vehicle, considering factors like current battery levels, charging requirements, and the optimal routes calculated by Dijkstra's Algorithm. Additionally, it includes decision points for charging and waiting to ensure that vehicles reach their destinations efficiently. The integration of charging and waiting strategies within the algorithm allows for a more realistic simulation of electric vehicle navigation, accounting for the time spent on recharging and the waiting periods at charging stations.

Optimal Algorithm

The algorithm implements an optimized state-space search.

It takes into account, for each vehicle, the possibilites of:

Either charging at the present station (if it is not occupied, and if the vehicle does not have the least amount of charge required to reach the destination node from the present node via the shortest route between those two nodes, as given by the heuristic algorithm, without stopping for charging anywhere in between), or, waiting to charge at present station, or, moving on to any neighbouring node, provided that the vehicle has enough charge to move on to that node, that that neighbouring node has not been visited by the vehicle yet, and that the shortest path from that node to the destination of the vehicle does not pass through the current node (where the vehicle presently is).

After running passes through all the vehicles, the algorithm checks whether the present state is a valid solution, i.e., whether all vehicles have arrived.

If the present state is a valid solution, the algorithm compares the timestamp of the present state with the current solution (a global variable), and if the former is found to be lesser among the two, it is set as the new timestamp, and the present state is deepcopied to the solution state (also a global variable), after all of which, the algorithm goes back and continues the search.

If the present state is not a valid solution, the algorithm increments the state timestamp, and continues the search.

Code:

```
from copy import deepcopy

import sys

import contextlib


sys.setrecursionlimit(2000)


period = 1

adj = []

shortestPaths = [[]]

currentSolution = float("inf")

solutionState = None



def print_results(vehicle, solution_state):

print("Vehicle " + str(vehicle.id) + ":")

print("Heuristic Underestimate (Distance): " + str(shortestPaths[vehicle.source][vehicle.destination][0]) + "m")

print("Heuristic Underestimate (Time): " + str(vehicle.heuristicUnderestimate) + "s")

print("Heuristic Route: " + " -> ".join(shortestPaths[vehicle.source][vehicle.destination][1]))

if solution_state:

print("Optimal Schedule/Route: " + " -> ".join(solution_state.vehicleStates[vehicle.id - 1].route))
```

```
print("Optimal Time: " + str(solution_state.vehicleStates[vehicle.id - 1].timestamp) + "s\n")
else:
print("Error computing optimal schedule and time. Please try varying the time period.\n")



def driver(filename='testcases.txt', output_file='output.txt'):
global adj
global shortestPaths
global period
global solutionState

with open(filename, 'r') as file:
lines = file.readlines()

index = 0

with open(output_file, 'w') as output:
with contextlib.redirect_stdout(output): # Redirecting stdout using a context manager
# n -> number of cities (nodes / vertices); m -> number of roads (edges); k -> number of vehicles
n, m, k = map(int, lines[index].split(" "))
index += 1
period = int(lines[index])
index += 1

# edges -> local adjacency list
edges = [[] for _ in range(n)]
vehicles = []

for i in range(m):
a, b, d = map(int, lines[index].split(" "))
```

```
index += 1
edges[a - 1].append([b - 1, d])
edges[b - 1].append([a - 1, d])

adj = edges

dijkstraAll(n)

# implementing Dijkstra's Algorithm here instead of inside Vehicle class to avoid passing edges as a parameter.
for i in range(k):
a, b = map(int, lines[index].split(" "))
index += 1
c, d, e, f, g = map(float, lines[index].split(" "))
index += 1
dist, path = shortestPaths[a - 1][b - 1][0], shortestPaths[a - 1][b - 1][1]
vehicles.append(Vehicle(i + 1, a - 1, b - 1, c, d, e, f, g, dist, path))

initialState = State(n, vehicles)

generate(initialState)

print("\nAssuming all distance and time measurements are provided in metres and seconds, respectively.")
print("\nOptimal Time for all vehicles to reach their destinations is " + str(round(currentSolution, 2)) + "s ± " + str(period) +
"s.\n")
for i in range(k):
print_results(vehicles[i], solutionState)

return
```

```
def dijkstraAll(n):
global shortestPaths
global adj
shortestPaths = [[0 for j in range(n)] for i in range(n)]
for i in range(n):
for j in range(n):
a, b = dijkstrasAlgorithm(i, j, adj)
shortestPaths[i][j] = [a, b]


def generate(state, currentVehicle=0):
global currentSolution
global shortestPaths
global solutionState

if not state.feasible:
return

k = len(state.vehicleStates)

if currentVehicle == k:

t = state.timestamp
state.timestamp += period

if state.success():
for vehicleState in state.vehicleStates:
state.timestamp = max(state.timestamp, vehicleState.timestamp)
```

```
if state.timestamp < currentSolution:
currentSolution = min(currentSolution, state.timestamp)
solutionState = deepcopy(state)
state.timestamp = t
return

else:
generate(state)

state.timestamp -= period

else:
vehicleState = state.vehicleStates[currentVehicle]

if vehicleState.arrived:
generate(state, currentVehicle + 1)
return

if vehicleState.timestamp <= state.timestamp:
vehicleState.timestamp = state.timestamp

# charge the vehicle at current node
if vehicleState.charge < vehicleState.info.maxCapacity and (
shortestPaths[vehicleState.position][vehicleState.info.destination][0]) / (
vehicleState.info.averageSpeed) > vehicleState.charge / vehicleState.info.dischargingRate:
# charge
if state.nodeState[vehicleState.position] == 0 or state.nodeState[
vehicleState.position] == vehicleState.info.id:
state.nodeState[vehicleState.position] = vehicleState.info.id
t = vehicleState.charge
```

```
t1 = vehicleState.route
vehicleState.charge = min(vehicleState.charge + vehicleState.info.chargingRate * period,
vehicleState.info.maxCapacity)
if vehicleState.route[-1][0] != 'C':
vehicleState.route.append('Charge for ' + str(period) + 's')
else:
vehicleState.route[-1] = vehicleState.route[-1][:11] + str(
int(vehicleState.route[-1][11:-1]) + period) + "s"
vehicleState.timestamp += period
generate(state, currentVehicle + 1)
vehicleState.timestamp -= period
vehicleState.charge = t
vehicleState.route = t1
state.nodeState[vehicleState.position] = 0
# wait
else:
vehicleState.timestamp += period
t = vehicleState.route
if vehicleState.route[-1][0] != 'W':
vehicleState.route.append('Wait for ' + str(period) + 's')
else:
vehicleState.route[-1] = vehicleState.route[-1][:9] + str(
int(vehicleState.route[-1][9:-1]) + period) + "s"
generate(state, currentVehicle + 1)
vehicleState.timestamp -= period
vehicleState.route = t

# move on
state.nodeState[vehicleState.position] = 0
canMoveOn = False
```

```
for edge in adj[vehicleState.position]:
if edge[1] > (vehicleState.charge / vehicleState.info.dischargingRate) * (
vehicleState.info.averageSpeed) or edge[0] in vehicleState.visitedMemo: continue
a = str(vehicleState.position)
b = shortestPaths[edge[0]][vehicleState.info.destination]
if str(vehicleState.position + 1) in shortestPaths[edge[0]][vehicleState.info.destination][1]:
continue
canMoveOn = True
t1 = vehicleState.timestamp
t = edge[1] / vehicleState.info.averageSpeed
vehicleState.timestamp += t
t2 = vehicleState.charge
vehicleState.charge -= t * vehicleState.info.dischargingRate
t3 = vehicleState.position
vehicleState.position = edge[0]
t4 = vehicleState.arrived
vehicleState.arrived = edge[0] == vehicleState.info.destination
vehicleState.route.append(str(edge[0] + 1))
vehicleState.visitedMemo.add(edge[0])
generate(state, currentVehicle + 1)
vehicleState.visitedMemo.remove(edge[0])
vehicleState.route.pop()
vehicleState.arrived = t4
vehicleState.position = t3
vehicleState.charge = t2
vehicleState.timestamp = t1

if not canMoveOn:
vehicleState.canArrive = False
```

```
return



# Modified Dijkstra's Algorithm using Min-Heaps | O((n + m) * log(n)) time | O(n) space,

# where n is the number of vertices and m is the number of edges in the input graph.

# Is used to calculate shortest path time, which in turn, together with required charging time for said shortest path, is used to generate a heuristic underestimate.

def dijkstrasAlgorithm(start, end, edges):

numberOfVertices = len(edges)


minDistances = [float("inf") for _ in range(numberOfVertices)]

minDistances[start] = 0


predecessors = [None for _ in range(numberOfVertices)]


minDistancesHeap = MinHeap([[idx, float("inf")] for idx in range(numberOfVertices)])

minDistancesHeap.update(start, 0)


while not minDistancesHeap.isEmpty():

vertex, currentMinDistance = minDistancesHeap.remove()


if currentMinDistance == float("inf"):

break


for edge in edges[vertex]:

destination, distanceToDestination = edge


newPathDistance = currentMinDistance + distanceToDestination

currentDestinationDistance = minDistances[destination]

if newPathDistance < currentDestinationDistance:
```

```
minDistances[destination] = newPathDistance
minDistancesHeap.update(destination, newPathDistance)
predecessors[destination] = vertex

shortestPath = [str(end + 1)]
vertex = end
while predecessors[vertex] is not None:
shortestPath.append(str(predecessors[vertex] + 1))
vertex = predecessors[vertex]
shortestPath.reverse()

return (-1, shortestPath) if minDistances[end] == float("inf") else (minDistances[end], shortestPath)



class Vehicle:
def __init__(self, id, source, destination, initialBattery, chargingRate, dischargingRate, maxCapacity,
averageSpeed,
shortestDistance, shortestRoute):
self.id = id
self.source = source
self.destination = destination
self.initialBattery = initialBattery
self.chargingRate = chargingRate
self.dischargingRate = dischargingRate
self.maxCapacity = maxCapacity
self.averageSpeed = averageSpeed
self.shortestRoute = shortestRoute
self.minRoadTime = shortestDistance / averageSpeed
self.minChargingTime = max(0, (self.minRoadTime - self.initialBattery / self.dischargingRate) * (
self.dischargingRate / self.chargingRate))
```

```
self.needsToBeCharged = self.minChargingTime > 0
self.heuristicUnderestimate = self.minRoadTime + self.minChargingTime



class MinHeap:
def __init__(self, array):
self.vertexMap = {idx: idx for idx in range(len(array))}
self.heap = self.buildHeap(array)

def isEmpty(self):
return len(self.heap) == 0

def buildHeap(self, array):
firstParentIdx = (len(array) - 2) // 2
for currentIdx in reversed(range(firstParentIdx + 1)):
self.siftDown(currentIdx, len(array) - 1, array)
return array

def siftDown(self, currentIdx, endIdx, heap):
childOneIdx = currentIdx * 2 + 1
while childOneIdx <= endIdx:
childTwoIdx = currentIdx * 2 + 2 if currentIdx * 2 + 2 <= endIdx else -1
if childTwoIdx != -1 and heap[childTwoIdx][1] < heap[childOneIdx][1]:
idxToSwap = childTwoIdx
else:
idxToSwap = childOneIdx
if heap[idxToSwap][1] < heap[currentIdx][1]:
self.swap(currentIdx, idxToSwap, heap)
currentIdx = idxToSwap
childOneIdx = currentIdx * 2 + 1
```

```
else:
return

def siftUp(self, currentIdx, heap):
parentIdx = (currentIdx - 1) // 2
while currentIdx > 0 and heap[currentIdx][1] < heap[parentIdx][1]:
self.swap(currentIdx, parentIdx, heap)
currentIdx = parentIdx
parentIdx = (currentIdx - 1) // 2

def remove(self):
if self.isEmpty():
return

self.swap(0, len(self.heap) - 1, self.heap)
vertex, distance = self.heap.pop()
self.vertexMap.pop(vertex)
self.siftDown(0, len(self.heap) - 1, self.heap)
return vertex, distance

def swap(self, i, j, heap):
self.vertexMap[heap[i][0]] = j
self.vertexMap[heap[j][0]] = i
heap[i], heap[j] = heap[j], heap[i]

def update(self, vertex, value):
self.heap[self.vertexMap[vertex]] = (vertex, value)
self.siftUp(self.vertexMap[vertex], self.heap)
```

```
class State:

def __init__(self, n, vehicles):

self.timestamp = 0

self.nodeState = [0] * n

self.vehicleStates = [VehicleState(vehicle) for vehicle in vehicles]


def feasible(self):

feasible = True

for vehicleState in self.vehicleStates:

feasible = feasible and vehicleState.canArrive

return feasible


def success(self):

success = True

for vehicleState in self.vehicleStates:

success = success and vehicleState.arrived

return success



class VehicleState:

def __init__(self, vehicle):

self.arrived = not vehicle.needsToBeCharged

self.timestamp = vehicle.heuristicUnderestimate if self.arrived else 0

self.info = vehicle

self.charge = vehicle.initialBattery

self.position = vehicle.source

self.route = [str(vehicle.source + 1)]

if self.arrived:

self.route = vehicle.shortestRoute

self.canArrive = True
```

```
self.visitedMemo = set()

self.visitedMemo.add(vehicle.source)


driver()
```

Input format:

Please provide your input in the following format:

All values provided in a single line must be separated by a single space.

First line should contain the number of cities (n), the number of connected roads (any one direction) and the number of electric vehicles (k).

Second line should contain the time period used to discretize time. 1s is a good place to start. Lesser values yield more accurate results, but tend to be more prone to errors.

Follow this with n lines describing the roads, each containing the source, the destination, and the distance between, in order.

Lastly, enter k pairs of lines describing the vehicles, the first line of each containing the source and destination, in order, and the second, the initial battery charge, the charging rate, the discharging rate, the maximum battery capacity, and the average speed of the vehicle, in order.

This Python code implements an algorithm for optimizing the navigation of a fleet of electric vehicles between cities, considering various parameters such as charging station availability, vehicle battery levels, distances, and time constraints. The code comprises several classes and functions to achieve this optimization:

1. Dijkstra's Algorithm and Heuristic Approach:

- The `dijkstrasAlgorithm` function implements a modified version of Dijkstra's Algorithm using Min-Heaps. It calculates the shortest paths between nodes in a road network, considering distances and times. The calculated paths are stored in the `shortestPaths` matrix.

- The `generate` function uses a recursive approach to generate feasible routes for each vehicle. It incorporates charging, waiting, and moving on decision points based on the vehicle's state, such as battery level and optimal routes.

2. Vehicle and State Classes:

- The `Vehicle` class represents individual vehicles, storing information such as initial battery level, charging rate, discharging rate, maximum capacity, and average speed. It also calculates minimum road time and charging time, determining if a vehicle needs to be charged.

- The `VehicleState` class maintains the state of each vehicle during the simulation, including its arrival status, timestamp, charge level, position, and route.

3. MinHeap Class:

- The `MinHeap` class is a data structure used in Dijkstra's Algorithm for efficient heap operations.

4. State Class:

- The `State` class represents the overall state of the system, including the timestamp, node states, and states of individual vehicles.

5. Driver Function:

- The `driver` function reads input from a file (`testcases.txt` by default), initializes parameters, and executes the algorithm. It redirects the output to a file (`output.txt` by default) for further analysis.

- The results, including heuristic underestimates, heuristic routes, optimal schedules/routes, and optimal times for each vehicle, are printed to the console.

6. Global Variables:

- Global variables like `adj` (adjacency list representing the road network), `shortest Paths` (matrix storing shortest paths between nodes), `period` (time period for simulation), `currentSolution` (current optimal time), and `solution State` (state corresponding to the optimal solution) are used to maintain the algorithm's state during execution.

Output Generated

Results

The output generated are the heuristic distance and time and the routes for each vehicle in the fleet and calculates optimal routes for all the vehicles in consideration and preventing the waste in time for recharging vehicles and rerouting to the best possible route where there is availability of the charging stations.

Key Parameters and Considerations:

Several critical parameters and considerations play a pivotal role in the success of the algorithm. These include:

Charging Station Locations: Identifying optimal charging station locations along the routes is crucial. The algorithm considers the distribution of charging infrastructure to minimize deviations from the optimal path for recharging.

Vehicle-specific Parameters: Each vehicle's initial battery level, charging rate, discharging rate, maximum capacity, and average speed are considered. These parameters influence the energy consumption and overall travel time for each vehicle.

Heuristic Underestimate: The algorithm calculates a heuristic underestimate for each vehicle, incorporating the minimum road time and the required charging time. This estimate guides the decision-making process during route planning.

Feasibility Checks: The algorithm includes feasibility checks at each decision point to ensure that vehicles can reach their destinations on time. Feasibility is determined based on the combination of current battery levels, charging capabilities, and the availability of charging stations.

Justification for Algorithm Choice:

The choice of the algorithm is grounded in its ability to address the specific challenges posed by electric vehicles in an urban transportation context. Dijkstra's Algorithm provides an efficient way to find optimal paths in the road network, while the heuristic-driven approach enhances the algorithm's adaptability to electric vehicle constraints.

By integrating real-world parameters such as charging rates, battery levels, and station locations, the algorithm reflects the practical considerations of eco-driving intelligent vehicles. The focus on feasibility checks ensures that the algorithm produces viable solutions that adhere to time constraints and charging station availability.

In conclusion, the selected methodology combines established graph traversal techniques with tailored heuristics to devise an algorithm that not only optimizes energy-efficient routes for electric vehicles but also considers the practical constraints of urban transportation infrastructure. The integration of these elements allows for a comprehensive approach to intelligent vehicle navigation in an eco-friendly manner.

Conclusion

The project, titled "Electric Vehicle Routing Problem (EVRP)," aimed to design an algorithm for navigating a fleet of electric vehicles across city nodes, considering factors such as charging stations, vehicle battery, distance, and time. The following sections highlight the key findings, contributions, and suggestions for future work:

Research Findings:

- The implementation of an AI-driven algorithm leveraging Dijkstra's Algorithm and heuristic estimations proved effective in optimizing routes for electric vehicles.

- The algorithm successfully considered charging stations, battery levels, and travel times to generate eco-friendly and efficient routes for each vehicle.

Contributions:

- The project contributes a novel approach to intelligent vehicle navigation, emphasizing eco-friendly practices and energy optimization.

- The algorithm provides a robust framework for managing electric vehicle fleets, potentially reducing overall energy consumption and enhancing sustainability.

Recommendations for Future Work:

1. **Dynamic Charging Station Allocation**: Explore methods for dynamically allocating charging stations based on real-time traffic conditions, demand, and energy availability. This could enhance adaptability to changing scenarios and further optimize energy consumption.

2. **Integration of Traffic Data:** Integrate real-time traffic data into the algorithm to enhance route planning. This addition could provide more accurate predictions and improve overall traffic management.

3. **Machine Learning for Predictive Analysis:** Investigate the use of machine learning techniques to predict optimal charging times and locations based on historical data. This predictive analysis could further refine the algorithm's decision-making process.

4. **Multi-Agent System Implementation:** Consider extending the project to a multi-agent system, allowing vehicles to communicate and collaborate for more efficient route planning. This could lead to a more adaptive and cooperative fleet navigation system.

5. **Environmental Impact Assessment:** Conduct an in-depth environmental impact assessment of the proposed algorithm to quantify its contributions to sustainability. This could include measurements of carbon footprint reduction, energy savings, and overall environmental benefits.

Conclusion Summary:

In conclusion, the project presents an innovative solution for eco-driving intelligent vehicles, showcasing the effectiveness of AI-based algorithms in optimizing routes for electric vehicle fleets. The findings contribute to the growing field of intelligent transportation systems with a focus on sustainability. The recommendations for future work outline potential avenues for improvement and expansion, ensuring the continuous evolution of the algorithm to meet the dynamic challenges of urban mobility and environmental conservation.

References

^Controversial, see Moshe Sniedovich (2006). "Dijkstra's algorithm revisited: the dynamic programming connexion". Control and Cybernetics. 35: 599–620. and below part.

^ Jump up to:a b Cormen et al. 2001

^ Jump up to:a b Fredman & Tarjan 1987

^ Richards, Hamilton. "Edsger Wybe Dijkstra". A.M. Turing Award. Association for Computing Machinery. Retrieved 16 October 2017. At the Mathematical Centre a major project was building the ARMAC computer. For its official inauguration in 1956, Dijkstra devised a program to solve a problem interesting to a nontechnical audience: Given a network of roads connecting cities, what is the shortest route between two designated cities?

^ Jump up to:a b c Frana, Phil (August 2010). "An Interview with Edsger W. Dijkstra". Communications of the ACM. 53 (8): 41–47. doi:10.1145/1787234.1787249.

^ Jump up to:a b Dijkstra, E. W. (1959). "A note on two problems in connexion with graphs". Numerische Mathematik. 1: 269–271. CiteSeerX 10.1.1.165.7577. doi:10.1007/BF01386390. S2CID 123284777.

^ Jump up to:a b c d e Mehlhorn, Kurt; Sanders, Peter (2008). "Chapter 10. Shortest Paths" (PDF). Algorithms and Data Structures: The Basic Toolbox. Springer. doi:10.1007/978-3-540-77978-0. ISBN 978-3-540-77977-3.

^ Szcześniak, Ireneusz; Jajszczyk, Andrzej; Woźna-Szcześniak, Bożena (2019). "Generic Dijkstra for optical networks". Journal of Optical Communications and Networking. 11 (11): 568–577. arXiv:1810.04481. doi:10.1364/JOCN.11.000568. S2CID 52958911.

^ Szcześniak, Ireneusz; Woźna-Szcześniak, Bożena (2023), "Generic Dijkstra: Correctness and tractability", NOMS 2023-2023 IEEE/IFIP Network Operations and Management Symposium, pp. 1–7, arXiv:2204.13547, doi:10.1109/NOMS56928.2023.10154322, ISBN 978-1-6654-7716-1, S2CID 248427020

Cormen, Thomas H.; Leiserson, Charles E.; Rivest, Ronald L.; Stein, Clifford (2009) [1990]. Introduction to Algorithms (3rd ed.). MIT Press and McGraw-Hill. ISBN 0-262-03384-4.