International Journal of Grid and Distributed Computing
Vol. 14, No. 1, (2021), pp. 1022-1033

Finding an Optimal Route for Multimodal Transportation Using Machine Learning

Ms. Charvi Sanjay Suri[1], Dr. Avinash Agarwal[2], Mr. Ajit Dharmik[3]
[12]*Department of Computer Science & Engineering, Shri Ramdeobaba College of Engineering and Management, Nagpur, Maharashtra, India*
[3]*6Simplex Software Solutions Pvt. Ltd., Nagpur, Maharashtra, India*
surics@rknec.edu[1], agrawalaj@rknec.edu[2], ajitdharmik@6simplex.co.in[3]

Abstract

When it comes to travelling in a new city, the most important factors to consider are time, cost, and distance. The aim is to develop an application that will assist a user in getting from point A to point B for the least amount of money and time while avoiding or reducing emissions. Aside from that, the atmosphere and protection are taken into account. Machine learning is used to make this method an optimal solution. Various applications include weather forecasts for the region we are travelling through, but this app displays a dynamic weather forecast for the area. Checking the weather does not need any additional software. Furthermore, this application displays safe zones, those that are less prone to incidents and are successful, so that users can prevent obvious problems. Since traffic plays such an important role in travel, this app also informs you about it.

Keywords: *GIS, Multimodal integration, Machine Learning, A* Algorithm*

1. Introduction

Traveling to or in a city is always a challenge when a person is new to it. Therefore we need certain support to get to know about how to travel in the city without spending much money as well as time while traveling from one place to another in a city. In simple words, an optimal path with low cost and time is a basic requirement. In addition to that if a pollution-free path or a path with comparatively less pollution would be the cherry on the cake. To bring all three criteria together in an application, an A-star algorithm has been used. It is an algorithm to find out the optimal path between probable routes. There may be 3-4 paths, but this algorithm gives the best results.

Time, cost, and distance are the main objectives taken into consideration while addressing the issue of travel in a new city. The aim is to create an application to help a user to travel from source to destination at the lowest cost and time via pollution-free or with less pollution. In addition to that weather and safety is also taken into consideration. For making this system optimal, machine learning is used. Various applications support weather reports of the area we are travelling in, but this application shows the dynamic weather report of the area within itself. No extra application is required for checking the weather. In addition to that this application shows safe regions, the ones which are less prone to accidents and are active so that users can avoid getting into obvious troubles. Traffic plays an important role in travel; this application takes care of informing about it too. Django service is called using the python code which consist of the A* algorithm logic.

Transportation plays a vital role in the development of the social as well as the economic status of the city. Public Transport is the mass transportation of people from one place to another through Bus, Rail, and metro, with efficient speed, punctuality, frequency, facilities, comfort, convenience, and reliability. Public transport or public

transit is a shared passenger transport service that is available for use by the general public, as distinct from modes such as hired buses, metro rails, local trains which are not shared by strangers without private preparatory measures. Most of the public transport runs to a scheduled timetable with the most frequent services running to headway.

The need for convenient, reliable, and cost-effective regular commuting has led to an increase in the use of multimodal public transportation. There has been a lot of research into single-mode and single-criterion route selection based on travel time or cost. However, little has been written about a real-time integrated model for multimodal and multi-criteria route selection.

2. Literature Review

In 2013, Rafael Rodríguez-Puente and Manuel S Lazo-Cortés stated the following views on GIS and A* algorithm.

Since the 1980s and 1990s, the use of Geographic Information Systems has risen dramatically. We may mention shortest paths search as one of their most challenging applications. Several researches on shortest path search show that graphs can be used to achieve this goal. A Geographic Information System (GIS) is a computer system that can handle geo-referenced data in a practical sense. The information associated with geographic coordinates (longitude and latitude) is referred to as this type of data. The shortest path quest has been extensively researched. Many applications can be found in a variety of scientific fields, especially in GIS. Since the road networks used by GIS to respond to the above requests are typically broad, with thousands of streets, it's important to pay close attention to how such data is processed.

One of the most well-known shortest path search algorithms is Dijkstra's algorithm. The shortest path search in massive graphs is not well suited to this algorithm. This is why, in order to decrease the run time of shortest path search, several authors have suggested numerous modifications to Dijkstra's algorithm using heuristics. The A* algorithm is one of the most widely used heuristic algorithms. Its main aim is to minimize run time by reducing the search space. In reduced graphs, this article proposes a modification of Dijkstra's shortest path search algorithm. It demonstrates that the cost of the path discovered in this study is the same as the cost of the path discovered in the original graph using Dijkstra's algorithm. The results of finding the shortest path using the proposed algorithm, as well as Dijkstra's and A* algorithms, are compared. This comparison shows that using the proposed method, the optimal path can be found in a comparable or even faster time than using heuristic algorithms.

Several authors have suggested various modifications to Dijkstra's algorithm. Some of these algorithms use heuristics to minimize shortest path search run time, and they can be classified as follows:

1. without data preprocessing:

- A* (A-star) algorithm

- Bidirectional search

2. with data preprocessing:

- Reach-Based Pruning

- Landmark-A*

- Highway Hierarchies

- Edge flags

- Geometric containers

- Pre-computed Cluster Distances (PCD)

The A* algorithm is one of the most widely used heuristic algorithms. Its main aim is to minimize run time by narrowing the search space and evaluating only the vertices with the best chances of appearing in the shortest path. The heuristic function used to evaluate the order in which vertices are visited has an effect on the results obtained by this algorithm. The computational complexity is reduced to O if the chosen heuristic is optimal (n). The A* algorithm is commonly used for shortest path search because of this. [1]

In 2019, XIAO-HUAN LIU and his mates have proposed the best route selection algorithm to solve the problem of path planning for intelligent driving vehicles in the presence of restricted driving, accidents, and traffic congestion. They created a new method of best-path selection with length priority based on prior knowledge and used a reinforcement learning approach, as well as improved the search direction setting of the A* shortest path algorithm in the program. This best route planning algorithm can effectively assist various types of intelligent driving vehicles in selecting the best path in a traffic network with restricted height, distance, and weight, incidents, and traffic jams.

Path selection is critical for intelligent driving vehicles. This section will build a length-first best path algorithm based on a reinforcement learning technique known as OPABRL (Optimized Path Algorithm Based on Reinforcement Learning).

We must solve two problems when driving an intelligent vehicle: route planning and path selection. To make the system easier to understand, we'll go over the reinforcement learning approach based on previous experience and the upgrade rules first: Assuming that the intelligent vehicle is still travelling on a road of a certain distance, route planning can be thought of as the issue of planning one or more routes that can successfully reach the end point from the starting point while avoiding the road environment's obstacles. The first part of the problem can be solved using the shortest path algorithm.

We consider rasterizing the road network in the second section, and determining if it is an obstacle based on whether the grid is safe or not. The stumbling block consists primarily of three cases: traffic restrictions, collisions, and traffic congestion. The limits primarily apply to three conditions: height limit, width limit, and weight limit; traffic incidents are classified as mild, moderate, or high; and traffic congestion is classified as slight, slow, or heavy. The grid is called an obstacle if it appears in any of the above. When the machine determines that the safe grid is not an obstruction, it divides the situation into two categories: correct and incorrect judgment. The correct judgment is that the grid meets the path's conditions and can be used as a choice by the optimal path selection step; the incorrect judgment is that the grid is dangerous while the hazard is not identified and the vehicle passes through it. Because of the misidentification, the grid's overall cost would rise.

Some steps are taken to create a new topology, such as boundary learning, environment map generation, route preprocessing and planning, and best path selection and determination. In the process, some theorems are applied.

We propose a best route planning algorithm for intelligent driving vehicles that is suitable for traffic congestion, collisions, and temporary limits, based on the reinforcement learning algorithm of machine learning and the judgment of prior information, combined with the searching optimized A* algorithm. To convert the cost, the best path determination and virtual equivalence ratio are proposed. The proposed algorithm has obvious advantages in terms of path length and virtual equivalence ratio, and the algorithm itself has an advantage in terms of running time and search performance, according to simulations and actual tests. Given the rapid advancement of intelligent

driving vehicle technology, finding a robust cost path planning approach based on intelligent decision making is critical. [2]

In 2020 Fatemeh Salehi Rizi, Joerg Schloetterer and Michael Granitzer proposed that many graph algorithms and applications depend on computing shortest path distances between nodes. Traditional exact approaches, such as breadth-first-search (BFS), do not scale up to today's large networks, which are rapidly changing. As a result, approximation methods must be discovered in order to allow scalable graph processing at a significant speedup. In this paper, we use deep learning techniques to learn vector embedding to estimate the shortest path distances in large graphs. We demonstrate that a feedforward neural network fed with embedding can accurately estimate distances with a low distortion error.

The approach they followed consists of following steps explained in brief:

1. Distance Approximation

An un weighted undirected graph with n nodes and m edges is G = (V, E). Any node in a graph is given a real-valued vector embedding by graph embedding techniques. The target is to use a feed forward neural network to approximate the distance between two nodes with the actual shortest path distance.

We must extract training pairs from the entire graph G in order to train the neural network. We begin by selecting a small number of l nodes to serve as landmarks, where l<<n. The real shortest distances from each landmark to all of the remaining nodes are then computed using BFS. It produces a total of l (n - l) training pairs.

By applying a binary operation to the vector embedding, such as subtraction, concatenation, average, and pointwise multiplication, we construct a joint representation as input to the neural network. Vectors from the training set are eventually used as input for a feed forward neural network. The input vectors are converted to a real-valued distance by the neural network. An input layer, a secret layer, and an output layer make up our feed forward network. The size of the input layer is determined by the binary vector embedding operation.

For the first two layers, we used the rectified unit (ReLU) as the activation function. ReLU does not have the issue of gradient vanishing, and it has been demonstrated that deep networks can be trained effectively using ReLU. The output layer is a single unit of softplus, which is a smoother version of ReLU with a range of [0, 1] since the network performs regression. The Mean Squared Error (MSE), which calculates the sum of the squares of variance between the estimator and what is predicted, is used to evaluate the predictor's efficiency. We use Stochastic Gradient Descent (SGD) as an optimizer because it is typically fast and effective for large-scale learning.

2. Computational Complexity

The proposed approach has a runtime complexity that is linear. First, we learn vector embedding, which has a pre-computation time of O (n), where n is the graph's number of nodes. The landmark-based scheme is then used to reduce the number of shortest path computations needed to determine the ground truth. We only need to compute a BFS tree for each landmark if we choose a small, constant number of landmarks. The resulting values are adequate to make up the training set since they represent the shortest distances between all remaining nodes and these landmarks.

The resulting values are adequate to make up the training set since they represent the shortest distances between all remaining nodes and these landmarks. We have l (n - l) training pairs with l nodes as landmarks, where l<< n. It takes O (l (n + m) time to learn BFS on unweight sparse graphs, and it takes O (n + m) time to learn BFS on weight

sparse graphs. The benefit of using a graph embedding is that a feed forward neural network will answer a distance query between two nodes u, v in a short period of time, regardless of graph size, (i.e.) O (1) time. So it takes O to calculate the shortest path distances from a starting node u to all other nodes (n).

To today's vast graphs with millions of nodes and billions of edges, traditional methods for computing shortest path distances no longer scale. They propose a new method that approximates node distances by first embedding graphs into an embedding space, inspired by landmarked-based approaches. They fed their vectors into a feed forward neural network using two recent graph embedding techniques. The outcomes are outstanding. For the vast majority of node pairs, our approach yields the shortest distances, matching the most valid ground facts. And it does so efficiently, with results arriving in a linear time frame. [3]

In 2018 Prashant Thombre proposed that in the field of computer science, path finding is a well-studied subject. The discovery and plotting of an optimal route between two points on a plane is known as the path-finding problem. The current algorithms for solving this problem are largely static and depend heavily on prior environmental expertise. They also need a deterministic climate. However, in real-world path-finding applications, the environment is often unknown and stochastic, with many competing goals. In such instances, the algorithms described above fail to produce useful results. In this project, we investigate and implement Voting Q-Learning (VoQL), a model-free, on-policy learning algorithm, for solving the many-objective path-finding problem. The VoQL algorithm is used to evaluate a set of optimal policies in this project. For action-selection, this algorithm employs a variety of voting methods borrowed from the field of social choice theory. For the first time, the output of alternative voting methods is analyzed and evaluated in addition to operating with the current methods for VOQL.[4]

In 2020, Rachel Mundeli Murekatete & Takeshi Shirabe proposed one of the most basic functions of raster-based geographic information systems is the selection of suitable paths or sequences of cells from a grid of cells. It is commonly assumed that the optimality of a path can be evaluated by the sum of the weighted lengths of all its segments weighted, (i.e.) by the underlying cell values, in order for this function to work. However, if those values are calculated on a scale that does not allow for arithmetic operations, the validity of this statement must be challenged. This paper compares two models, the minisum and minimax path models, which aggregate the values of the cells associated with a path using the sum and maximum functions, respectively, through computational experiments with randomly generated artificial landscapes. The minisum path model appears to be successful if the path search can be translated into the traditional least-cost path problem, which aims to find a path between two terminuses on a ratio-scaled raster cost surface with the shortest cost-weighted range. The minimax path model, on the other hand, is found to be mathematically sound when cost values are calculated on an ordinal scale and functionally useful when the problem is concerned with the maximization of some desirable condition such as suitability rather than the minimization of cost.

The aim of this paper is to provide a theoretical account of why using the traditional least-cost path model can be problematic, as well as an alternative, and to compare their merits and limitations using a series of computational experiments with a variety of randomly generated hypothetical cost or suitability surfaces.

Evaluation and optimization of paths are done in the following steps stated below:

1. Shortest path problem

2. Variants of shortest path problem

3. Least-cost paths on raster cost surfaces

4. Most-suitable paths on raster suitability surfaces

This research has the practical consequence that various computational approaches to dealing with the complexity and subjectivity inherent in route planning exist. One approach, as described in the literature, is to try different models of a cost (or suitability) surface on each of which a least-cost (or most-suitable) path is obtained. [5]

3. Proposed Methodology

Finding the low pollution route of the desired destination from the current source at low cost and the minimum time is the main goal of this application. A graph data structure is used to solve this problem. Here, the edges are the route between two stops (i.e.) source and destination. For nodes, we have bus stops, bus stations; metro stations, auto stands, and cab pickup and drop points. If we consider the world to be a graph, then all path finding problems boil down to figuring out how to get from a starting node in a graph to a destination node. This can be accomplished by creating a graph search algorithm for the given problem that searches the graph from an arbitrary node and explores the adjacent vertices of the visited nodes until it reaches the destination node. The nodes are point data in the shape file created using Google Earth. [6] All the shape file data can be converted to geojson online anytime; we can see the specifications such as name of the location, Latitudes, longitudes, exact address of the location, etc.

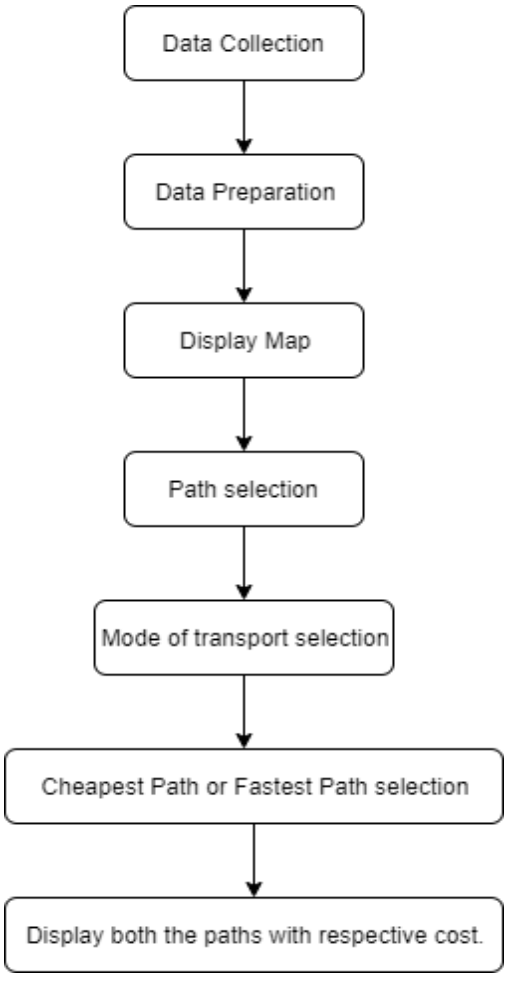

Figure 1. Flow of the Working of the Application

Path finding attempts to answer two key questions: how to find any path between two nodes in a graph and how to find any path between two nodes in a graph. Another query that a path finding problem can address is determining the most efficient route between the start and destination nodes. Most of the time, we're concerned with the above problem, in which we find an optimal path that avoids obstacles and minimizes deviations from the optimal path as much as possible before arriving at our destination. This is a more complicated and difficult problem to solve than finding a path between the two nodes since it is an optimization problem. Many algorithms are used to find out an optimal path like Dijkstra's Algorithm, Floyd War shall algorithm, Bellman-Ford algorithm, etc. In this application, we will be using the A* (A-star) algorithm. Finding the shortest path can be done by any of the above algorithms but finding the path with less pollution can be done properly with the A* algorithm. The heuristic value can be used as the pollution percentage of a certain bus stop, metro station, or auto stand which is considered as nodes here. Features like traffic, safety, and weather, which are dynamic to regular changes, are supported through an external API.

4. Implementation

4.1. Data Attribution

Data collection is done by using Google earth manually by pinpointing the location of all the pickup and drop off locations (i.e.) various bus stops, metro stations and auto stands. You can display high-resolution aerial and satellite images, photos, elevation terrain, road and street labels, and business listings from your screen, which transforms into a portal to anywhere. [6]

Figure 2. Google Earth Portal with Location Markers Which Helps To Make a Shape File

Geo-server will be used as a database for giving input to the application in the form of layers and output will be displayed on the web application. Geo-Server is a Java-based open source software server that lets users exchange and edits geospatial data. It publishes data from any big spatial data source using open standards, and it is designed for interoperability. Geo-Server is a community-driven project that is created, tested, and

funded by people and organizations from all over the world. Geo-Server is a high-performance certified compliant Web Map Service that implements the Open Geospatial Consortium's (OGC) Web Feature Service (WFS) and Web Coverage Service (WCS) standards (WMS). Geo-Server is an essential part of the Geospatial Web. [7]

4.2. Algorithms and Libraries

In computer science, the A* algorithm is one of the algorithms used to find the shortest path. It can be used to solve problems like the Traveling Salesman Problem, for example. It's a "best match" algorithm for determining which nodes are the most efficient for reaching the goal node from any given node. The A* algorithm falls into the category of admissible heuristic algorithms.

The distance is calculated using the equation

$f(n) = g(n) + h(n)$,

f (n): computational heuristic function,

g (n): cost of reaching to the current node from the starting node,

h (n): estimated distance to arrive at the target node from the current node.

The addition is used in the algorithm, which has a fairly simple structure. The node with the highest priority in the algorithm that uses a priority queue as a data structure is the node with the lowest value of f (n). In simple words the algorithm works in the given steps:

1. In the first step, the algorithm selects the lowest value (and thus the most important) node (that is, it goes to the next node) and removes it from the queue.

2. The values of all nodes that are adjacent to this node are modified (there is now a cost to come to this node, which is calculated using the function f (n)).

3. The algorithm repeats the steps above until the goal is met (i.e., until the target node is in the priority queue or until no node remains in the queue). [8]

OSMnx is a Python package that uses OpenStreetMap to import political/administrative boundary geometries, building footprints, and street networks. It allows researchers to consistently create, project, visualize, and analyze non-planar Complex Street networks by building a city's or neighborhoods walking, driving, or biking network with a single line of Python code, including node elevations and street grades. Python was chosen because it is widely used, beginner-friendly, strong, fast, free, and open-source. For rich network analytic capabilities, beautiful and easy visualizations, and quick spatial queries with R-tree indexing, OSMnx is developed on top of Python's NetworkX, matplotlib, and geopandas libraries. [9]

NetworkX is a Python package for creating, manipulating, and studying complex networks' structure, dynamics, and functions. [10] Matplotlib is a Python library that allows you to create static, animated, and interactive visualisations. Matplotlib is a plotting library for Python and NumPy, the Python numerical mathematics extension. It uses a general-purpose GUI toolkit to provide an object-oriented API for embedding plots into applications. [11] GeoPandas is a free open source project that makes it easier to work with geospatial data in Python. GeoPandas allows spatial operations on geometric forms by extending the datatypes used by pandas. Shapely is in charge of geometric operations. Geopandas also uses fiona to access files and matplotlib to plot data. [12]

4.3. Methodology

There are several ways to look at the multimodal network. It can be divided into four categories based on their physical characteristics: road, rail, water, and air. On the other side, it can be divided into private (e.g., walking, biking, and driving) and public

modes (e.g., public transportation) (e.g. bus, train, and metro). The functional view has the benefit of emphasizing the service provided to a traveler. Private networks, which include both physical nodes and physical connections, provide continuous service at any time. Public transportation networks, on the other hand, provide discrete services based on time tables, with physical nodes (such as stops and stations) visible but physical links usually hidden. As a result, the functional viewpoint is appropriate for modelling a multimodal transportation network. [13]

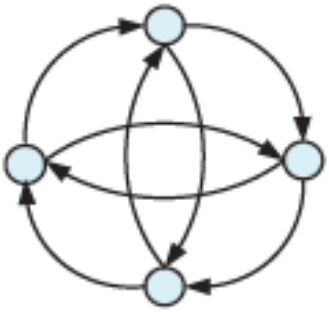

Figure. 3 Abstract fragment for private transport [13]

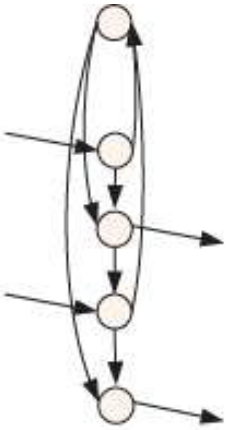

Figure 4. Abstract fragment for public transport. [13]

In the above figures the nodes are considered as the stops and edges are considered as roads or tracks to be travelled. In a super network, there are two types of nodes: physical nodes that represent positions which require X, Y coordinates, and event nodes that represent arrival and departure events at specific stops or stations. Event type, event time, and service-related factors (e.g. bus path, but travel, stop sequence) are required attributes for the above nodes. Distance, time, speed, monetary costs, pollution, efficiency, and general cost are all required or possible attributes for physical links.

It should be noted that there are two types of transfers: one within the same mode (e.g., transfer from one bus line to another) and the other between modes (e.g., transfer from bus to train), both of which are depicted in figure 3 by dashed lines. In mode transfers, the foot (pedestrian) network is critical: all transfer links are linked to this network because walking is often involved in such transfers. Adding transfer connections can be done in a number of ways. Each node in a layer is connected to the nearest neighbor node in the foot network, which is an easy solution.

To test the algorithm, we gathered public transportation and road network data for the Nagpur area. Car nodes, car connections, bike nodes, bike links, foot nodes, foot links, bus stops, bus time tables, metro stops, and metro timetables are among the information available. As a result, the modes of walking, biking, driving, taking the subway, and taking the bus have been chosen. We presumed that walking speeds are 4.7 km/h, cycling speeds are 11.5 km/h, and car speeds are limited to the maximum speed allowed on the road. A trip from Burdi Bus Stop to Nagpur's Yeshwantrao Chavan College of Engineering (YCCE) is considered.

5. Result

The tables shown below displays the time taken by and distance travelled by various combinations of the modes of transport and routes. The cost of public transports are under the local government for all of the locations, it does not vary in years, also timetable probably same for years. Walking is essential in any case of travel. There are private bikes and car rentals whose charges may vary according to time of the day. Here we have two tables to show results for both fastest as well as for shortest paths.

Table 1. Difference between time and distance of shortest and fastest path

Sr. No.	Modes	Shortest Path		Fastest Path	
		Time(hr)	Distance(km)	Time(hr)	Distance(km)
1.	Foot	2.965	14.827	3.392	16.958
2.	Bike	0.682	14.909	0.758	16.102
3.	Car	0.282	15.006	0.269	15.847
4.	Foot + Bus	0.437	19.913	0.408	21.876
5.	Foot + Metro	0.378	16.925	0.323	16.925
6.	Foot + Bus + Bike	0.484	20.280	0.423	21.382
7.	Bike + Metro + Foot	0.518	19.634	0.445	23.12
8.	Bike + Metro + Car	0.343	16.030	0.294	16.04

Using the above results the user will select the optimal path for him. There will be pictorial representation of these combinations. From Fig (4) to Fig (11) you can see example of the spatial view of the routes with the path. From the above table we come to a conclusion that, Walking is pretty essential part of the travel process. Travelling through car utilizes lowest time as once we get into car we have advantage of the speed as well as there is no waiting at the stops. Whereas metro can be listed as the fastest mode of transport in the city, even with the combination of foot and bike we get a pretty good lead.

The discussion to choose optimal path is in the hands of user. Its user choice completely what kind of mode of transport he chooses and also the time he spends at the bus and metro stops.

Figure 5. Foot Route

Figure 6. Bike Route

Figure 7. Car Route

Figure 8. Foot + Bus Route

Figure 9. Foot + Metro Route

Figure 10. Foot + Bus + Bike

Figure 11. Bike + Metro + Foot

Figure 12. Bike + Metro + Car

6. Conclusion

In this project, a GIS application has been created for estimating optimal cost and time for the given source and destination keeping in minds various other factors too such as pollution, safety, traffic and weather. This application is useful for new visitors and tourists in the city to find places using multiple modes of transport and not getting into any trouble while doing so.

For doing this Regression model is used as Django service with A* algorithm. There are still more advancements that can be made in this application such as we can use this for planning the whole India tour in low cost with hotel and dining costs included.

References

[1] "Algorithm for shortest path search in Geographic Information Systems by using reduced graphs", Rodríguez-Puente and Lazo-Cortés, SpringerPlus, 2013, DOI: 10.1186@2193-1801-2-291

[2] "A New Algorithm of the Best Path Selection Based on Machine Learning ", Xiao-Huan Liu , De-Gan Zhang , Hao-Ran Yan, Yu-Ya Cui , And Lu Chen, IEEE Xpress, 2019, DOI: 0.1109/ACCESS.2019.2939423

[3] "Shortest Path Distance Approximation using deep learning Techniques", Fatemeh Salehi Rizi, Joerg Schloetterer and Michael Granitzer, 2020, DOI: arXiv: 2002.05257v1

[4] "Multi-objective Path Finding Using Reinforcement Learning", Prashant Thombare, 2018, DOI:10.31979/etd.2ntb-4j8q

[5] "An experimental analysis of least-cost path models on ordinal-scaled raster surfaces", Rachel Mundeli Murekatete & Takeshi Shirabe, 2020, DOI: 10.1080/13658816.2020.1753204

[6] https://www.google.com/intl/en_in/earth/

[7] https://docs.geoserver.org/

[8] "Usage of the A* Algorithm to Find the Shortest Path in Transportation Systems", E.DERE and A.DURDU, 2018, International Conference on Advanced Technologies, Computer Engineering and Science

[9] Boeing, G. 2017. "OSMnx: New Methods for Acquiring, Constructing, Analyzing, and Visualizing Complex Street Networks." Computers, Environment and Urban Systems 65, 126-139, doi:10.1016/j.compenvurbsys.2017.05.004

[10] https://networkx.org/

[11] https://matplotlib.org/

[12] https://geopandas.org/docs.html

[13] "A multimodal transport network model for advanced traveler information systems", Jianwei Zhang, Feixiong Liao, Theo Arentze, Harry Timmermans,2011, DOI:10.1016/j.sbspro.2011.08.037