ROUTE OPTIMIZATION FOR EMERGENCY VEHICLES

A PROJECT REPORT

Submitted by

SINDHUMATHI.B [REGISTER NO:211419104253]
SNEHA.V [REGISTER NO:211419104258]
SNEKA.R [REGISTER NO:211419104259]

in partial fulfillment for the award of the degree

of

BACHELOR OF ENGINEERING

IN

COMPUTER SCIENCE AND ENGINEERING

PANIMALAR ENGINEERING COLLEGE

(An Autonomous Institution, Affliated to Anna University, Chennai)

APRIL 2023

BONAFIDE CERTIFICATE

Certified that this project report **"ROUTE OPTIMIZATION FOR EMERGENCY VEHICLES"** is the bonafide work of **"SINDHUMATHI.B [211419104253], SNEHA.V [211419104258]** and **SNEKA.R [211419104259]"** who carried out the project work under my supervision.

SIGNATURE	**SIGNATURE**
Dr.L.JABASHEELA,M.E.,Ph.D.,	**Dr.K.VALARMATHI,M.E.,Ph.D.,**
HEAD OF THE DEPARTMENT	**SUPERVISOR**
	PROFESSOR
DEPARTMENT OF CSE,	DEPARTMENT OF CSE,
PANIMALAR ENGINEERING COLLEGE,	PANIMALAR ENGINEERING COLLEGE,
NASARATHPETTAI,	NASARATHPETTAI,
POONAMALLEE,	POONAMALLEE,
CHENNAI-600 123.	CHENNAI-600 123.

Certified that the above candidates were examined in the End Semester Project Viva-Voce Examination held on ...11.04.2023….

INTERNAL EXAMINER **EXTERNAL EXAMINER**

DECLARATION BY THE STUDENT

We **SINDHUMATHI.B (211419104253), SNEHA.V (2114191014258), SNEKA.R (211419104259)** hereby declare that this project report titled **"ROUTE OPTIMIZATION FOR EMERGENCY VECHICLES"**, under the guidance of **DR.K.VALARMATHI,M.E,PH.D.,** is the original work done by us and we have not plagiarized or submitted to any other degree in any university by us.

SINDHUMATHI.B

SNEHA.V

SNEKA.R

ACKNOWLEDGEMENT

We would like to express our deep gratitude to our respected Secretary and Correspondent **Dr.P.CHINNADURAI, M.A., Ph.D.** for his kind words and enthusiastic motivation, which inspired us a lot in completing this project.

We express our sincere thanks to our beloved Directors **Tmt.C.VIJAYARAJESWARI**, **Dr.C.SAKTHI KUMAR,M.E.,Ph.D** and **Dr.SARANYASREE SAKTHI KUMAR B.E.,M.B.A.,Ph.D.,** for providing us with the necessary facilities to undertake this project.

We also express our gratitude to our Principal **Dr.K.Mani, M.E., Ph.D.** who facilitated us in completing the project.

We thank the Head of the CSE Department, **Dr.L.JABASHEELA , M.E.,Ph.D.,** for the support extended throughout the project.

We would like to thank our **Project Guide, Dr.K.VALARMATHI,M.E.,Ph.D.,** and all the faculty members of the Department of CSE for their advice and encouragement for the successful completion of the project.

SINDHUMATHI.B
SNEHA.V
SNEKA.R

ABSTRACT

In this modern era getting struck in busy traffic for hours makes human impatient. Movement of emergency vehicles among these busy routes has become more tougher. So there is an immediate need for an optimized traffic flow control. This work delivers the study of traffic flow analysis in busy areas. Implementation of a suitable algorithm to analyze quick route to reach the destination for emergency vehicle is being analyzed here. This method is based on graph theory, where the data in the data sheet is first converted into graph with nodes and edges. Here, nodes represent the area and the edge represents the time required to travel through. The edges are given with different weight i.e. the distance and the traffic percentage in that route. Based on the value of these data the total time required to travel through the route is calculated. Based on this calculated value the shortest path is obtained by using the Dijkstra’s algorithm.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

CHAPTER 1
INTRODUCTION

Approximately 60% of the people get struck in the traffic in their day-to-day life which results in major problems. Transport sector is important for the economic growth of our nation. Because of these delays in transportation sector it leads to some critical situation that affect the business of many private and public sectors. The delay and loss of a business sector is very mere when compared to the delay in an emergency vehicle as it completely deals with human lives. Even a small delay will lead to critical situation of a persons, sometimes may also leads to death. To introduce an algorithm that preciously finds the shortest path and guide the emergency vehicle to reach the destination at faster rate. The solution helps not only emergency vehicle but also the common people who wishes to reach their destination on time. As a result, no one can have to wait for a long time in the traffic by finding the better ways to reach their destination.

1.1 OVERVIEW

The objective of this work is to predict the traffic flow in urban areas and to provide an efficient way for movement of emergency vehicles to reach their destination on time. To introduce an algorithm that preciously finds the shortest path and guide the emergency vehicle to reach the destination at faster rate. The solution helps not only emergency vehicle but also the common people who wishes to reach their destination on time.

1.2 PROBLEM DESCRIPTION

Due to heavy traffic congestion, most of the patients who are travelling on an emergency vehicles like ambulance, get struck in the traffic which leads to delay in reaching the hospital on time. Because of this reason, many patients condition becomes severe and sometimes leads to death. In order to prevent this critical situation an solution has been deployed to detect and prevent the traffic flow specifically for emergency vehicles.

CHAPTER 2
LITERATURE SURVEY

[1] Traffic flow prediction using machine learning and deep learning. Journal of Big Data volume 8, Article number: 152 (2021), Dec 04, 2021.

In recent years, there has been increased interest in the use of machine learning (ML) and deep learning (DL) in order to improve traffic flow forecasting and prediction. This literature review explores the application of ML and DL to the Internet of Things (IoT) and Intelligent Transportation Systems (ITS). It examines how ML and DL can be used to improve existing features, as well as to bridge the gap between the system of IoT and ITS, which may result in improved performance. Additionally, it is shown that ML and DL can improve algorithms, and that local urban authorities can benefit from their application.

[2] IoT security with Deep Learning-based Intrusion Detection Systems. 4th ICDS , 21-23, October 2020.

The IoT security deep learning based intrusion detection system has a major role in modern times. This system has been developed to help protect against security threats and researchers are looking into ways to integrate the system into real network traffic to help detect cyber attacks. New datasets are also being introduced to the IOT environment in order to reduce false positives and negatives. Algorithms are being developed and tested to give the intrusion detection system a powerful upgrade.

[3] IoT-based traffic prediction and traffic signal control system for smart city. ResearchGate,19 May 2021

The proposed traffic prediction and signal control system for smart cities utilizes the OWENN algorithm, Intel 80,286 microprocessor, and five stages of data collection in order to reduce traffic congestion that commonly occurs in metropolitan cities. This system was compared with existing ENN, CNN, NN and ANFIS techniques, using MAE, RMSE, and measure metrics to measure accuracy. The results showed an accuracy of 98.23% and a measure of 96.69%. Compared to existing techniques, this system provided lower efficiency, but uses Wi-Fi communication to improve energy usage and charging solutions for the future.

[4] KCLP: A k-Means Cluster-Based Location Privacy Protection Scheme in WSNs for IoT. IEEE Wireless Communications, December 2018.

KCLP is a K-Means Cluster-Based Location Privacy Protection Scheme in WSNs for loT, designed to reduce the problems associated with wireless sensor networks. This scheme creates a fake source, which functions similarly to a real source, and a fake sink, which also performs the same role. The KCLP increases safety time and reduces delay while consuming less energy. It has been compared to SLP and RBR, with results indicating that it is more efficient than both.

[5] An LSTM-Based Deep Learning Approach for Classifying Malicious Traffic at the Packet Level. IEEE ICASI , 19 August 2019.

This paper proposes a novel approach utilizing an LSTM model to classify malicious traffic at the packet level, with the aim of boosting the detection process of malicious attacks. It is competitive to prior literature that detects malicious traffic on the flow level and is inspired by the increasing prevalence of malicious attacks year by year. It offers the advantage of not requiring preprocessed packets to create flows and thus reduces the delay of the intrusion detection system (IDS).

[6] Sun, D. Design and implementation of traffic incident warning and release device under Internet of Vehicles. Dig. Technol. Appl.39(12), 201–203 (2021).

For analyzing the actual traffic conditions and calculating the traffic volume, density and traffic speed, a traffic prediction model is established and updated iteratively to modify the prediction model parameters. Based on this model, the congestion degree is estimated at the current road section, thus, an intelligent decision-making and the coordinated optimization methods are proposed.

[7] Traffic Prediction: Methods, Analysis, and Future Directions. Published in: IEEE Transactions on Intelligent Transportation Systems (Volume: 23, Issue: 6, June 2022)

The purpose of this paper is to provide a comprehensive survey on deep learning-based approaches in traffic prediction from multiple perspectives. Specifically, we first summarize the existing traffic prediction methods, and give a taxonomy. Second, we list the state-of-the-art approaches in different traffic prediction applications. Third, we comprehensively collect and organize widely used public datasets in the existing literature to facilitate other researchers. Furthermore, we give an evaluation and analysis by conducting extensive experiments to compare the performance of different methods on a real-world public dataset.

[8] Short-Term Prediction of City Traffic Flow via Convolutional Deep Learning. Published in: IEEE Access (Volume: 10)

The short-term predictions (10-60 minutes) of traffic flow data is a complex nonlinear task that has been the subject of many research efforts in past few decades. Accessing traffic flow data is mandatory for a large number of applications that have to guarantee a high level of services such as traffic flow analysis, traffic flow reconstruction, which in their turn are used to compute predictions needed to perform what-if analysis, forecast routing.

[9] Large-Scale Network Imputation and Prediction of Traffic Volume Based on Multi-Source Data Collection System. First published online March 28, 2023 in SAGE journals.

In this paper, we propose a multi-variable spatio-temporal learning technique based on multi-source traffic state information, which was realized by adopting Attention-based Spatial–Temporal Graph Convolutional Networks (ASTGCN). The proposed imputation method cooperatively aggregates spatial and temporal correlation from two different types of detectors into an integrated framework, which allows us to predict missing volume regardless of the missing rate. Moreover, the study was conducted on a large-scale network that contains the entire road characteristics. Daejeon city has served as a case study to demonstrate the performance, and the results show that the mean absolute error of the proposed method is under 12 vehicles/5 min.

CHAPTER 3
SYSTEM ANALYSIS

3.1 EXISTING SYSTEM

The Dijkstra’s algorithm used here finds the shortest path based on the distance between the two areas through different routes. This algorithm is used in many companies such as google maps, apple maps, android auto etc.. It cannot find the traffic analysis of any particular area. All nodes that are still immobile are considered to a load.

3.2 PROPOSED SYSTEM

The modified algorithm calculates the shortest path based on the value of time taken to reach the destination based on the calculation of both the distance between the source and the destination and also with the average traffic percentage in that particular route. Able to find the traffic analysis of any particular area. Fastest route to reach the destination.

3.3 FEASIBILITY STUDY

The objective of feasibility study is not only to solve the problem but also to acquire a sense of its scope. During the study, the problem definition was crystallized and aspects of the problem to be included in the system are determined. Consequently, benefits are estimated with greater accuracy at this stage. The key considerations are:

- Economic feasibility
- Technical feasibility
- Operational feasibility

Economic Feasibility

Economic feasibility studies not only the cost of hardware, software is included but also the benefits in the form of reduced costs are considered here. This project, if installed will certainly be beneficial since there will be reduction in manual work and increase in the speed of work.

Technical Feasibility

Technical feasibility evaluates the hardware requirements, software technology, available personnel etc., as per the requirements it provides sufficient memory to hold and process.

Operational Feasibility

This is the most important step of the feasibility study this study helps us predict the operational ability of the system that is being developed. This study also helps us analyze the approach towards which the system must be developed by which development effort is reduced. Proposed system is beneficial only if they can be turned into information systems that will meet the organization requirements. Only by spending time to evaluate the feasibility, do we reduce the chances from extreme embarrassments at larger stager of the project. Effort spends on a feasibility analysis that results in the cancellation of a proposed project is not a wasted effort.

3.4 HARDWARE REQUIREMENTS

- Processor : Pentium 4
- RAM : 512 MB and above
- Hard Disk : 40 GB and above

3.5 SOFTWARE REQUIREMENTS

- Operating System : Windows XP.
- Platform : Jupyter Notebook
- Backend : Anaconda
- Language : Python

SOFTWARE DESCRIPTION

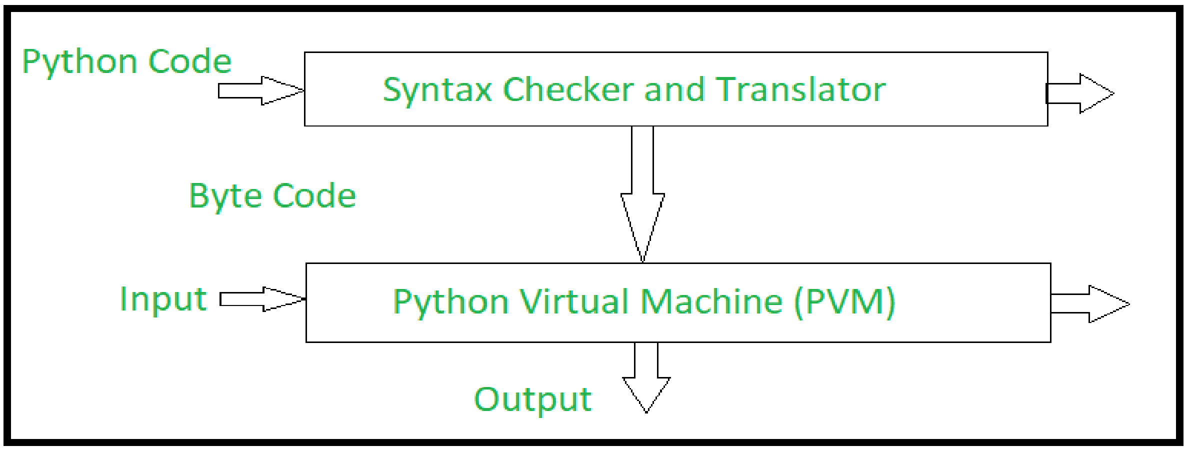
Python

Overview

Python can be used on a server to create web applications. Python can be used alongside software to create workflows. Python can connect to database systems. It can also read and modify files. Python can be used to handle big data and perform complex mathematics. Python can be used for rapid prototyping, or for production-ready software development. Python works on different platforms (Windows, Mac, Linux, Raspberry Pi, etc). Python has a simple syntax similar to the English language. Python has syntax that allows developers to write programs with fewer lines than some other programming languages. Python runs on an interpreter system, meaning that code can be executed as soon as it is written. This means that prototyping can be very quick. Python can be treated in a procedural way, an object-oriented way or a functional way.

Working with python

Python is an object-oriented programming language like Java. Python is called an interpreted language. Python uses code modules that are interchangeable instead of a single long list of instructions that was standard for functional programming languages. The standard implementation of python is called “cpython”. It is the default and widely used implementation of Python. Python doesn’t convert its code into machine code, something that hardware can understand. It actually converts it into something called byte code. So within python, compilation happens, but it’s just not into a machine language. It is into byte code (.pyc or .pyo) and this byte code can't be understood by the CPU. So we need an interpreter called the python virtual machine to execute the byte codes.



The Python source code goes through the following to generate an executable code

Step 1: The python compiler reads a python source code or instruction. Then it verifies that the instruction is well-formatted, i.e. it checks the syntax of each line. If it encounters an error, it immediately halts the translation and shows an error message.

Step 2: If there is no error, i.e. if the python instruction or source code is well-formatted then the compiler translates it into its equivalent form in an intermediate language called "Byte code".

Step 3: Byte code is then sent to the Python Virtual Machine(PVM) which is the python interpreter. PVM converts the python byte code into machine-executable code. If an error occurs during this interpretation then the conversion is halted with an error message.

Anaconda

Overview

Anaconda is a great package that already contains many Python packages, making it easy to enter the Python world. Additionally, you can create environments in Python that contain different versions of Python packages. For example, if your program only runs on versions of Matplotlib prior to Python 2.7, you can create a separate workspace for this program and switch back to Python 3 with one click. In addition, switching between Tensorflow 2.0 and Tensorflow 1.15 will also be easier, so you will eventually be able to switch between versions without any problems (otherwise it can cause quite a headache). Miniconda is a barebones version of Anaconda. Working on servers with limited disk space. When Anaconda (conda) and Jupyter Notebook (Jupyter Lab) are set up properly, their combination makes for a perfect team that can easily switch between deep learning conda environments.

Jupyter Notebook

Overview

Jupyter Notebook is an open source web application that lets you create and share documents with live code, equations, visualizations, and text. The Jupyter Notebook is maintained by Project Jupyter staff. Jupyter Notebooks is a project derived from his IPython project, where previously he had an IPython notebook project itself. The name Jupyter comes from core programming languages that support Julia, Python, and R. Jupyter comes with an IPython kernel that allows you to program in Python, but there are currently over 100 other kernels available. as well. Jupyter Notebooks are very useful not only for learning and teaching programming languages like Python, but also for sharing data. Google and Microsoft have their own versions of his Notebook that can be used to create and share notebooks on Google Collaboratory.

CHAPTER 4
SYSTEM DESIGN

4.1 ER DIAGRAM

An Entity Relationship (ER) Diagram is a type of flowchart that illustrates how "entities" such as people, objects or concepts relate to each other within a system. ER diagrams are related to data structure diagrams (DSDs), which focus on the relationships of elements within entities instead of relationships between entities themselves.

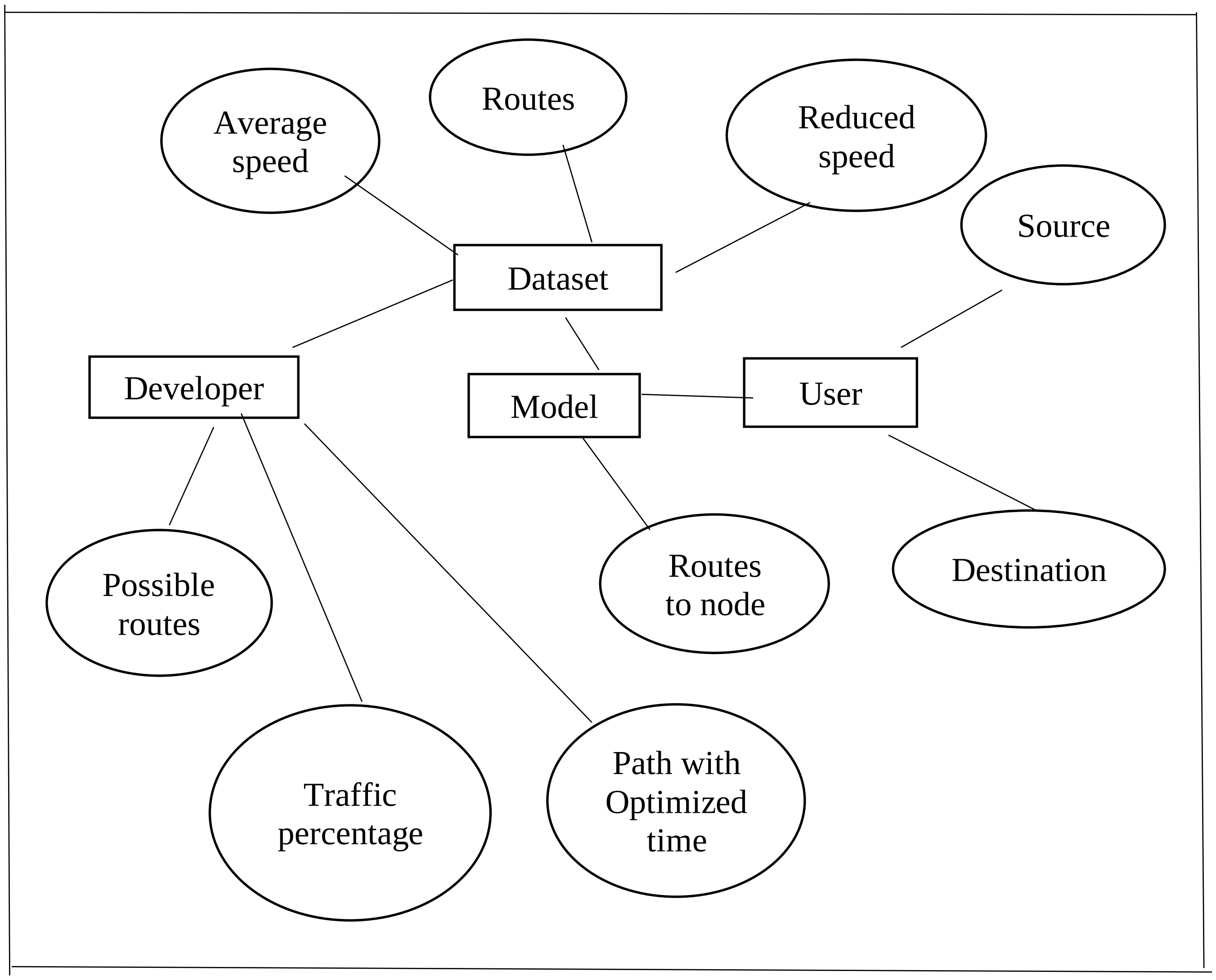


Figure 4.1 ER Diagram

4.2 DATA DICTIONARY

This is normally represented as the data about data. It is also termed as metadata some times which gives the data about the data stored in the database. It defines each data term encountered during the analysis and design of a new system. Data elements can describe files or the processes. Following are some rules, which defines the construction of data dictionary entries:

1. Words should be defined to understand for what they need and not the variable need by which they may be described in the program.
2. Each word must be unique. We cannot have two definition of the same client.
3. Aliases or synonyms are allowed when two or more enters shows the same meaning.
4. A self-defining word should not be decomposed. It means that the reduction of any information in to subpart should be done only if it is really required that is it is not easy to understand directly.

Data dictionary includes information such as the number of records in file, the frequency a process will run, security factor like pass word which user must enter to get excess to the information.

4.3 TABLE NORMALIZATION

Table 4.3.1 Table normalization

SOURCE	DESTINATION	DISTANCE	AVERAGE SPEED	REDUCED SPEED	TIME
28 AVE	SCHERMERHORN ST	50	60	9.0	58
28 AVE	ARLINGTON AVE	40	60	13.8	51
28 AVE	QUEENS BLVD	25	60	12.0	31
SCHERMERH ORN ST	ARLINGTON AVE	15	60	13.8	19
ARLINGTON AVE	ARTHUR KIL RD	25	60	18.0	35
ARLINGTON AVE	FULTON AVE	25	60	16.8	34
QUEENS BLVD	PITKIN AVE	40	60	18.6	57
ARTHUR KIL RD	FOREST AVE	40	60	17.4	56
FOREST AVE	FULTON AVE	15	60	18.6	21
TIDES LN	FULTON AVE	40	60	11.4	49
TIDES LN	WILLIS AVE	25	60	13.8	32
NORTON ST	PITKIN AVE	20	60	9.0	23
PRESIDENT ST	CROPSEY AVE	50	60	10.2000001	60

4.4 DATA FLOW DIAGRAM

A data-flow diagram is a way of representing a flow of data through a process or a system. The DFD also provides information about the outputs and inputs of each entity and the process itself. A data-flow diagram has no control flow there are no decision rules and no loops.

0-Level DFD

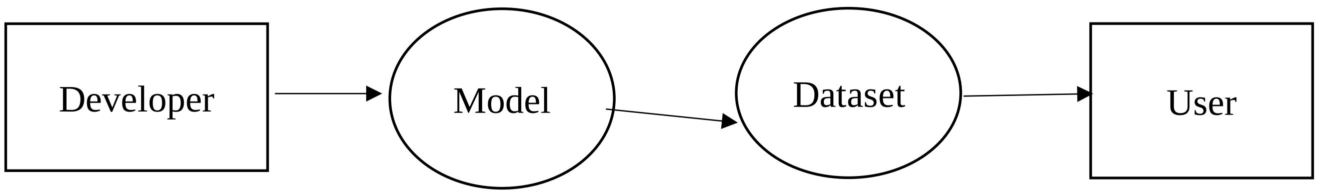


Figure 4.4.1 DFD Level 0

1-Level DFD

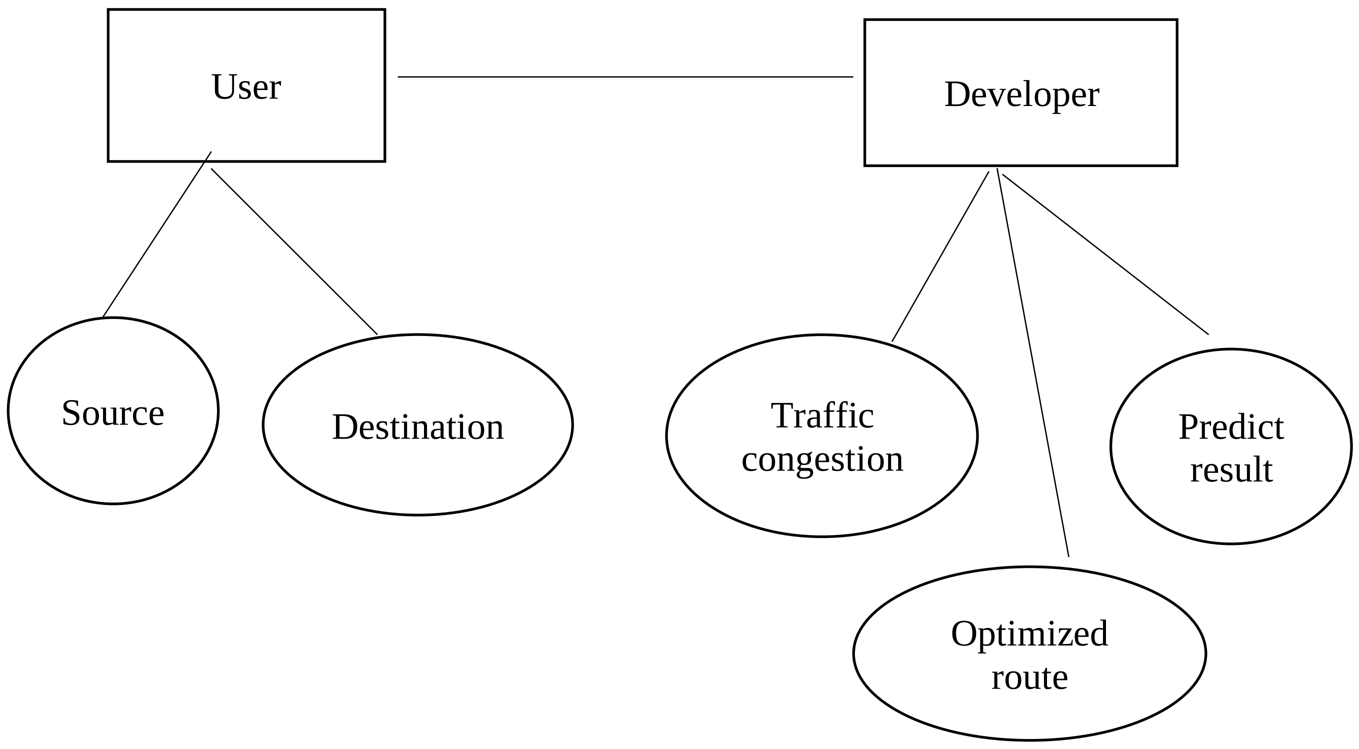


Figure 4.4.2 DFD Level 1

2-Level DFD

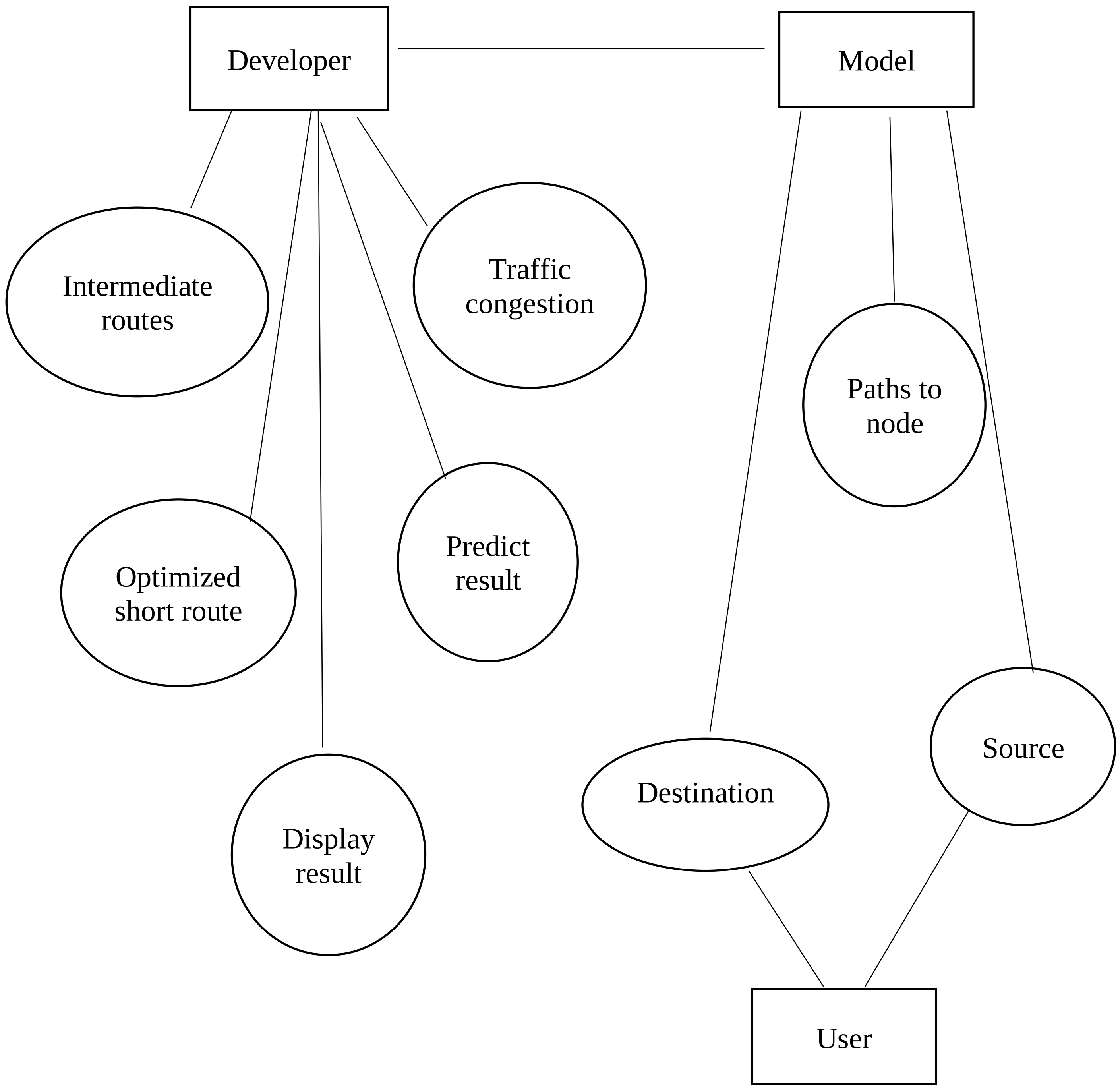


Figure 4.4.3 DFD Level 2

4.5 UML DIAGRAMS

USECASE DIAGRAM

A use case diagram shows various use cases and different types of users the system has and will often be accompanied by other types of diagrams as well. The use cases are represented by either circles or ellipses.

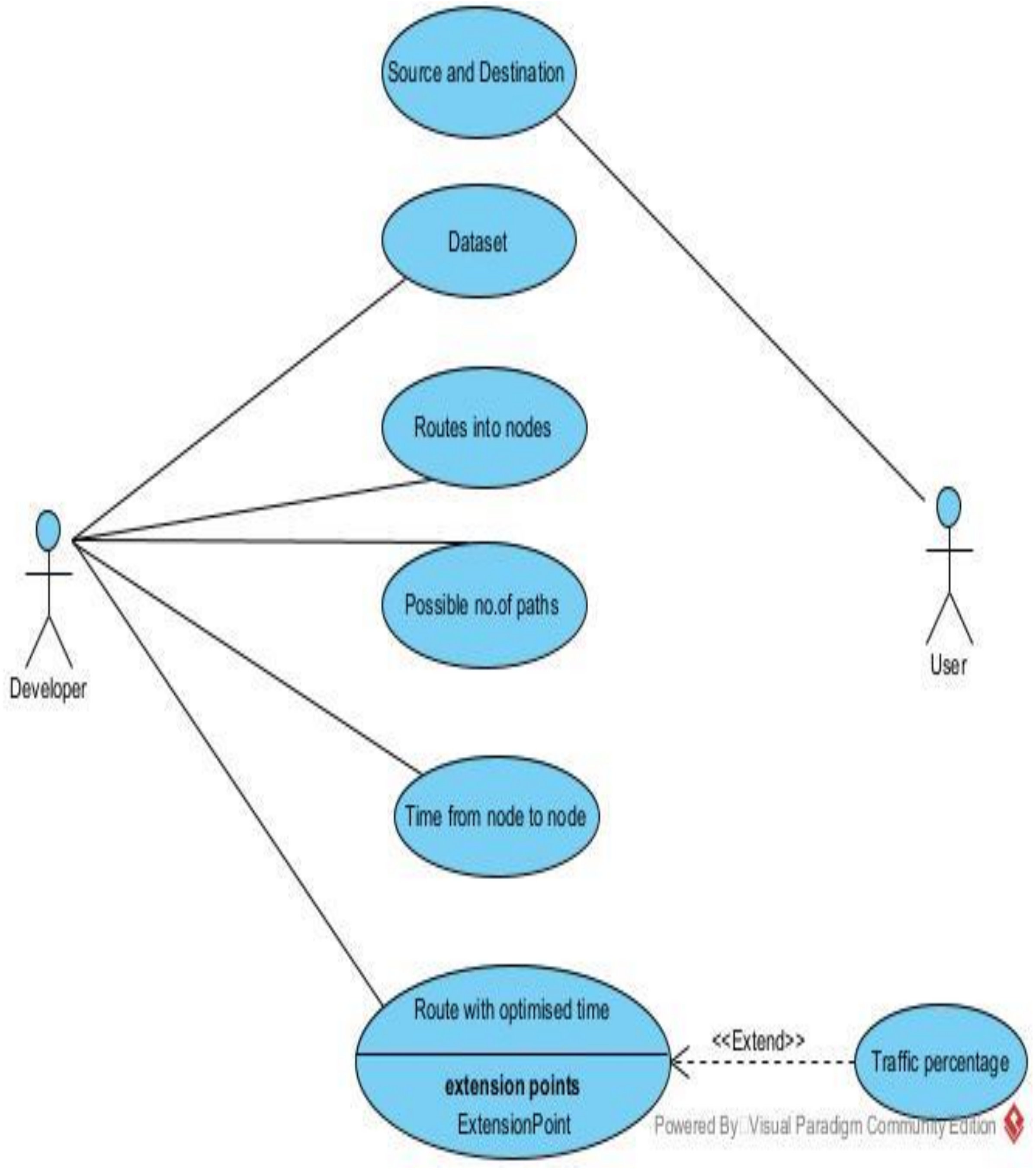


Figure 4.5.1 Use case Diagram

CLASS DIAGRAM

A class diagram is a type of static structure diagram that describes the structure of a system by showing the system's classes, their attributes, operations, and the relationships among objects.

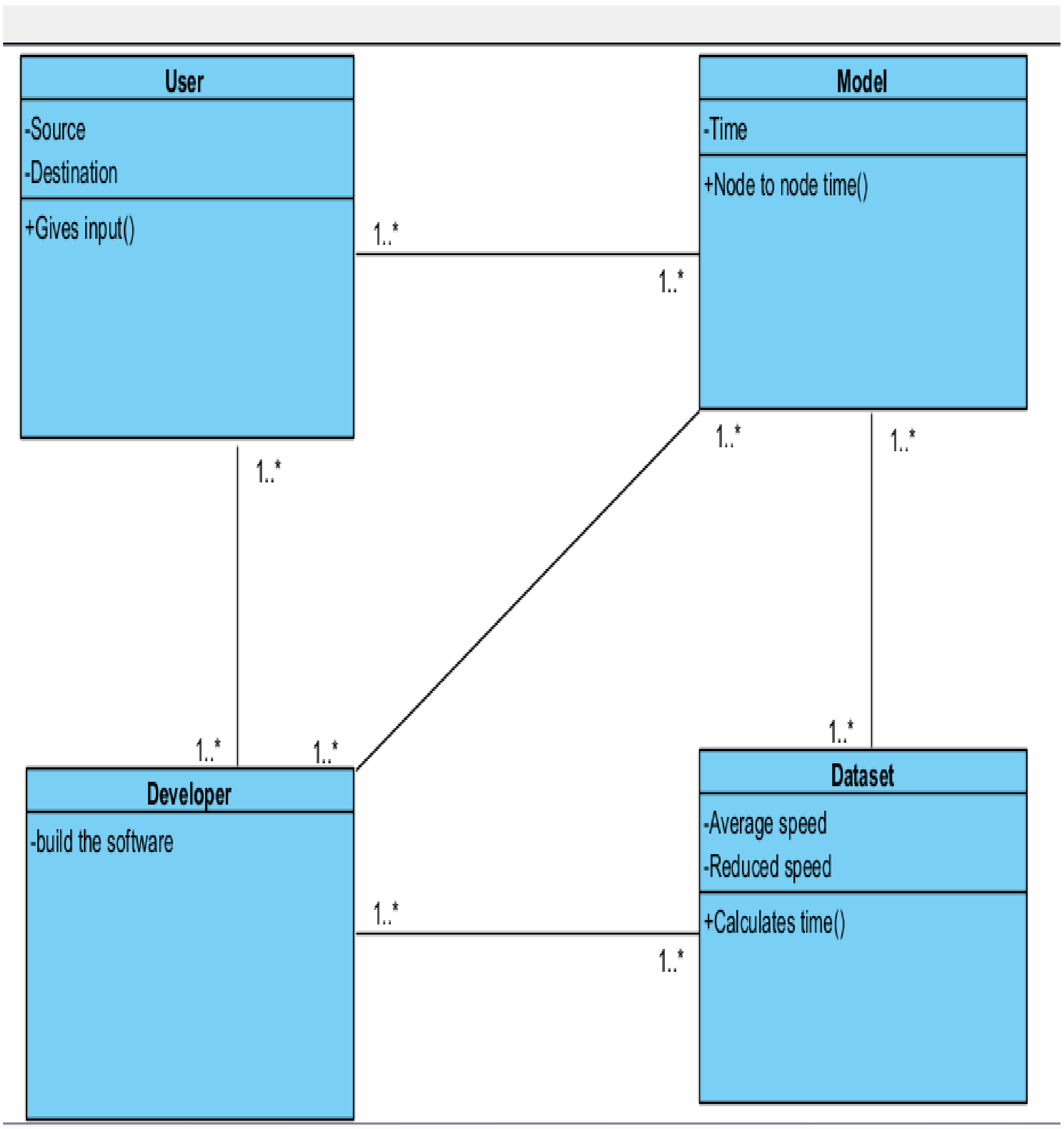


Figure 4.5.2 Class Diagram

ACTIVITY DIAGRAM

An activity diagrams are graphical representations of workflows of stepwise activities and actions with support for choice, iteration and concurrency. Activity diagram is basically a flowchart to represent the flow from one activity to another activity.

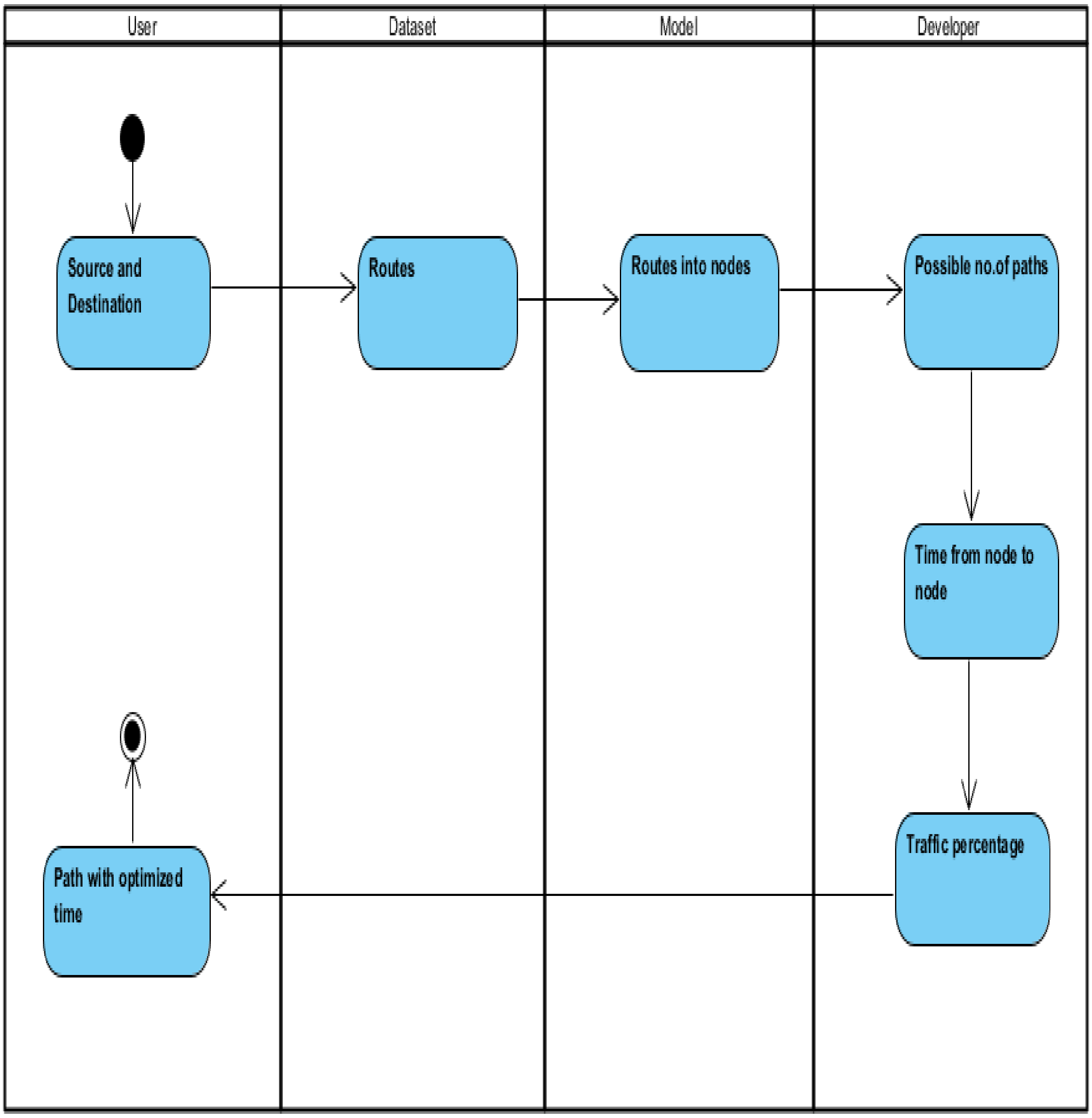
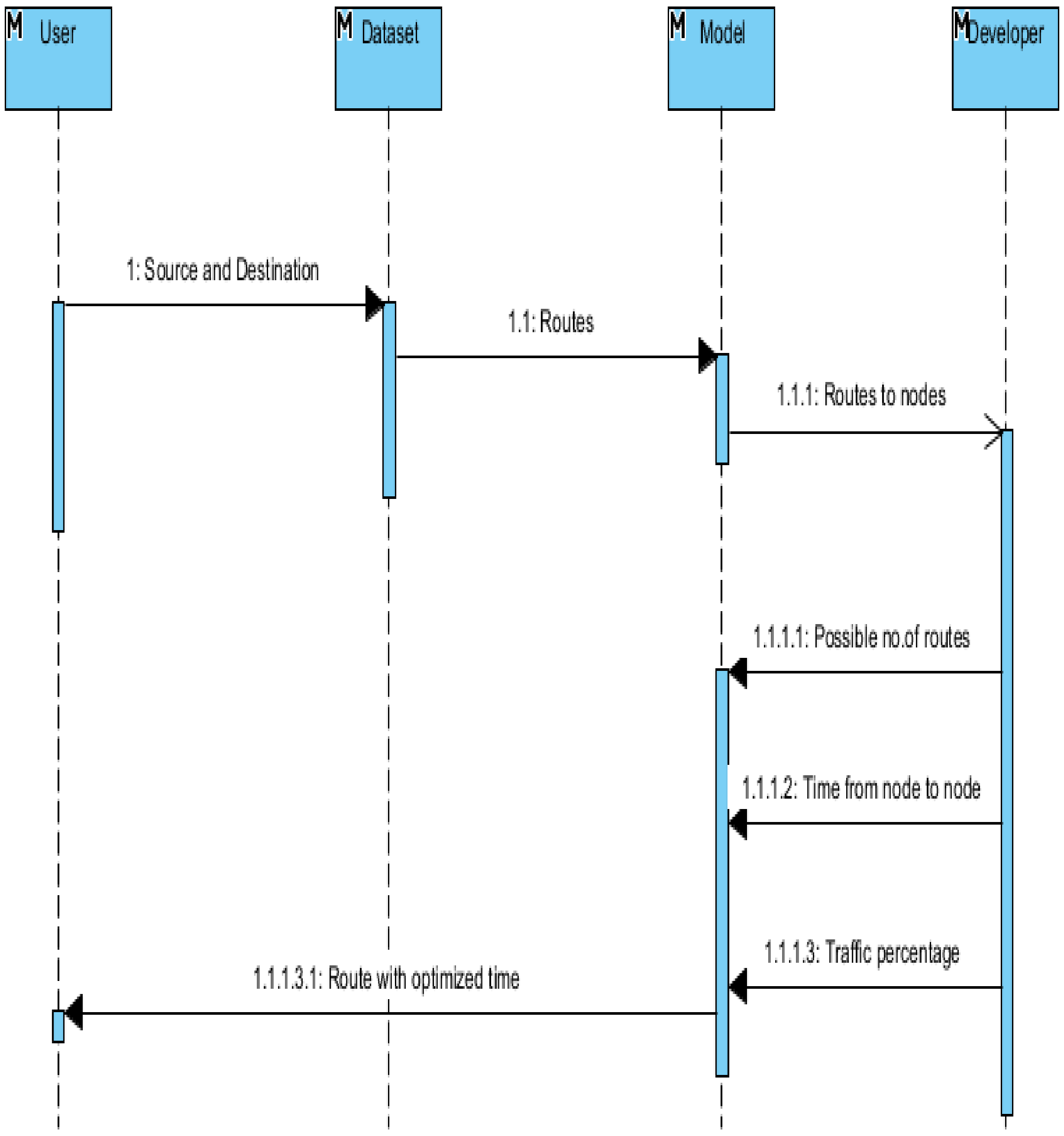


Figure 4.5.3 Activity Diagram

SEQUENCE DIAGRAM

A sequence diagram or system sequence diagram shows process interactions arranged in time sequence in the field of software engineering. It depicts the processes involved and the sequence of messages exchanged between the processes needed to carry out the functionality.



User
Dataset
Model
Developer
1: Source and Destination
1.1: Routes
1.1.1: Routes to nodes
1.1.1.1: Possible no.of routes
1.1.1.2: Time from node to node
1.1.1.3: Traffic percentage
1.1.1.3.1: Route with optimized time

Figure 4.5.4 Sequence Diagram

COLLABORATION DIAGRAM

A collaboration diagram, also known as a communication diagram, is an illustration of the relationships and interactions among software objects in the Unified Modeling Language (UML).

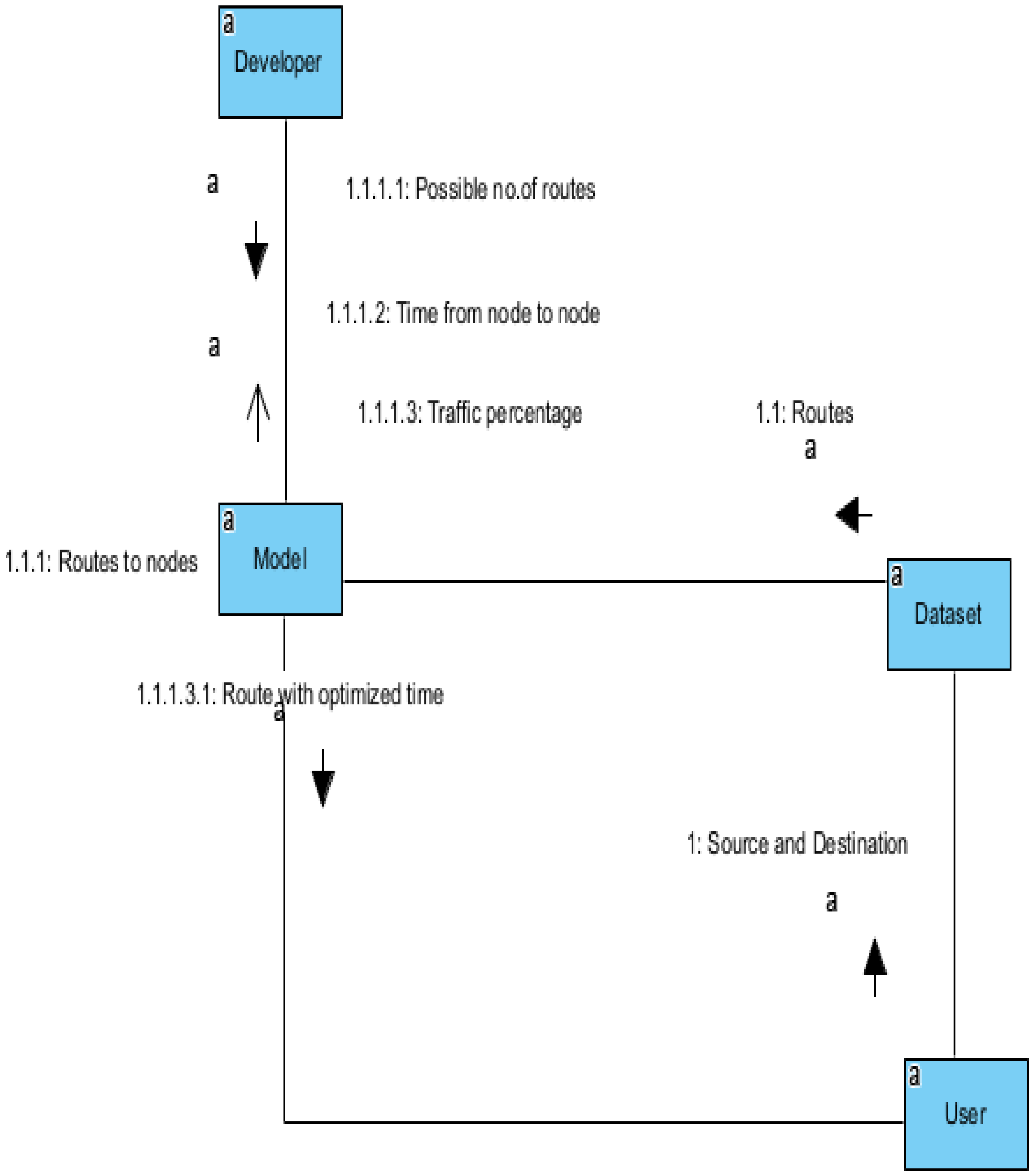

a
Developer
a
1.1.1.1: Possible no.of routes
1.1.1.2: Time from node to node
a
1.1.1.3: Traffic percentage
1.1.: Routes
a
a
Model
1.1.1: Routes to nodes
a
Dataset
1.1.1.3.1: Route with optimized time
a
1: Source and Destination
a
a
User

Figure 4.5.5 Collaboration Diagram

CHAPTER 5

SYSTEM ARCHITECTURE

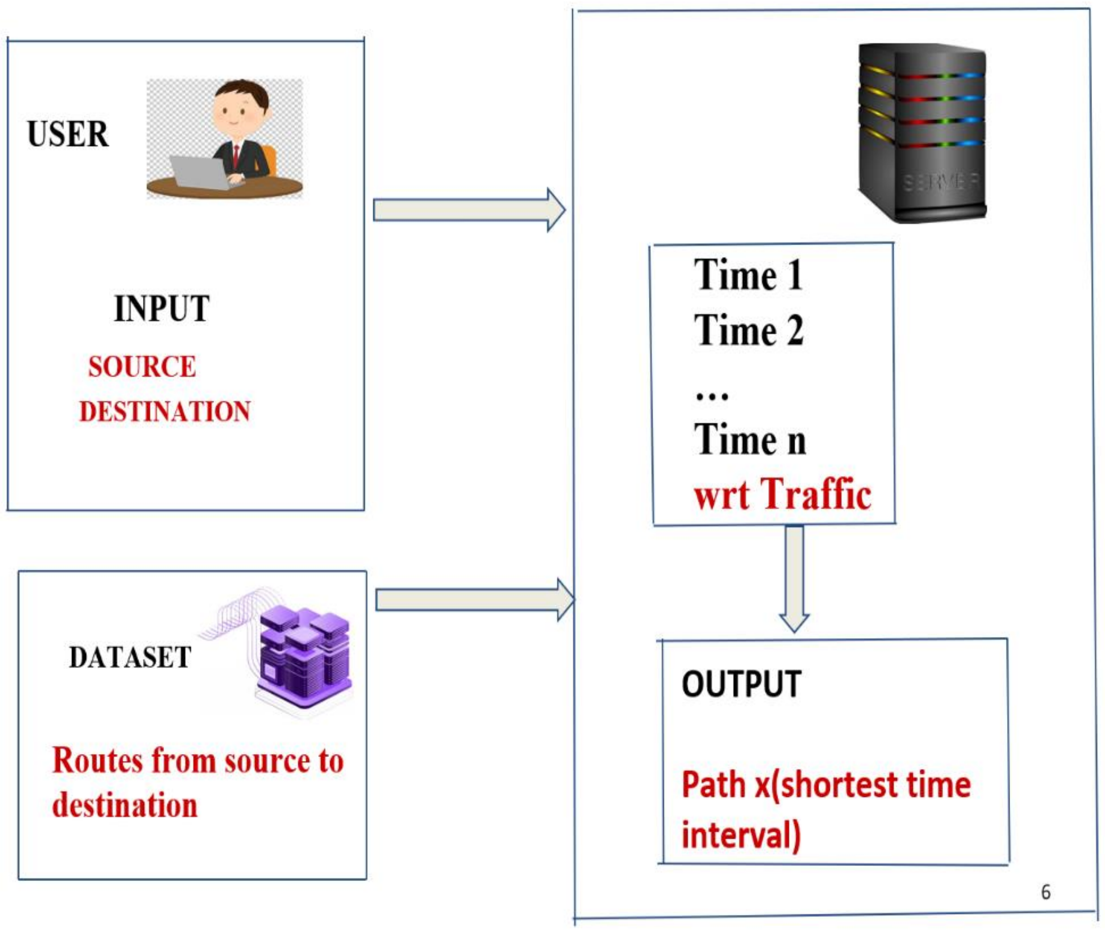


Fig 5.1 ARCHITECTURE OVERVIEW

The main reason to final this concept is to detect the traffic flow control in an efficient way by calculating the total time required to travel from one place to another rather than finding the distance between the two places. It is more efficient then the existing Dijkstra’s algorithm by producing the accurate result with the calculation of total time required to cover the required distance between the two places. This system finds the shortest path on the basis of both the distance between the two nodes and also the traffic congestion in that particular areas.

5.1 MODULE DESIGN SPECIFICATION

GRAPH MODULE

In the graph module, a detailed data about the road network is first converted into the graph. The time taken to move from one location to another is obtained from the calculation of the distance and weightage of the traffic in that route along with the standard average speed calculation. Dijkstra’s algorithm is applied on the graph where one input is taken from the origin and the other from the destination and the algorithm returns the fastest route and the shortest time a vehicle is likely to take. Several tests are conducted by running the program at different times.

TIME MODULE

The time is used to calculate the time taken to move from one location to another is obtained from the calculation of the distance and weightage of the traffic in that route along with the standard average speed calculation.

$$\mathbf{T = D / (S - ((S*P)/100))}$$

D - Distance between two nodes

P - Percentage of traffic flow

S - Average speed

T – Time taken to reach destination

The system is used to give the output as the fastest route the approximate time to reach the destination. The output results in each and every area that has been covered to travel from origin to destination along with their respective distance and time required to cross that corresponding intermediate areas. e, the algorithm considers the route with the minimum time and returns it together with the total approximate time in minutes.

5.2 ALGORITHMS

5.2.1 GRAPH ALGORITHM

In graph algorithm, a detailed data about the road network is first converted into the graph. Graphs are used to model connections between objects, people, or entities. They have two main elements: nodes and edges. Nodes represent objects and edges represent the connections between these objects. The time taken to move from one location to another is obtained from the calculation of the distance and weightage of the traffic in that route along with the standard average speed calculation. Dijkstra's algorithm is applied on the graph where one input is taken from the origin and the other from the destination and the algorithm returns the fastest route and the shortest time a vehicle is likely to take. Several tests are conducted by running the program at different times.

Graphs can be:

- **Undirected -** if for every pair of connected nodes, you can go from one node to the other in both directions.
- **Directed -** if for every pair of connected nodes, you can only go from one node to another in a specific direction. We use arrows instead of simple lines to represent directed edges.

5.2.2 DIJKSTRA’S ALGORITHM

Dijkstra's Algorithm finds the shortest path between a given node (which is called the "source node") and all other nodes in a graph. This algorithm uses the weights of the edges to find the path that minimizes the total distance (weight) between the source node and all other nodes. It is a single source shortest paths algorithm. It means that it finds the shortest paths from a single source vertex to all other vertices in a graph. It is a greedy algorithm and works for both directed and undirected, positively weighted graphs (a graph is called positively weighted if all of its edges have only positive weights).

STEPS

1. Mark the ending vertex with a distance of zero. Designate this vertex as current.

2. Find all vertices leading to the current vertex. Calculate their distances to the end. Since we already know the distance the current vertex is from the end, this will just require adding the most recent edge. Don’t record this distance if it is longer than a previously recorded distance.

3. Mark the current vertex as visited. We will never look at this vertex again.

4. Mark the vertex with the smallest distance as current and repeat from step 2.

5.3.3 TIME ALGORITHM

Time algorithm is used to develop the section of a road network where the user gives the input as the origin and the destination. Thus, the system applies the Dijkstra's algorithm to give the output as the fastest route the approximate time to reach the destination. The output results in each and every area that has been covered to travel from origin to destination along with their respective distance and time required to cross that corresponding intermediate areas.

The algorithm considers the route with the minimum time and returns it together with the total approximate time in minutes. All the areas in that particular shortest route is calculated and shown in the output. This helps us to keep track the areas and reach the destination by easily identifying the routes and areas.

The system is used to give the output as the fastest route the approximate time to reach the destination. The output results in each and every area that has been covered to travel from origin to destination along with their respective distance and time required to cross that corresponding intermediate areas. The algorithm considers the route with the minimum time and returns it together with the total approximate time in minutes

CHAPTER 6

SYSTEM IMPLEMENTATION

6.1 Server-side coding

```
import numpy as np

import pandas as pd

from csv import reader

from csv import DictReader


source = input("enter source: ")

destination =input("enter destination: ")


from collections import defaultdict

class Graph():

    def __init__(self):

        self.edges = defaultdict(list)

        self.weights = {}

    def add_edge(self, from_node, to_node, weight):

        # Note: assumes edges are bi-directional

        self.edges[from_node].append(to_node)

        self.edges[to_node].append(from_node)

        self.weights[(from_node, to_node)] = weight
        self.weights[(to_node, from_node)] = weight
graph = Graph()


with open('data.csv', 'r') as read_obj:
    csv_dict_reader = DictReader(read_ob
    Average = 60
    for row in csv_dict_reader:
        Distance = int(row['distance'])
        Percentage = int(row['traffic percentage'])
        Decreased_speed = Average * (Percentage/100)
        Actual_speed = Average - Decreased_speed
        Total_time = Distance / Actual_speed
        Total_time = int(Total_time * 60)
        print("Source        Destination        Distance     Average Speed    Reduced
speed    Actual speed    time")
        print("{}      {}       {}          {}            {}            {}            {}".format
(row['from'], row['to'], Distance, Average, Decreased_speed, Actual_speed,
Total_time))
        graph.add_edge(row['from'], row['to'], Total_time)


def dijsktra(graph, initial, end):
    # shortest paths is a dict of nodes
    # whose value is a tuple of (previous node, weight)
    shortest_paths = {initial: (None, 0)}
    current_node = initial
    visited = set()

    while current_node != end:
        visited.add(current_node)
        destinations = graph.edges[current_node]
        weight_to_current_node = shortest_paths[current_node][1]
        for next_node in destinations:
        weight = graph.weights[(current_node, next_node)] +
weight_to_current_node

        if next_node not in shortest_paths:
                shortest_paths[next_node] = (current_node, weight)
        else:
                current_shortest_weight = shortest_paths[next_node][1]
                if current_shortest_weight > weight:
                    shortest_paths[next_node] = (current_node, weight)
                next_destinations = {node: shortest_paths[node]for node in
shortest_paths if node not in visited}
```

```
            if not next_destinations:

            return "Route Not Possible"


        # next node is the destination with the lowest weight

        current_node = min(next_destinations, key=lambda k:
    next_destinations[k][1])


        # Work back through destinations in shortest path

        path = []

        total=0

       while current_node is not None:

           path.append(current_node)


           #print(current_node)

           next_node = shortest_paths[current_node][0]

           if next_node is not None:

               print("from : {} , to : {} , Time : {} minutes".format
    (next_node,current_node,graph.weights[(next_node, current_node)]))

               total = total+graph.weights[(next_node, current_node)]

               current_node = next_node
    # Reverse path

    print("Total Time = {} minutes".format(total))

    path = path[::-1]

    return path


dijsktra(graph, source, destination)


path1=dijsktra(graph, source, destination)


print('Source : '+path1[0])

for x in range(len(path1)):

    if x != 0 and x != len(path1)-1:

        print(path1[x])

print('Destination : '+path1[len(path1)-1])
```

CHAPTER 7
SYSTEM TESTING

The software, which has been developed, has to be tested to prove its validity. Testing is considered to be the least creative phase of the whole cycle of system design. In the real sense it is the phase, which helps to bring out the creativity of the other phases makes it shine.

7.1 UNIT TESTING

This is the first level of testing. In these different modules are tested against the specifications produced during the design of the module. During this testing the number of arguments is compared to input parameters, matching of parameter and arguments etc. It is also ensured whether the file attributes are correct, whether the Files are opened before using, whether Input/output errors are handled etc. Unit Test is conducted using a test Driver usually.

7.2 INTEGRATION TESTING

Integration testing is a systematic testing for constructing the program structure, while at the same time conducting test to uncover errors associated within the interface. Bottom-up integration is used for this phase. It begins construction and testing with atomic modules. This strategy is implemented with the following steps.

- Low-level modules are combined to form clusters that perform a specific software sub function.
- The cluster is tested.
- Drivers are removed and clusters are combined moving upward in the program structure.

7.3 TEST CASES AND REPORTS

Table 7.3.1 Test cases

TEST CASE ID	TEST CASE TO BE PERFORMED	EXPECTED RESULT	OBTAINED RESULT	PASS/FAIL	REMARKS
TC01	Dataset	Successful import	Successful import	Pass	Imported successfully
TC02	Dataset	Preprocess successful	Preprocess successful	Pass	Preprocessed successfully
TC03	Dataset	Model creation	Model created	Pass	Model created successfully
TC04	Model	Successful compilation	Successful compilation	Pass	Model compiled successfully
TC05	Input (source and destination)	Input obtained without error	Input obtained without error	Pass	Input obtained successfully
TC06	Route optimization	Found intermediate routes	Found intermediate routes	Pass	Found successfully
TC07	Output (Shortest route)	Shortest route found	Shortest route found	Pass	Output obtained successfully

CHAPTER 8
CONCLUSION

8.1 RESULTS AND DISCUSSIONS

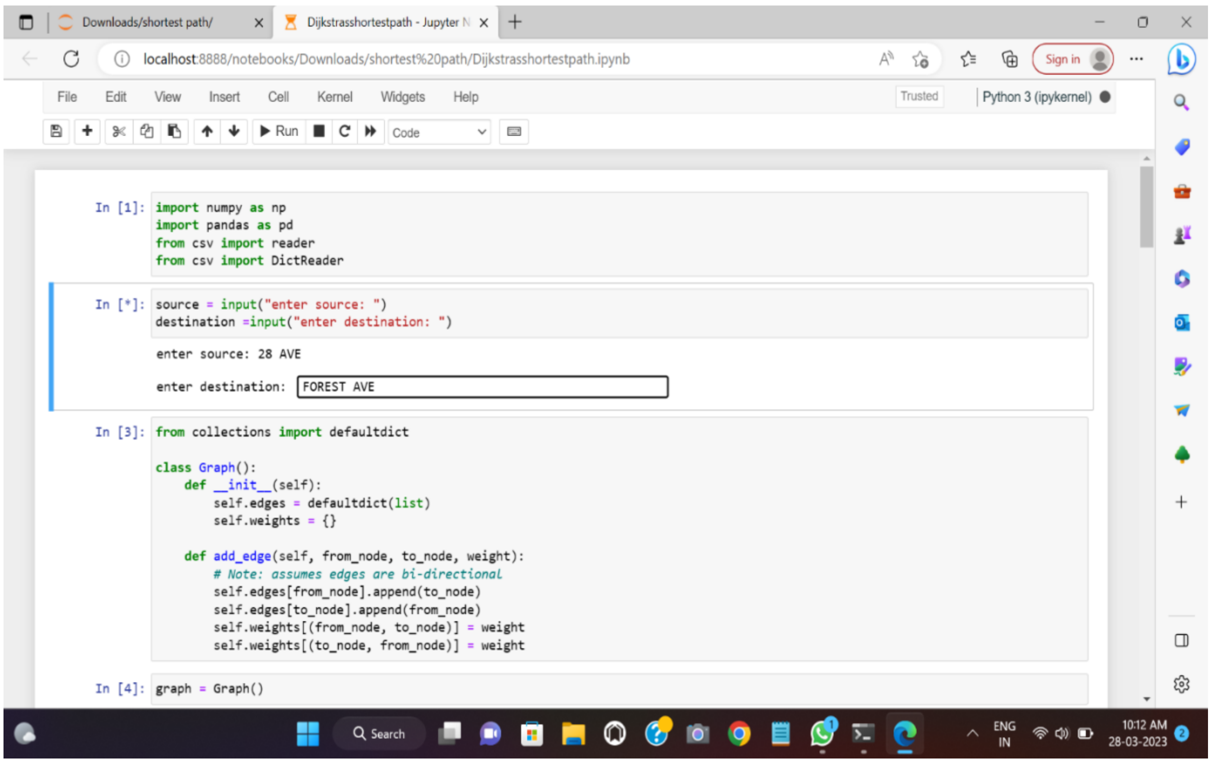
The system is used to develop the section of a road network where the user gives the input as the origin and the destination. Thus, the system applies the dijkstra's algorithm to give the output as the fastest route the approximate time to reach the destination. The output results in each and every area that has been covered to travel from origin to destination along with their respective distance and time required to cross that corresponding intermediate areas. The algorithm considers the route with the minimum time and returns it together with the total approximate time in minutes. As seen in the result, all the areas in that particular shortest route is calculated and shown in the output. This helps us to keep track the areas and reach the destination by easily identifying the routes and areas. This system helps not only the regular vehicle but also the emergency vehicle to reach the hospital very soon and helps the patient to save their lives.

8.2 CONCLUSION AND FUTURE ENHANCEMENTS

In conclusion, it is evident that in the current world with developing technology, efficiency and reliability are critical when developing new systems and implement in the real world. There is a high need to save time especially on the roads during critical situation and also obtain reliable information on the traffic situation that can be relied upon. The system developed shows that the fastest route from one point to another can be determined with a suitable algorithm and also helps to save many lives by its efficient traffic flow control methodology. This meets the main objective of the paper.

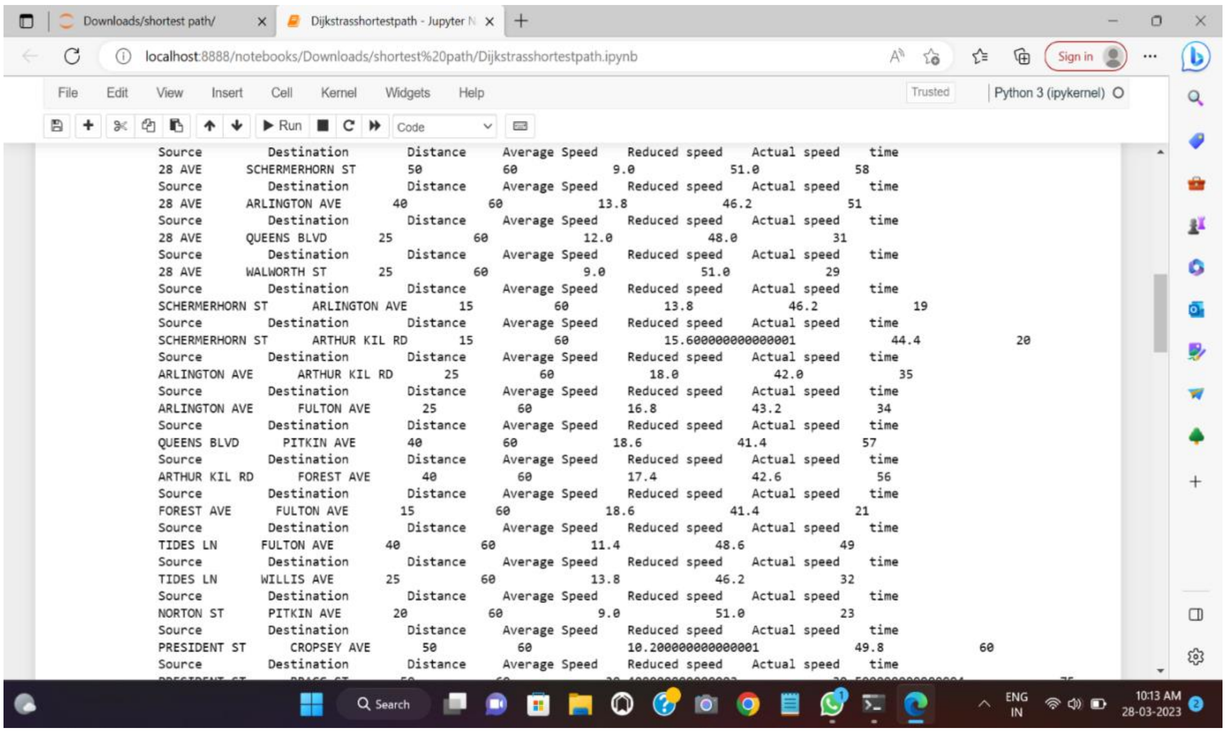
APPENDICES

A.1 SAMPLE SCREENSHORTS



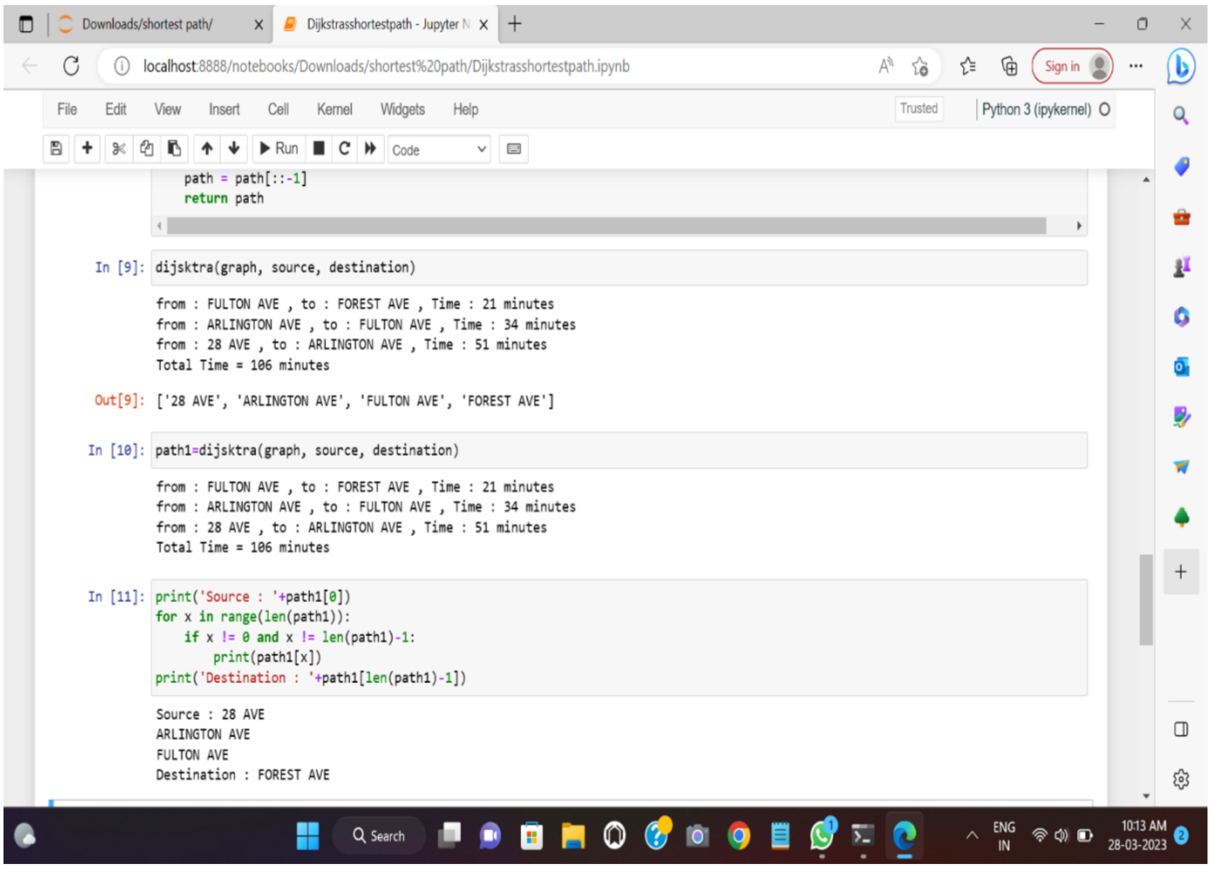
Downloads/shortest path/
Dijkstrasshortestpath - Jupyter N
localhost:8888/notebooks/Downloads/shortest%20path/Dijkstrasshortestpath.ipynb
File Edit View Insert Cell Kernel Widgets Help
Trusted
Python 3 (ipykernel)
Run
Code

```
In [1]: import numpy as np
        import pandas as pd
        from csv import reader
        from csv import DictReader

In [*]: source = input("enter source: ")
        destination =input("enter destination: ")

        enter source: 28 AVE

        enter destination: FOREST AVE

In [3]: from collections import defaultdict

        class Graph():
            def __init__(self):
                self.edges = defaultdict(list)
                self.weights = {}

            def add_edge(self, from_node, to_node, weight):
                # Note: assumes edges are bi-directional
                self.edges[from_node].append(to_node)
                self.edges[to_node].append(from_node)
                self.weights[(from_node, to_node)] = weight
                self.weights[(to_node, from_node)] = weight

In [4]: graph = Graph()
```



Downloads/shortest path/
Dijkstrasshortestpath - Jupyter N
localhost:8888/notebooks/Downloads/shortest%20path/Dijkstrasshortestpath.ipynb
File Edit View Insert Cell Kernel Widgets Help
Trusted
Python 3 (ipykernel)
Run
Code

```
Source        Destination        Distance     Average Speed    Reduced speed    Actual speed    time
28 AVE      SCHERMERHORN ST        50           60              9.0            51.0             58
Source        Destination        Distance     Average Speed    Reduced speed    Actual speed    time
28 AVE      ARLINGTON AVE      40           60             13.8            46.2             51
Source        Destination        Distance     Average Speed    Reduced speed    Actual speed    time
28 AVE      QUEENS BLVD      25           60             12.0             48.0             31
Source        Destination        Distance     Average Speed    Reduced speed    Actual speed    time
28 AVE      WALWORTH ST      25           60             9.0             51.0             29
Source        Destination        Distance     Average Speed    Reduced speed    Actual speed    time
SCHERMERHORN ST      ARLINGTON AVE      15           60             13.8             46.2             19
Source        Destination        Distance     Average Speed    Reduced speed    Actual speed    time
SCHERMERHORN ST      ARTHUR KIL RD      15           60             15.600000000000001           44.4             20
Source        Destination        Distance     Average Speed    Reduced speed    Actual speed    time
ARLINGTON AVE      ARTHUR KIL RD      25           60             18.0             42.0             35
Source        Destination        Distance     Average Speed    Reduced speed    Actual speed    time
ARLINGTON AVE      FULTON AVE      25           60             16.8             43.2             34
Source        Destination        Distance     Average Speed    Reduced speed    Actual speed    time
QUEENS BLVD      PITKIN AVE      40           60             18.6             41.4             57
Source        Destination        Distance     Average Speed    Reduced speed    Actual speed    time
ARTHUR KIL RD      FOREST AVE      40           60             17.4             42.6             56
Source        Destination        Distance     Average Speed    Reduced speed    Actual speed    time
FOREST AVE      FULTON AVE      15           60             18.6             41.4             21
Source        Destination        Distance     Average Speed    Reduced speed    Actual speed    time
TIDES LN      FULTON AVE      40           60             11.4             48.6             49
Source        Destination        Distance     Average Speed    Reduced speed    Actual speed    time
TIDES LN      WILLIS AVE      25           60             13.8             46.2             32
Source        Destination        Distance     Average Speed    Reduced speed    Actual speed    time
NORTON ST      PITKIN AVE      20           60             9.0             51.0             23
Source        Destination        Distance     Average Speed    Reduced speed    Actual speed    time
PRESIDENT ST      CROPSEY AVE      50           60             10.200000000000001           49.8             60
Source        Destination        Distance     Average Speed    Reduced speed    Actual speed    time
```


Downloads/shortest path/
Dijkstrasshortestpath - Jupyter N
localhost:8888/notebooks/Downloads/shortest%20path/Dijkstrasshortestpath.ipynb
File Edit View Insert Cell Kernel Widgets Help
Trusted
Python 3 (ipykernel)
Run
Code

```
        path = path[::-1]
        return path
```

In [9]:
```
dijsktra(graph, source, destination)
```

```
from : FULTON AVE , to : FOREST AVE , Time : 21 minutes
from : ARLINGTON AVE , to : FULTON AVE , Time : 34 minutes
from : 28 AVE , to : ARLINGTON AVE , Time : 51 minutes
Total Time = 106 minutes
```

Out[9]:
```
['28 AVE', 'ARLINGTON AVE', 'FULTON AVE', 'FOREST AVE']
```

In [10]:
```
path1=dijsktra(graph, source, destination)
```

```
from : FULTON AVE , to : FOREST AVE , Time : 21 minutes
from : ARLINGTON AVE , to : FULTON AVE , Time : 34 minutes
from : 28 AVE , to : ARLINGTON AVE , Time : 51 minutes
Total Time = 106 minutes
```

In [11]:
```
print('Source : '+path1[0])
for x in range(len(path1)):
    if x != 0 and x != len(path1)-1:
        print(path1[x])
print('Destination : '+path1[len(path1)-1])
```

```
Source : 28 AVE
ARLINGTON AVE
FULTON AVE
Destination : FOREST AVE
```

REFERENCES

[1] Agafonov, A., & Myasnikov, V. (2017). Efficiency comparison of the routing algorithms used in centralized traffic management systems. Procedia Engineering, Vol. 201, 265-270.

[2] Agafonov, A., & Myasnikov, V. (2018). Vehicle routing algorithms based on a route reservation approach. Journal of Physics: Conference Series, Vol. 1069, 1-9.

[3] Mu, H., Yu, J., & Liu, L. (2009). Shortest path algorithm for road network with traffic restriction. IEEE 2nd International Conference on Power Electronics and Intelligent Transportation System, China.

[4] Nha, V., Djahel, S., & Murphy, J. (2012). A comparative study of vehicles' routing algorithms for route planning in smart cities. IEEE First International Workshop on Vehicular Traffic Management for Smart Cities, Ireland.

[5] Noor Afiza Mat Razali, Nuraini Shamsaimon, Khairul Khalil Ishak. Traffic flow prediction using machine learning and deep learning. Journal of Big Data volume 8, Article number: 152 (2021), Dec 04, 2021.

[6] Idriss Idrissi, Mostafa Azizi, Omar Moussaoui. IoT security with Deep Learning-based Intrusion Detection Systems. 4th ICDS , 21-23, October 2020.

[7] N. Nithya, B. S. Kumar and S. Suriya, "Smart Traffic Density and Emergency Signal Controller," 2022 International Conference on Power, Energy, Control and Transmission Systems (ICPECTS), Chennai, India, 2022, pp. 1-5, doi: 10.1109/ICPECTS56089.2022.10047520.

[8] Guangjie Han, Hao Wang, Mohsen Guizani, Sammy Chan. KCLP: A kMeans Cluster-Based Location Privacy Protection Scheme in WSNs for IoT. IEEE Wireless Communications, December 2018.

[9] W. Yue, C. Li, G. Mao, N. Cheng and D. Zhou, "Evolution of road traffic congestion control: A survey from perspective of sensing, communication, and

computation," in China Communications, vol. 18, no. 12, pp. 151-177, Dec. 2021, doi: 10.23919/JCC.2021.12.010.

[10] B. Roy, S. Patnaik and P. Dutta, "Congestion Detection Techniques in Road Network," 2021 Smart City Challenges & Outcomes for Urban Transformation (SCOUT), Bhubaneswar, India, 2021, pp. 252-255, doi: 10.1109/SCOUT54618.2021.00060.

[11] M. Akhtar, M. Raffeh, F. ul Zaman, A. Ramzan, S. Aslam and F. Usman, "Development of congestion level based dynamic traffic management system using IoT," 2020 International Conference on Electrical, Communication, and Computer Engineering (ICECCE), Istanbul, Turkey, 2020, pp. 1-6, doi: 10.1109/ICECCE49384.2020.9179375.

[12] X. Duan, J. Xu, Y. Chen and R. Jiang, "Analysis of influencing factors on urban traffic congestion and prediction of congestion time based on spatiotemporal big data," 2020 International Conference on Big Data, Artificial Intelligence and Internet of Things Engineering (ICBAIE), Fuzhou, China, 2020, pp. 75-78, doi: 10.1109/ICBAIE49996.2020.00022.

[13] D. Avramoni, A. Iovanovici, A. -M. Ilienescu and L. Prodan, "Short term traffic congestion prediction using publically available traffic data: a case study on Timisoara," 2022 IEEE 20th Jubilee World Symposium on Applied Machine Intelligence and Informatics (SAMI), Poprad, Slovakia, 2022, pp. 000121-000126, doi: 10.1109/SAMI54271.2022.9780813.

[14] K. Rajan and K. Sampath Kumar, "Optimization of Traffic Congestion in Smart Cities Using Residual Convolutional Neural Network," 2022 IEEE International Conference on Data Science and Information System (ICDSIS), Hassan, India, 2022, pp. 1-4, doi: 10.1109/ICDSIS55133.2022.9915860.

[15] N. Shukla, D. Garg, S. Singh and C. Upadhyaya, "Traffic Congestion Management using Camera and Geolocation," 2022 6th International Conference on Trends in Electronics and Informatics (ICOEI), Tirunelveli, India, 2022, pp. 68-73, doi: 10.1109/ICOEI53556.2022.9776886.

[16] Z. He, B. Ren and C. He, "Identification of influencing factors of urban traffic congestion based on ordered Logistic regression," 2022 7th International Conference on Intelligent Computing and Signal Processing (ICSP), Xi'an, China, 2022, pp. 914-918, doi: 10.1109/ICSP54964.2022.9778490.

[17] S. G. Rao, R. RamBabu, B. S. A. Kumar, V. Srinivas and P. V. Rao, "Detection of Traffic Congestion from Surveillance Videos using Machine Learning Techniques," 2022 Sixth International Conference on I-SMAC (IoT in Social, Mobile, Analytics and Cloud) (I-SMAC), Dharan, Nepal, 2022, pp. 572-579, doi: 10.1109/I-SMAC55078.2022.9987342.

[18] S. Neelakandan, M. A. Berlin, Sandesh Tripathi, V. Brindha Devi. IoTbasedtraffic prediction and traffic signal control system for smart city. ResearchGate,19 May 2021 .