Reduction in cost through efficient path planning in electric vehicles

Final Report

Submitted by:-

Kavita	19NA10011
Yathartha Soneji	19HS20051
Sneha Kharya	19HS20044

1. **Problem Statement**

Problem Description

One of the primary issues that the world is currently experiencing is the depletion of fossil fuel sources, environmental degradation, an energy crisis, and the catastrophic implications of climate change. Electric vehicles (EVs) can drastically alter future road traffic and, to some part, overcome the difficulties that we presently face because of their efficiency and capacity to run on regenerative energy sources such as solar and wind power. One of the distinguishing features of electric cars is their ability to recover part of their kinetic and/or potential energy during deceleration stages. However, limited battery capacities and long recharge durations hinder the more extensive and widespread use of EVs. Reports show that an average EV has a cruising range of about only 160-190 Kms.

The distinctive qualities of EVs influence search algorithms used in navigation systems and route planners. Because of variables such as low battery capacity, long recharging times, and limited cruise range, the goal switches to identifying energy-efficient routes rather than merely short and quick ones. The objective of driving EVs in an energy-efficient manner raises severe computational hurdles for navigation systems and route planners.

According to the problem statement, we are given a city network where we need to route a set of electric vehicles from their respective sources to destinations subjected to constraints, such that max{Tr} is minimized, where t=Tr is the time when an EV Pr reaches its destination. Our search algorithm is inspired by the Dijkstra algorithm, which has a time complexity O(n^2), to find the shortest path between the nodes in the graph, which ultimately helps us get the optimal solution.

Formal Problem Statement

The main goal is to find a set of paths for EVs in a network of cities using a Dijkstra and A* algorithm.

Seen from a single EVs perspective: It wants an optimal path through the network of cities for which the time taken to move from source city to destination city is minimum.

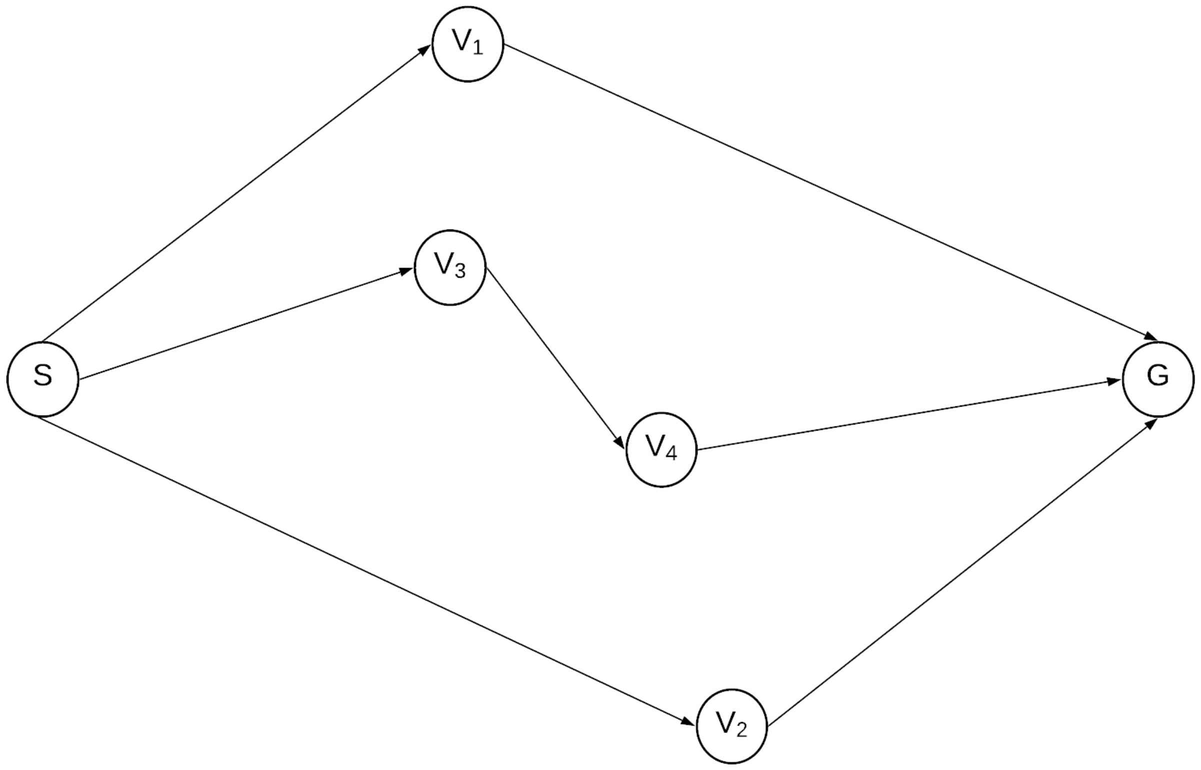
Seen from a multi EVs system perspective:

We want an optimal path for all EVs. So, the total time taken by the EV that takes the maximum time to reach its destination(or we can say the EV which will reach its destination at last) should be minimized. In notations, if

$\{T_1, T_2, T_3, \ldots.., T_n\}$ is the time taken by the EVs respectively, then the max$\{T_r\}$ should be minimized.

2. AI Modelling

To keep the problem simple let us start with finding the optimal path of a single EV. So, initially, we can use a greedy method to find an optimal path from the source node to the destination node through the network of cities that is **inspired** by Dijkstra's Algorithm which uses **heuristic** as the **distance between the nodes**. The figure below represents an example of this situation



Before analysing the problem further let us define the variables we will be using:

1. D:- Total distance (from source to destination)
2. s:- Average speed
3. S:- Source node
4. G:- Destination node
5. B:- Battery charge status initially
6. c:- Charging rate for battery at a charging station (energy per unit time)

7. h:- Discharging rate of battery while travelling (distance travel per unit charge)
8. M:-Maximum battery capacity
9. T:- Optimal time for rth EV

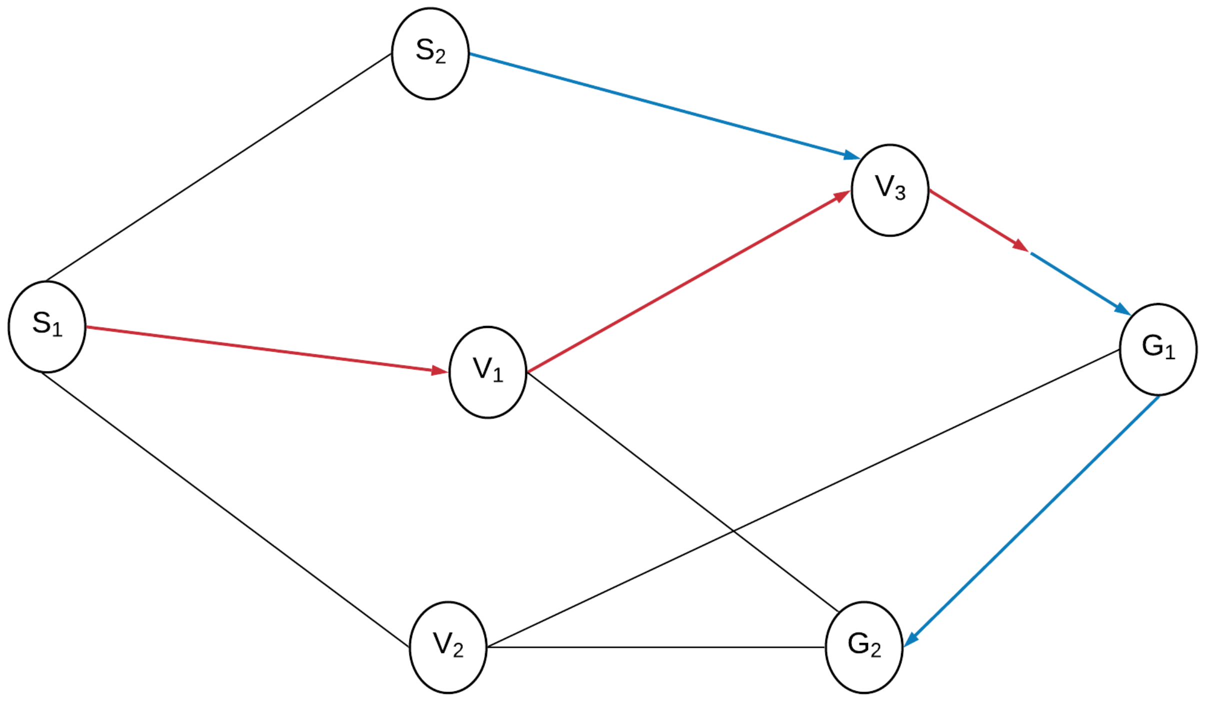
Suppose, the arrowed path shows the optimal path generated by our search Algorithm
The time taken by our EV to cover the total distance D at an average speed s will be D/s. But wait! it does not incorporate the time taken to charge the battery. The vehicle may charge any number of times before reaching the station. Hence the total time taken would be:

Time taken = $\sum$ charging time at different stations + D/s

charging time at one station = Charge needed / c

Charge needed = (D-d)/h

Now, let us move on to the case of two EVs. Suppose in addition to our previous EV EV1 we now have another EV EV2. The situation can be seen in the below graph where the red arrows represent the path of EV1 and the blue arrows represent the optimal path of EV2.(**Note** we have not written the distance between the cities but it is expected by the reader to suppose that there are some values for them still they aren't shown in the figure but are taken into account while implementing the our search algorithm).



Again, let us go through the variables we will be using:

Before analysing the problem further let us define the variables we will be using:

1. D_r:- Total distance for rth EV (from source to destination)
2. s_r:- Average speed for rth EV
3. S_r:- Source node for rth EV
4. G_r:- Destination node for rth EV
5. B_r:- Battery charge status initially for rth EV
6. c_r:- Charging rate for battery at a charging station for rth EV(energy per unit time)
7. h_r:- Discharging rate of battery while travelling for rth EV(distance travel per unit charge)
8. M_r:-Maximum battery capacity for rth EV
9. T_r: Optimal time for rth EV

Initially we assume the paths of V1 and V2 to be independent. So just like the previous case the time taken by the vehicles will be:

Time taken by EV_1 (T_1) = $\sum$ charging time at different stations + D_1/s

Time taken by EV_2 (T_2) = $\sum$ charging time at different stations + D_2/s

Since both T_1 and T_2 are optimized so max{T_1, T_2} is optimized.

Here, we are neglecting the case when both the vehicles come to the same charging station at the same time. If they arrive at the same station at the same time then we have to give preference to any one of the EV. In order to minimise the maximum time we will have to minimise max{T_1, T_2}. So the EV have the more T_r will be given preference for charge first.

Note: if a car has lower Tr but has reached a city where it needs to be charged before the preferred one(i. E whose D/s is more)), then , it can start its charging while the other has to come, as soon as the more preferred comes, it will stop, and the preferred one will start charging. Now the other one has two options, either it can wait for the preferred one charging completes or choose another path that takes less time than waiting time.

Let's suppose that T_1<T_2. Then EV_2will be the preferred EV for charging. This will change the Time taken by EV_1. If EV_1 waits the time taken will be:

t_a = $\sum$ $charging\ times\ at\ different\ stations$+ waiting time + D/s

But if it takes another path the time taken will be

t_b= $\sum$ $charging\ times\ at\ different\ stations$ + D_{new}/s

Here, D_{new} will be calculated by again applying Dijkstara at the charging station where it was waiting to destination

Hence the optimal time for EV_1 will be the minimum of the above two cases, i.e

T_1 = min(t_a,t_b)

Since the path of T_2 is already optimised the time taken will be the same as previously discussed.

T_2 = $\sum$ $charging\ times\ at\ different\ stations + D/s$

Since T_2(AS WE SUPPOSED) is the max{T_1, T_2} and is optimized, so we can say that max{T_1, T_2} is optimized.

Note: T_1 is not optimized here but our goal is to optimize max{T_1, T_2} and as we calculated T_2 was max{ T_1, T_2} and T_2 is optimized so we have succeeded in our goal.

Now we will Generalize the above discussion for a multi EV system. Suppose we have given a graph of cities{v_1,v_2,v_3,v_4,....,v_n} and a set of EVs {EV_1,EV_2,... EV_n}The other variables used are the same as given in the case of two EVs

The sequence of steps followed will be:

S1 - we will first apply our search algorithm for each EV separately and generate their shortest time taking the path from their respective sources to their respective destinations considering that no issues are to be resolved there.

S2 - We will check where they need to charge separately for each as we did for one EV system and resolve charging issues that were concerned for one EV system.

S3 - Check all colliding points (colliding points here means the cities where multiple EVs are to be charged, the same issues we looked after in the two EVs system).

S4- Resolve them by giving preferences to the EVs which have more time to reach their destination. Also, we will check all the different cases as stated in the two EV systems above. i.e check different paths or wait there only, compare and proceed accordingly, the resolving methods are briefly discussed in the above two EVs system.

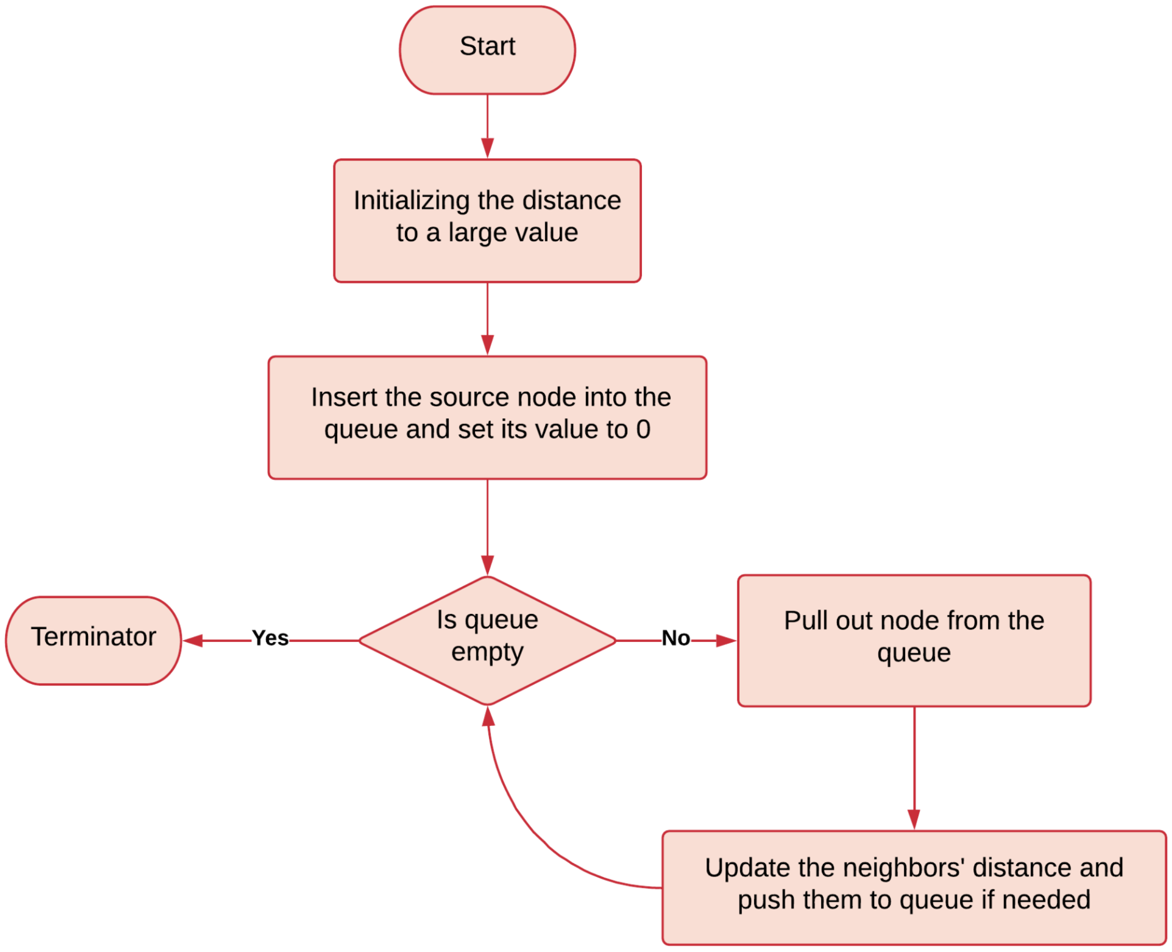
Since max{T_r} will be getting preference wherever there is an issue and will follow its original optimized path so our goal to minimize max{T_r} is achieved.

This was the complete thought process and analysis done by our team to solve the problem. The next section will be about the proposed solution to the problem.

3. Solution Approach

According to the problem statement, we were asked to find an electric vehicle route that minimised max{Tr}, the time taken to reach the destination from the start of the journey at t = 0 by all vehicles from their respective sources to destinations, and we were also asked to find the best algorithm to do so. There are various algorithms that are used for the electric vehicle routing problem but every algorithm used in electric vehicle routing follows the same fundamental flow chart:

1. start at the source node
2. examine the subsequent nodes one by one
3. travel to the node that is closest to the source node.


Start
Initializing the distance to a large value
Insert the source node into the queue and set its value to 0
Is queue empty
Yes
Terminator
No
Pull out node from the queue
Update the neighbors' distance and push them to queue if needed

The first step is to initialize the distances between any two nodes with a large number and then insert the source node into the queue. We next perform iterations until the queue gets empty. For each iteration, we remove a node from a queue that has the shortest distance from the source node.

After that, we visit all the neighbours of the visited node and look at the new distance we have been able to reach. If a new distance is better than an old one, we change the distance of the node which we are on and push it into the queue. Finally, the algorithm moves forward to perform one more iteration until the queue is empty.

We obtain the shortest path from the source node to all the other nodes in the graph at the completion of the algorithm. However, there is a difference between algorithms depending on whether the graphs to be evaluated are unweighted or weighted, and we can't utilise any algorithm for either situation. We have a few algorithms, but the two primary algorithms are A* and Dijkstra's algorithm, which finds the shortest path in unweighted graphs and weighted graphs, respectively.

1. **Dijkstra Algorithm:** In a graph with weighted edges, Dijkstra's Algorithm seeks to find the shortest path between two nodes. Only if the edge weights are positive does it operate. It operates by creating the shortest path from the source node to each node one at a time. When we are given weighted graphs, it is not necessarily the case that the shortest path is between nearby nodes. However, a neighbour with a shorter edge cannot be reached in any other shorter way, because all other edges have larger weights, so walking on them alone can increase the distance. This is the principle behind the Dijkstra Algorithm, which uses a greedy approach, in which we select the next node with the shortest path at each step. We update the cost and add surrounding nodes to the queue at each step, and we do it with the help of the priority queue. However, in the case of unweighted graphs, this algorithm may not be the best. When we arrived at a node through a different path in an unweighted network, we were confident that the initial path was the shortest, which is not necessarily the case with weighted graphs.

   If we arrive at a node via a shorter path, we must update its distance and add it to the queue, implying that we can visit the same node several times and add it to the queue numerous times. As a result, we should compare the visited node's cost to its stored value. If the distance between the visited node and the one currently recorded is greater, this indicates that the supplied node was added earlier, then we have found a shorter path and updated it.

   The Dijkstra algorithm ends as soon as the destination node is labelled, allowing it to discover the shortest path. We only visit each node's neighbours once, just as we only visit edges once. Also, we used a priority queue that has the time complexity of O(n^2) for push and pop operations.

2. **A* Algorithm:** The A* Algorithm is a version of the Dijkstra algorithm, except instead of using the optimum search mechanism, it uses heuristics. It decreases processing time by limiting the search space. For real-time applications, extensions of A* such as RTA* and LRTA* have been developed. They use the heuristic function to calculate the direct distance between the source

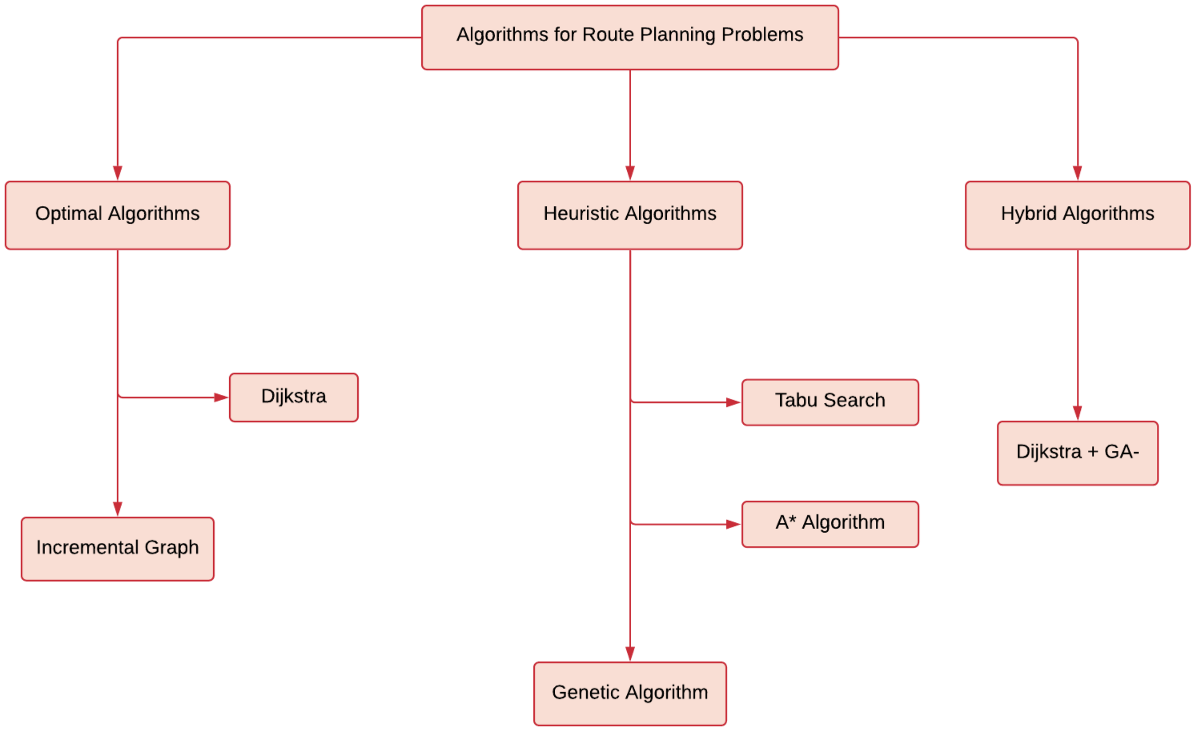
and the destination. We usually aim to limit the number of visited edges when dealing with unweighted graphs. In weighted graphs with identical weights for all edges, A* works very well, i.e. it finds the shortest path appropriately. However, this isn't the case for many graphs since the weights for all of the edges in the graphs aren't the same.

The main approach of the A* is to always start from the node which we already reached as it is in FIFO (First in first out) and it uses a queue.

3. **Genetic Algorithms (GA):** Routing and optimization search problems are solved using Genetic Algorithms. It employs meta-heuristics and, as the name implies, it simulates the way species evolve and adapt to their environment in accordance with the Darwinian principle of natural selection. When applied to vehicle routing issues, it creates a random heuristic population and repeats the cycle a number of times, and the classical solution is updated. It is feasible to analyse the route in a short amount of time utilising other routes and limitations in the search since it has routes during a population search.
4. **Tabu Search:** Tabu search is a local search-based meta-heuristic that may be used to solve route planning issues with several iterations. In every iteration, the best solution neighbouring to the present node is updated to the current solution, even if leads to the increment of the solution cost. As a result, is not the best method to use since a bad optimal solution is mitigated. It keeps track of the values it has visited in a list, making it easier to avoid visiting the same node repeatedly. The search for new solutions in this algorithm terminates after a certain number of iterations and it doesn't care about improving the best-known solution.

5. **Hybrid Genetic Algorithms:** To solve multi-objective problems, Hybrid Genetic Algorithm combines the Genetic Algorithm with Dijkstra Algorithm. It provides solutions that meet three criteria: optimal travel time, driving ease, and optimal distance driven along the route. The Dijkstra algorithm is used to determine the initial routes when using the Genetic Algorithm in the road system. This approach generates subsequent routes from the initial population using a Genetic Algorithm.

Above we defined some of the few algorithms used for route planning problems. These algorithms are classified into three categories based upon their technique to explore the solution space. They are classified as Optimal approach-based algorithms, heuristic approach-based algorithms, and hybrid approach-based algorithms. Moreover, A* is a generalized case of Djikstra with heuristic value, h(n) = 0



Optimal approach-based algorithms ensure that the best solution is found by examining the whole solution set. Dijkstra and Incremental Graph are two of the most common optimum approach-based algorithms. Both of them look for the shortest route between two nodes. The approximation optimal solution, which is near to the global optimal solution, is obtained using heuristic approach-based algorithms, which investigate all accessible options. Tabu search, A* Algorithm, and Genetic Algorithm are some of the most often utilised heuristic approach-based algorithms. These identify routes based on constraints that take less time to compute and provide the optimum route for the given constraints.

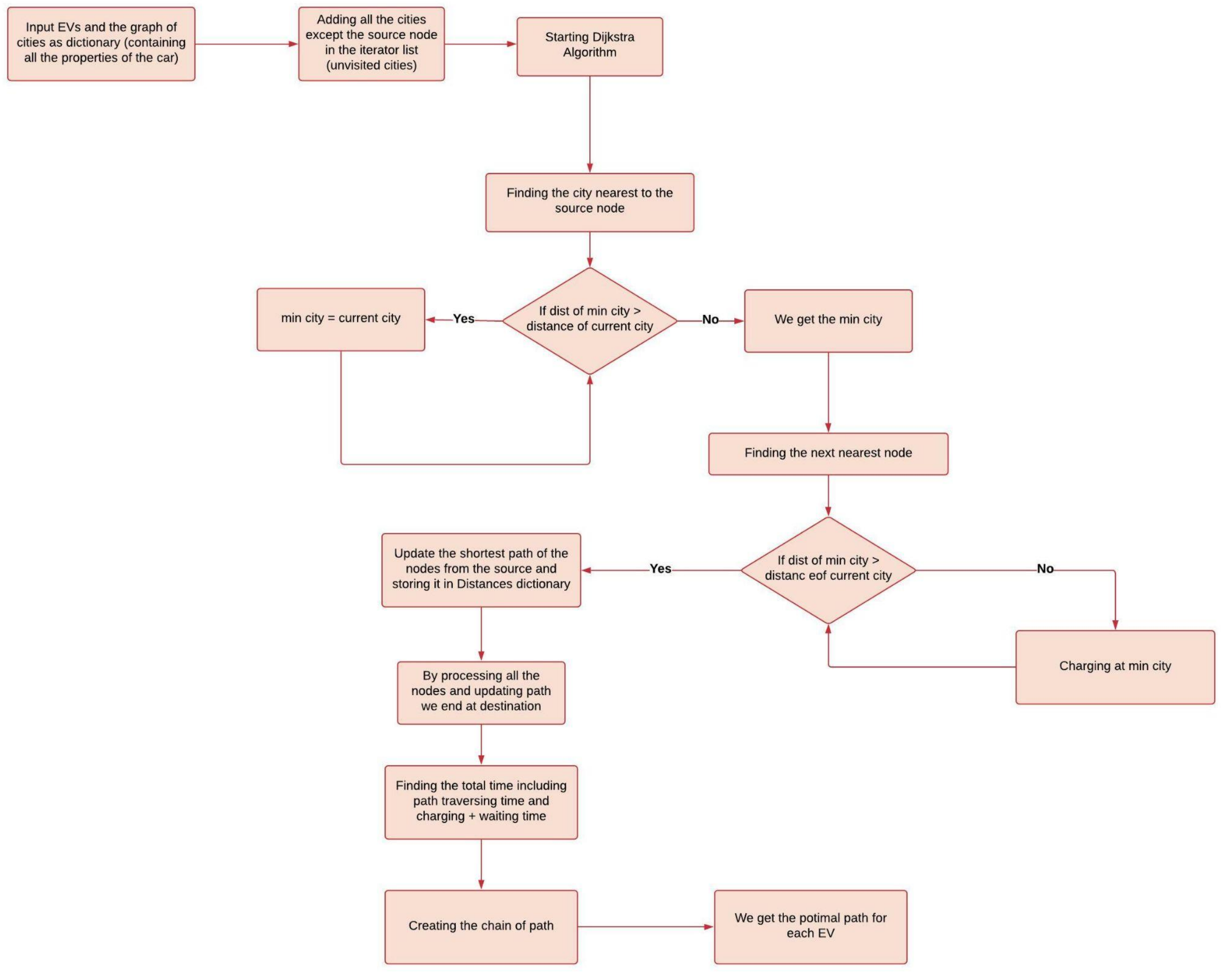
We used the hybrid approach-based algorithm for our problem statement, and we used the properties of both the Djikstra and GA because we had to consider various metrics that are best implemented using a hybrid approach for planning electric vehicle's route such that max{Tr}, the time taken to reach the destination from the start of the journey at t = 0 by all the vehicles from their respective sources to destinations, is minimised, and we had to find the optimal route. We can acquire the best feasible path with the least amount of energy by using this method. We also discussed the performance of other routing strategies.

In the road map graph, we applied the Djikstra algorithm's efficiency. For this one-to-one problem, it is better to use Dijkstra + GA to find the shortest path because it terminates as soon as we find the destination node, and it is better than others because other algorithms can only find the shortest path after exploring all the paths, and those who don't do that give solutions that are close to optimal but not globally optimal.

It's also a heuristic algorithm with a particular case where h(n) equals 0. For starters, because edge costs may be negative as in the case of A* Algorithm due to recuperation, except in our case of the algorithm.

Second, since charging time affects edge costs in our scenario. Third, because our problem's battery capacity is restricted, edge cost is not just the sum of edge costs. We also can't utilise the Dijkstra Algorithm directly since the battery can become entirely depleted at times, which is a metric to solve. We can't utilise the A* method directly since it provides the best solution for unweighted and weighted with equal weight networks, saves the visited nodes in a simple queue, and visits all nodes with costs less than the goal's costs while being slower than the others.

DETAILED ARCHITECTURE OF OUR ALGORITHM


Input EVs and the graph of cities as dictionary (containing all the properties of the car)
Adding all the cities except the source node in the iterator list (unvisited cities)
Starting Dijkstra Algorithm
Finding the city nearest to the source node
min city = current city
Yes
If dist of min city > distance of current city
No
We get the min city
Finding the next nearest node
Update the shortest path of the nodes from the source and storing it in Distances dictionary
Yes
If dist of min city > distanc eof current city
No
Charging at min city
By processing all the nodes and updating path we end at destination
Finding the total time including path traversing time and charging + waiting time
Creating the chain of path
We get the potimal path for each EV

4. APPENDIX

Pseudo-code we used for finding the best possible optimal and minimum energy usage path:

```
[S] = Dijkstra(Graph, source, target time[source]):
    create node set V
For each node v in Graph:
    dist[v] ← Infinity
    prev[v]← Undefined
    add v to Q
End for
dist[source] ← 0
uo ← source
While Q is not empty:
    u ← node in Q with min dist[u]
    remove u from Q
    If prev[u] is defined
        uo ← prevlu]
        cost(uo, u) ← output of Eq. (1) with To
        timeȩu] ← timeȩuo] + cost(uo, u)
    End if
    For each neighbor v of u:
        cost (u, v) ← output of Eq. (1) with time[u]
        alt ← dist[u] + cost(u, v)
        If alt < dist[v]:
            dist[v] ← alt
            prev[v] ← u
            time[v] ← time[v] + cost(u, v)
        End if
    End for
End while
return dist[], prev[]
```

```
S ← empty sequence
u ← target
While prev[u] is defined:
    insert u at the beginning of S
    u ← prev[u]
End while
insert u at the beginning of S
```

Pseudo-code for Constrained Generic shortest path:

```
input: A directed graph G = (V.E), weight function e: E → Z, source vertex strategy S,
maximum capacity Cmax and initial Us,
output: A prefix bounded shortest path tree from s with respect to absorp
begin
for each vertex v in V do
    d(v) ← ∞;
    p(v) ← null;
d(s) ← 0 + Us;
Q ← {s};
while Q ≠ (/) do
    choose u from Q with strategy S;
    Q ← Q \ {u};
    for each successor v of u do
        d' ← d(u) + c(u, v);
        d' ← max (d',0);
        if d' < d(v) and d' ≤ Cmax then
            d(v) ← d';
            p(v) ← u;
            Q ← Q U {v};
end
```

5. REFERENCES

- F. B. Zhan and C. E. Noon, Shortest Path Algorithms: An Evaluation Using Real Road Networks, Transportation Science, vol. 32, pp. 65-73, Feb. 1998.
- R. E. Korf, Real-time heuristic search, Artificial Intelligence, vol. 42, pp. 189-211, Mar. 1990.
- F. Glover and M. Laguna, Tabu Search, July 1997.