PROJECT LOAD-UNLOAD: NEW APPROACH TO SUPPLY CHAINS AND TRANSPORTATION OF GOODS USING SEARCH ALGORITHMS

Nivedit Jain (B18CSE039)[1]

[1]*Department of Computer Science and Engineering*
Indian Institute of Technology, Jodhpur

Project Report for completion of Artificial Intelligence Course (CS314) (BTech Pre-Final Year, Trimester 1, Academic Year 2020-2021) by Dr Yashaswi Verma, Indian Institute of Technology, Jodhpur.

I INTRODUCTION

Without Supply Chain and Transportation, any modern economy cannot service. All businesses are dependent on them. However, with each passing day, they are becoming more and more vital for us, especially in times of pandemic, without efficient transportation and supply chains many people would have suffered extremely hard. This sector is on a boom since a long time and would continue to grow even further in upcoming years. One of the reason being is rising eCommerce. We cannot even imagine great companies like Amazon, Flipkart, etc without efficient Supply Chain and Transportation sector. Millions of tones of packets both small and large flow through the system everyday valuation in billions of dollars with hundreds of thousands of vehicles both large and small moving from one place to another. To make importance much more concrete, among 29 Unicorns from India [1], 3 (BlackBuck, Rivigo and Delhivery) are working in this field, and 7 (Snapdeal, Paytm, Udaan, Bigbasket, Lenskart, FirstCry, Nykaa) are completely dependent on it. Not only the StartUp world all other business also won't be able to function with tools of Supply Chain And Transportation.

We aim to make this system even better and improve margins for companies simultaneous reducing running costs of vehicles using an aggregator service for inter-city supply of goods using unutilized space available in vehicles already running on the road, and reducing the running cost of these vehicles as some money would be paid for successful delivery. Our goal is to make the process most efficient for business and vehicle owners.

Such a scheme would be highly useful and could add solve a number of problems, few such problems we can think of are

- This could would could significantly reduce the cost of transportation of goods as the cost would be shared with the person already travelling to some place. Hence it would be extremely useful to support MSMEs (Micro Small and Medium Enterprises), not only MSMEs it could be useful for big cooperates and individuals, thus supporting economies especially for developing nations.
- This scheme could also be provided very effective for eCommerce businesses and customers as the delivery cost would be reduced using this scheme.
- It would also reduce the travel cost of people travelling by road. Also, it could also potentially increase the income for various drivers, like truck drivers, bus drivers etc, who hardly make money.
- If we are able to implement we could try to reduce hundreds of thousands of good carriers from the road, simultaneous solving the problem of congestion on-road and positively affecting our environment in a number of ways, like reducing emissions, number vehicles and roads itself.

As we will discuss in later sections that this problem is computationally hard to implement. Also in practical implementations data which we would be working on would be extremely large making is quite tough to solve this problem. Geographical data for practical implementations would also be huge.

The approach we are using to handle this problem is solving subproblems with relaxed constrains using various algorithms like Uniform Cost Search etc and trying to build upon the simple version to solve the bigger problem.

If we are able to implement a real-time system which is capable of finding the most optimal solution, it would be truly novel and according to us would be able to revolutionize the industry with an extremely positive impact on our environment.

II PROBLEM DEFINITION

We planned to implement our aggregator service (Load-Unload) as following, (please see Figure 5 for reference)

- We assumed that there are a good number of vehicles of any type (private cars, taxis, commercials, trucks, busses, vans etc) are going from city A to city B at any given point of time.
- Each such vehicle has a maximum unoccupied volume V_V and the vehicle can carry more load V_M (max recommended weight of vehicle − present weight), both V_V and V_M are such that it won't affect

* jain.22@iitj.ac.in

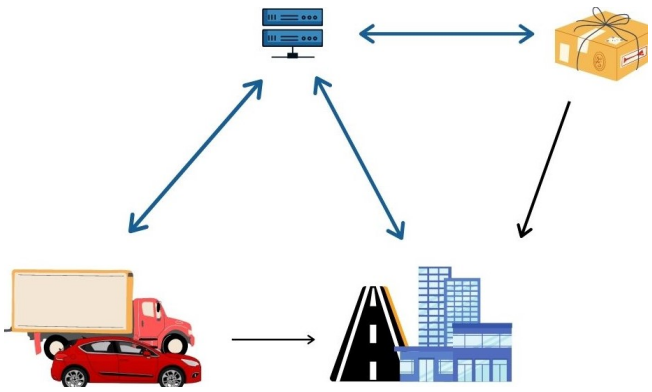

Figure 1: Working Flow of the planned system

the safety of the vehicle in any way (eg, overloading is prevented).

- Vehicles before starting their journey register to our servers and provide all this information to us, using any internet-connected device in the vehicle (eg, Drivers Mobile). After this driver will go to a pickup centre of the area, address of the nearest pickup centre would be sent to drivers by the servers.

- All the packets or boxes (we will refer them as parcels) which need to be shipped to some location are sent to a pickup with following details for each parcel, weight P_W, volume P_V, price to deliver P_M, and destination of the parcel P_D. All this information is sent to our server.

- We assume that some $x\%$ of P_M goes to the driver taking the parcel to destination. Let's denote this quantity by P_C. We can easily find P_C as

$$P_C = \frac{x \times P_M}{100} \tag{1}$$

- For each vehicle, we define a special function called the cost function, we can develop different cost functions as per our needs, we have used the following function as cost function,

$$cf([A\dots B], TIME, \{PC_1\dots PC_n\}) = c_1 len([A\dots B]) + c_2 TIME - c_3 \sum_{i=1}^{i=n} PC_i \tag{2}$$

where cf denoted the cost function, $[A\dots B]$ denotes the path of the vehicle, $TIME$ denotes the journey time, $\{PC_1\dots PC_n\}$ denotes the multi-set of PC of parcels assigned to the vehicle where PC_i denotes PC of i^{th} parcel, $len()$ is a function which gives the length of path (distance) and c_1, c_2 and c_3 are some constants such that $c_i \geq 0\ \forall i$, in this case they denote the priority of different factors and we would like user to be able to change them as per his/her need.

In the above cost function formulation we have assumed that all parcels picked are delivered, if system picks an assignment in which there is any undelivered parcel we reject that assignment, alternatively one can define a cost function to handle such cases by a very slight modification as

$$cf([A\dots B], TIME, \{PC_1\dots PC_n\}, U) = c_1 len([A\dots B]) + c_2 TIME - c_3 \sum_{i=1}^{i=n} PC_i + mSize(U) \tag{3}$$

U denotes the set of undelivered parcels and $Size(U)$ denotes the number of parcels in U, we make sure that here m is a very large positive number which could never be achieved by any accepted cost function.

Note that cost function can also be negative, if it means that vehicle owner/driver is actually earning from the trip.

- We would also allow vehicles to reduce V_V and V_M, ensuring maximum comfort and flexibility to vehicle owners, (eg, say in car when a family is going for picnic more space is needed, so reduce V_V and V_M, but one may want to keep V_V and V_M unchanged while solo travel, similarly for commercial vehicles drivers might want to reduce V_V and V_M if their vehicles are partially occupied already and might want it to be unchanged if the vehicle is not occupied at all.)

- We also define a quantity T_S, denoting expected time for a vehicle to unload at any pickup point while dropping parcels and at the end destination. Similarly at the beginning of the journey let us consider loading time at L_S. Let us consider that T_S and L_S are constant for each parcel. So we can write time as

$$TIME = \sum_{i=1}^{i=k} Time_i + m \times T_S + n \times L_S \tag{4}$$

where $Time_i$ denotes the expected running time on road i and k is total number of roads (edges, in the graph we will construct), n is number of parcels handled by vehicle and m is number of parcels delivered by vehicle.

As soon as a vehicle arrives at a pickup centre our server must be able to compute the best path for the driver and should be able to assign packets to the driver such that the cost function for that vehicle is minimum.

There are many other variants and modification of the problem one may think off, like in out approach we are trying to minimize the cf for each vehicle, but could formulate the problem as

$$min(cf_i) \tag{5}$$

such systems would be more effective but would be even harder to compute. This kind of formulation make the problem even more tougher to solve in a real time scenario.

Another possible variant could be allowing vehicles to pick up at multiple stops during the travel, we keep this variant for future works.

Data Generation and Description

We could easily generate any random graph and could be able to run our experiments on them, however we wanted to have a data which closely represents road networks (inter-city highways) of a country. So we are generating our data, in the following steps

- We first generate a random tree of N nodes using Prüfer Sequence [2][3]. We generate as it will ensure that there exists at least one route between any two locations, which is usually the case, you can go from any city of a country to any other city.
- We now generate weights for edges generated in the previous step, we assign weights to edges using a normal distribution as we observed that in the real world most of the distance between neighbouring cities is almost near an average value but some are very near and some are very far away, so generated edge weights using a normal distribution would be a good choice. We generate edge weight (D_1) using the following equations,

$$D_1 \sim \mathcal{N}(\mu_1, \sigma_1^2) \tag{6}$$

$$D_1 > 0 \tag{7}$$

$$| D_1 - \mu_1 | \leq 3\sigma_1 \tag{8}$$

  we would be easily able to generate an appropriate values within few iterations as we know that

$$P(| D_1 - \mu_1 | \leq 3\sigma_1) \approx 0.9973 \tag{9}$$

- We now divide all cities in our model in 3 categories, metros, non-metros and remote locations, we connect all metro cities with each other adding new edges, we do not add new edges for non-metro cities, and we also add new edges from metros to non-metros, All these cities are randomly selected. We also randomly select from the uniform distribution number of non-metro cities connected to a metro city. All the weights of new edges as per the following equation

$$D_2 \sim U((1-\alpha)minDis(A,B),\ (1+\alpha)maxDis(A,B)) \tag{10}$$

  here $minDis(A,B)$ and $maxDis(A,B)$ denote minimum and maximum distance on graph between A and B nodes, along already existing edges and α is some factor in range $(0,1)$. By α we are trying to depict that there could not be a very large change in road distance between 2 cities.
- Lastly we need to assign a sense of time to each edge, we define time by

$$time = \frac{distance}{c \times max(degree(A), degree(B)) + S} \tag{11}$$

$$S \sim \mathcal{N}(\mu_2, \sigma_2^2) \tag{12}$$

$$S > 0 \tag{13}$$

$$| S - \mu_2 | \leq 3\sigma_2 \tag{14}$$

  basic motivation is usually highly connected cities have better road infrastructure and we are using a normal distribution (about 60 Km/Hour) to simulate the real world.

We run our experiments on the data generated by our method, also we tested on random graphs.

State Definition

Let us assume that start pickup centre is A, there are P parcels at start pickup centre and destination pickup centre is B.

Now if we look closely to represent any possible state we need the position of the vehicle and a set of parcels in the vehicle at that particular time. We represent sate as a tuple (S),

$$S = (City,\ \{P_1, \ldots, P_n\}) \tag{15}$$

Let us analyze the number of possible states, let there be N cities, vehicle can be at any one the city so first parameter could be any city, however there could be any number of parcels from P, hence number of possible second parameter could be,

$$\binom{P}{0} + \binom{P}{1} + \cdots + \binom{P}{P} = 2^P \tag{16}$$

So total number of possible states becomes,

$$number(S) = N \times 2^P \tag{17}$$

However, all these states are not possible as number of parcels in the set must follow the following constrains,

$$\sum_{i=1}^{i=m} weight(P_i) \ \leq V_M \tag{18}$$

$$\sum_{i=1}^{i=m} volume(P_i) \ \leq V_V \tag{19}$$

However we define a special state for start state, imagine the condition before the vehicle enters the first pickup facility, it would not have any parcels and we define the city for this as a imaginary city, let's $city_0$, so maximum possible states now becomes,

$$number(S) = N \times 2^P + 1 \tag{20}$$

$$start\ state = (city_0,\ \{\}) \tag{21}$$

$$end\ state = (B,\ \{P_1, \ldots, P_m\}) \tag{22}$$

One might think that in our definition why end state parcel set is not empty, the reason being our cost function (see equation 3) we are using, end state definition should be changed if you are using cost function defined by equation

2.

Computational Hardness

Let us assume that there exists an Algorithm, A, to exactly solve this problem.

Claim .1 *We can use Algorithm A to solve Travelling Sales Man Problem.*

Proof. Let us take an instance of our problem with following parameters

- Set our Start Location (S) to our Destination (D).
- Set c_2 (equation 3) to 0, and set c_1 and c_3 to 1.
- Let our input graph has N cities, we assign initially available parcels at start pickup center $N-1$, such that each parcel for a different city and follows following constraints,

$$\sum_{i=0}^{i=N} P_{Wi} \leq V_M \tag{23}$$

$$\sum_{i=0}^{i=N} P_{Vi} \leq V_V \tag{24}$$

$$\sum_{i=0}^{i=N} P_{Ci} \geq len([A \ldots B]) \tag{25}$$

This constructions and parameters setting could be done in polynomial time and further we could notice, if we set our problem like this, the most optimal solution is going to each city and coming back to origin with the path of shortest distance. So, we can use Algorithm A to solve Travelling Salesman Problem also.

Claim .2 *Our problem is at least as Hard as Travelling Salesman Problem*

Proof. As we can use an instance of our problem to solve Travelling Salesman Problem, this clearly means that our problem is at least as Hard as Travelling Salesman Problem.

As Travelling Salesman Problem is NP-Hard, our problem also belongs to NP-Hard Class.

Claim .3 *Alogrithm A could also be used to solve 0-1 Knapsack Problem*

Proof. Let us consider our cities and road to be a Linear Graph with start state and goal state on two different ends, with c_1 and c_2 be 0 and c_3 be 1. We can clearly see most optimal solution would be one which highest valued packages in our vehicle, which is a 0-1 Knapsack Problem.

Claim .4 *Our problem is at least as Hard as 0-1 Knapsack Problem*

Proof. Clearly from claim .3, an algorithm for our problem can be used to solve 0-1 Knapsack Problem, which is NP-Hard Problem

If you look carefully our problem is a sort of combination of a 0-1 Knapsack Problem and Traveling Sales Man Problem, however major focus should be some fast approximations for real time implementation.

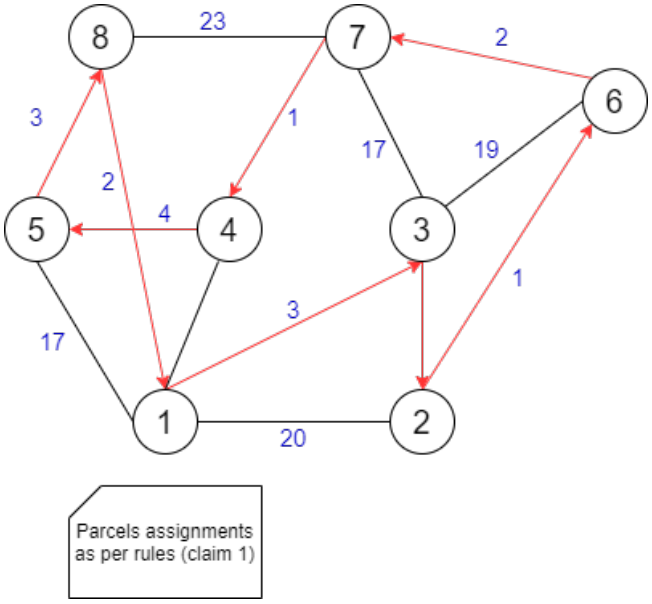

Figure 2: Solving TSP using our Algorithm A

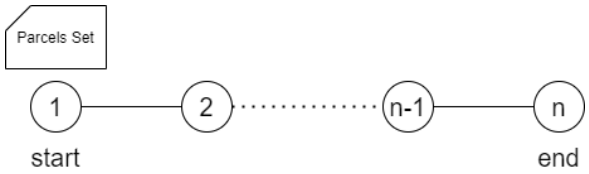

Figure 3: Visualization of solving knapsack using Algorithm A

III BACKGROUND SURVEY

We were not able to find the same problem, however, there exists certain similarity with Fleet Assignment Problem, based on our review majority of already existing researches were more focused on either an Integer Programming Approach or Multiple Optimisation Approach.

Our problem appears to be different from Fleet Assignment problem as in Fleet Assignment, we know the exact number of vehicles available and destination of each vehicle is not fixed. Another noticeable point could be one can run such Fleet Assignment algorithms once use the result till anything changes as other factors would usually be constant, however, it is not a case with our problem.

This problem is originally inspired by CarPool Problem or Ride Sharing Problem (see Acknowledgments section), however, we were also not able to find similar problem formulation in that scenario too.

Some variations of combination of 0-1 Knapsack problem with Travelling Sales Man problem exists. [4]. A large number of variants of Transportation Problems and Transportation Scheduling Problems do exists in literature. (we were not able to do a detailed literature review due to time constraints)

As per our knowledge no exact such problem has been studied or no such similar system is implemented yet any where in world and we too believe that such system could impact millions of life in India only.

IV DISCUSSION

We would like to come up with an algorithm to solve this problem with the most optimal possible solution within a

few seconds, if we can do so on a large geographic data say, complete road data on Indian, with a large number of parcels, along with handling multiple queries from several pickup centres around the country, we will call our approach novel and would be a practically implementable solution.

Our solution as discussed in Algorithms Section could be considered as Engineering Based Solution as we are trying to include all practical constrains and trying to simulate the real world. We are trying to generate the best solution as per resources available to us.

This problem is a huge scale problem when comes to implementing in real world, however we are not very convinced with our solution as it is not a very fast and optimal solution but it would be able to solve the problem with a given time limit for a small number of parcels and a small number of cities. If we take it as a long term research project, we probably would have been able to develop a solution which could be deployed and impact millions of life, simultaneously saving our precious environment.

V ALGORITHM and ALGORITHM COMPLEXITY

Iterative Deepening Based Approach

We try to modify Iterative Deepening DFS Algorithm, first, we need to define the state successor function, we define our state transition function T_f as,

$$T_f(S) = \begin{cases} (city_1,\ P_{in}) & : S.city = city_0 \\ (city_{edge},\ P_{in} - D_{city_{edge}}) & : S.city \neq city_0 \end{cases} \quad (26)$$

Here, P_{in} denotes the set of parcels present in vehicle, in

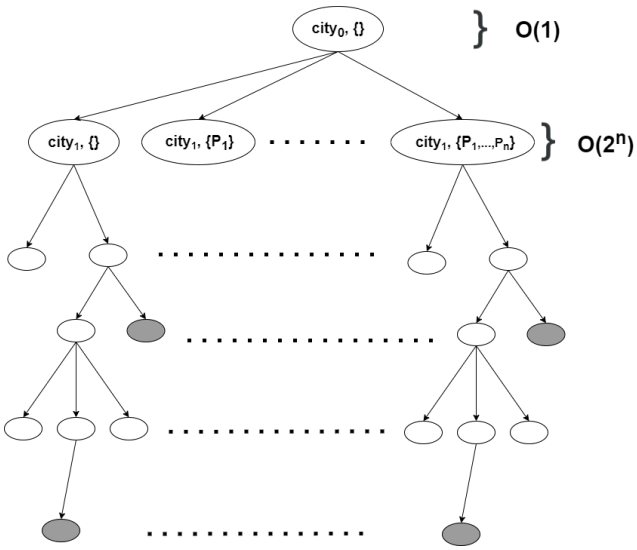

Figure 4: Visualization of successor Function

initial states ($S.city = city_0$) we are assigning parcels to the vehicle. In the second case when ($S.city \neq city_0$), we are trying to move vehicle, so we change *city* part of the tuple to one of its neighbouring (both cities have a direct road connection) city denoted by $city_{edge}$, also we need to remove parcels from the vehicle which are to be delivered at $city_{edge}$, $D_{city_{edge}}$ denotes set of parcels to be delivered at $city_{edge}$.

Here we are not running Iterative Deepening DFS search directly, our approach is a bit different. In general Iterative Deepening DFS, we stop as we have found a solution, however in this case we do not, we keep exploring the depths up to a defined limit and at one time we completely explore one depth to make sure the we find the best solution at that depth level.

For this we define few terms,

- $stepSize$, the number of levels in depth we can go down in one step of iterative deepening, that means we will give the best solution till that level of depth.
- $epochs$, the number of times we go down, that means, we will output the best solution till $depth = stepSize \times epochNumber + 1$ (In the first $epoch$ we go to $stepSize + 1$ depth, we do so to accommodate the special first state successor function.)

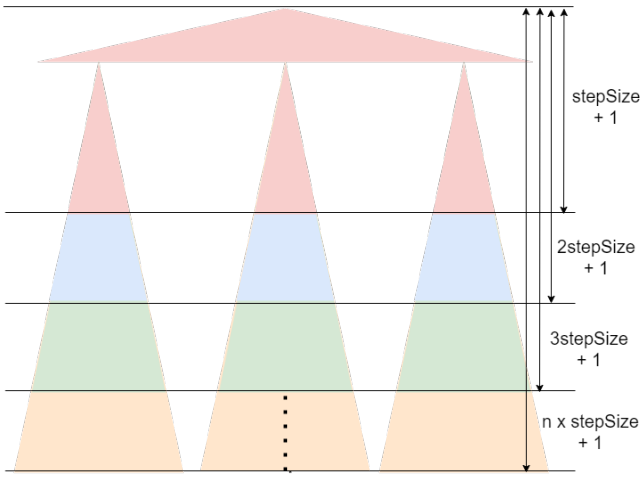

Figure 5: Visualization of Iterative Deepening Based Approach

We store all the best state found and choose the state with best (most optimal) cost-function (see equation 3), as number of epochs increase the chances to find the best solution also, we run certain minimum number of epochs to make sure that we at least find a goal state.

Here depths of goal states would be very uncertain as the vehicle's destination could be any city and branching factor is also extremely hard to predict as the graphs could be extremely random depending upon the country we are working on, however, the branching factor for the first state would be different from other states as our approach to first state is itself completely different, the branching factor of the first state would be $O(2^P)$.

As we will travel to each states in a epoch, so we can safely say that time complexity would be asymptomatically equal to number of total number of states till that epoch. We can safely say that in worst case all cities of the country would be connected forming a cliche, let there be N cities, so we can say that we are considering branching factor as N, for all nodes in search graph except the first node. We can clearly see that if our epoch number is $epoch_i$, then, number of states visited

$$\begin{aligned} &= 2^P \times \left(\frac{N^{epoch_i \times stepSize+1} - 1}{N-1}\right) \\ &= O(2^P \times N^{epoch_i \times stepSize}) \end{aligned} \quad (27)$$

if we are running our algorithm for K epochs then running time would be

$$O(2^P \times N^{stepSize}) + O(2^P \times N^{2\times stepSize}) + \dots \\ \dots + O(2^P \times N^{K\times stepSize}) = O(2^P \times N^{K\times stepSize}) \quad (28)$$

For memory complexity we not only need to store the fringe also we need to store the path in search tree for the best solution till now. So, it will take extra space in orders of tree depth along with original DFS solution. However one special thing to note here is that branching factor is not same for all, so for the first node in order to store all the children it will take at least $O(2^P)$ space, hence giving overall memory complexity as,

$$O(2^P + N \times (K \times stepSize) + K \times stepSize) \\ = O(2^P + N \times K \times stepSize) \quad (29)$$

One of the major advantages of iterative deepening algorithm is that we can adjust the number of epochs required as per computation resources we have, however as it is clear from space and memory complexity that this algorithm is not a very good approach.

However, if we could be able to not go to one point again and again then we could have a reduced number of states and could have implemented something like graph search.

Claim .5 *If there is a cycle in a graph then for optimal solution vehicle must take cycle once or do not take at all.*

Proof. Let us consider there is a graph of the road network, we start from A and our destination is some A', and there exists a city B which is a part of a cycle and also a part of a straight path from A to A', (see figure 6, here C is A' and straight path is $A \to B \to C$, however B is also a part of cycle $B \to E \to D \to E$). Let cf_i (see equation 3) be the cost function when took cycle i^{th} time. We cannot compare cf_0 and cf_1 without data, however, we can definitively that

$$cf_1 < cf_2 < \dots < cf_n \quad (30)$$

as if vehicle has any parcel to be delivered to any city in the cycle it will do it the first traversal of cycle, next iterations will only increase the both cost and time however no impact on the number of delivered parcels, which is highly discouraged by our cost function and will only increase the cost, won't reduce it. (see figure 6, if we have checked the path $B \to E \to D \to B$, we do not need to check the path $B \to E \to D \to B \to E \to D \to B$). Hence, our claim is proved.

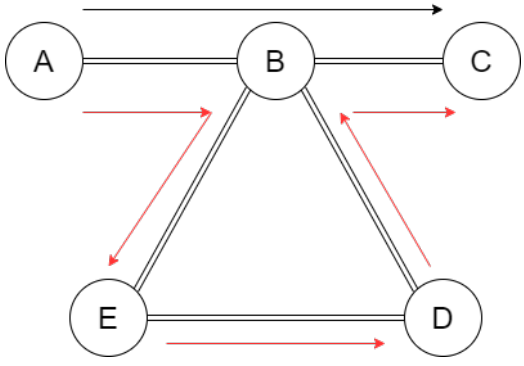

Figure 6: Example Cycle

Claim .6 *In order to find the most optimal solution, it is enough to run till* $\lceil \frac{N}{stepSize} \rceil$ *epochs.*

Proof From claim 1 we know that it is enough to check a cycle for only 1 traversal, so we can take at max N edges for traversal between cities, which mean that total depth of search tree would be $N+1$, as

$$N + 1 = epoch_{max} * stepSize + 1 \quad (31)$$

$$epoch_{max} = \lceil \frac{N}{stepSize} \rceil \quad (32)$$

eg, consider figure 6, here max search tree depth we should consider is along the path $A \to B \to E \to D \to B \to C$, which is 5, ei, number of vertices.

We use Claim 1 and Claim 2 to improve efficiency of our algorithm, Claim 1 can be used for Pruning and hence we will be able to limit the number of branches to travel and Claim 2 will limit our depth of search. Also, we can check for impossible assignments at first depth to prune impossible branches.

For applying Pruning using Claim 1, we can maintain a set of visited edges, not that effectively Claim 1 means that travel each edge only once per branch of recursion tree.

Our Algorithm could also be parallelized efficiently.

VI EXAMPLE

Let us consider our city graph same as figure 6. Let our start location be our start location be A and destination be C. For simplicity for now we have assumed c_1 and c_3 to be 1 and c_2 be 0. Let m be 1,00,000 (see equation 3). V_v be 10 and V_M be 20. Let $stepSize$ be 2. So $epochs_{max}$ become 3.

We also assumed that at initial Pickup Centre following packets represented as (destination, volume, weight, P_C) :-: {P1 : (C, 5, 10, 100), P2 : (E, 5, 10, 50), P3 : (D, 7, 4, 140)}

Let edges with weights (distance only for now) be (from, to, distance) (we are not taking time into consideration here as c_2 is 0) {(A, B, 1), (B, C, 2), (B, E, 1), (E, D, 1), (B, D, 2)}

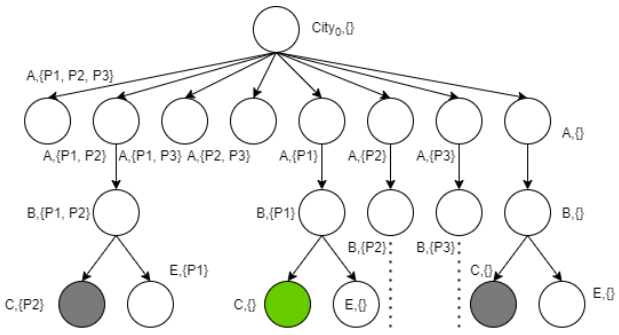

Figure 7: Search Graph after $epoch_1$ *the best solution shown in green till here, with cost of,* $(2+3)-(100) = -97$*, we pruned the the nodes as per rules and dotted lines indicate that there will be more search but was unable to draw due to space constrains while drawing. We have not shown B-D as it will behave exactly same as B-E in this case*

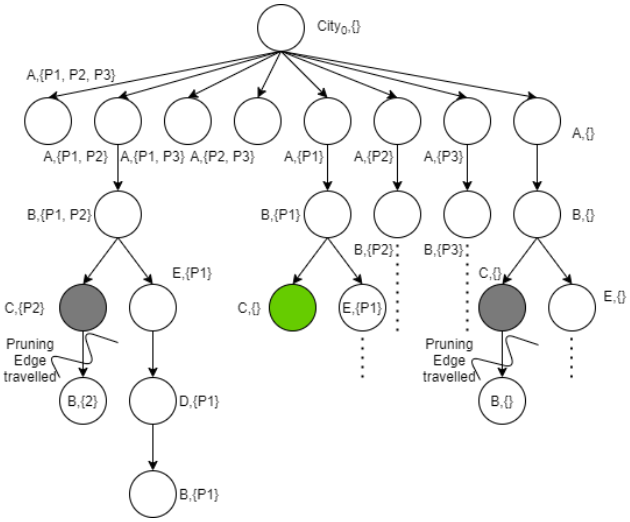

Figure 8: Search Graph after $epoch_2$ as we didn't reach a goal state in this epoch so best state remains the same.

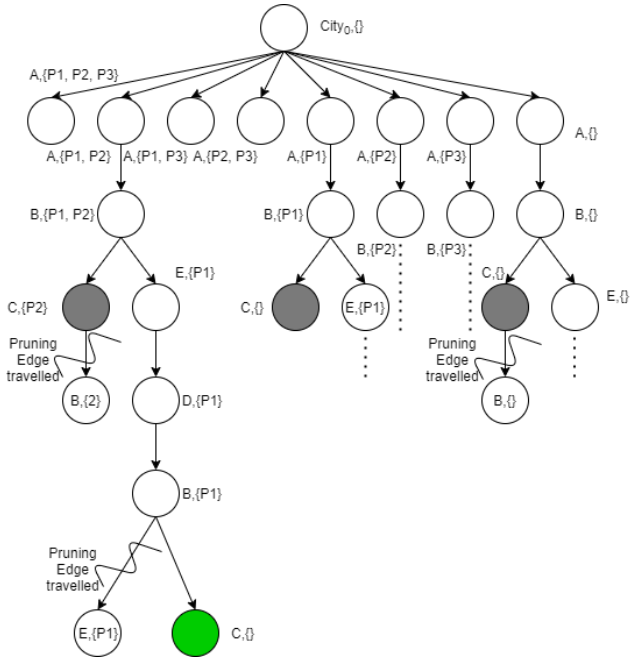

Figure 9: Search Graph after $epoch_3$, we are able to find a better state taking the bath $A \to B \to E \to D \to B \to C$, with packets { P1 , P2} with a cost of $(1+1+1+2+2)-(100+50) = -143$

VII SUMMARY AND CONCLUSION

We first formulated the problem and argued that if implemented in real life this could be extremely beneficial for a wide range of Stakeholders like Business, Casual Drivers, Environmentalist etc. We then mathematically defined the problem and were able to model the road structures between cities of a county. We have also proved the computation Hardness of the problem.

In the later section, we were able to develop an Iterative Deepening Depth First Seach based algorithm to exactly solve the problem in the presence of enough computation resources. However, our Algorithm can he parallelized well and we could also search depending upon our computation resource. One might also think of a metacomputing based approach where how much to search is calculated by another algorithm.

Our algorithm would be able to find the solution for a small number of cities (20-30) and a small number of packets (15-20) within a few minutes. However, time will rise exponentially as parameters are increased. Our algorithm would be able to find some suboptimal good solution within an hour for large graphs and small packets (100000 Cities and a small number of packets (15-30) by efficient running as we would be able to run on less computational resources mode.

The really hard part of this problem is to make a real-time (running in a few seconds) algorithms giving good solutions as the problem is very computationally hard (proved in section 2). However one can look upon certain patterns in distributions and queries to reduce the time complexity of calculating suboptimal solutions.

We can conclude that we were able to develop an approximately real-time system for a small number of cities and parcels. We could be able to deploy the algorithm for major cities only for the country with a not very high amount of parcel traffic. One could also be able to implement our algorithm for small and remote regions where large transportation, promoting local trade, promoting the economy of these regions.

VIII FURTHER CHALLENGES

As already discussed a lot in this report there are a very large number of benefits to solve and deploy this problem. If one could be able to completely solve this, it could soon be a billion-dollar business with a large impact on society and the environment. However, making efficient real-time algorithms is still a challenge. One might think to develop good approximation algorithms to speed up the process. One might be able to think of heuristic and meta heuristic based approaches. One can formulate and try version of this problem, and implement it. One can also think of making better pruning rules thus saving time. Also, it could be though of trying different approaches other than search-based approaches.

We would like to add again that if given enough time and no constrains this could be taken as a major research project we would have been able to come up with a better(faster and memory-efficient) approach.

IX ACKNOWLEDGMENTS

We would not have be able to come up with this idea if not worked with Dr. Arti Pandey, Assistant Professor, Department of Mathematics, IIT Ropar on Carpooling Problem as intern in summers of 2019.

A very special thanks my colleague and friend, Rajat Sharma (B18CSE043) for review and valuable suggestions.

REFERENCES

[1] https://tracxn.com/d/unicorn-corner/india

[2] H. Prüfer. *Neuer Beweis eines Satzes über Permutationen*

[3] https://en.wikipedia.org/wiki/Pr%C3%BCfer_sequence

[4] A. P. U. Siahaan. Adjustable Knapsack in Travelling Salesman Problem Using Genetic Process. International Journal of Science Technoledge **4,** 46–55 (Sept. 2016).