Distributed Approach to Solve Large-scale Multi-vehicle Routing Problem in the Presence of Static and Dynamic Obstacles

Submitted in fulfillment of the requirements

for the degree of

Master of Technology

By

Ankit Singh

Roll no. 213070029

Under the guidance of

Prof. Dwaipayan Mukherjee

Department of Electrical Engineering

Specialization: Control and Computing (EE-2)

INDIAN INSTITUTE OF TECHNOLOGY, BOMBAY

Abstract

This report addresses the large-scale open vehicle routing problem where a group of vehicles begins from the start depots and traverses to a set of targets while ensuring all the targets are visited at least once by some vehicle .The objective is to solve large-scale routing problem in a dynamic environment within the reasonable time.A cost function is designed which is able to handle variable costs associated with the road.Cost variation is introduced because traffic on the road varies with time and for traffic estimation, a Machine Learning model is trained. Exact method is not able to handle a dynamic environment with large-scale so a distributed approach is being used with real-time cooperation amongst the vehicles. Distribution and cooperation mechanisms are hosted over ROS(Robot operating system). Some heuristics are used to cover the targets like the vehicle only checks for its 1^{st} neighbors and travels to the node which produces minimum cost for traversal.

Contents

1 Introduction **8**

2 Mathematical Formulation for Optimal Solution using Exact Method **10**

2.1 MILP Formulation[11] 10

3 Cost Function **13**

3.1 Limitations of Exact Method 13

3.2 Cost Function Design 13

4 Distributed approach for multi-vehicle path planning **17**

4.1 Brief introduction to ROS 17

4.2 Clusters formation based on the geometric location of nodes 21

4.3 Implementaion of greedy approach 22

4.4 Dynamic obstacles 22

4.5 Dynamic weights 23

4.6 Cooperative approach implemenation 27

5 Simulation Results for Test Case **29**

5.1 Building ROS network 29

5.2 Vehicle Path without cooperation 30

5.3 Vehicle path with cooperation 31

6 Contribution and Future Work **34**

6.1 Contribution 34

6.2 Future Work 34

7 Simulations details **35**

List of Figures

2.1 a. In-edges and out-edges of S b. Case when S = i . . 11

3.1 Sample problem [4] 14
3.2 Feasible Solution [4] 14

4.1 Ros FileSystem [11] 18
4.2 Rosmaster and inter-node communication management [11] 19
4.3 Rqt graph 21
4.4 Nodes distribution via K-means clustering[13] 21
4.5 Dynamic Obstacle 1 22
4.6 Dynamic Obstacle 2 22
4.7 Dataset of vehicles taking time 23
4.8 Predicted time taken by the vehicles 23
4.9 Showing regression model accuracy 23
4.10 Actual time-taken by vehicles 25
4.11 Actual time-taken 26
4.12 Predicted time taken 26

5.1 Running ROS master 29
5.2 Fully active distributed system 29
5.3 RQT graph showing Communication network 29
5.4 vehicle-2 path in cluster 2 if no cooperation 30
5.5 vehicle-1 path in cluster 1 without cooperation 30
5.6 vehicle-3 path in cluster 3 without cooperation 30
5.7 All Vehicle path in actual problem without cooperation . . 30
5.8 vehicle-2 cost 30
5.9 vehicle-1 cost 30
5.10 vehicle-3 cost 31
5.11 vehicle-2 path in cluster 2 with cooperation 31
5.12 vehicle-1 path in cluster 1 with cooperation 31
5.13 vehicle-3 path in cluster 3 with cooperation 32
5.14 All vehicle routes in actual problem with cooperation avoiding static and dynamic obstacles 32
5.15 Path of vehicles with cooperation and no obstacle 32

5.16 vehicle-1 cost in presence of dynamic obstacle 32
5.17 vehicle-2 cost in presence of dynamic obstacle 32
5.18 vehicle-3 cost in presence of dynamic obstacle 33
5.19 vehicle-1 cost in absence of dynamic obstacle 33
5.20 vehicle-2 cost in absence of dynamic obstacle 33
5.21 vehicle-3 cost in absence of dynamic obstacle 33

Thesis Approval

This thesis entitled **Distributed Approach to Solve Large-scale Multi-vehicle Routing Problem in the Presence of Static and Dynamic Obstacles** by **Ankit Singh** is approved for the degree of **Master of Technology**.

Examiners:

. . . .

. . . .

. . . .

. . . .

Supervisor:

. . . .

Chairperson:

. . . .

Date: . . . .

Place: . . . .

Declaration of the Academic Ethics

I declare that this written submission represents my ideas in my own words and where others' ideas or words have been included, I have adequately cited and referenced the original sources. I also declare that I have adhered to all principles of academic honesty and integrity and have not misrepresented or fabricated or falsified any idea/ data/ fact/ source in my submission. I understand that any violation of the above will be a cause for disciplinary action by the Institute and can also evoke penal action from the sources which have not been properly cited or from whom proper permission has not been taken when needed.

Date: 27-June-2023

Ankit Singh
(Roll No: 213070029)

ACKNOWLEDGEMENT

I would like to express my gratitude to my supervisor **Prof. Dwaipayan Mukherjee** for his constant support and guidance throughout the learning. His prompt suggestions with enthusiasm have helped me to complete the dissertation. I would also like to acknowledge Electrical Department, IIT Bombay for providing me with necessary resources to carry out the project.

Finally, and most importantly, I would like to thank the almighty for it is under His grace that we live, learn and flourish.

Date: 27-June-2023

Ankit Singh
(Roll No: 213070029)

Chapter 1

Introduction

In real world applications like transportation, logistics and supply chain management Vehicle Path Planning[6] is a core problem. Vehicle Path Planning is a combinatorial optimization problem, where the goal is to find optimal routes to each vehicle from the depots to a set of targets subject to constraints.VRP(Vehicle Routing Problem) has different variants in the literature. VRP with open routes was first discussed in [5] and was formally defined as Open vehicle routing problem(OVRP) in. MDOVRP[7] is an extension of VRP in which vehicles start from more than one depot and do not return to the depots after visiting the last target on its route. MDOVRP finds its application where the vehicles are not required to return to the same depots like delivery of packages.

Heterogeneity of vehicles is classified in Doshi et.al[4] as structural heterogeneity and functional heterogeneity. Structurally heterogeneous vehicles may differ in design, dynamics, fuel consumption, maximum speeds, maximum payload capacity, etc. Vehicles are said to be functionally heterogeneous if a target can't be visited by all the vehicles due to their functionalities. Equipping vehicles with different sensors may cause functional heterogeneity in vehicles. Further Sundar et. al[3] partition the set of targets into targets(common) that can be visited by any vehicle and targets(vehicle-specific) that can be visited only by specific vehicles. In our work we consider functionally homogeneous vehicles and targets are taken as common targets which can be visited by any vehicle.

Multi-depot open vehicle routing problem with vehicle target constraints (MDOVRP-VTC) differs from MDOVRP in three aspects. a) In MDOVRP vehicle routes end in targets whereas vehicles end in depots in MDOVRP-VTC. b) A target can be visited by all the vehicles in MDOVRP but in case of MDOVRP-VTC there may be targets that can only be visited by specific vehicles. c) A target must be visited only once in MDOVRP whereas multiple visits to a target is allowed in MDOVRP-VTC, although for future work I considered an environment where multiple vehicles covering all the targets considering vehicles starts from a depot and end at a depot.

MDOVRP-VTC problem with a single depot and all common targets reduces to a well-known Traveling Salesman Problem(TSP) which is a NP-Hard problem. The exact solution for such problems is based on mixed integer linear programming(MILP) formulations and branch.The exact methods available can't solve most real-world instances of the problem to optimality within a reasonable time because of the exponential time complexity. So a heuristic approach is being used, which assumes the vehicle only cares about its neigbors and will travel to the node which produces lowest cost. To solve a large scale problem we used distributed approach and solves the whole problem with cooperation. Distribution and cooperation mechanisms are hosted over ROS(Robot operating system)[12].

The remainder of this report is organized as follows.In chapter 2, formulated and simulated the Exact method, chapter 3 defines the cost function design ,chapter 4 shows the distributed approach for multi vehicle path planning,chapter 5 contains simulation Results for various test cases,chapter 6 talks about contribution and future scope and chapter 7 contains the code for the whole project.

Chapter 2

Mathematical Formulation for Optimal Solution using Exact Method

Problem Definition

Assumption 2.1 The vehicles are functionally homogeneous.

Assumption 2.2 A direct path exists between every pair of targets but not between depot.

Assumption 2.3 A vehicle cannot visit any depot on its path from start depot to stop depot.

Given a fleet of functionally homogeneous vehicles stationed at the depots and a set of targets, the goal is to find a path for each vehicle that start and end at the depots assigned to it such that

1. Each target is visited at least once by some vehicle.
2. The vehicle-target constraints are satisfied.
3. The total cost of the paths travelled by all the vehicles is minimum.

Satisfying the vehicle target constraints means, every vehicle should start from a depot and end at a depot while travelling to targets in between. Figure 2.1 shows a test scenario with a feasible solution.

2.1 MILP Formulation[11]

Let N denote the set of targets, $D = \{d1, d2, ..., dn\}$ represent the set of the depots and the total number of vehicles be m. Every vehicle is associated with an ordered pair (d_{sk}, d_{fk}), where d_{sk} is the start depot and d_{fk} is the stop depot for the vehicle k.

The connectivity between the depots and targets are represented as a directed graph G = (V, E) where V = $D \cup N$ is the node set and E is the edge set. The cost incurred by a vehicle to traverse the edge $(i, j) \in E$ is represented by C_{ij} and for each vehicle k a variable x_{ij}^k is associated with each edge (i, j), whose value is 1 if the edge (i, j) is traversed by vehicle k and 0 otherwise. For each vehicle k, we associate with each node i a binary variable y_{ik}, which takes a value of 1 when node i is visited by vehicle k and 0 otherwise.

For any $S \subset V$, we define $\delta^+(S) = \{(i, j) \in E : i \in S, j \notin S\}, \delta^-(S) = (i, j) \in E : i \notin S, j \in S$ Figure 2.2 shows $\delta^+(S)$ and $\delta^-(S)$ which are set of edges exiting and entering set S. For any $\tilde{E} \subseteq E$, we define

$$x^k \tilde{E} = \sum_{(i,j) \in E} x_{ij}^k$$

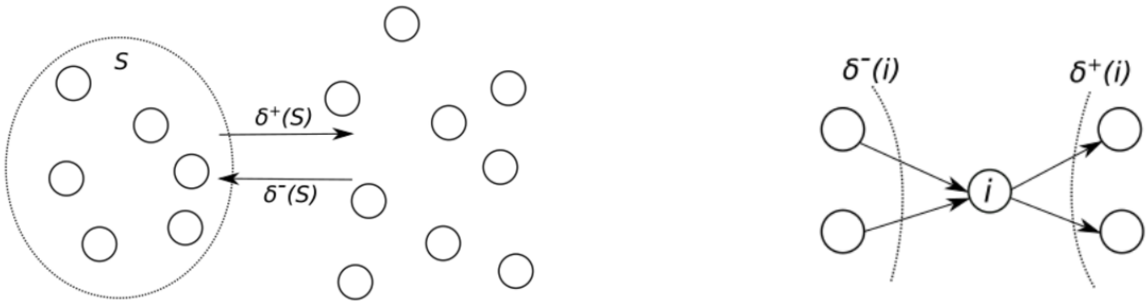

Figure 2.1: a. In-edges and out-edges of S b. Case when S = i

A. Objective:

$$J(x) : \sum_{k=1}^{m} \sum_{(i,j) \in E} c_{ij} x_{ij}^k$$

where E is edge set

The objective[11] minimizes the total cost of the paths traveled by all the vehicles in order to visit all the targets at least once.

B. Degree constraints:

$$x^k \delta^+(S)) = x^k \delta^-(i) \qquad \forall i \in T, k \in 1, ..., m \tag{2.1}$$

The constraint (2.1) ensures that for a vehicle, the in-degree for a target is equal to its out-degree.

C. Vehicle-target constraints:

$$\sum_{k=1}^{m} y_i^k \geq 1 \quad \forall i \in T \tag{2.2}$$

The constraints (2.2) ensure that each target is visited by at least one vehicle

D. Vehicle-depot assignments:

$$x^k(\delta^+(d_{sk})) = 1 \qquad \forall k \in 1, ..., m \tag{2.3}$$

$$x^k(\delta^-(d_{fk})) = 1 \qquad \forall k \in 1, ..., m \tag{2.4}$$

$$x^k(\delta^+(d)) = 0 \qquad \forall k \in 1, ..., m, d \in D \setminus d_{sk} \tag{2.5}$$

Constraints (2.3) to (2.5) ensure that each vehicle visits at least one target and that its path starts and ends at the depots assigned to it.

E. Connectivity constraints:

$$x^k(\delta^+(S)) \geq y_i^k \qquad\qquad \forall i \in S, S \subseteq T, k \in 1, ..., m \tag{2.6}$$

The constraints in (2.6) eliminate sub-tours of any subset of targets for each vehicle. They also ensure that for each vehicle, its path remains connected.

F. Variable restrictions:

$$x_{i\,j}^k \in \{0, 1\} \qquad\qquad \forall (i, j) \in E, k \in 1, ..., m \tag{2.7}$$

$$y_i^k \in \{0, 1\} \qquad\qquad \forall i \in T, k \in \{1, ..., m\} \tag{2.8}$$

The constraints in (2.7) and (2.8) denote the binary restrictions on the decision variables.

The above MILP formulation provides insights for understanding the MDOVRPVTC properties better and helps in solving the problem to optimality. Further, it can lead to the computation of lower bounds which provides an estimation of the quality of sub-optimal solution.

Chapter 3

Cost Function

3.1 Limitations of Exact Method

- **Scalability**: As the number of nodes increases in the vehicle routing problem, exact method struggle to solve in reasonable time because complexity of exact algorithm is exponential. The number of solutions for a vehicle routing problem is of order n!, where n is number of nodes. so the time required to solve the problem grows exponentially as the size of the problem increase, making it ineffective and impractical to find an optimal solution for such cases.
- **Intensive Computation**: Solving large-scale vehicle routing problem through exact method requires significant computational resources and time. Solvers like CPLEX are unable to handle the problem after a certain point of scale of the problem
- **Difficulty in Handling Dynamic changes**: Consider if the weights of edges change in real-time or connectivity of the nodes changes then Exact methods are not well-suited for such kind of dynamics.

Due to the limitations of the exact method, it is not able to solve the problem at large scale, it is also ineffective if the conditions of the problem change in real-time. So a heuristic approach is being used to get the best solution for the problem which includes a trade-off between optimality and efficiency.so, we need to design a cost function that not only penalizes the distance but also the time a vehicle takes to cover a particular edge.

3.2 Cost Function Design

Consider a scenario we are assigned to a task where we need to deliver packages at a different location(nodes) in a city and in that city we have depots where all the packages are stored. To deliver all the packages we have multiple vehicles so our task is to deliver all the packages in such a way it produces minimum delivery cost for us. Fig 3.1 Shows a sample for delivery location and Fig 3.2 shows an optimal solution with minimum cost.

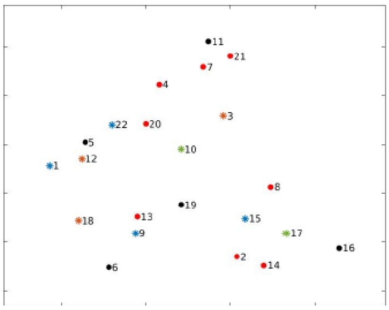

Figure 3.1: Sample problem [4]

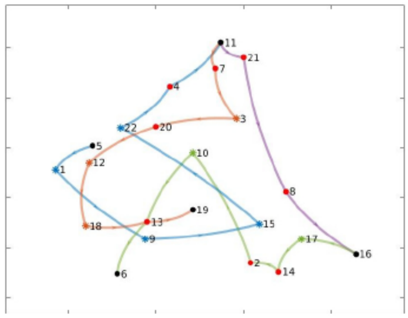

Figure 3.2: Feasible Solution [4]

The next possible question could be, how to assign a cost value to a path(edge) between two delivery locations so that the best path can be chosen for a vehicle. Available literature only considered the distance between the two nodes to get an idea of the cost and completely ignored the idea of the time taken to cover the path.

Consider a situation where we have two paths (path-1 and path-2) from location A to location B . Path-2 distance is large in comparison to path1, but path1 is blocked by heavy traffic because of an accident on the road then, of course, in this scenario we will not take that road which has less distance (path1). So we can say the time a vehicle takes to cover a path also matters and must be included in the cost function.

Let c_{ij} be the cost when we travel between the node i and j and x_{ij} is the decision variable.

So the cost can be formulated as

$$J(t, x) = \sum_{k=1}^{m} \sum_{(i,j)\in E} C_{ij}(t) x_{ij}^{k}$$

where E is edge set

Calculation for C_{ij} :

C_{ij} broadly depends on the distance between the nodes and the time taken by the vehicle to cover the path. Further analysis of the time taken can help to realize that the time taken by the vehicle further depends mainly on two factors:

1. The Quality of the road
2. The Traffic on the road

So, C_{ij} can be written as

$$C_{ij} = \alpha_1 \cdot d_{ij} + \alpha_2 \cdot q_{ij} + \alpha_3 \cdot r_{ij}(t)$$

where d_{ij} is the distance between the nodes i and j.
q_{ij} is the relative quality factor of the road (i, j)
$r_{ij}(t)$ is the relative traffic factor at time t
α_1 , α_2, α_3 are the penalising factors

The next possible question could be, how to make all these factors accountable.

A) Distance :
Distance between the nodes is easily available. A huge dataset is available for distances between the two points, We can use directly the values of distances through a pipeline that can directly feed the distance data in our system.

B) Quality of the road:
Manual checks for every road quality itself is a very tedious and time-consuming task, so we need to develop a way to get an estimate for the quality of the road. A common observation is that if road quality is not good, then the vehicle will take more time to cover the same distance as for a good-quality road. So,vehicles taking the time to cover an edge can be used to approximate the relative road quality.
The quality of a road can be approximated by data analysis of vehicle time history with following steps:

1. Reference road (like express highways in a city) distance d with best quality and no traffic, calculate the average ideal time the vehicles take to cover(T^{ideal}).
2. Calculate average ideal time for every road (T_{ij}^{ideal}).
3. For any road, consider a set of some lowest times taken by the vehicles as per availability of data and calculate an average of the set (T_{ij}^{low}).
4. The difference between T_{ij}^{ideal} and T_{ij}^{low} can provide an estimate for the quality of the road.

Since the quality of the road is not decaying very fast, so quality can be considered as a constant quantity for a road.

$$q_{ij} = \frac{(T_{ij}^{low} - T_{ij}^{ideal})}{T_{ij}^{ideal}}$$

C) Traffic on the Road:

Traffic on the road is time specific for eg. In the morning, like 5 am, the vehicle density on the road is expected to be less than at 10 am. A dataset[9] that shows the time taken by the vehicle in different time frames throughout the day, can be used to mimic the traffic behavior using Machine learning techniques. This ML model can be used to predict the time -taken by the vehicle at a particular time t.
Traffic on the road can be quantified by following steps:

1. Generate a data set containing variables such as edge-id, day, time slot, and time taken to cover.
2. A machine learning regression model(XGboost) is trained that takes inputs such as edge-id, day, and time slot and can be used to predict the time a vehicle will take when it traverses the edge (i, j) as $T_{ij}(t)$.
3. The difference in T^{low} and $T_{ij}(t)$ can provide a reasonable estimate for the traffic on the road at that time.

The relative traffic factor can be calculated as:

$$r_{ij} = \frac{(T_{ij}(t) - T_{ij}^{low})}{T_{ij}^{low}}$$

So C_{ij}(t) can be written as

$$C_{ij}(t) = \alpha_1 \cdot d_{ij} + \alpha_2 \cdot \frac{(T_{ij}^{low} - T_{ij}^{ideal})}{T_{ij}^{ideal}} + \alpha_3 \cdot \frac{(T_{ij}(t) - T_{ij}^{low})}{T_{ij}^{low}}$$

Where

- $\alpha_1 = 1$
- $\alpha_2 = 0.25 \cdot d_{ij}$
- $\alpha_3 = 0.5 \cdot d_{ij}$

So, the Cost function can be written as

$$J(t, x) = \sum_{k=1}^{m} \sum_{(i,j)\in E} d_{ij}(1 + 0.25 \cdot \frac{(T_{ij}^{low} - T_{ij}^{ideal})}{T_{ij}^{ideal}} + 0.5 \cdot \frac{(T_{ij}(t) - T_{ij}^{low})}{T_{ij}^{low}}) \cdot x_{ij}^{k}$$

Chapter 4

Distributed approach for multi-vehicle path planning

The main problem with Exact method is that it is not able to solve a large-scale problem and the reason for this is a vehicle needs to explore all the possible ways to travel at each step, for a fully connected graph it is impossible to explore all the paths for a common feature machine, so initial allocation of nodes to the vehicles can play an important role to decrease the computation requirement so that each vehicle now has no need to see a full graph but a subgraph only of nodes which are allocated to that particular vehicle and traverse through in each graph independently but having proper communication between the vehicle, to implement this idea a distributed network can be used like ROS(robot operating system), which help to disintegrate a bigger problem into smaller pieces and solve the whole problem with a cooperation among the vehicles. All clusters can communicate with each other but it is not mandatory that they should also be connected to each other.

4.1 Brief introduction to ROS

The ROS Robotics Operating System[10] is a freely available open-source framework widely used in robotics research and development. It facilitates the creation of complex applications for robots by means of a set of tools, libraries, and conventions:

- Conceptual Framework: ROS is based on an infrastructure where functions are split up into independent nodes communicating with one another via messages passed between identified topics. This decoupling provides flexibility to develop and integrate the different components in a modular way.

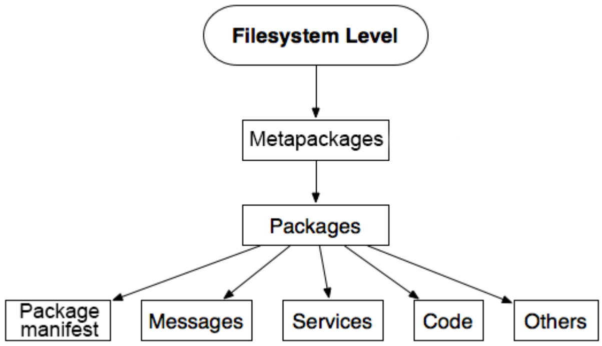

Figure 4.1: Ros FileSystem [11]

- Message Passing: Nodes in ROS communicate by publishing and subscribing to messages on specific topics. There are several types of data in messages, for example, sensor readings, control commands or custom data structures. This messaging system enables seamless communication between nodes regardless of programming language or physical location.
- Package System: ROS organizes software components into packages, which are self-contained units containing nodes, libraries, configuration files, and other resources. In order to foster collaboration in the robotics community, packages could be easy to share and reuse.
- Tools and libraries: In order to simplify the development of robots, ROS offers a wide range of tools and libraries. The roscore, which will be a key node for ROS network management and the roslaunch tool used to launch several nodes in order to set up an environment are some of the main components. In addition, ROS offers a set of libraries that cover essential functionalities like robot control, perception, mapping, and navigation.
- Visualisation and Debugging: To assist in the design and analysis of robot systems, ROS comes with visualization and debugging capabilities. The 3D visualization of sensor data, robot models, and trajectory is possible using the RViz tool. An adaptable graphical interface for monitoring and troubleshooting ROS nodes is offered by the rqt suite.
- Ros message: Messages are the primary data types for communication between nodes in the ROS (Robot Operating System). A message is a data structure that specifies the information being transmitted

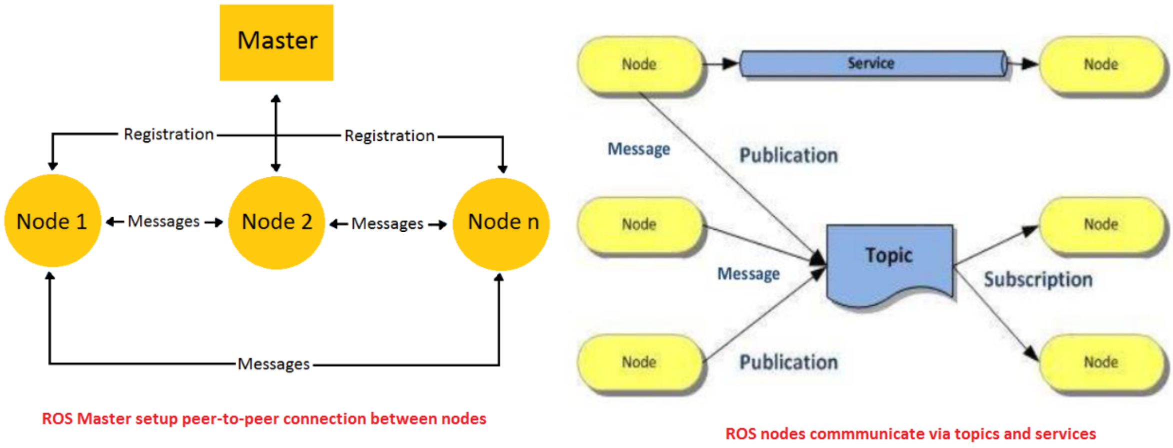

Figure 4.2: Rosmaster and inter-node communication management [11]

between various ROS system components. Data of many kinds, including sensor readings, instructions, status updates, and more, may be represented by messages.
The.msg file format, which defines the message's fields and data types, is used to define ROS messages. These message definitions are normally kept in the msg directory of a package. The following are some essentials regarding ROS messages:

- Message Types: A variety of built-in message types are available in ROS for various data formats, including the std.msgs standard messages package for basic data kinds including strings, integers, floats, and time.
- Custom messages: In addition to the built-in message types, you can define your own custom message types using the .msg file format. Custom messages allow you to define message structures that meet the needs of your application.
- Fields and Data Types: A message consists of one or more fields, where each field has a name and a data type. The data types can be basic types like integers, floats, and strings, or more complex types like arrays, nested messages, or custom message types.
- Publishing and Subscribing: Nodes can publish messages on a specific topic, allowing other nodes to subscribe to those topics and receive the published messages. This publish-subscribe mechanism enables communication and data sharing between different parts of a ROS system.
- Message Serialization: Messages are serialized and deserialized when sent over the ROS network. Serialization is the process

of converting a message into a compact binary format for transmission, and deserialization is the reverse process of reconstructing the message on the receiving end.

- Ros node: The rosnode command-line tool allows you to perform various operations related to nodes in a ROS system. Use rosnode list to get a list of all active nodes in the ROS system. The command rosnode graph generates a visual representation of the node graph, showing the connections between nodes based on their topic communications.

- Roscore: The roscore command is used to start the core infrastructure of the ROS system. It initializes the ROS Master, which acts as a central registry for all ROS nodes, topics, services, and parameters. Running roscore starts the ROS Master, which is a crucial component for ROS communication. It must be running for other ROS nodes to connect and communicate. The ROS Master provides a central hub for nodes to discover each other and exchange information. It facilitates the establishment of communication channels for topics, services, and parameters. Before launching or using other ROS tools, such as running nodes, publishing/subscribing to topics, or calling services, you need to ensure that roscore is running.

- Ros topic: Rostopic is a command-line tool in ROS that provides a way to interact with topics. Topics are a fundamental communication mechanism in ROS, allowing nodes to exchange data by publishing and subscribing to messages on specific topics. rostopic list to get a list of all available topics in the ROS system. The command rostopic info <topicname> provides information about a specific topic, including its type, publisher(s), and subscriber(s).To manually publish messages to a topic, you can use rostopic pub <topicname> <messagetype> <args>. This command allows you to simulate data publishing to a topic for testing purposes. The rostopic echo <topicname> command subscribes to a topic and displays the messages published on that topic in real time.

- RQT graph: The rqtgraph command is used to launch the graphical tool called "rqtgraph" in ROS.It provides a visual representation of the ROS graph, showing the nodes and their connections (topics, services, and parameters).

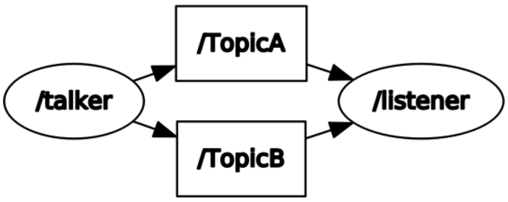

Figure 4.3: Rqt graph

4.2 Clusters formation based on the geometric location of nodes

Distribution needs to be done to allocate the nodes to specific vehicles based on the depots. For the allocation of the nodes K-means clustering Method is being used which is based on the distance of nodes from the depots. Consider we have 3 depots in a city so based on the distance of every node location from the depots we allocate that node to a particular depot.

Implementation of k-means clustering method

- Choose the number of clusters equal to depots, k.
- Select k points (clusters of size 1) at random.
- Calculate the distance between each point and the centroid and assign each data point to the closest cluster.
- Calculate the centroid (mean position) for each cluster.
- Keep repeating steps 3–4 until the clusters don't change or the maximum number of iterations is reached.

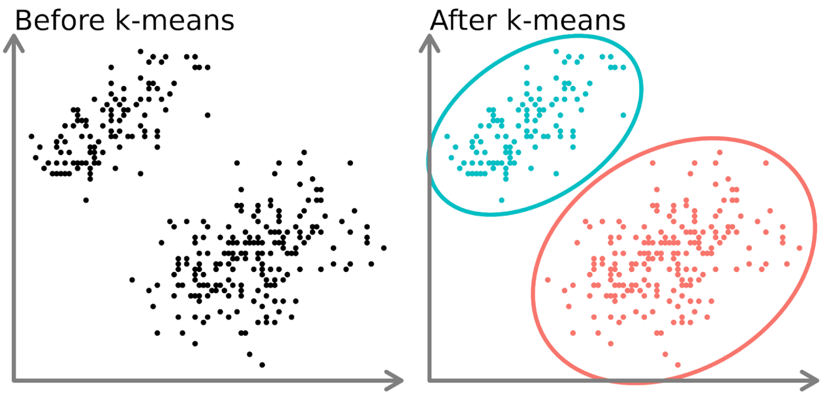

Figure 4.4: Nodes distribution via K-means clustering[13]

In this case, centroid is the depot location which is variable but depot location is fixed so we need to run above algorithm only once

4.3 Implementaion of greedy approach

Large-scale problem is disintegrated into smaller problems based on the number of depots, each cluster is now a smaller problem because a vehicle is not required to see the full graph, it traverses in its cluster which makes it more effective for a large-scale problem. For the traversal strategy in the cluster, Exact method can not be used because of its limitation to be ineffective in changing environment.

The idea is that if the vehicle is at a node, then we can only travel to that node that is connected to that particular node, so choose the edge with minimum cost.

This strategy is also classified as a greedy approach, which certainly will not provide an optimal solution but a good solution in real-time with changing environment. Each vehicle will cover the nodes in its cluster using a greedy approach.

4.4 Dynamic obstacles

Consider the situation if a vehicle is covering the nodes in a cluster but due to traffic, the road got blocked which means that a particular road can not be used and these types of changes are very random if the algorithm is not equipped with such scenarios then it will terminate immaturely which makes the solution of no use. To overcome a situation like this we need to route the vehicle in such a way that it can avoid the situation. This is only possible if we explore the connectivity outside of its cluster and use this vehicle to cover the node of another cluster.

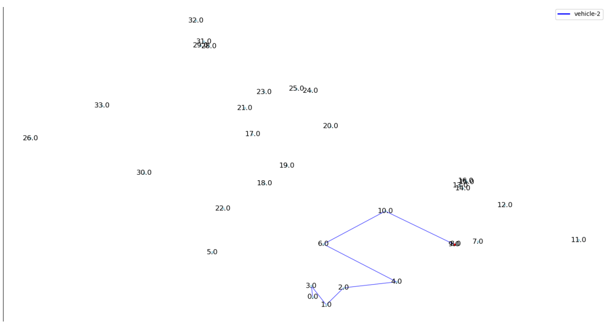

Figure 4.5: Dynamic Obstacle 1

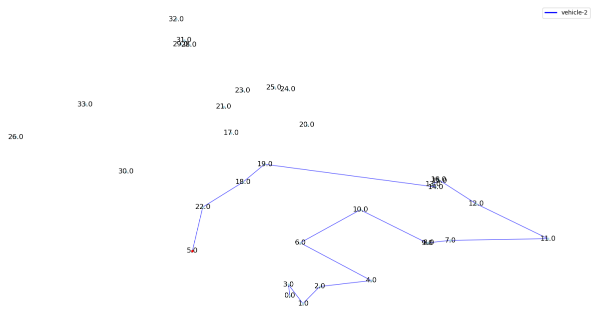

Figure 4.6: Dynamic Obstacle 2

4.5 Dynamic weights

As we discussed in the 2nd chapter the cost of traveling on a road may not be the same as it can vary with time. Now the agenda is how can we implement this variable cost in our solution. As the idea suggested in the 2nd chapter the cost variation is mainly introduced due to traffic on the road. Traffic on the road is variable but very much predictable, So a machine-learning method can be deployed to predict the possible time a vehicle can take to traverse the road. we got the dataset of 135 nodes with 1,00,000+ instances which can be used to train an ML regression model. we have used the Xgboost gradient decent method which helps to get a RMSE equal to 1.7. which is not an excellent case but provides good results over a large dataset

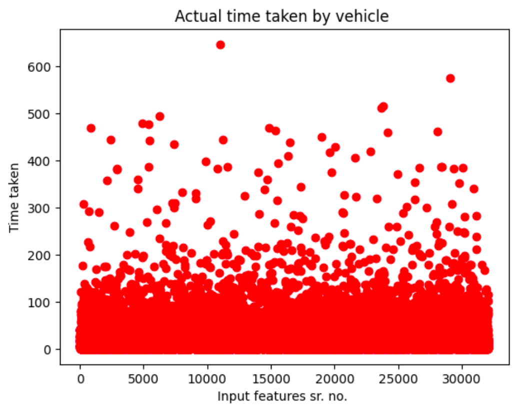

Figure 4.7: Dataset of vehicles taking time

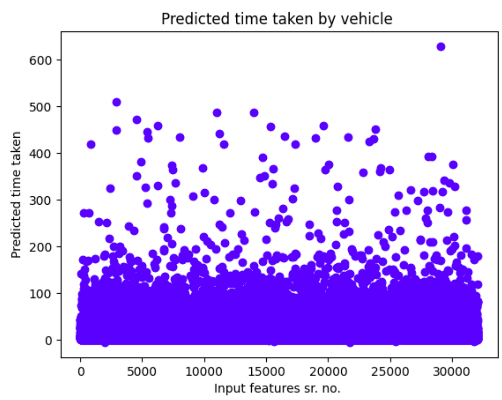

Figure 4.8: Predicted time taken by the vehicles

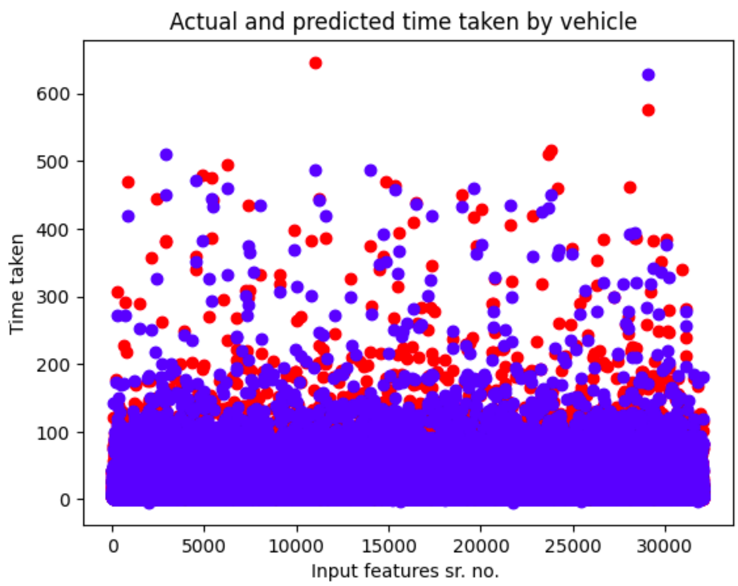

Figure 4.9: Showing regression model accuracy

Use of travel time:-

Using the ML regression model time taken by a vehicle to cover the road is a prior known quantity that can be used to draw a lot of useful insights from this data. It can provide an idea of traffic on the road which we can quantify and use this traffic scale to estimate the cost of travel in real time. Also, it can help to get an idea of the quality of the road.

Here are the proposed method to get the road quality factor and traffic factor on the road.

Quality of the road:

Manual checks for every road quality itself is a very tedious and time-consuming task, so we need to develop a way to get an estimate for the quality of the road. A common observation is that if road quality is not good, then the vehicle will take more time to cover the same distance as for a good-quality road. So,vehicles taking the time to cover an edge can be used to approximate the relative road quality.

The quality of a road can be approximated by data analysis of vehicle time history with following steps:

1. Reference road (like express highways in a city) distance d with best quality and no traffic, calculate the average ideal time the vehicles take to cover(T^{ideal}).
2. Calculate average ideal time for every road (T_{ij}^{ideal}).
3. For any road, consider a set of some lowest times taken by the vehicles as per availability of data and calculate an average of the set (T_{ij}^{low}).
4. The difference between T_{ij}^{ideal} and T_{ij}^{low} can provide an estimate for the quality of the road.

Since the quality of the road is not decaying very fast, so quality can be considered as a constant quantity for a road.

$$q_{ij} = \frac{(T_{ij}^{low} - T_{ij}^{ideal})}{T_{ij}^{ideal}}$$

Here is the analysis of a road from the dataset[9]
Road-ID: 3.37790628002851E+018 and $(i, j) = (4, 6)$

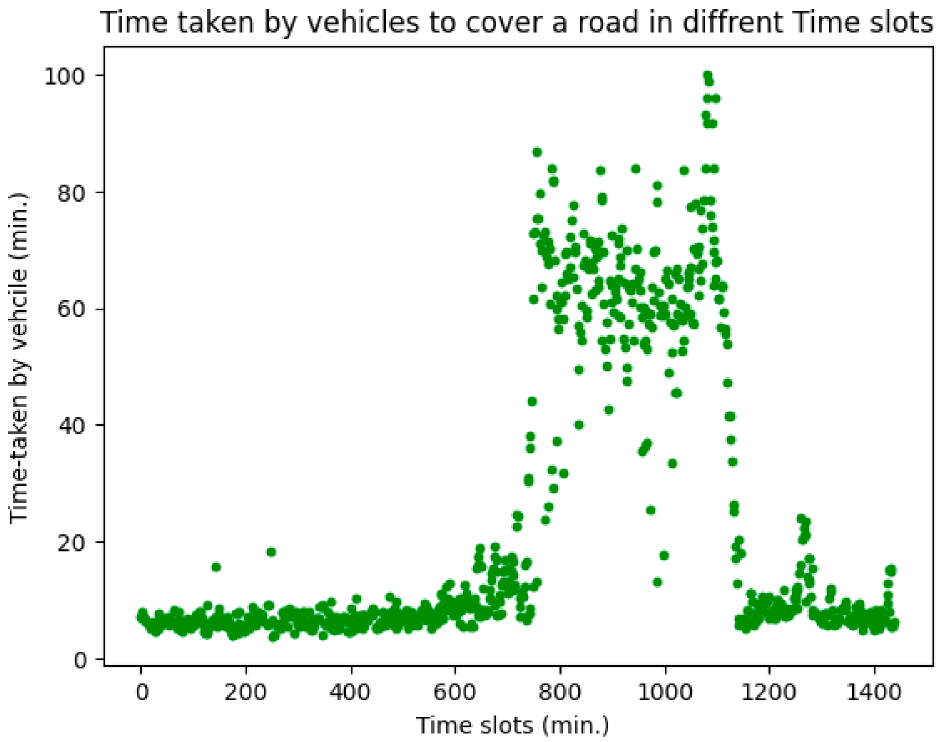

Figure 4.10: Actual time-taken by vehicles

$T_{46}^{ideal} = 2.4$min.
$T_{46}^{low} = 3.7$min.

so the relative quality factor :

$$q_{46} = \frac{3.7 - 2.4}{2.4} = 0.54$$

Traffic on road:
Traffic on the road is time specific for eg. In the morning, like 5 am, the vehicle density on the road is expected to be less than at 10 am. A dataset[9] that shows the time taken by the vehicle in different time frames throughout the day, can be used to mimic the traffic behavior using Machine learning techniques. This ML model can be used to predict the time -taken by the vehicle at a particular time t.
Traffic on the road can be quantified by following steps:

1. Generate a data set containing variables such as edge-id, day, time slot, and time taken to cover.

2. A machine learning regression model(XGboost) is trained that takes inputs such as edge-id, day, and time slot and can be used to predict the time a vehicle will take when it traverses the edge (i, j) as $T_{ij}(t)$.
3. The difference in T^{low} and $T_{ij}(t)$ can provide a reasonable estimate for the traffic on the road at that time.

The relative traffic factor can be calculated as:

$$r_{ij}(t) = \frac{(T_{ij}(t) - T_{ij}^{low})}{T_{ij}^{low}}$$

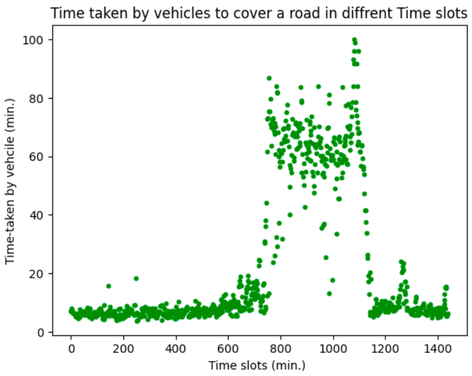

Figure 4.11: Actual time-taken

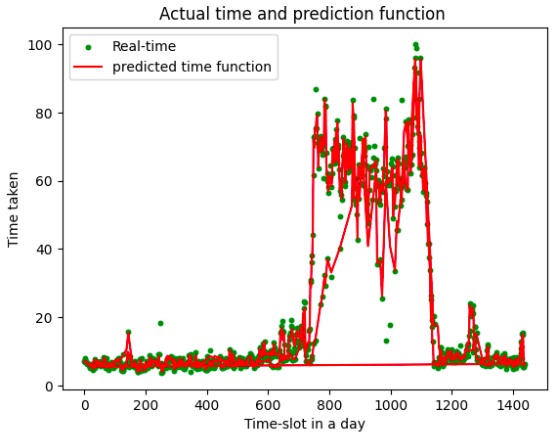

Figure 4.12: Predicted time taken

$T_{46}^{ideal} = 2.4\text{min.}$
$T_{46}^{low} = 3.7\text{min.}$

$T_{46}(11a.m) = 10.3\text{min.}$

$T_{46}(6p.m) = 100\text{min.}$

So the relative traffic factor :

$$r(1, 11a.m) = \frac{10.3 - 3.7}{3.7} = 1.78$$

and C_{ij}(t) can be written as

$$C_{ij}(t) = \alpha_1 \cdot d_{ij} + \alpha_2 \cdot q_{ij} + \alpha_3 \cdot r_{ij}(t)$$

Where d_{ij} is in km and

- $\alpha_1 = 1/km.$
- $\alpha_2 = 0.12/km \cdot d_{ij}$

- $\alpha_3 = 0.25/km \cdot d_{ij}$

So

$$C_{46}(11a.m) = 6.04$$

$C_{46}(t)$ at $6p.m$ will be:

$$C_{46}(6p.m) = 30.25$$

It shows that traveling to the road in the evening around $6p.m$ produces more cost than in morning around $11a.m$.

The cost function :

$$J(t,x) = \sum_{k=1}^{m} \sum_{(i,j)\in E} (1 \cdot d_{ij} + 0.25 \cdot d_{ij} \cdot \frac{(T_{ij}^{low} - T_{ij}^{ideal})}{T_{ij}^{ideal}} + 0.5 \cdot d_{ij} \cdot \frac{(T_{ij}(t) - T_{ij}^{low})}{T_{ij}^{low}}) \cdot x_{ij}^{k}$$

Based on the calculated cost in real-time we traverse the vehicle so that it can provide the best solution in a real-time scenario

4.6 Cooperative approach implemenation

Simple clustering of the nodes and solving independently each cluster facilitates very less control over the full-scale problems, consider the situation if a vehicle is not able to move further due to a roadblock then all the remaining nodes will remain unvisited and another situation can be if we have density variation of the nodes in the cluster, for e.g 2 cluster one with high density, so the vehicle with low density has fewer nodes to travel, it will finish its work very early while another vehicle has work to travel if there is no cooperation then it will not be an effective use of resources.

So a cooperative approach needs to be implemented, ROS is a distributed network in which every vehicle can communicate with another vehicle if a vehicle got stuck or completed its task then it will travel to another cluster.

we used the dataset from the Alibaba website of 135 nodes. which helps to create the graph. ROS network has different ros-nodes, each handling a different and unique functioning. And there is a unique flow of information, Weight-rosnode handles the weights of the edges. The k-clustering node performs the k-means clustering method and distributes the subgraph information to different nodes based on the number of clusters, next, we have cluster nodes that uniquely solve the subproblems with bidirectional communication between each cluster node.Then every cluster node is connected with a final-output rosnode which gathers the data of each vehicle's travel history and generates a graph of the path taken by each vehicle.

Chapter 5

Simulation Results for Test Case

5.1 Building ROS network

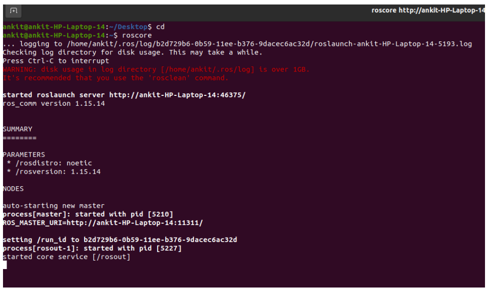

Figure 5.1: Running ROS master

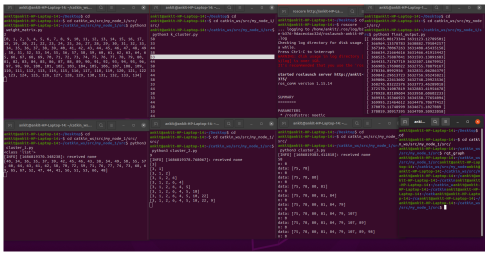

Figure 5.2: Fully active distributed system

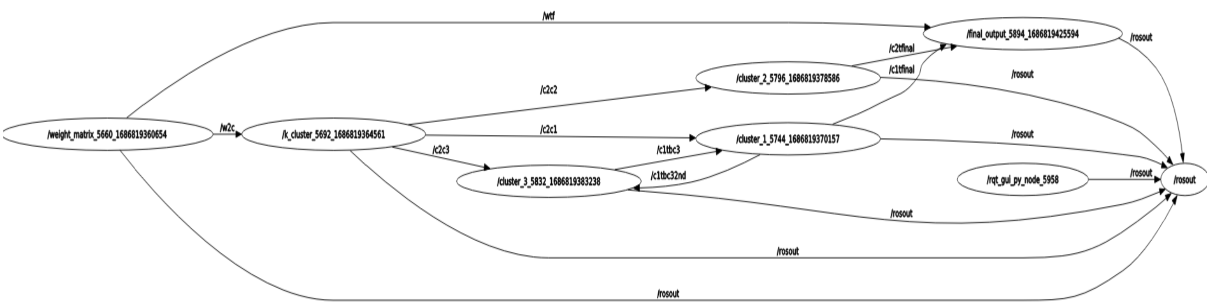

Figure 5.3: RQT graph showing Communication network

5.2 Vehicle Path without cooperation

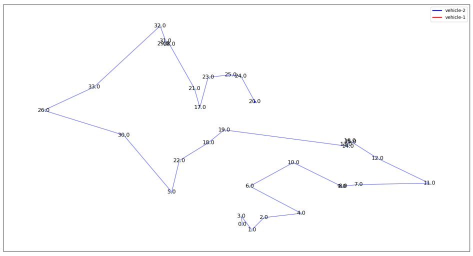

Figure 5.4: vehicle-2 path in cluster 2 if no cooperation

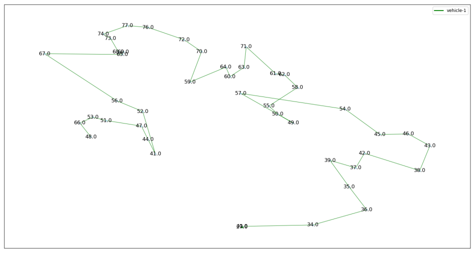

Figure 5.5: vehicle-1 path in cluster 1 without cooperation

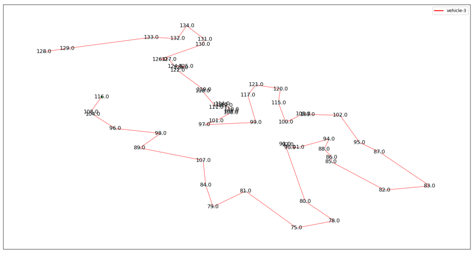

Figure 5.6: vehicle-3 path in cluster 3 without cooperation

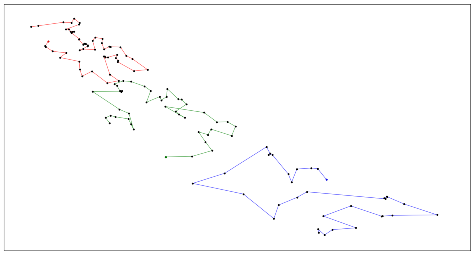

Figure 5.7: All Vehicle path in actual problem without cooperation

Cost for each vehicle

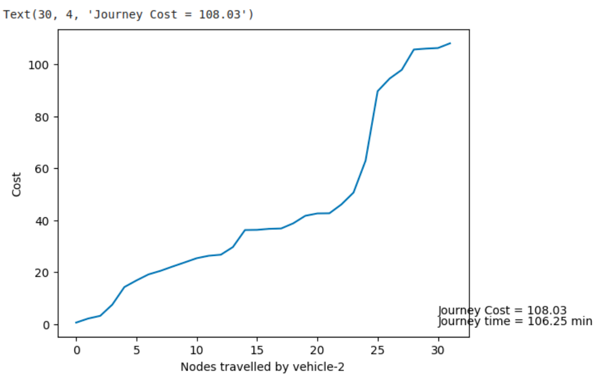

Figure 5.8: vehicle-2 cost

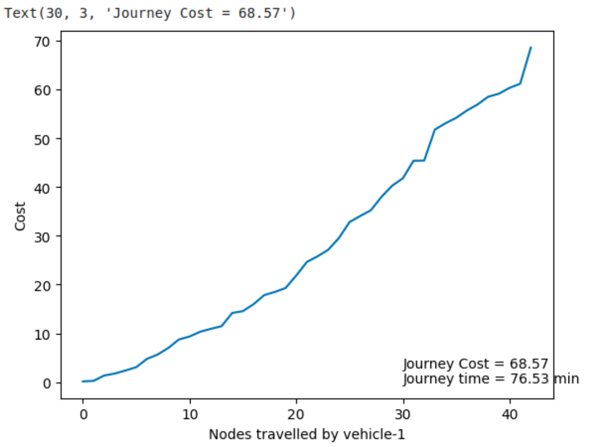

Figure 5.9: vehicle-1 cost

```
Text(30, 3, 'Journey Cost = 57.88')
```

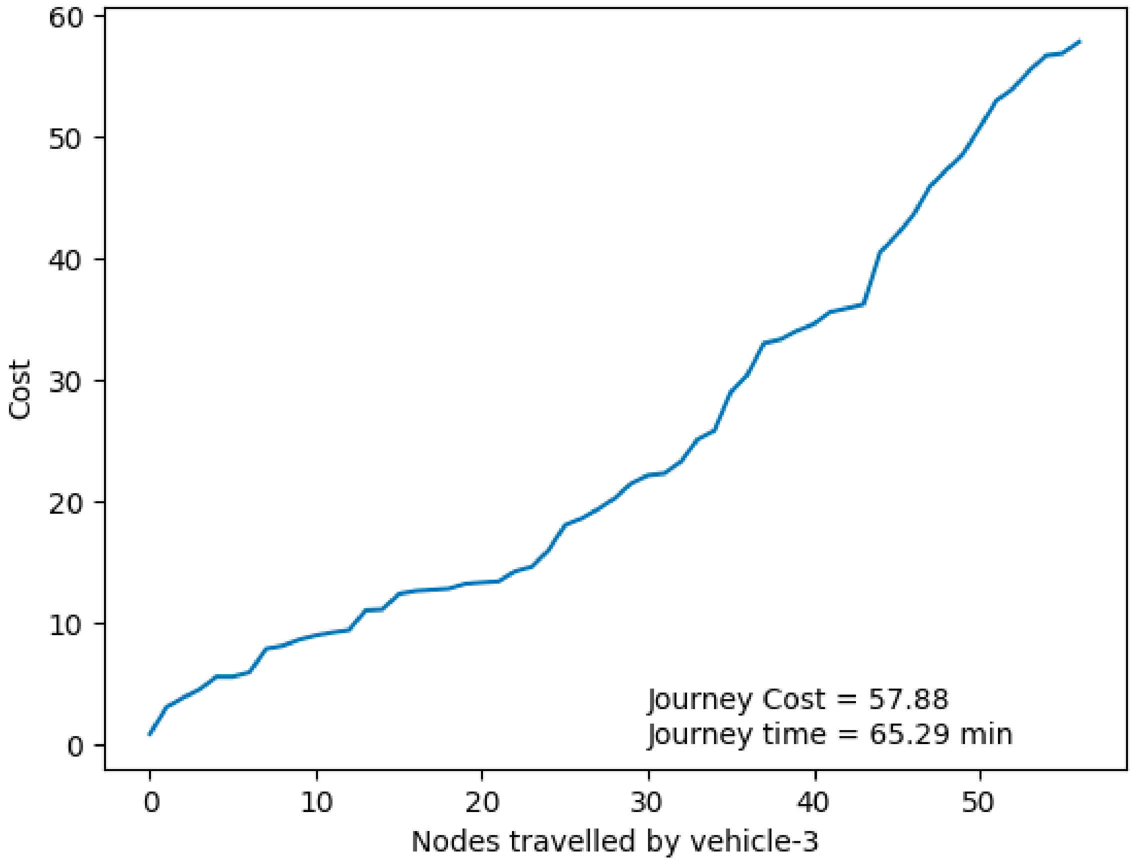

Figure 5.10: vehicle-3 cost

5.3 Vehicle path with cooperation

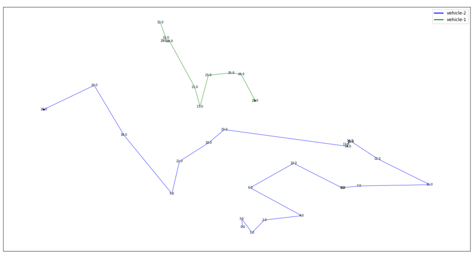

Figure 5.11: vehicle-2 path in cluster 2 with cooperation

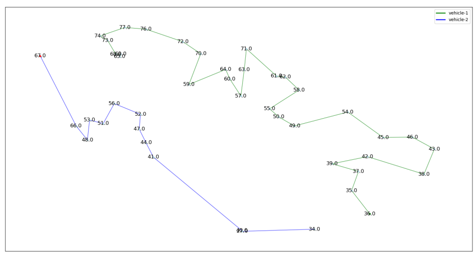

Figure 5.12: vehicle-1 path in cluster 1 with cooperation

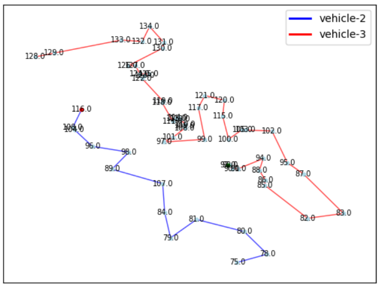

Figure 5.13: vehicle-3 path in cluster 3 with cooperation

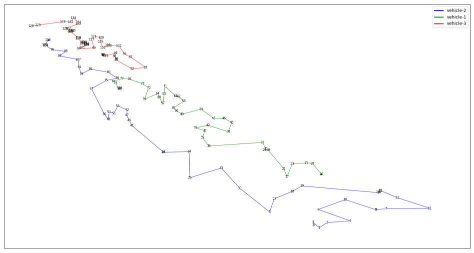

Figure 5.14: All vehicle routes in actual problem with cooperation avoiding static and dynamic obstacles

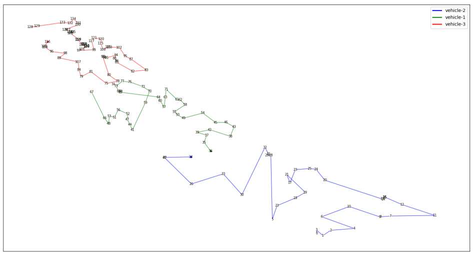

Figure 5.15: Path of vehicles with cooperation and no obstacle

Cost for each vehicle with and without obstacles

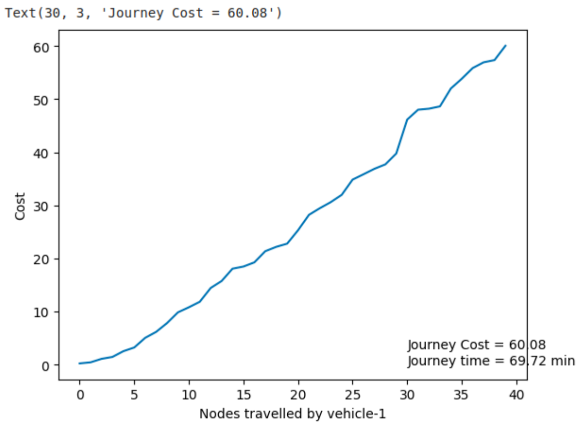

Figure 5.16: vehicle-1 cost in presence of dynamic obstacle

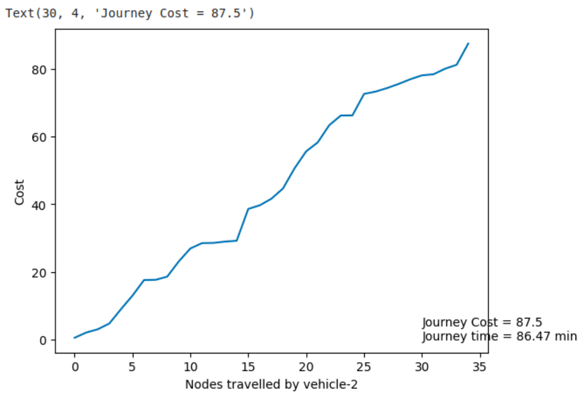

Figure 5.17: vehicle-2 cost in presence of dynamic obstacle

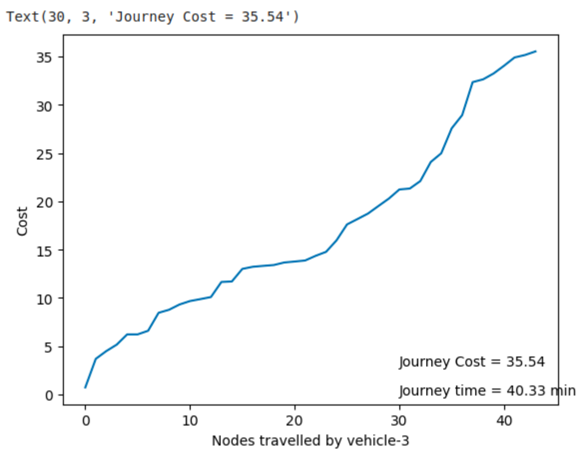

Figure 5.18: vehicle-3 cost in presence of dynamic obstacle

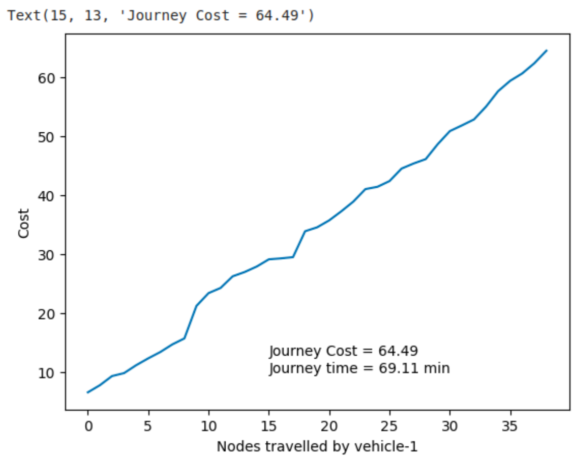

Figure 5.19: vehicle-1 cost in absence of dynamic obstacle

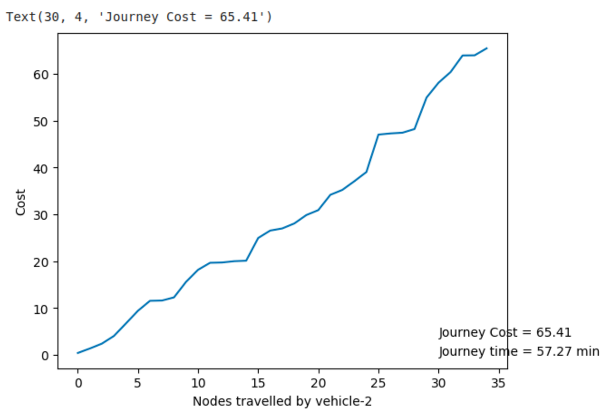

Figure 5.20: vehicle-2 cost in absence of dynamic obstacle

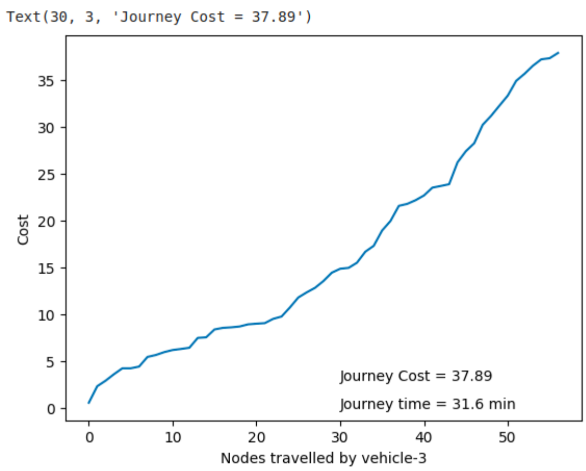

Figure 5.21: vehicle-3 cost in absence of dynamic obstacle

Chapter 6

Contribution and Future Work

6.1 Contribution

We introduced a cooperative strategy with an online algorithm, which modifies its way of traversal with the variation in the graphing environment

- Modified the cost function which penalizes distance of the road and time taken by vehicle to cover the road
- Disintegrated large-scale problem into simple problems using an unsupervised K-means clustering method.
- Build an ML model which is able to predict the time, the vehicle will take to cover a particular road.
- Implemented a distributed network (Robot operating system -ROS) to solve the large-scale problem in dynamic environment with load balancing through cooperation

6.2 Future Work

- we are using the greedy approach of degree one in each cluster to travel which may lead to the immature termination of the algorithm. So, a higher degree of greedy approach can be tested for a better solution.
- The available dataset has very few features, with a better dataset a good accuracy model can be produced which helps to predict travel time with more accuracy.
- In our case we are assuming for two nodes there can be only two paths in between but this problem can further be modified if we have more than two paths.

Chapter 7

Simulations details

```
#!/usr/bin/env python3



import rospy
from std_msgs.msg import Float64MultiArray,MultiArrayDimension,MultiArrayLayout
from sklearn.cluster import KMeans
import networkx as nx
import matplotlib.pyplot as plt
import pandas as pd
import math
import random

dataset= pd.read_csv('/home/ankit/Downloads/LosAngeles/LosAngeles_Edgelist.csv')
co = dataset[['XCoord','YCoord']].drop_duplicates()
coordinates= co.iloc[:135,:].values.tolist()



print(len(coordinates[0]))
# create a graph from the DataFrame
G = nx.Graph()
for i, (x, y) in enumerate(coordinates):
    G.add_node(i, pos=(x, y))

for i in range(len(coordinates)):
    distances = []
    for j in range(len(coordinates)):
        if i != j:
            x1, y1 = coordinates[i]
            x2, y2 = coordinates[j]
            distance = round(math.sqrt((x2 - x1)**2 + (y2 - y1)**2)/1000,2)  # Eucl
            distances.append((j, distance))
    distances.sort(key=lambda x: x[1])
    for j, distance in distances[:135]:
        G.add_edge(i, j, weight=distance)

plt.figure(figsize=(10,10))
pos = nx.get_node_attributes(G, 'pos')


nx.draw_networkx(G, pos, with_labels=True,node_size = 10)
labels = nx.get_edge_attributes(G, 'weight')
nx.draw_networkx_edge_labels(G, pos, edge_labels=labels)
```

```
A_sparse = nx.adjacency_matrix(G,nodelist=sorted(G.nodes())).todense()


A_sparse[70][64],A_sparse[70][59] = A_sparse[70][59],A_sparse[70][64]
A_sparse[37][42],A_sparse[37][39] = A_sparse[37][39],A_sparse[37][42]
A_sparse[63][71],A_sparse[63][61] = A_sparse[63][61],A_sparse[63][71]





rospy.init_node("weight_matrix",anonymous=True)
pub2 = rospy.Publisher('w2c',Float64MultiArray,queue_size=10)
pub4 = rospy.Publisher('w2c2nd',Float64MultiArray,queue_size=10)
pub3 = rospy.Publisher('wtf',Float64MultiArray,queue_size=10)
rate = rospy.Rate(2)
while not rospy.is_shutdown():
    my_msg = Float64MultiArray()
    my_msg.layout = MultiArrayLayout()
    my_msg.layout.dim.append(MultiArrayDimension(label="rows", size=len(coordinates)
    my_msg.layout.dim.append(MultiArrayDimension(label="cols", size=len(coordinates
    array_1d = []
    for row in coordinates:
        array_1d.extend(row)
    my_msg.data = array_1d



    my_msg2 = Float64MultiArray()
    my_msg2.layout = MultiArrayLayout()


    my_msg2.layout.dim.append(MultiArrayDimension(label="rows", size=len(A_sparse))
    my_msg2.layout.dim.append(MultiArrayDimension(label="cols", size=len(A_sparse[0
    array_1d2 = []

    for row in A_sparse:
        array_1d2.extend(row)
    my_msg2.data = array_1d2

    pub2.publish(my_msg)
    pub4.publish(my_msg2)
    pub3.publish(my_msg)
    rate.sleep()
```

```
#!/usr/bin/env python3



import rospy
from std_msgs.msg import Float64MultiArray,MultiArrayDimension,MultiArrayLayout
from sklearn.cluster import KMeans
import networkx as nx
import matplotlib.pyplot as plt
import pandas as pd
import math
import xgboost as xgb
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_squared_error
```

```
from sklearn.preprocessing import OneHotEncoder
from sklearn.preprocessing import MinMaxScaler
from sklearn.compose import ColumnTransformer
from sklearn.preprocessing import OneHotEncoder
import numpy as np
import random
new_dataset = pd.read_csv('full_dataset.csv')

X = new_dataset.iloc[:, :-1].values
Y = new_dataset.iloc[:, -1].values




ct = ColumnTransformer(transformers=[('encoder', OneHotEncoder(sparse= False), [0,3]
X = np.array(ct.fit_transform(X))



# Split the data into training and test sets
X_train, X_test, y_train, y_test = train_test_split(X, Y, test_size=0.3, random_stat


sc = MinMaxScaler()
X_train[:, -2:] = sc.fit_transform(X_train[:, -2:])
X_test[:, -2:] = sc.transform(X_test[:, -2:])

model = xgb.XGBRegressor(objective ='reg:squarederror', learning_rate = 0.5, max_dep
model.fit(X_train, y_train)



y_pred = model.predict(X_test)



def update1(msg):
    x= msg.data
    no =[]
    for i in range(x):
        no.append(random.randint(1,10))

    dt = Float64MultiArray()
    dt.data = no
    pub2.publish(dt)

def update2(msg):
    x= msg.data
    no =[]
    for i in range(x):
        no.append(random.randint(0.01,5))

    dt = Float64MultiArray()
    dt.data = no
    pub3.publish(dt)



def update3(msg):
    x= msg.data
    no =[]
```

```
    for i in range(x):
        no.append(random.randint(1,10))

    dt = Float64MultiArray()
    dt.data = no
    pub4.publish(dt)


rospy.init_node("Time_cost",anonymous=True)
pub2 = rospy.Publisher('t2c1',Float64MultiArray,queue_size=10)
pub4 = rospy.Publisher('t2c3',Float64MultiArray,queue_size=10)
pub3 = rospy.Publisher('t2c2',Float64MultiArray,queue_size=10)

sub1 =rospy.Subscriber('t2bc1', Float64MultiArray, callback=update1)
sub2 =rospy.Subscriber('t2bc2', Float64MultiArray, callback=update2)
sub3 =rospy.Subscriber('t2bc3', Float64MultiArray, callback=update3)
rate = rospy.Rate(2)


while not rospy.is_shutdown( ):
    my_msg = Float64MultiArray()
    my_msg.layout = MultiArrayLayout()
    my_msg.layout.dim.append(MultiArrayDimension(label="rows", size=len(coordinates)
    my_msg.layout.dim.append(MultiArrayDimension(label="cols", size=len(coordinates
    array_1d = []
    for row in coordinates:
        array_1d.extend(row)
    my_msg.data = array_1d



    my_msg2 = Float64MultiArray()
    my_msg2.layout = MultiArrayLayout()


    my_msg2.layout.dim.append(MultiArrayDimension(label="rows", size=len(A_sparse))
    my_msg2.layout.dim.append(MultiArrayDimension(label="cols", size=len(A_sparse[0
    array_1d2 = []
    for row in A_sparse:
        array_1d2.extend(row)
    my_msg2.data = array_1d2

    pub2.publish(my_msg)
    pub4.publish(my_msg2)
    pub3.publish(my_msg)
    rate.sleep()
```

```

#!/usr/bin/env python3

import rospy
from new_msg.msg import batterystatus
from std_msgs.msg import Float64MultiArray,MultiArrayDimension,MultiArrayLayout
import numpy as np
import matplotlib.pyplot as plt
import networkx as nx
import pandas as pd
from sklearn.cluster import KMeans
import math

# getting adhecency matrix
```

```python
Adjecency_matrix = None
def callback_weight(msg):
    global Adjecency_matrix
    dims2 = msg.layout.dim
    shape2 = (dims2[0].size, dims2[1].size)
    Adjecency_matrix = np.array(msg.data).reshape(shape2)
    print(Adjecency_matrix[70][59])




z=0
# getting cooridnates and performing K means clustring
def callback(msg):
    global Adjecency_matrix ,z
    dims = msg.layout.dim
    shape = (dims[0].size, dims[1].size)
    coordinates = np.array(msg.data).reshape(shape)

    if Adjecency_matrix is not None:
          kmeans = KMeans(n_clusters=3,n_init=10,random_state= 0)  # change the numb
          kmeans.fit(coordinates)
          labels = kmeans.labels_

          G2 = nx.from_numpy_array(Adjecency_matrix)

          cluster_0 = [coord for i, coord in enumerate(coordinates) if labels[i] ==
          cluster_0_node = [i for i, coord in enumerate(coordinates) if labels[i] ==
          msg1.layout = MultiArrayLayout()
          msg1.layout.dim.append(MultiArrayDimension(label="rows", size=len(cluster_
          msg1.layout.dim.append(MultiArrayDimension(label="cols", size=len(cluster_
          array_1d = []
          for row in cluster_0:
               array_1d.extend(row)
          msg1.data = array_1d
          pub1.publish(msg1)



          msg1_node.layout = MultiArrayLayout()
          msg1_node.data = cluster_0_node
          pub6.publish(msg1_node)

          subgraph_adjacency = Adjecency_matrix[np.ix_(cluster_0_node, cluster_0_nod
          subgraph = nx.from_numpy_array(subgraph_adjacency)
          node_mapping = {i: node for i, node in enumerate(cluster_0_node)}
          # Set the edge weights of the subgraph from the parent graph
          print(node_mapping)
          for u, v, edge in subgraph.edges(data=True):
               u_parent = node_mapping[u]
               v_parent = node_mapping[v]
               edge['weight'] = Adjecency_matrix[u_parent, v_parent]


          # weighted_adjacency_matrix = nx.to_numpy_array(subgraph)
          weighted_adjacency_matrix = subgraph_adjacency
          print(weighted_adjacency_matrix[37][26])
          msg1_ad.layout = MultiArrayLayout()
          msg1_ad.layout.dim.append(MultiArrayDimension(label="rows", size=len(weig
          msg1_ad.layout.dim.append(MultiArrayDimension(label="cols", size=len(weig
          array_ad = []
```

```
        for row in weighted_adjacency_matrix:
            array_ad.extend(row)
        msg1_ad.data = array_ad
        # rospy.loginfo(msg1_ad.data)
        pub5.publish(msg1_ad)



        cluster_1 = [coord for i, coord in enumerate(coordinates) if labels[i] ==
        cluster_1_node = [i for i, coord in enumerate(coordinates) if labels[i] ==
        msg2.layout = MultiArrayLayout()
        msg2.layout.dim.append(MultiArrayDimension(label="rows", size=len(cluster_
        msg2.layout.dim.append(MultiArrayDimension(label="cols", size=len(cluster_
        array_1d2 = []
        for row in cluster_1:
            array_1d2.extend(row)
        msg2.data = array_1d2
        pub2.publish(msg2)



        msg2_node.layout = MultiArrayLayout()
        msg2_node.data = cluster_1_node
        pub7.publish(msg2_node)

        subgraph_adjacency2 = Adjecency_matrix[np.ix_(cluster_1_node, cluster_1_no
        subgraph2 = nx.from_numpy_array(subgraph_adjacency2)
        node_mapping2 = {i: node for i, node in enumerate(cluster_1_node)}
        # Set the edge weights of the subgraph from the parent graph
        for u, v, edge in subgraph2.edges(data=True):
            u_parent = node_mapping2[u]
            v_parent = node_mapping2[v]
            edge['weight'] = Adjecency_matrix[u_parent, v_parent]


        weighted_adjacency_matrix2 = subgraph_adjacency2

        msg2_ad.layout = MultiArrayLayout()
        msg2_ad.layout.dim.append(MultiArrayDimension(label="rows", size=len(weigh
        msg2_ad.layout.dim.append(MultiArrayDimension(label="cols", size=len(weigh
        array_ad2 = []
        for row in weighted_adjacency_matrix2:
            array_ad2.extend(row)
        msg2_ad.data = array_ad2
        # rospy.loginfo(msg1_ad.data)
        pub8.publish(msg2_ad)











        cluster_2 = [coord for i, coord in enumerate(coordinates) if labels[i] ==
        cluster_2_node = [i for i, coord in enumerate(coordinates) if labels[i] ==
        print(len(cluster_2_node))
        print(len(cluster_1_node))
        print(len(cluster_0_node))
```

```
        msg3.layout = MultiArrayLayout()
        msg3.layout.dim.append(MultiArrayDimension(label="rows", size=len(cluster_
        msg3.layout.dim.append(MultiArrayDimension(label="cols", size=len(cluster_
        array_1d3 = []
        for row in cluster_2:
            array_1d3.extend(row)
        msg3.data = array_1d3
        pub3.publish(msg3)



        msg3_node.layout = MultiArrayLayout()
        msg3_node.data = cluster_2_node
        pub9.publish(msg3_node)

        subgraph_adjacency3 = Adjecency_matrix[np.ix_(cluster_2_node, cluster_2_no
        subgraph3 = nx.from_numpy_array(subgraph_adjacency3)
        node_mapping3 = {i: node for i, node in enumerate(cluster_2_node)}
        # Set the edge weights of the subgraph from the parent graph
        for u, v, edge in subgraph2.edges(data=True):
            u_parent = node_mapping3[u]
            v_parent = node_mapping3[v]
            edge['weight'] = Adjecency_matrix[u_parent, v_parent]


        weighted_adjacency_matrix3 = subgraph_adjacency3
        # print(weighted_adjacency_matrix)
        msg3_ad.layout = MultiArrayLayout()
        msg3_ad.layout.dim.append(MultiArrayDimension(label="rows", size=len(weigh
        msg3_ad.layout.dim.append(MultiArrayDimension(label="cols", size=len(weigh
        array_ad3 = []
        for row in weighted_adjacency_matrix3:
            array_ad3.extend(row)
        msg3_ad.data = array_ad3
        # rospy.loginfo(msg1_ad.data)
        pub10.publish(msg3_ad)


        com_mat_no =[]
        com_mat_coo=[]
        com_mat_no.extend(cluster_0_node)
        com_mat_no.extend(cluster_2_node)

        msg4_node.layout = MultiArrayLayout()
        msg4_node.data = com_mat_no
        pub12.publish(msg4_node)


        com_mat_coo.extend(cluster_0)
        com_mat_coo.extend(cluster_2)
        pos = {i: cord for i, cord in enumerate(com_mat_coo)}
        subgraph_adjacency4 = Adjecency_matrix[np.ix_(com_mat_no, com_mat_no)]
        subgraph4 = nx.from_numpy_array(subgraph_adjacency4)
        node_mapping4 = {i: node for i, node in enumerate(com_mat_no)}
        # Set the edge weights of the subgraph from the parent graph
        for u, v, edge in subgraph4.edges(data=True):
            u_parent = node_mapping4[u]
            v_parent = node_mapping4[v]
            edge['weight'] = Adjecency_matrix[u_parent, v_parent]


```

```
            weighted_adjacency_matrix4 = subgraph_adjacency4
            # print(weighted_adjacency_matrix)
            msg4_ad.layout = MultiArrayLayout()
            msg4_ad.layout.dim.append(MultiArrayDimension(label="rows", size=len(weig
            msg4_ad.layout.dim.append(MultiArrayDimension(label="cols", size=len(weig
            array_ad4 = []
            for row in weighted_adjacency_matrix4:
                 array_ad4.extend(row)
            msg4_ad.data = array_ad4
            # rospy.loginfo(msg1_ad.data)
            pub11.publish(msg4_ad)
            # if z==0:
            #       z=1
            #       G3 = nx.from_numpy_array(weighted_adjacency_matrix4)
            #       nx.draw_networkx(G3, pos, with_labels=False,node_size = 10)
            #       plt.title('2nd TSP Graph')
            #       plt.show()




rospy.init_node('k_cluster', anonymous=True)
sub =rospy.Subscriber('w2c',Float64MultiArray, callback)
sub2 =rospy.Subscriber('w2c2nd',Float64MultiArray, callback_weight)


pub1 = rospy.Publisher('c2c1', Float64MultiArray, queue_size=10)
pub5 = rospy.Publisher('c2c12nd', Float64MultiArray, queue_size=10)
pub6 = rospy.Publisher('ctc13rd',Float64MultiArray, queue_size=10)
pub11 = rospy.Publisher('c2c14th', Float64MultiArray, queue_size=10)
pub12 = rospy.Publisher('c2c15th', Float64MultiArray, queue_size=10)



pub2 = rospy.Publisher('c2c2', Float64MultiArray, queue_size=10)
pub7 = rospy.Publisher('c2c22nd', Float64MultiArray, queue_size=10)
pub8 = rospy.Publisher('c2c23rd', Float64MultiArray, queue_size=10)


pub3 = rospy.Publisher('c2c3', Float64MultiArray, queue_size=10)
pub9 = rospy.Publisher('c2c32nd', Float64MultiArray, queue_size=10)
pub10 = rospy.Publisher('c2c33rd', Float64MultiArray, queue_size=10)


pub4 = rospy.Publisher('c2c4', Float64MultiArray, queue_size=10)

rate = rospy.Rate(2) # 10hz
msg1= Float64MultiArray()
msg1_node= Float64MultiArray()
msg1_ad = Float64MultiArray()


msg2 =Float64MultiArray()
msg2_node= Float64MultiArray()
msg2_ad = Float64MultiArray()


msg3= Float64MultiArray()
msg3_node= Float64MultiArray()
msg3_ad = Float64MultiArray()

```

```python

msg4_ad =Float64MultiArray()
msg4_node = Float64MultiArray()
rospy.spin()
```

```python
#!/usr/bin/env python3

import rospy
from new_msg.msg import batterystatus,costmatrix
from std_msgs.msg import Float64MultiArray
import numpy as np
import matplotlib.pyplot as plt
import networkx as nx
import pandas as pd
from sklearn.cluster import KMeans
import math
import heapq
from matplotlib.animation import FuncAnimation
import random

Flag = False
shere =None
trav=[]
coordinates =[]
delay = rospy.Duration(1.0)

coordinates =[]
def k_cluster(msg):
    global coordinates
    dims = msg.layout.dim
    shape = (dims[0].size, dims[1].size)
    coordinates = np.array(msg.data).reshape(shape)



Adj_mat = []
def Adjecency(msg):
    global Adj_mat
    dims = msg.layout.dim
    shape = (dims[0].size, dims[1].size)
    Adj_mat = np.array(msg.data).reshape(shape)


node_nos = []
def node_no(msg):
    global node_nos
    node_nos = np.array(msg.data)
    # print(node_nos)
    node_map = {i: node_nos for i, cod in enumerate(node_nos)}
    # print(node_map[0])


com = []
def combined(msg):
    global com
    dims = msg.layout.dim
    shape = (dims[0].size, dims[1].size)
    com = np.array(msg.data).reshape(shape)
    # print(len(com))

com_node =[]
```

```
def comnode(msg):
    global com_node
    com_node = np.array(msg.data)
    # print(com_node)


visitfrom3rd = []
newa=True
def cluster_3(msg):
    global visitfrom3rd,newa
    arr= list(msg.data)
    if newa:

      visitfrom3rd=arr
      newa=False
    else:
        visitfrom3rd.append(arr[-1])

time_pen =None
def tcost(msg):
    global time_pen
    time_pen  = msg.percentage


dmy = 0

def perform(cod ,ad_mt,nono,delay):
    global dmy,trav,time_pen
    if len(cod) != 0 and len(ad_mt) != 0  and len(nono) != 0:

                coords = {nono[i]: tuple(cod) for i, cod in enumerate(cod)}
                if dmy ==0:
                    dmy =1
                    # nx.draw_networkx(sub1,coords, with_labels=False,node_size = 1
                    # plt.show()

                    def greedy_traversal(matrix, start_node,delay):
                        global trav
                        N = len(matrix)  # number of nodes
                        trav =[int(nono[start_node])]
                        visited = [start_node]
                        vis = set()
                        vis.add(start_node)
                        current_vertex = start_node
                        queue = []
                        print(matrix[7][8])
                        print(matrix[7][34])
                        ch_wt =0
                        while len(visited)< N:
                            ad_vt = np.where(matrix[current_vertex] > 0)[0]
                            traffic =[]
                            road_quality =[]
                            unvis = [v for v in ad_vt if v not in vis]
                            for i in range(len(unvis)):
                                traffic.append(random.uniform(0.01,2))

                            for i in range(len(unvis)):
                                road_quality.append(random.uniform(0.01,5))

                            if len(unvis) ==0:
                                break
```

```
                    nx_vx = unvis[0]
                    lw_val = matrix[current_vertex][0]
                    for i in  range(len(unvis)):
                        new_val = matrix[current_vertex][unvis[i]] + (0.25*m
                        if new_val < lw_val:
                            lw_val = new_val
                            nx_vx = unvis[i]
                    # nx_vx = min(unvis , key = lambda v : (matrix[current_v
                    ch_wt +=lw_val

                    visited.append(nx_vx)
                    vis.add(nx_vx)
                    trav.append(int(nono[nx_vx]))
                    # rospy.sleep(delay)
                    print(ch_wt)
                    print(trav)
                    current_vertex = nx_vx
                return trav


        my_no = np.where(node_nos == 65)[0][0]

        traversal = greedy_traversal(ad_mt, my_no,delay)
        def animate_traversal(graph, vehicle1_path):
                fig, ax = plt.subplots()
                edge_color = ['green','blue']
                edge_label = ['vehicle-1','vehicle-2']

                # Create a custom legend
                legend_element = [plt.Line2D([0], [0], color=color, labe
                plt.title('Cluster-1')
                def update(frame):
                    ax.clear()
                    nx.draw_networkx(graph, pos, with_labels=True, node_

                    # Draw vehicle 1's path

                    vehicle1_subpath = vehicle1_path[:frame+1]
                    nx.draw_networkx_edges(graph, pos, edgelist=[(u, v)
                    nx.draw_networkx_nodes(graph, pos, nodelist=[vehicle
                    plt.legend(handles=legend_element)
                #     if frame >= len(vehicle1_path):

                #      fram = frame -len(vehicle1_path)
                #      vehicle2_subpath = vehicle2_path[:fram+1]
                #      nx.draw_networkx_edges(graph, pos, edgelist=[(u,
                #      nx.draw_networkx_nodes(graph, pos, nodelist=[vehi

                #     # Terminate the animation

                #     if frame == len(vehicle1_path) +len(vehicle2_path)
                #         ani.event_source.stop()

                # ani = FuncAnimation(fig, update, frames=(len(vehicle1_

                # plt.show()

                # Print traversed paths
                # print("Vehicle 1's path:", vehicle1_path)
                    if frame == len(vehicle1_path) - 1:
                          ani.event_source.stop()
```

```
                            ani = FuncAnimation(fig, update, frames=len(vehicle1_pat

                            plt.show()

                    G = nx.Graph()
                    for node_id, coords in zip(nono, cod):
                        G.add_node(node_id, pos=coords)

                    pos = nx.get_node_attributes(G, 'pos')

                    # animate_traversal(G, path1,path2)

                    animate_traversal(G, traversal)


    else:
         rospy.loginfo("received none")


co_trav =[]
co_visited = [shere]
print(type(co_visited))
myval=0
def newfun(myarr,pub6,x):
    global visitfrom3rd,trav,co_trav,myval,co_visited
    # print(myarr)
    if visitfrom3rd:
        if len(node_nos) !=0:
                if Flag ==True:
                    N = len(com)  # number of nodes
                    queue = []
                    # print(visited)
                    for neighbor, weight in enumerate(com[myarr]):
                        if weight and neighbor != myarr:
                            heapq.heappush(queue, (weight, neighbor))
                    # print(queue)
                    while queue:
                        # print(visitfrom3rd[0])
                        weight, node = heapq.heappop(queue)
                        if com_node[node] not in co_visited:
                          if com_node[node] not in trav:
                           if com_node[node] not in visitfrom3rd:
                            #  print(visitfrom3rd[0])
                             co_visited.append(com_node[node])
                             x.data= co_visited[1:]
                             pub6.publish(x)
                             print(co_visited[1:])
                             rospy.sleep(delay)
                             for neighbor, weight in enumerate(com[node]):
                                if weight and neighbor not in co_visited:
                                    heapq.heappush(queue, (weight, neighbor))
```

```
def nodo():

    rospy.init_node('cluster_1', anonymous=True)


    rospy.Subscriber('c2c1', Float64MultiArray, callback=k_cluster)
    rospy.Subscriber('c2c12nd', Float64MultiArray, callback= Adjecency)
    rospy.Subscriber('ctc13rd',Float64MultiArray, callback = node_no)
    # rospy.Subscriber('c1tbc2', costmatrix, callback=cluster_2)
    rospy.Subscriber('c2c14th', Float64MultiArray, callback=combined)
    rospy.Subscriber('c2c15th', Float64MultiArray, callback=comnode)

    rospy.Subscriber('c1tbc3', costmatrix, callback=cluster_3)

    # rospy.Subscriber('t2c1', batterystatus, callback=tcost)
    # print(box2)
    # rospy.Subscriber('c1tbc4', Float64MultiArray, callback=cluster_4)
    pub6 = rospy.Publisher('c1tbc32nd', costmatrix,queue_size= 10)
    pub2 = rospy.Publisher('c1tc2', batterystatus, queue_size=10)
    pub3 = rospy.Publisher('c1tc3', Float64MultiArray, queue_size=10)
    pub4 = rospy.Publisher('c1tc4', Float64MultiArray, queue_size=10)
    pub5 = rospy.Publisher('c1tfinal', Float64MultiArray, queue_size=10)

    # pub7 = rospy.Publisher('t2bc1', batterystatus, queue_size=10)


    delay = rospy.Duration(1.0)


    rate = rospy.Rate(1) # 10hz

    while not rospy.is_shutdown():
        x= costmatrix()
        perform(coordinates,Adj_mat,node_nos,delay)
        newfun(shere,pub6,x)

        rate.sleep()
    rospy.spin()

if __name__ == '__main__':
    try:

        nodo()
    except rospy.ROSInterruptException:
        pass
```

```
#!/usr/bin/env python3

import rospy
from new_msg.msg import batterystatus,costmatrix
from std_msgs.msg import Float64MultiArray,MultiArrayDimension,MultiArrayLayout
import numpy as np
import random
import matplotlib.pyplot as plt
import networkx as nx
import pandas as pd
from sklearn.cluster import KMeans
import math
```

```
import heapq
from matplotlib.animation import FuncAnimation

Flag = False
shere =None
visited=[]
coordinates =[]
delay = rospy.Duration(1.0)
def k_cluster(msg):
    global coordinates
    dims = msg.layout.dim
    shape = (dims[0].size, dims[1].size)
    coordinates = np.array(msg.data).reshape(shape)



Adj_mat = []
a= random.randint(5,15)
print(a)
def Adjecency(msg):
    global Adj_mat,a
    dims = msg.layout.dim
    shape = (dims[0].size, dims[1].size)
    Adj_mt = np.array(msg.data).reshape(shape)
    # x= Adj_mt[8][9]
    # for i in range(len(Adj_mt)):
    #     for j in range(len(Adj_mt)):
    #         if i==a:
    #             Adj_mt[i][j]=0

    Adj_mat = Adj_mt


com = []
def combined(msg):
    global com
    dims = msg.layout.dim
    shape = (dims[0].size, dims[1].size)
    com = np.array(msg.data).reshape(shape)
    # print(len(com))

com_node =[]
def comnode(msg):
    global com_node
    com_node = np.array(msg.data)
    # print(com_node)


node_nos = []
def node_no(msg):
    global node_nos
    node_nos = np.array(msg.data)
    node_map = {i: node_nos for i, cod in enumerate(node_nos)}
    # print(node_map[0])


visitfrom3rd = []
newa=True
def cluster_3(msg):
    global visitfrom3rd,newa
    arr= list(msg.data)
```

```
    if newa:

      visitfrom3rd=arr
      newa=False
    else:
        visitfrom3rd.append(arr[-1])





dmy = 0

def perform(cod ,ad_mt,nono,delay):
    global dmy,visited
    if len(cod) != 0 and len(ad_mt) != 0  and len(nono) != 0:

                coords = {nono[i]: tuple(cod) for i, cod in enumerate(cod)}
                if dmy ==0:
                    dmy =1
                    # nx.draw_networkx(sub1,coords, with_labels=False,node_size = 10
                    # plt.show()

                    def greedy_traversal(matrix, start_node,delay):
                        # global visited
                        # N = len(matrix)  # number of nodes
                        # trav =[]
                        # visited = [start_node]
                        # queue = []

                        # for neighbor, weight in enumerate(matrix[start_node]):
                        #     if weight and neighbor != start_node:
                        #         heapq.heappush(queue, (weight, neighbor))

                        # while queue:
                        #     weight, node = heapq.heappop(queue)
                        #     if node not in visited:
                        #         visited.append(node)
                        #         trav.append(int(nono[node]))
                        #         rospy.sleep(delay)
                        #         print(trav)
                        #         for neighbor, weight in enumerate(matrix[node]):
                        #           if not np.any(matrix[node]):
                        #             queue.clear()

                        #             if weight and neighbor not in visited:
                        #                 heapq.heappush(queue, (weight, neighbor))
                        global trav
                        N = len(matrix)  # number of nodes
                        trav =[int(nono[start_node])]
                        visited = [start_node]
                        vis = set()
                        vis.add(start_node)
                        current_vertex = start_node
                        queue = []

                        ch_wt =0
                        while len(visited)< N:
                            ad_vt = np.where(matrix[current_vertex] > 0)[0]
                            traffic =[]
                            road_quality =[]
```

```
                        unvis = [v for v in ad_vt if v not in vis]
                        for i in range(len(unvis)):
                            traffic.append(random.uniform(0.01,2))

                        for i in range(len(unvis)):
                            road_quality.append(random.uniform(0.01,5))

                        if len(unvis) ==0:
                            break
                        nx_vx = unvis[0]
                        lw_val = matrix[current_vertex][0]
                        for i in  range(len(unvis)):
                            new_val = matrix[current_vertex][unvis[i]] + (0.25*r
                            if new_val < lw_val:
                                lw_val = new_val
                                nx_vx = unvis[i]
                        # nx_vx = min(unvis , key = lambda v : (matrix[current_v
                        ch_wt +=lw_val

                        visited.append(nx_vx)
                        vis.add(nx_vx)
                        trav.append(int(nono[nx_vx]))
                        # rospy.sleep(delay)
                        print(ch_wt)
                        print(trav)
                        current_vertex = nx_vx


                    return trav


                # Call the function
                traversal = greedy_traversal(ad_mt, 0,delay)
                # a=26
                # pathmod = path1[:path1.index(a)+1]
                def animate_traversal(graph, vehicle1_path):
                        fig, ax = plt.subplots()
                        edge_color = ['blue','green']
                        edge_label = ['vehicle-2','vehicle-1']

                        # Create a custom legend
                        legend_element = [plt.Line2D([0], [0], color=color, labe

                        def update(frame):
                            ax.clear()
                            nx.draw_networkx(graph, pos, with_labels=True, node_

                            # Draw vehicle 1's path
                            vehicle1_subpath = vehicle1_path[:frame+1]
                            nx.draw_networkx_edges(graph, pos, edgelist=[(u, v)
                            nx.draw_networkx_nodes(graph, pos, nodelist=[vehicle
                            plt.legend(handles=legend_element)

                        #     # Draw vehicle 2's path
                        #     if frame >= len(vehicle1_path):

                        #      fram = frame -len(vehicle1_path)
                        #      vehicle2_subpath = vehicle2_path[:fram+1]
                        #      nx.draw_networkx_edges(graph, pos, edgelist=[(u,
                        #      nx.draw_networkx_nodes(graph, pos, nodelist=[vehi
                        #     plt.legend(handles=legend_element)
```

```
                            #     # Terminate the animation

                            #     if frame == len(vehicle1_path) +len(vehicle2_path)
                            #         ani.event_source.stop()

                            # ani = FuncAnimation(fig, update, frames=(len(vehicle1_
                            # plt.show()

                                if frame == len(vehicle1_path) - 1:
                                      ani.event_source.stop()
                            ani = FuncAnimation(fig, update, frames=len(vehicle1_pat

                            plt.show()

                            # Print traversed paths
                            # print("Vehicle 1's path:", vehicle1_path)

                    G = nx.Graph()

                    for node_id, coords in zip(nono, cod):
                        G.add_node(node_id, pos=coords)

                    pos = nx.get_node_attributes(G, 'pos')

                    animate_traversal(G, traversal)


    else:
         rospy.loginfo("received none")





co_trav =[]
co_visited = [shere]
myval=0
def newfun(myarr):
    global visitfrom3rd,visited,co_trav,myval,co_visited
    print(myarr)
    if visitfrom3rd:
        if len(node_nos) !=0:
                if Flag ==True:
                    N = len(com)  # number of nodes


                    co_trav =[com_node[int(myarr)]]


                    queue = []

                    for neighbor, weight in enumerate(com[myarr]):
                        if weight and neighbor != myarr:
                            heapq.heappush(queue, (weight, neighbor))
                    print(queue)
                    while queue:
                        # print(visitfrom3rd[0])
                        weight, node = heapq.heappop(queue)
                        if node not in co_visited:
                          if node not in visited:
                           if com_node[node] not in visitfrom3rd:
```

```
                                #   print(visitfrom3rd[0])
                                 co_visited.append(node)

                                 co_trav.append(com_node[node])
                                 print(co_trav)
                                 rospy.sleep(delay)
                                 for neighbor, weight in enumerate(com[node]):
                                    if weight and neighbor not in co_visited:
                                        heapq.heappush(queue, (weight, neighbor))















def nodo():
    rospy.init_node('cluster_2', anonymous=True)


    rospy.Subscriber('c2c2', Float64MultiArray, callback=k_cluster)
    rospy.Subscriber('c2c22nd', Float64MultiArray, callback=node_no)
    rospy.Subscriber('c2c23rd', Float64MultiArray, callback=Adjecency)


    rospy.Subscriber('c2c24th', Float64MultiArray, callback=combined)
    rospy.Subscriber('c2c25th', Float64MultiArray, callback=comnode)

    rospy.Subscriber('c2tbc3', costmatrix, callback=cluster_3)

    # rospy.Subscriber('c2tbc4', Float64MultiArray, callback=cluster_4)

    pub2 = rospy.Publisher('c1tbc2',costmatrix, queue_size=10)
    pub3 = rospy.Publisher('c2tc3', Float64MultiArray, queue_size=10)
    pub4 = rospy.Publisher('c2tc4', Float64MultiArray, queue_size=10)
    pub5 = rospy.Publisher('c2tfinal', Float64MultiArray, queue_size=10)

    rate = rospy.Rate(1.0) # 10hz

    while not rospy.is_shutdown():

        x= costmatrix()

        perform(coordinates,Adj_mat,node_nos,delay)

        # newfun(shere)

        rate.sleep()
    rospy.spin()

if __name__ == '__main__':
    try:
        nodo()
```

```
    except rospy.ROSInterruptException:
        pass
```

```
#!/usr/bin/env python3

import rospy
from new_msg.msg import batterystatus,costmatrix
from std_msgs.msg import Float64MultiArray
import numpy as np
import matplotlib.pyplot as plt
import networkx as nx
import pandas as pd
from sklearn.cluster import KMeans
import math
import heapq
from matplotlib.animation import FuncAnimation
import random


coordinates =[]
def k_cluster(msg):
    global coordinates
    dims = msg.layout.dim
    shape = (dims[0].size, dims[1].size)
    coordinates = np.array(msg.data).reshape(shape)



Adj_mat = []
def Adjecency(msg):
    global Adj_mat
    dims = msg.layout.dim
    shape = (dims[0].size, dims[1].size)
    Adj_mat = np.array(msg.data).reshape(shape)


node_nos = []
def node_no(msg):
    global node_nos
    node_nos = np.array(msg.data)
    # node_map = {i: node_nos for i, cod in enumerate(node_nos)}
    # print(node_map[0])

cvisit = []
def co_visitation(msg):
    global cvisit
    cvisit = msg.data




dmy = 0
def perform(cod ,ad_mt,nono,pub3,x,delay):
    global dmy,cvisit
    if len(cod) != 0 and len(ad_mt) != 0  and len(nono) != 0 :

                coords = {nono[i]: tuple(cod) for i, cod in enumerate(cod)}
                if dmy ==0:
                    dmy =1

                    def greedy_traversal(matrix, start_node,pub3,x,delay):
```

```
# N = len(matrix)  # number of nodes
# print(N)
# trav =[]
# trav.append(int(nono[start_node]))
# visited = [start_node]
# queue = []

# for neighbor, weight in enumerate(matrix[start_node]):
#     if weight and neighbor != start_node:
#         heapq.heappush(queue, (weight, neighbor))

# while queue:
#     weight, node = heapq.heappop(queue)
#     if node not in visited:
#       if nono[node] not in cvisit:
#         visited.append(node)
#         trav.append(int(nono[node]))
#         # print(trav)
#         x.data = trav

#         pub3.publish(x)
#         print(x)
#         # rospy.sleep(delay)

#         for neighbor, weight in enumerate(matrix[node]):
#             if weight and neighbor not in visited:
#                 heapq.heappush(queue, (weight, neighbor))
global trav
N = len(matrix)  # number of nodes
trav =[int(nono[start_node])]
visited = [start_node]
vis = set()
vis.add(start_node)
current_vertex = start_node
queue = []

ch_wt =0
while len(visited)< N:
    ad_vt = np.where(matrix[current_vertex] > 0)[0]
    traffic =[]
    road_quality =[]
    unvis = [v for v in ad_vt if v not in vis]
    for i in range(len(unvis)):
        traffic.append(random.uniform(0.01,2))

    for i in range(len(unvis)):
        road_quality.append(random.uniform(0.01,5))

    if len(unvis) ==0:
        break
    nx_vx = unvis[0]
    lw_val = matrix[current_vertex][0]
    for i in  range(len(unvis)):
        new_val = matrix[current_vertex][unvis[i]] + (0.25*
        if new_val < lw_val:
            lw_val = new_val
            nx_vx = unvis[i]
    # nx_vx = min(unvis , key = lambda v : (matrix[current_
    ch_wt +=lw_val

    visited.append(nx_vx)
```

```
                vis.add(nx_vx)
                trav.append(int(nono[nx_vx]))
                # rospy.sleep(delay)
                print(ch_wt)
                print(trav)
                current_vertex = nx_vx


            return trav
        # Call the function
        traversal = greedy_traversal(ad_mt, 0,pub3,x,delay)
        print(traversal)
        def animate_traversal(graph, vehicle1_path):
                fig, ax = plt.subplots()
                edge_color = ['blue','red']
                edge_label = ['vehicle-2','vehicle-3']
                # Create a custom legend
                legend_element = [plt.Line2D([0], [0], color=color, labe

                def update(frame):
                    ax.clear()
                    nx.draw_networkx(graph, pos, with_labels=True, node_

                    # Draw vehicle 1's path
                    vehicle1_subpath = vehicle1_path[:frame+1]
                    nx.draw_networkx_edges(graph, pos, edgelist=[(u, v)
                    nx.draw_networkx_nodes(graph, pos, nodelist=[vehicle
                    plt.legend(handles=legend_element)
                    # # Draw vehicle 2's path

                #      if frame >= (max(len(vehicle1_path), len(vehicle2_
                #       x= max(len(vehicle1_path), len(vehicle2_path)) -
                #       fram = frame -x
                #       vehicle2_subpath = vehicle2_path[:fram+1]
                #       nx.draw_networkx_edges(graph, pos, edgelist=[(u,
                #       nx.draw_networkx_nodes(graph, pos, nodelist=[vehi
                #      # plt.legend(handles=[legend_element])
                #      # Terminate the animation

                #      if frame == max(len(vehicle1_path), len(vehicle2_p
                #          ani.event_source.stop()

                # ani = FuncAnimation(fig, update, frames=max(len(vehicl
                # plt.show()

                # Print traversed paths
                # print("Vehicle 1's path:", vehicle1_path)
                    if frame == len(vehicle1_path) - 1:
                          ani.event_source.stop()
                ani = FuncAnimation(fig, update, frames=len(vehicle1_pat

                plt.show()
        G = nx.Graph()
        for node_id, coords in zip(nono, cod):
            G.add_node(node_id, pos=coords)

        pos = nx.get_node_attributes(G, 'pos')

        animate_traversal(G, traversal)


```

```
    else:
         rospy.loginfo("received none")



def nodo():
    rospy.init_node('cluster_3', anonymous=True)


    rospy.Subscriber('c2c3', Float64MultiArray, callback=k_cluster)
    rospy.Subscriber('c2c32nd', Float64MultiArray, callback=node_no)
    rospy.Subscriber('c2c33rd', Float64MultiArray, callback=Adjecency)
    rospy.Subscriber('c1tbc32nd', costmatrix,callback=co_visitation)
    # rospy.Subscriber('c1tc3', Float64MultiArray, callback=cluster_2)
    # rospy.Subscriber('c2tc3', Float64MultiArray, callback=cluster_3)
    # rospy.Subscriber('c3tbc4', Float64MultiArray, callback=cluster_4)
    # pub2 = rospy.Publisher('c1tbc3', batterystatus, queue_size=10)
    pub3 = rospy.Publisher('c1tbc3', costmatrix, queue_size=100)
    # pub4 = rospy.Publisher('c3tc4', Float64MultiArray, queue_size=10)
    # pub5 = rospy.Publisher('c3tfinal', Float64MultiArray, queue_size=10)
    delay = rospy.Duration(1.0)



    rate = rospy.Rate(1) # 10hz


    while not rospy.is_shutdown():



        x= costmatrix()
        perform(coordinates,Adj_mat,node_nos,pub3,x,delay)

        rate.sleep()
    rospy.spin()

if __name__ == '__main__':
    try:
        nodo()
    except rospy.ROSInterruptException:
        pass
```

```
import rospy
from new_msg.msg import batterystatus,costmatrix
from std_msgs.msg import Float64MultiArray
import numpy as np
import matplotlib.pyplot as plt
import networkx as nx
import pandas as pd
from sklearn.cluster import KMeans
import math
import heapq
from matplotlib.animation import FuncAnimation
coordinates = []
def callback_data1(msg):
    global coordinates
    dims = msg.layout.dim
    shape = (dims[0].size, dims[1].size)
    coordinates = np.array(msg.data).reshape(shape)
def callback_data2(data):
```

```
    rospy.loginfo(data)
def callback_data3(data):
    rospy.loginfo(data)
def callback_data4(data):
    rospy.loginfo(data)
def callback_data5(data):
    rospy.loginfo(data)

def fullgraph(cordn):
    print(cordn)
    graph = nx.Graph()
    for i, (x, y) in enumerate(cordn):
       graph.add_node(i, pos=(x, y))
    plt.figure(figsize=(10,10))
    pos = nx.get_node_attributes(graph, 'pos')
    nx.draw_networkx(graph, pos, with_labels=False,node_size = 10)


    fig, ax = plt.subplots()
    edge_color = ['blue','green','red']
    edge_label = ['vehicle-2','vehicle-1','vehicle-3']

    # Create a custom legend
    legend_element = [plt.Line2D([0], [0], color=color, label=label, linewidth=2) fo

    def update(frame):
        ax.clear()
        nx.draw_networkx(graph, pos, with_labels=True, node_color='lightyellow', noo

        # Draw vehicle 1's path

        vehicle1_subpath = vehicle1_path[:frame+1]
        nx.draw_networkx_edges(graph, pos, edgelist=[(u, v) for u, v in zip(vehicle1
        nx.draw_networkx_nodes(graph, pos, nodelist=[vehicle1_subpath[-1]], node_col
        plt.legend(handles=legend_element)



        vehicle2_subpath = vehicle2_path[:frame+1]
        nx.draw_networkx_edges(graph, pos, edgelist=[(u, v) for u, v in zip(vehicle2
        nx.draw_networkx_nodes(graph, pos, nodelist=[vehicle2_subpath[-1]], node_col


        vehicle3_subpath = vehicle3_path[:frame+1]
        nx.draw_networkx_edges(graph, pos, edgelist=[(u, v) for u, v in zip(vehicle3
        nx.draw_networkx_nodes(graph, pos, nodelist=[vehicle3_subpath[-1]], node_col
        # Terminate the animation

        if frame == max(len(vehicle1_path) ,len(vehicle2_path),len(vehicle3_path)) -
            ani.event_source.stop()

    ani = FuncAnimation(fig, update, frames=max(len(vehicle1_path) ,len(vehicle2_pat
    plt.show()





x=0
def listner():
    global x
```

```
    rospy.init_node("final_output",anonymous=True)
    rospy.Subscriber('wtf',Float64MultiArray,callback=callback_data1)
    rospy.Subscriber('c1tfinal',Float64MultiArray,callback=callback_data2)
    rospy.Subscriber('c2tfinal',Float64MultiArray,callback=callback_data3)
    rospy.Subscriber('c3tfinal',Float64MultiArray,callback=callback_data4)
    rospy.Subscriber('c4tfinal',Float64MultiArray,callback=callback_data5)
    rate = rospy.Rate(1.0) # 10hz

    while not rospy.is_shutdown():
        if len(coordinates) != 0:
          if x ==0:
              x=1
              fullgraph(coordinates)

        # newfun(shere)

        rate.sleep()
    rospy.spin()

if __name__ == '__main__':
    try:
        listner()
    except rospy.ROSInterruptException:
        pass
```

References

[1] Konstantakopoulos, Grigorios D., Sotiris P. Gayialis, and Evripidis P. Kechagias. Vehicle routing problem and related algorithms for logistics distribution: A literature review and classi

cation." Operational Research (2020): 1-30.

[2] Sundar, Kaarthik, and Sivakumar Rathinam. Algorithms for heterogeneous, multiple depot, multiple unmanned vehicle path planning problems." Journal of Intelligent Robotic Systems 88.2-4 (2017): 513-526.

[3] Pillac, Victor, et al. review of dynamic vehicle routing problems." European Journal of Operational Research 225.1 (2013): 1-11.

[4] Omara, Fatma A., and Mona M. Arafa. algorithms for task scheduling problem." Foundations of Computational Intelligence Volume 3. Springer, Berlin, Heidelberg, 2009. 479-507.

[5] Gerhard Reinelt, - A Traveling Salesman Problem Library", ORSA Journal on Computing, Volume 3, Number 4, Fall 1991, pages 376-384.

[6] Dantzig, George B., and John H. Ramser. truck dispatching problem." Management science 6.1 (1959): 80-91.

[7] TSP Dataset,https://people.sc.fsu.edu/jburkardt/datasets/cities/cities.html

[8] IMF eLibrary, Road quality, and speed score https://doi.org/10.5089/9798400210440.001

[9] Dataset of 135 nodes @misc title= url=https://tianchi.aliyun.com/dataset/dataDetail?dataId=1079 author=Tianchi, year=2018

[10] Aaron Martinez ,Enrique Fernández.Learning ROS for Robotics Programming.Published by Packt Publishing Ltd. Livery Place 35 Livery Street Birmingham B3 2PB, UK. ISBN 978-1-78216-144-8

[11] Doshi, Riddhi, et al. "Approximation algorithms and heuristics for a 2depot, hetero- geneous Hamiltonian path problem." International Journal of Robust and Nonlinear Control 21.12 (2011): 1434-1451

[12] ROS neotic documentation https://www.ros.org/

[13] Ameera Jaradat,Bara'a Matalkah,Waed Aldiabat."Solving Traveling Salesman Problem using Firefly algorithm and K-means Clustering" International Joint confernce on Electrical Engineering and Information Technology (2019)