AIFA ASSIGNMENT SPRING 2021

ELECTRIC VEHICLES ROUTING PROBLEM

SOLUTION REPORT

Submitted by

SIDDHARTH RANJAN BAJPAI(19HS20043)

NAMYA SWARNKAR(19HS20029)

ADITYA TIWARI(19MA20001)

TUSHAR KHOKHAR(19HS20046)

TABLE OF CONTENT

OVERVIEW

In this document, we first describe the goals and objectives. Then explain certain terminologies. Then we analyze the problem by breaking it into smaller and simpler parts and combine it all and then propose a solution that intends to optimize the time which is asked in the problem.

Introduction

Dwindling fossil fuel reserves, environmental pollution, energy crisis, and the severe consequences that arise with climate change is one of the major problems that the world is facing at present. Electric vehicles(EV), due to their efficiency and their ability to run on regenerative energy sources such as solar and wind power, definitely have the potential to significantly shape the road traffic of the future and thus resolve the problems that we currently face to a certain extent. One of the unique things about electric vehicles is that they can recover some of their kinetic and/or potential energy during deceleration phases. However, more extensive and widespread use of EVs is still hindered by limited battery capacities and long recharge durations. Reports show that an average EV has a cruising range of about only 160-190 Kms.

The unique characteristics of EVs have an impact on search algorithms used in navigation systems and route planners. Due to the factors such as limited battery capacities, long recharge durations and limited cruising range, the aim shifts towards finding energy-efficient routes and not just only short and fast routes. The goal of energy-efficient driving of EVs creates formidable algorithmic challenges 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(n2), in order to find the shortest path between the nodes in the graph, which ultimately helps us in getting the optimal solution.

Problem Description: Goal and Objective

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

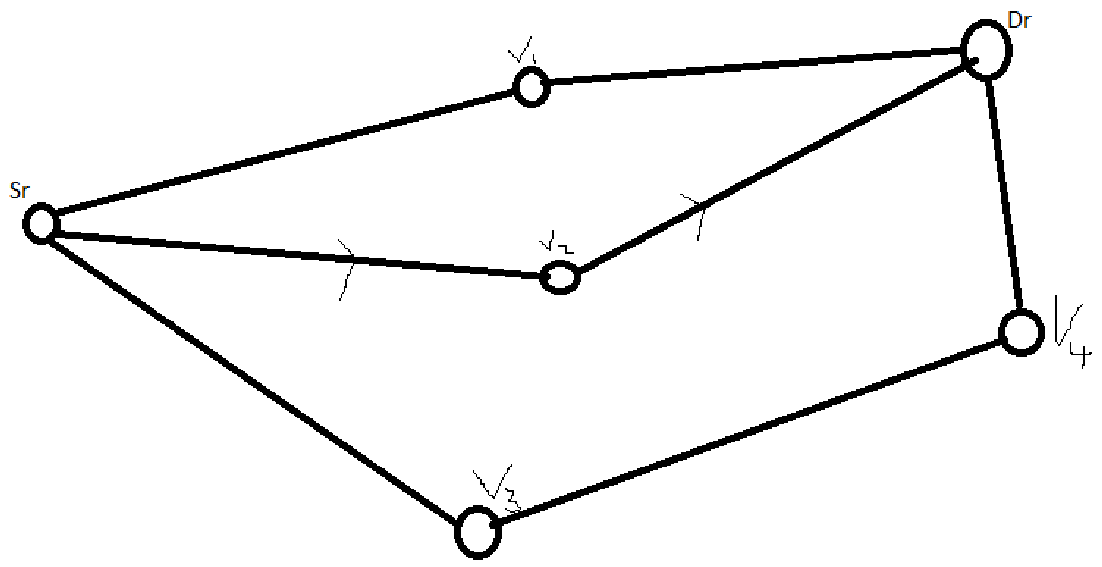
Seen from a single EVs perspective: It wants an optimal path through the network of cities for which the time taken to move from source city to destination city is minimum.

Seen from a multi EVs system perspective: We want an optimal path for all EVs for which the EV that takes the maximum time to reach its destination(or we can say the EV which will reach its destination at last), so its total time taken should be minimized. In notations, if {T1,T2,....Tn} is the time taken by the EVs respectively then the max{Tr} should be minimized.

PROBLEM ANALYSIS

Let us break the problem into simpler parts.
Consider that there is only one EV and its source and the destination node are given.
The graph of cities has been defined and the properties of the EV(which are Br,cr,dr, Mr,s) are given.

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**. Figure below represents an example of this situation.



Suppose, the arrowed path shows the optimal path generated by our search Algorithm (Note: the path generated by our Dijkstra inspired search algorithm is the most optimal will be proved in the later section of the document)

Suppose the total distance that EV has to travel is D from source to destination through the optimal path. But wait the Battery of the EV can go dead in between the path. So we have to check that can the EV with the initial Battery Status travel from source to destination or not. For this, we have two cases -
<u>Case1</u>: It can travel from source to destination with the initial battery status: so it's a SUCCESS!

Total time taken = $\frac{D}{s}$(where D is total distance and s is the average speed)

<u>Case2</u>: It cannot travel from source to destination with the initial battery status: This means it will stop somewhere in the path., let say at some d from source, mathematically writing

B = $\frac{d}{s}$(B is initial battery status)

So, intuitively what we can do is charge the battery at the previous city up to which it can travel the rest of the distance. Mathematically,

Charging needed = $\frac{D-d}{h}$(*h* is discharging rate of the EV suppose)

Charging time =$\frac{Charging\ needed}{c}$ (*c* is charging rate of the EV at a station)

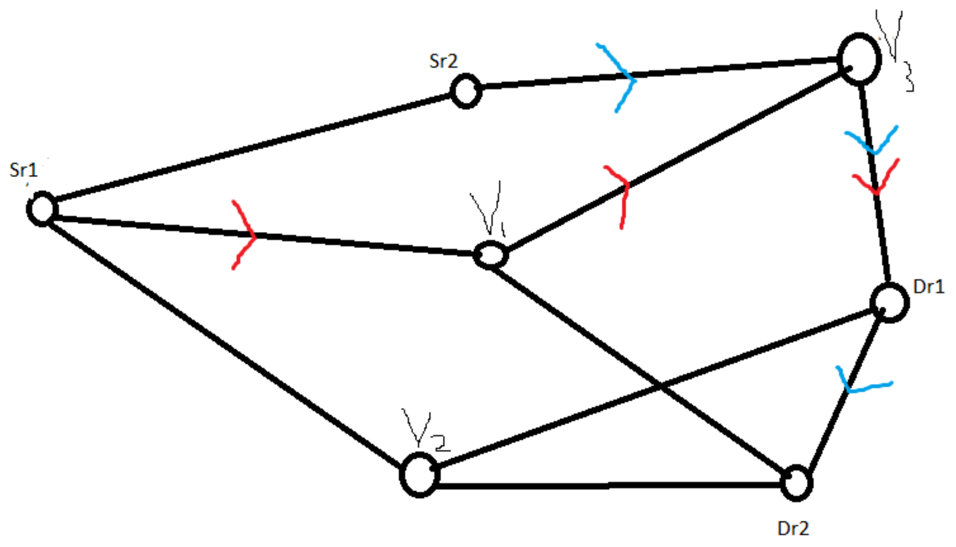
Time taken = *charging time* + $\frac{D}{s}$

Now, after charging at a city, again two more cases arise :
<u>CaseA</u>: It can travel the rest with a battery status of less than or equal to max battery capacity. So its a SUCCESS! and *Total Time taken* = *charging time* + $\frac{D}{s}$
<u>CaseB</u>: It charges fully but still cannot travel the rest of the distance.So what we can do is again charge it at a previous station after which,on the way, its battery discharges completely. And then again check as we checked above again and again till it reaches its destination node. Therefore, in this case,

Total Time taken = $\sum charging\ times\ at\ different\ stations$ + $\frac{D}{s}$

This was all for the case when there is only one EV.

Now suppose there are two EVs P1 and P2 respectively. The graph of cities is given and every other detail(property) of each EV is given. The below figure represents an example of this type of situation.



Suppose the red arrows represent the optimal path of EV P1 generated by our search algorithm and the blue arrows represent the optimal path of EV P2 generated by our search algorithm. (**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 i.e our Dijkstra inspired search algorithm also generates the paths by taking the distances between cities as heuristic values).

As we did, we initially apply our search Algorithm to both the vehicles and generate their separate optimal paths independently of each other. They are shown in the figure as red and blue arrows.
The cases which arise are :
Case1:They both don't have to charge anywhere: this means they can travel from source node to destination node with their initial battery status.

Total time taken by each EV = $\frac{D}{s}$(where D is total distance and s is the average speed)

Let the time taken by P1 and P2 are T1 and T2 respectively. Since both are optimized so, max{T1, T2} is optimized

Case2: One has to charge and the other doesn't need to charge for traveling from its source node to destination node respectively: we can apply the same resolving method for charging as we did for a single EV system above and can optimize the time of EV which is to be charged at some points. The other EV which need not be charged is already optimized and we don't have to resolve anything

Let P1 needs charging

So, T1 = $\sum charging\ times\ at\ different\ stations$ + $\frac{D}{s}$

And, T2 = $\frac{D}{s}$(where D is total distance and s is the average speed)

Since both are optimized so max{T1, T2} is optimized.
for traveling from its source node to destination node respectively

<u>Case3</u>: Both needs to be charged for traveling from its source node to destination node respectively: In this case, two cases arise again, we will resolve each case one by one-
<u>CaseA</u>: They don't charge at the same city at any point of time. No issue in this case as we can apply the same resolving method for charging as we did for a single EV system above and can optimize the time of EVs which is to be charged at some points respectively.
Let T1 and T2 be time taken by P1 and P2 respectively

$$T1 = \sum charging\ times\ at\ different\ stations + \frac{D}{s}$$

$$T2 = \sum charging\ times\ at\ different\ stations + \frac{D}{s}$$

Since both T1 and T2 are optimized, max{T1,T2 } will be optimized.

<u>CaseB</u>: One is charging while the other came at the station and the other has to charge at the same station as well. So what we can do is we give preference for charging to one that needs more time to reach the destination as we want to minimize max{Tr}. So, whichever EV has more Tr, it will be given preference.

Note: if a car has lower Tr but has reached 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.

That is the time taken for the waiting one will be (if it waits)

$$T1 = \sum charging\ times\ at\ different\ stations + \text{waiting time} + \frac{D}{s}$$

Time taken (if it takes another path)

$$T1 = \sum charging\ times\ at\ different\ stations + \frac{Dnew}{s}$$ (Dnew will be calculated by again applying Dijkstara at the charging station where it was waiting to destination)

We will compare both the time and the time which is minimum will be optimal for it.
So in this case the preferred one's time is already optimized and given by

$$T2 = \sum charging\ times\ at\ different\ stations + \frac{D}{s}$$

And the one which was less preferred will be compared in two different situations that is if it waits or chooses another optimal path from that charging node to the destination node

Since T2(AS WE SUPPOSED) is the max{T1, T2} and is optimized, so we can say that max{T1, T2} is optimized.
Note: T1 is not optimized here but our goal is to optimize max{T1, T2} and as we calculated T2 was max{T1, T2} and T2 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{v1,v2,v3,....,vn} and set of EVs {P1,P2,...Pr} and their properties a) Sr - source node (b) Dr - destination node (c) Br - battery charge status initially (d) cr - charging rate for battery at a charging station (energy per unit time) (e) dr - discharging rate of battery while traveling (distance travel per unit charge) (f) Mr - maximum battery capacity (g) sr - average traveling speed (distance per unit time) been provided.

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

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

PROPOSED SOLUTION and PROOFS

We have done an analysis in the problem analysis section of the document. Please have a read to it. Most part of the solution and algorithm has been done in the analysis part.
It is clear from the above analysis section what our algorithm is, and how it will be processing the data to provide optimal paths. We have used a search algorithm inspired by Dijkstra Algorithm which uses distance between cities as heuristic to find optimal paths for

EVs.In this section, we will be explaining and prove how it gives optimal paths of a node to any other node in a graph.

```
Dijkstra(G, s):
{
        for all u ∈ V \ {s},
        d(u) = ∞
        d(s) = 0
         R = {}
        while R != V
        pick u not in R with smallest d(u)
         R = R ∪ {u}
        for all vertices v adjacent to u if d(v) > d(u) + `(u, v)
        d(v) = d(u) + `(u, v)
}
```

Let d(v) be the label found by the algorithm and let δ(v) be the shortest path distance from s-to-v. We want to show that d(v) = δ(v) for every vertex v at the end of the algorithm, showing that the algorithm correctly computes the distances. We prove this by induction on |R| via the following lemma:

Lemma: For each x ∈ R, d(x) = δ(x).

Proof by Induction: Base case (|R| = 1): Since R only grows in size, the only time |R| = 1 is when R = {s} and d(s) = 0 = δ(s), which is correct.

Inductive hypothesis: Let u be the last vertex added to R. Let R0 = R∪ {u}. Our I.H. is: for each x ∈ R0 , d(x) = δ(x).

Using the I.H.: By the inductive hypothesis, for every vertex in R0 that isn’t u, we have the correct distance label. We need only show that d(u) = δ(u) to complete the proof.

Suppose for a contradiction that the shortest path from s-to-u is Q and has length

$$`(Q) < d(u).$$

Q starts in R0 and at some leaves R0 (to get to u which is not in R0). Let xy be the first edge along with Q that leaves R0. Let Qx be the s-to-x subpath of Q. Clearly:

$$`(Qx) + `(xy) \leq `(Q).$$

Since d(x) is the length of the shortest s-to-x path by the I.H., d(x) ≤ `(Qx), giving us

$$d(x) + `(xy) \leq `(Qx).$$

Since y is adjacent to x, d(y) must have been updated by the algorithm, so

$$d(y) \leq d(x) + `(xy).$$

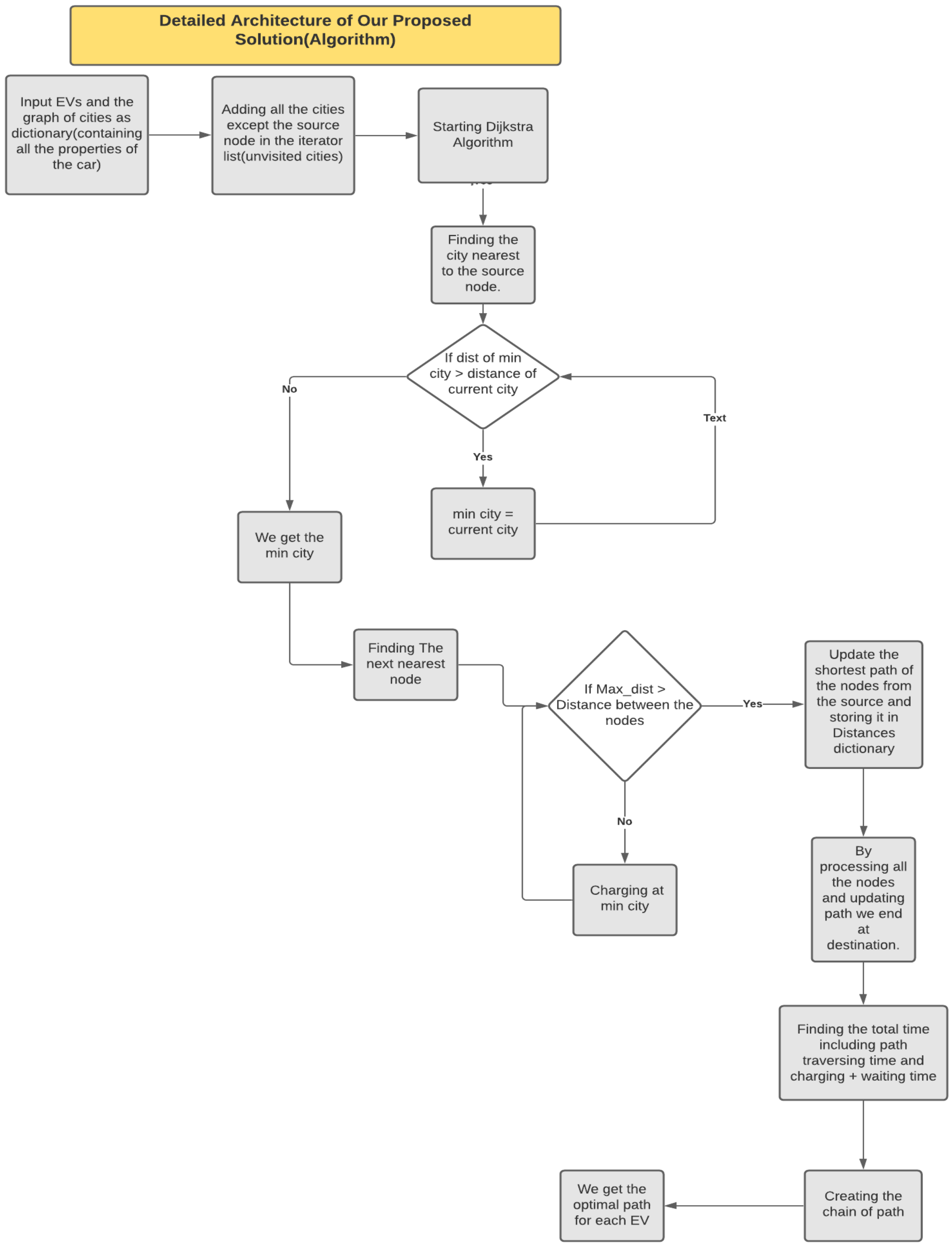
Finally, since u was picked by the algorithm, u must have the smallest distance label:

$$d(u) \leq d(y).$$

Combining these inequalities in reverse order gives us the contradiction that d(x) < d(x). Therefore, no such shorter path Q must exist and so d(u) = δ(u).
This lemma shows the algorithm is correct by “applying” the lemma for R = V.

Hence we proved that it gives the most optimal path from one node to any node of a graph.

DETAILED ARCHITECTURE

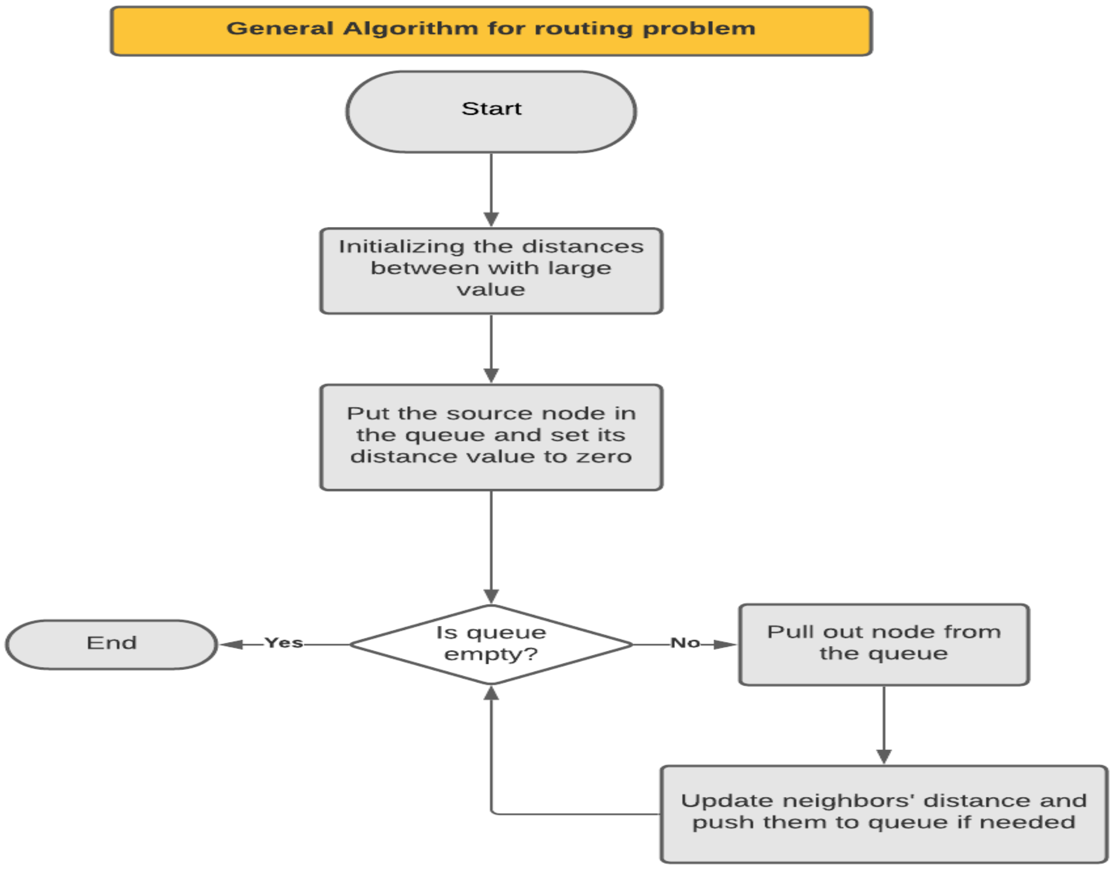

Detailed Architecture of Our Proposed Solution(Algorithm)
Input EVs and the graph of cities as dictionary(containing all the properties of the car)
Adding all the cities except the source node in the iterator list(unvisited cities)
Starting Dijkstra Algorithm
Finding the city nearest to the source node.
If dist of min city > distance of current city
No
Yes
Text
min city = current city
We get the min city
Finding The next nearest node
If Max_dist > Distance between the nodes
Yes
No
Update the shortest path of the nodes from the source and storing it in Distances dictionary
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 optimal path for each EV

ANALYSIS AND COMPARISONS OF POSSIBLE SEARCH ALGORITHMS FOR OUR PROBLEM

According to the problem statement we are given, we were required to find the Electric vehicle's route such that max{Tr}, 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 minimized and we were asked to find the optimal algorithm to perform the same task. There are various electric vehicle routing algorithms about which we have discussed, analyzed, and compared below, but the main algorithms which fall under this electric vehicle routing are A* and Dijkstra's algorithm.

GENERAL ALGORITHM:

The basic flow chart of every algorithm used in electric vehicle routing is to start from the source node, explore the consecutive nodes one by one and each next step we go is the node that is closest to the source node.


General Algorithm for routing problem
Start
Initializing the distances between with large value
Put the source node in the queue and set its distance value to zero
Is queue empty?
Yes
End
No
Pull out node from the queue
Update 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 neighbors 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.

VARIOUS ALGORITHMS FOR ROUTE PLANNING PROBLEM:

And at the end of the algorithm, we get the shortest path from the source node to all the other nodes in the graph. But the difference between algorithms comes in when graphs given to evaluate are either unweighted or weighted and thus we can't use any algorithm for any case. We have few algorithms but the two main algorithms A* which calculate the shortest path in unweighted graphs and another main algorithm, Dijkstra's algorithm does the same thing in weighted graphs.

1. **Dijkstra Algorithm:**
   Dijkstra's Algorithm seeks to find the shortest path between two nodes in a graph with weighted edges. It works only if the edge weights are positive. It works by building the shortest path to every node from the source node, one node at a time. When we are given weighted graphs, it is not always the situation that neighboring nodes will always have the shortest path. However, a neighbor 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. And this is the idea of the Dijkstra Algorithm which comes up with the greedy approach i.e in every step taken we choose the next node with the shortest path. At every step, we update the cost and add neighboring nodes to the queue and we achieve this using h e priority queue. But this algorithm might not be the best in the case of unweighted graphs. In unweighted graphs, when we reached a node from a different path, we were sure that the first route should be the shortest route which is not always true in weighted graphs. If we reached the node with a shorter path, we must update its distance and add it to the queue which implies that we visit the same node multiple times and the same node can be added multiple times. Thus we should compare the cost of the visited node with its stored value. If the distance of the visited node is larger than the one already stored, which means the given node was added in earlier, then found a shorter path and updated it.
   Dijkstra algorithm terminates as soon as it labels the destination node, leading it to find the shortest path and we visit each node neighbors only once as we are visiting edges only once, also we used a priority queue that has the time complexity of O(n2) for push and pop operations.

2. **A* Algorithm:**
   A* Algorithm is a variant of the Dijkstra algorithm but the difference is that it uses heuristics function rather than the optimal search mechanism rather than optimal search mechanism. It restricts the search space and reduces computational time. There are extensions of A* like RTA* and LRTA* proposed for real-time applications. They take the direct distance between the source and destinations s the heuristic function. Also while dealing with unweighted graphs we always try to reduce the number of visited edges. A* works perfectly well i.e it finds the shortest path correctly in weighted graphs where all the edges have the same weights for all the edges. But

this is not the case for many graphs as we don't have the same weights for all the edges in the graphs given.
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.

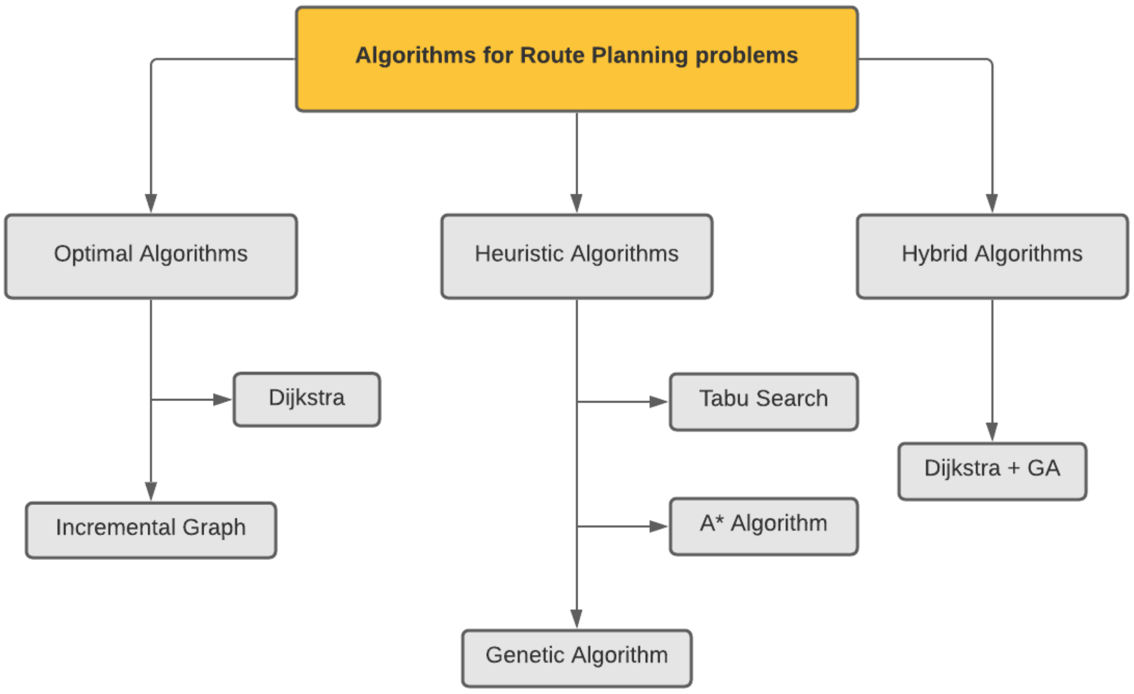
3. **Genetic Algorithms (GA):**
   Genetic Algorithms are used to routing and optimization search problems. It uses meta-heuristics and as the name goes it simulates the way species evolve and adapt to their environment, according to the Darwinian principle of natural selection. In beginning, it generates the random heuristic population and repeats the cycle a number of times, and the classical solution is modified when applying to vehicle routing problems. Since it has routes during a search in population thus it is possible to evaluate the route in a short period of time using other routes and constraints in the search.

4. **Tabu Search:**
   Tabu search is a local search-based meta-heuristic applied to route planning problems that run through several iterations. In every iteration, the best solution neighboring to present node is updated to the current solution, even if leads to the increment of the solution cost. Thus is not the best method to approach with as a bad optimal solution is mitigated. It stores the visited values in a list, which helps to avoid visiting the same node twice. In this algorithm search for the next solutions stops after a fixed number of consecutive iterations and doesn't care to improve the best-known solution.

5. **Hybrid Genetic Algorithms:**
   Hybrid Genetic Algorithm uses Genetic Algorithm with Dijkstra Algorithm to solve the multi-objective problems. It gives solutions fulfilling three objectives: travel time, ease of driving, and distance traveled in the route. To use the Genetic Algorithm in the road system, this method uses the Dijkstra algorithm to calculate the initial routes. From the initial population, this method uses Genetic Algorithm to generate subsequent routes.

CLASSIFICATION OF ALGORITHMS FOR ROUTE PLANNING PROBLEM:

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 guarantee to find the optimal solution by exploring the whole solution set available. The most common optimal approach-based algorithms are Dijkstra and Incremental Graph. They both find the shortest path between two nodes. Heuristic approach-based algorithms explore all available solutions and find the approximate optimal solution which is close to the global optimal solution. Some commonly used heuristic approach-based algorithms are Tabu search, A* Algorithm, and Genetic Algorithm. These find the routes based on constraints such that the computation time is less and gives the best route possible for the given constraints. And the last hybrid approach-based algorithms use the properties of both optimal approach-based algorithms and heuristic approach-based algorithms and uses the strength of both of these approaches.

Issues and challenges

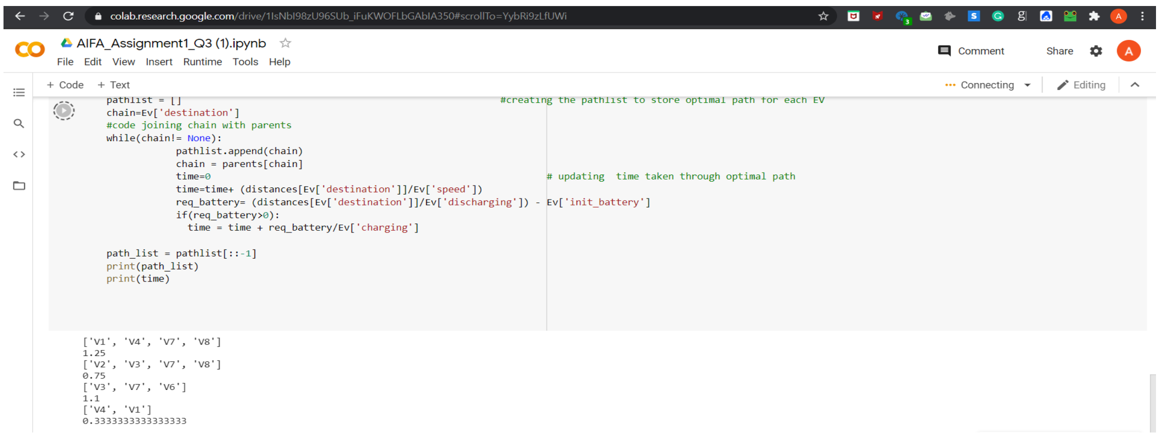
As we try to solve the problem, we encounter many issues and challenges that we try to overcome. One of the major challenges that we wish to resolve through our code is the wastage of energy when the chosen path is not the optimal one. Our code takes the average traveling speed, the charging rate for batteries at a charging station into account which are not unique for all the EVs. The waiting time, which is different for different EVs, is also a big challenge that we face and try to overcome. Since the problem is so diverse, the chances of not moving along the optimal path are very high which in turn leads to wastage of energy which in itself is limited.

The dynamic traffic environment is also a big challenge in the route planner and navigation systems problems. The loss of energy to the environment due to various reasons such as rolling, friction, aerodynamic resistance is also an issue that comes in our way.

RESULT

The algorithm we used for our problem statement is the hybrid approach-based algorithm, and we have used the properties of both the Djikstra and GA as we had to consider various metrics which are best implemented using a hybrid approach for planning electric vehicle's route such that max{Tr}, 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 minimized and we had to find the optimal algorithm for the route. Using this we are able to get best possible path with minimum energy usage. Also, we mentioned above the performance of various routing problem algorithms. We used the efficiency of the Djikstra algorithm in the road map graph. It is better to use Dijkstra + GA for finding the shortest path for this one-to-one problem, it terminates as we find the destination node and it is better than others because other algorithms are able to find the shortest path only after exploring all the paths and some who doesn't do that gives solution close to optimal one but not globally optimal.
Also, it is a heuristic algorithm with a special case where the value of h(n) is 0. Firstly because edge costs may be negative as in the case of A* Algorithm due to recuperation, except in our case of the algorithm. Secondly, because edge costs here in our problem depend upon charging time. Third, because the battery capacity in our problem is limited due to which edge cost is not just the sum of edge costs. Also, we can't apply the Dijkstra Algorithm directly because the battery can get fully discharged which at some points, which is a metric to be solved. Also, we can't use the A* algorithm directly because it gives optimal solution for unweighted and weighted with equal weight graphs, it stores the visited nodes in the simple queue, also it visits all the nodes that have costs less than the costs of the goal and is slower than others.

AN EXAMPLE OF AN OUTPUT IS SHOWN IN THE FIGURE


colab.research.google.com/drive/1IsNbl98zU96SUb_iFuKWOFLbGAbIA350#scrollTo=YybRi9zLfUWi
AIFA_Assignment1_Q3 (1).ipynb
File Edit View Insert Runtime Tools Help
Comment
Share
+ Code + Text
Connecting
Editing

```
pathlist = []                                    #creating the pathlist to store optimal path for each EV
chain=Ev['destination']
#code joining chain with parents
while(chain!= None):
            pathlist.append(chain)
            chain = parents[chain]
            time=0                                    # updating  time taken through optimal path
            time=time+ (distances[Ev['destination']]/Ev['speed'])
            req_battery= (distances[Ev['destination']]/Ev['discharging']) - Ev['init_battery']
            if(req_battery>0):
              time = time + req_battery/Ev['charging']

path_list = pathlist[::-1]
print(path_list)
print(time)
```

```
['V1', 'V4', 'V7', 'V8']
1.25
['V2', 'V3', 'V7', 'V8']
0.75
['V3', 'V7', 'V6']
1.1
['V4', 'V1']
0.3333333333333333
```

Conclusion

Optimal routing for electrical vehicles with rechargeable batteries will become increasingly important in the future.

Due to their limited range, EVs are recharged at charging stations many times while in transit in order to ensure that the trip is completed. We have also discussed the best method for optimally assigning charging stations for EVs with charging demand.

We got an optimal path for a set of inputs with their minimum traversal time. And this helped in minimum consumption of energy which helps in energy conservation.

APPENDIX

Here is the 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
              time̦u] ← time̦uo] + 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
```

REFERENCES

1. 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.
2. R. E. Korf, Real-time heuristic search, Artificial Intelligence, vol. 42, pp. 189-211, Mar. 1990.
3. F. Glover and M. Laguna, Tabu Search, July 1997.
4. T.-Y. Liao and T.-Y. Hu, An object-oriented evaluation framework for dynamic vehicle routing problems under real-time information, Expert Systems with Applications, vol. 38, pp. 12548-12558, Sept. 2011.
5. M. Gendreau, J.-Y. Potvin, O. Braumlaysy, G. Hasle, and A. L kketan- gen, Metaheuristics for the Vehicle Routing Problem and Its Extensions: A Categorized Bibliography, in The Vehicle Routing Problem: Latest Advances and New Challenges (B. Golden, S. Raghavan, and E. Wasil, eds.), vol. 43 of Operations Research/Computer Science Interfaces, pp.43-169, Boston, MA: Springer US, 2008.
6. Vi Tran Ngoc Nha, Soufiene Djahel and John Murphy Lero, UCD School of Computer Science and Informatics, Ireland {vi.tran-ngoc-nha, soufiene.djahel, j.murphy}@ucd.ie
7. Andreas Artmeier, Julian Haselmayr, Martin Leucker, Martin Sachenbacher Technische Universit¨at M¨unchen, Department of Informatics Boltzmannstraße 3, 85748 Garching, Germany {artmeier,haselmay,sachenba,leucker}@in.tum.de