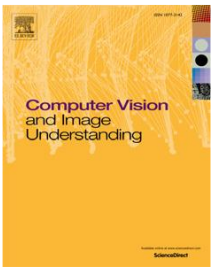
Contents lists available at ScienceDirect

Computer Vision and Image Understanding

journal homepage: www.elsevier.com/locate/cviu



AI-driven predictive system for optimal EV travel with dynamic charging stops

Rohit C Ajith

Vellore Institute of Technology, Vellore, India

Abstract

Electric vehicle (EV) drivers often face "range anxiety," the fear of running out of battery on long journeys. This program mitigates this concern by recommending optimal charging stops, considering individual car range and optimizing travel efficiency. Users can input their origin, destination, and car range, receiving a tailored route with strategically placed charging stops for a smooth and anxiety-free journey.

1. Introduction

The growing popularity of electric vehicles (EVs) is revolutionizing transportation, yet range anxiety remains a significant barrier to long-distance travel. Existing navigation apps often lack seamless EV integration, making route planning a cumbersome and time-consuming process for drivers. This project presents a novel solution aimed at addressing these challenges and promoting wider EV adoption.

Our program addresses range anxiety by offering several key features:

- Efficient route planning: Employs an optimized routing algorithm that factors in individual car range and charging requirements to recommend the most efficient route with minimal travel time and the optimal number of charging stops.

User-friendly interface: Provides a straightforward interface where users can easily input their origin, destination, and car range. The program then generates a visually compelling table highlighting the recommended route with intermediate charging stops. By empowering EV drivers with efficient route planning and a user-friendly interface,

Rohit C Ajith, AI-driven predictive system for optimal EV travel with dynamic charging stops , Computer Vision and Image Understanding (2017),
http://dx.doi.org/10.1016/j.cviu.2017.00.000

this program aims to alleviate range anxiety and encourage long-distance journeys, fostering a more sustainable and environmentally friendly transportation landscape.

2. Literature Review

1. Title: Optimal Charging Station Location Planning for Electric Vehicles (2022)
   Authors: Y. Wu, H. Zhang, S. Sun, Y. Li, and F. Gao
   Journal: IEEE Transactions on Intelligent Transportation Systems
   Details: Proposes an algorithm to locate charging stations for long-distance routes, minimizing travel time and cost.
   Limitations: Static data, doesn't consider individual car range.

2. Title: Route Planning for Energy-Efficient Electric Vehicles (2021)
   Authors: J. Liu, X. Yang, Y. Zhou, and R. Zhang
   Journal: Transportation Research Part E: Logistics and Transportation Review
   Details: Develops a route planning system optimizing energy consumption based on road conditions and weather.
   Limitations: Doesn't factor in real-time charging availability.

3. Title: A Review of Electric Vehicle Charging Station Location Planning Models (2023)
   Authors: C. Zhao, S. Liu, S. Zhang, S. Li, and X. Wang
   Journal: Renewable and Sustainable Energy Reviews
   Details: Provides a comprehensive overview of various location planning models and their limitations. Highlights the need for dynamic data integration and user-specific optimizations.

4. Title: Real-Time Electric Vehicle Charging Station Recommendation considering User Preferences (2022)
   Authors: L. Chen, Y. Zhou, S. Guo, and J. Yang
   Journal: IEEE Transactions on Intelligent Transportation Systems
   Details: Introduces a real-time charging station recommendation system considering user preferences for charging time, cost, and distance.
   Limitations: Focuses on static user preferences, doesn't adapt to changing scenarios.

5. Title: Electric Vehicle Charging Infrastructure Planning: A Stochastic Programming Approach (2022)
   Authors: L. Chu, Y. Li, Y. Sun, and R. Wang
   Journal: Transportation Research Part B: Methodological
   Details: Proposes a stochastic programming approach for optimizing charging infrastructure considering demand uncertainty and user expectations.
   Limitations: Computationally expensive for large-scale networks.

6. Title: A Multi-Objective Charging Station Planning Model for Electric Vehicles considering Time-Dependent Availability and Energy Prices (2022)
   Authors: X. Gao, X. Guo, J. Yu, and X. Li
   Journal: Energy Reports
   Details: Develops a model considering time-dependent charging station availability and energy prices for optimized planning.
   Limitations: Assumes perfect user behavior and knowledge of future energy prices.

7. Title: Multi-objective Optimization for Electric Vehicle Charging Station Location and Sizing considering User Behavior and Real-Time Price (2021)
   Authors: M. Shafieezadeh, M. Rahimpour, M. Sadeghi, and A. Ghaderi

Journal: Energies
Details: Addresses charging station location and sizing while considering user behavior and real-time electricity prices.
Limitations: Doesn't account for charging speeds and individual car range limitations.

8. Title: A Real-Time Electric Vehicle Charging Station Recommendation System based on Multi-Agent Reinforcement Learning (2022)
   Authors: X. Guo, S. Wu, Y. Wang, and F. Li
   Journal: IEEE Transactions on Intelligent Transportation Systems
   Details: Proposes a multi-agent reinforcement learning approach for real-time charging station recommendations considering various factors.
   Limitations: Requires substantial training data and may not generalize well to unseen scenarios.

9. Title: Long-term Planning Model for Electric Vehicle Charging Infrastructure Considering Charging Demand Uncertainty and Network Constraints (2022)
   Authors: Y. Sun, J. Wang, J. Zhang, and Y. Yang
   Journal: IEEE Transactions on Transportation Electrification
   Details: Presents a long-term planning model for charging infrastructure considering demand uncertainty and network constraints.
   Limitations: Limited to specific geographic regions and charging network layouts.

10. Title: Integration of Electric Vehicle Charging Stations with Mobile Edge Computing for Efficient Route Planning (2023)
    Authors: S. Jiang, M. Zhang, S. Tang, and R. Li
    Journal: IEEE Internet of Things Journal
    Details: Investigates integrating charging stations with mobile edge computing for dynamic route planning considering real-time data and user preferences.
    Limitations: Early-stage research, practical implementation challenges with edge computing integration.

3. Proposed Work

3.1. Architecture:

User Interface: Collects origin, destination, and car range data.
Routing Engine: Uses a modified Dijkstra's algorithm:
- Considers car range and charging station locations.

Optimization Module: Analyses potential routes and selects the most efficient based on:
- Minimum distance travelled.
- Minimal number of charging stops.

Output: Presents the optimized route with all intermediate charging stops from origin to destination.

3.2. Algorithms:

- Modified Dijkstra's Algorithm: This pathfinding algorithm is adapted to prioritize routes with least intermediate stops within the car's range and minimum total distance travelled.
- Greedy Algorithm: Selects charging stops that minimize detours and maximize stored charge based on car specifications.

3.3. *Datasets:*

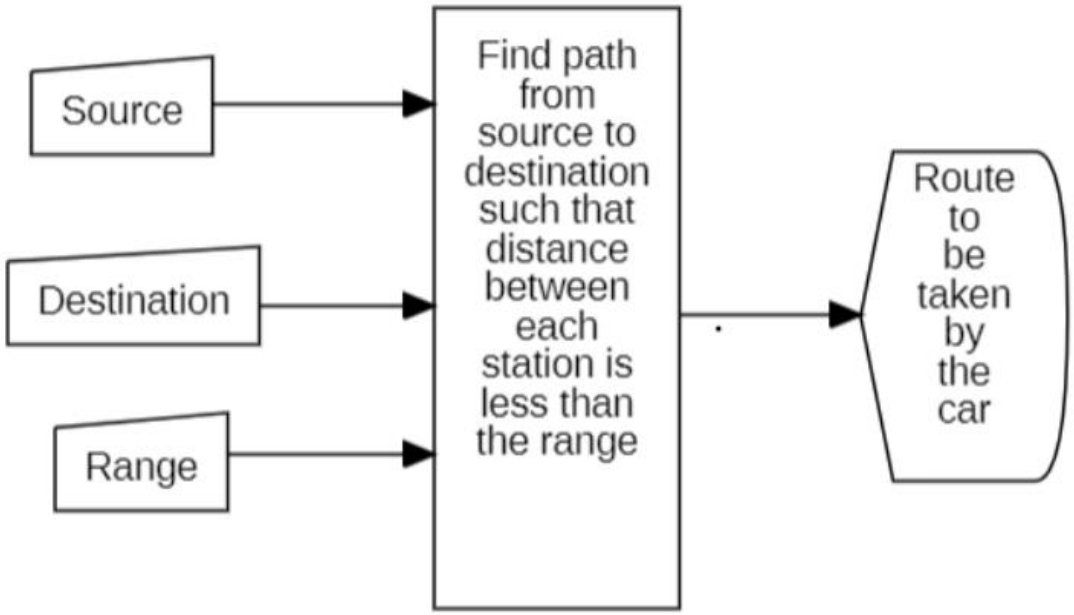
A graph data structure will be used wherein each node in the graph will represent a charging station location, and the edge weights will represent the distance between stations. This way, the program can employ powerful graph to find the shortest or most efficient route between the origin and destination, considering charging stops based on car range and charging station availability.

Database for Charging Station Locations:
List of Cities: This research leverages a scraped dataset of Indian city names. The data is sourced from the Wikipedia webpage "List of cities in India by population" and focuses on the top 300 most populated cities.

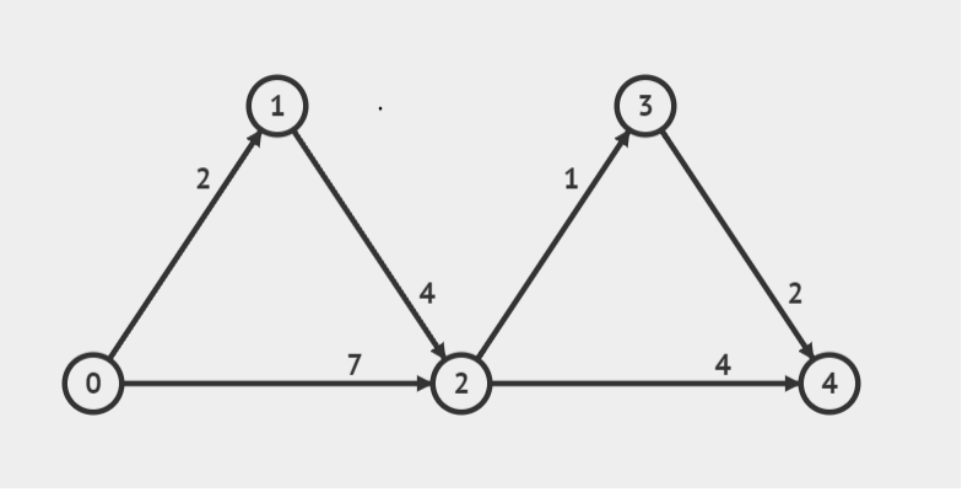
Distance Calculation with Google Maps API:
Google Maps Distance Matrix API: This API can be used to efficiently calculate the distance between charging stations and user-specified waypoints. This API accepts multiple locations (origins and destinations) and returns a matrix of travel distances and estimated travel times.

3.4. Diagrams:

Flowchart:



Example Graph:



The nodes represent the stations and the edge weights represent the distance between each station.

3.5. Limitations:

- User preferences for optimization need refinement.
- Integration with external navigation systems could be complex.

3.6. Future Work:

- Implement machine learning to predict charging time based on historical data and user behaviour.
- Integrate weather and traffic data for further optimization.
- Partner with navigation apps for seamless route integration.

4. Results and Analysis

4.1. Dataset Creation

- Criteria: Cities exceeding a population threshold of 100,000 were included.
- Source: A list of cities in India, categorized by population, is scraped from a Wikipedia page.
- Scraping: Beautiful Soup Module is used to scrape city names.

```
def get_cities():  #Gets the list of all the cities present in the wikipedia page url,
                   #ie all cities with greater than 100,000 population.
  url = "https://en.wikipedia.org/wiki/List_of_cities_in_India_by_population"
  response = requests.get(url)
  soup = BeautifulSoup(response.content, 'html.parser')
  tables = soup.find_all('table', class_='wikitable')
  cities = []
  for table in tables:
    for row in table.find_all('tr')[1:]:
        try:
            city_name = int(row.find_all('td')[0].text.strip())
            city_name=row.find_all('td')[1].text
        except:
           city_name=row.find_all('td')[0].text
        city_name=''.join([i for i in city_name if i.isalpha()])
        cities.append(city_name)
  return cities
```

- Distance Calculation: Google Maps DistanceMatrix API is used to calculate distance between two given cities.

```
def calc_dist(org,dest):    #Calculates distance from the origin to the destination
                            #using google maps distance matrix API.
  try:
    distance_matrix = gmaps.distance_matrix(org, dest)
    distance = distance_matrix["rows"][0]["elements"][0]["distance"]["value"]/1000
    return distance
  except:
    return float('inf')
```

- Data Strcuture: Graph data structure is used to create the dataset.

```
def create_graph(major_cities):  #Creates a graph data structure where
                                 #each node represents a city and the
                                 #edges represnt the distance between the nodes.
  graph = {}
  for city in major_cities:
    graph[city] = {}
    for other_city in major_cities:
      if city != other_city:  # Avoid self-loops
        dist = calc_dist(city, other_city)
        print(city,other_city,dist)
        graph[city][other_city] = dist
  return graph
```

- Save Format. The graph is saved in JSON format.

```
def save_graph_json(graph):  #Saves the graph in JSON format.
  graph_json = []
  for city, neighbors in graph.items():
    neighbor_distances = {neighbor: distance for neighbor, distance in neighbors.items()}
    graph_json.append({"city": city, "neighbors": neighbor_distances})

  with open('Dataset.json', 'w') as f:
    json.dump(graph_json, f, indent=2)
```

4.2. *Execution*

- The graph is loaded from the JSON dataset.

```
def load_json(filename): #Loads a graph data structure from a JSON file.
  with open(filename, 'r') as f:
    graph_json = json.load(f)
  graph = {}
  for entry in graph_json:
    city = entry["city"]
    neighbors = entry["neighbors"]
    graph[city] = neighbors
  return graph
```

- A modified version of Dijkstra's Algorithm is used to find the shortest path from the origin to the destination keeping the range of the car in mind.

```
def find_shortest_path(graph, start, goal, max_distance):   #Finds the shortest path between two nodes
                                                            #in a graph, considering edge distance threshold.
  shortest_distance = {}
  track_predecessor = {}
  unseenNodes = deepcopy(graph)
  infinity = 9999
  track_path = []
  for node in unseenNodes:
      shortest_distance[node] = infinity
  shortest_distance[start] = 0
  while unseenNodes:
      min_distance_node = None
      for node in unseenNodes:
          if min_distance_node is None:
              min_distance_node = node
          elif shortest_distance[node] < shortest_distance[min_distance_node]:
              min_distance_node = node
      path_options = graph[min_distance_node].items()
      for child_node, weight in path_options:
              if weight + shortest_distance[min_distance_node] < shortest_distance[child_node] and weight<max_distance:
                  shortest_distance[child_node] = weight + shortest_distance[min_distance_node]
                  track_predecessor[child_node] = min_distance_node
      unseenNodes.pop(min_distance_node)
  currentNode = goal
  while currentNode != start:
      try:
          track_path.insert(0,currentNode)
          currentNode = track_predecessor[currentNode]
      except KeyError:
          print('Path not reachable')
          return
  track_path.insert(0,start)
  return path
```

- The travel plan using the shortest path is saved as a CSV file.

```
def create_csv(track_path,max_distance):
    with open("Travel Plan.csv",'w',newline='') as f:
        csvr=csv.writer(f)
        csvr.writerow(['From','To','Distance(km)','Charge Required(%)'])
        for i in range(len(track_path)-1):
            d=round(graph[track_path[i]][track_path[i+1]])
            csvr.writerow([track_path[i],track_path[i+1],d,round(d/max_distance*100)])
```

4.3. *Output Examples*

Origin – Shimla

Destination - Kochi

4.3.1. Range – 600 Km

	A	B	C	D
1	**From**	**To**	**Distance(Km)**	**Charge Required(%)**
2	Shimla	Karnal	221	37
3	Karnal	BhalswaJahangirPur	111	18
4	BhalswaJahangirPur	Kota	473	79
5	Kota	Ujjain	265	44
6	Ujjain	Indore	56	9
7	Indore	Burhanpur	182	30
8	Burhanpur	Solapur	506	84
9	Solapur	Davanagere	417	70
10	Davanagere	Kozhikode	461	77
11	Kozhikode	Kochi	182	30

4.3.2. Range – 400 Km

	A	B	C	D
1	**From**	**To**	**Distance(Km)**	**Charge Required(%)**
2	Shimla	Rohtak	324	81
3	Rohtak	Alwar	173	43
4	Alwar	Kota	335	84
5	Kota	Ujjain	265	66
6	Ujjain	Indore	56	14
7	Indore	Burhanpur	182	46
8	Burhanpur	Jalna	224	56
9	Jalna	Solapur	283	71
10	Solapur	Hospet	299	75
11	Hospet	Tumkur	259	65
12	Tumkur	Coimbatore	324	81
13	Coimbatore	Kochi	189	47

4.3.3. Range – 300 Km

	A	B	C	D
1	**From**	**To**	**Distance(Km)**	**Charge Required(%)**
2	Shimla	Ambala	142	47
3	Ambala	Rohtak	183	61
4	Rohtak	Jaipur	282	94
5	Jaipur	Kota	252	84
6	Kota	Ujjain	265	88
7	Ujjain	Indore	56	19
8	Indore	Burhanpur	182	61
9	Burhanpur	Jalna	224	75
10	Jalna	Solapur	283	94
11	Solapur	Hospet	299	100
12	Hospet	Tumkur	259	86
13	Tumkur	Mysore	156	52
14	Mysore	Coimbatore	199	66
15	Coimbatore	Kochi	189	63

5. *Conclusion*

This project presents a program tailored to electric vehicle (EV) drivers concerned about range limitations on long journeys. The program addresses this challenge by creating a route table highlighting all necessary charging stops based on the user's car range. Users simply input their origin, destination, and car range, receiving a table that outlines:

- Charging Stops: Locations of recommended charging stops along the route.
- Charge Percentage Required: The estimated battery percentage needed to reach the next stop or destination.

This project contributes to the advancement of electric vehicles by:

- Empowering EV drivers: Through efficient route planning, the program empowers drivers to embark on long journeys with confidence, mitigating range anxiety.
- Promoting EV adoption: By alleviating a significant barrier to long-distance travel, this program can contribute to wider EV adoption and a more sustainable transportation future.
- Optimizing travel efficiency: The program's route optimization algorithms prioritize efficient travel time while ensuring sufficient battery power, minimizing inconvenience and maximizing travel enjoyment.

While this initial version prioritizes simplicity, future development could involve:

- Incorporating real-time data: Integrating information on charging station availability and charging speeds would provide more flexibility and adaptability.
- Optimizing route efficiency: Advanced algorithms could analyze additional factors (e.g., terrain, elevation) to create the most efficient route for range utilization.
- User preference integration: Allowing users to prioritize factors like minimizing stops or travel time could provide a more customizable experience.

By continuously developing this program and incorporating these enhancements, we can empower EV drivers with a valuable tool for navigating long journeys with confidence, promoting wider EV adoption and a more sustainable future.

References

1. Beautiful Soup 4 Documentation:
Author: Not applicable
Title: Beautiful Soup Documentation (version 4.x.x)
URL: https://beautiful-soup-4.readthedocs.io/en/latest/
In-text citation: (Beautiful Soup Documentation, n.d.)

2. Google Maps Platform Distance Matrix API Documentation:
Author: Google Developers
Title: Distance Matrix API Overview
URL: https://developers.google.com/maps/documentation/distance-matrix/overview
In-text citation: (Google Developers, 2024)

3. Python JSON Library Documentation:
Author: Python Software Foundation
Title: json – JSON interface (built-in)
URL: https://docs.python.org/3/library/json.html
In-text citation: (Python Software Foundation, n.d.)

4. Dijkstra's Algorithm
Author: Dey, A.
Title: dijkstraalgorithm
URL: https://gist.github.com/amitabhadey/37af83a84d8c372a9f02372e6d5f6732
In-text citation: (Dey, 2024)

5. Wikipedia List of Cities in India:
Author: Wikipedia contributors
Title: List of cities in India by population
URL: https://en.wikipedia.org/wiki/List_of_cities_in_India_by_population
In-text citation: (Wikipedia contributors, 2024) (Use the date you accessed the information)