Logo

Description automatically generated with low confidence

Software Engineering Department

Ort Braude College

Capstone Project Phase B

Bus Routes Scheduler

Project Number: 22-2-D-3

Supervisor:

Zeev Frenkel

Wafa Ghanem 315842427

[Wafagha3@gmail.com](mailto:Wafagha3@gmail.com)

Leran Dagash 315668392

[Leran158@gmail.com](mailto:Leran158@gmail.com)

# Table of Contents

Contents

[Table of Contents 2](#_Toc129165890)

[Abstract 4](#_Toc129165891)

[1. Introduction 5](#_Toc129165892)

[1.1 The problem 5](#_Toc129165893)

[1.2 Purpose 5](#_Toc129165894)

[1.3 Proposed solution 5](#_Toc129165895)

[2. Background and Related Work 6](#_Toc129165896)

[2.1 Graph Traversal 6](#_Toc129165897)

[2.2 Shortest Path Algorithms 6](#_Toc129165898)

[2.3 Neural Networks 7](#_Toc129165899)

[2.3.1 Perceptron 7](#_Toc129165900)

[2.3.2 non-Linearity 7](#_Toc129165901)

[2.3.3 Loss-function 8](#_Toc129165902)

[2.3.4 Cross Entropy Loss 8](#_Toc129165903)

[2.3.5 Backpropagation 9](#_Toc129165904)

[3. The Machine Learning Model 10](#_Toc129165905)

[3.1 The Model 10](#_Toc129165906)

[3.2 data generation 10](#_Toc129165907)

[3.3 The Machine Learning Model 12](#_Toc129165908)

[3.3.1 Fully Connected Layers 12](#_Toc129165909)

[3.3.2 Activation function 13](#_Toc129165910)

[3.3.3 SoftMax: 13](#_Toc129165911)

[3.3.4 Dropout: 13](#_Toc129165912)

[3.4 Evaluating the Model 14](#_Toc129165913)

[3.4.1 Destination arrival rate 14](#_Toc129165914)

[3.4.2 Path weight sum 14](#_Toc129165915)

[3.4.3 Path edge count 15](#_Toc129165916)

[4. Research / Engineering Process 15](#_Toc129165917)

[4.1 Process 15](#_Toc129165918)

[4.2 Product 17](#_Toc129165919)

[4.2.1 Algorithm for optimization 17](#_Toc129165920)

[4.2.3 GUI 18](#_Toc129165921)

[4.2.4 Class diagram 19](#_Toc129165922)

[4.2.5 Use case diagram 20](#_Toc129165923)

[4.2.6 Activity diagram 21](#_Toc129165924)

[5. Evaluation/Verification 22](#_Toc129165925)

[6. Results and analysis 23](#_Toc129165926)

[6.1 Training results 23](#_Toc129165927)

[6.2 Testing results 23](#_Toc129165928)

[6.3 Results 24](#_Toc129165929)

[7. Conclusion and further work 25](#_Toc129165930)

[8. User guide: 26](#_Toc129165931)

[8.1 Packages used: 26](#_Toc129165932)

[8.2 Training the model & statistics. 26](#_Toc129165933)

[8.3 Using the model. 26](#_Toc129165934)

[8.4 Using anvil server. 27](#_Toc129165935)

[9. References 27](#_Toc129165936)

# Abstract

Public transportation, especially bus routing and scheduling has been a huge problem for a long time. The problem with public transportation is that it is not so reliable. It is rarely precise regarding timing and sometimes forces people to take multiple buses for them to reach their destination. This caused many to abandon this method of transportation and move to self-transportation methods. Most bus routes are based on experience and common knowledge. Efficiency of the bus routes network is affected by many factors that make it hard for scoring and optimization. In our project we deal with this problem by machine learning approach. With this project, we now have a pilot program for bus routes optimization. For simplicity, we deal only with simulated data: network of roads, dependence of traffic on time, distribution of passenger’s needs, preferences, and costs. Efficiency of the solution is scored based on the criterion that includes average time of transportation, considers preferences, and includes the total cost of the bus routes network maintenance.

# 1. Introduction

## 1.1 The problem

Bus transportation use in Israel is decreasing due to its unprecise timing and bad routing. Users who plan to go from point A to point B in a specific time do not get there in time or must change multiple buses to get to their destination. Since buses also share roads with cars, this tends to reduce the efficiency of bus travel, unless given a dedicated corridor like trains. Road congestion leads to frequent stopping, long journey times and delays in passenger pick-up and drop-off. Other issues include limited off-peak service, poor-quality shelters, and unreliable timetables. The Bus transportation system is still often based on the old techniques of planning. Therefore, it is often not suitable for today's requirements and users' needs. This is leading passengers to use their own transportation methods which is worsening traffic and increasing energy consumption and CO2 emission. Solving this problem would improve the quality of life overall and make the world a better place.

## 1.2 Purpose

The bus transportation system has been in use since a very long time and is based on “traditional routes”, sophisticated heuristics and not systematic surveys and personal appeals. With the technology evolution this planning method had a hard time keeping up, until recent years bus transportation companies started to use automated control of public transportation use (e.g., by sensors and magnet bus passes or by smartphone applications). These new technologies led to a whole new level of data gathering. Usage of machine learning based models for this big data should help to plan bus routes and schedules more efficiently. Our basic idea is to adapt for this problem some algorithms developed in the logistics field. They are using it to plan and schedule delivery routes for a fleet of drivers. This problem is very similar to the bus transportation problem. The difference between those two problems is that deliveries are delivered to their own houses, but people are delivered to bus stops. We can overcome this problem in the implementation which will be explained in the next chapter.

## 1.3 Proposed solution

We propose a machine learning based method that, upon receiving a map of the requested area, traffic data, and approximate number of passengers, optimizes routes for the buses. In our criterion we optimize tradeoff between minimizing of traveling time, minimizing the total cost and maximizing the number of users (passengers). We ensure the bus visits the original stops assigned to it.

We built a program in python that implements a MXnet machine learning model, we generated graphs using netwrokx, we preprocessed the graphs using our algorithm and Dijkstra algorithm, the code is written in google collab environment ([training code](https://colab.research.google.com/drive/194U7AcD6IvT0NEOytC-a_laPGWAL72zt) / [GUI code](https://colab.research.google.com/drive/1xjtO5_ASqfp9Q3la3t7lvO9_dXA6dbSg)).

Training code:

<https://colab.research.google.com/drive/194U7AcD6IvT0NEOytC-a_laPGWAL72zt>

Model with anvil:

<https://colab.research.google.com/drive/1xjtO5_ASqfp9Q3la3t7lvO9_dXA6dbSg?userstoinvite=leran158%40gmail.com&actionButton=1#scrollTo=MidrAlv9wycF>

# 2. Background and Related Work

## 2.1 Graph Traversal

Route planning can be categorized into a bigger top topic, graph traversal. Graph traversal has been well studied in the last 50 years. In computer science, graph traversal is the process of visiting each node in a graph. Based on the order of visiting nodes, graph traversal can be divided into two main categories, depth-first search [1], and breadth-first search [2]. Both have been applied to solve many important problems. Best-first search is also an important traversal strategy. Best-first algorithms traverse a graph by following the most “promising” nodes according to specific rules. Those rules are often called heuristics [3]. Heuristics are commonly used to speed up the search algorithms. Heuristics are also applied when the original algorithms fail to find the optimal solutions or when it is not necessary to find the best ones. A heuristic finds an approximate solution that is good enough yet much faster. Heuristics guide the search by trying to be” smart” about how close a node is away from the destination [4]. “Closer” nodes are explored first. It is rare that people drive aimlessly, meaning they usually have a destination. Meanwhile, people want to arrive at their destinations fast. This makes shortest path algorithms useful not only in real life but also the center of this work. Online map service providers such as Google Map and Bing Map rely on shortest path algorithms [5]. In this study, we use the output of shortest path algorithms as the ground truth when training a neural network and the benchmark for evaluation. Heuristics guided shortest path algorithms will be extensively discussed and applied to guide the training of the neural network.

## 2.2 Shortest Path Algorithms

Shortest path algorithms aim to find a path between a pair of nodes such that the sum of edge weights is minimized [6]. Floyd-Warshall, Bellman-Ford, Dijkstra’s, and A\* search are the most popular algorithms for finding the shortest paths. Floyd-Warshall algorithm [7] finds the shortest paths for all pairs of nodes in a graph. This is often overkill and suffers from the high computational cost *(Θ (V 3)* where *V* is the number of nodes). Bellman-Ford algorithm [8] is a single source shortest path algorithm that can handle negative edge weights with time complexity of *Θ (V E)* where E is the number of edges. This is a big advantage in many scenarios. However, in this study, we limit the scope in the case where the edge weights of road networks are positive real numbers. Moreover, the implementation of Bellman-Ford shares much resemblance with Dijkstra’s. It is unnecessary to extensively explore this algorithm and it is reasonable to focus on a smaller set of algorithms. Dijkstra’s algorithm finds shortest paths from one source to all other nodes. It is unable to handle negative edge weights, and this is acceptable in this study. Dijkstra’s algorithm has the time complexity of *O (E + V log V )*. A\* search algorithm is one of the best-first search algorithms. It has even been more widely used than Dijkstra in many scenarios due to its performance and accuracy. As an extension of Dijkstra’s algorithm, A\* search uses customized heuristics to guide the search. In theory, A\* search has better time complexity *(O(E))* than Dijkstra’s. The goal of Dijkstra’ s and A\* search algorithms is similar mathematically, *f(n) = g(n) + h(n)* where n is the current node that is being explored, *g(n)* is the total weights of the path from the starting point to node n, and *h(n)* is a customized heuristic graph traversal function that estimates the cost from n to the destination. Therefore, Dijkstra’s and A\* search try to find a path that minimizes *f(n)*. For Dijkstra’s algorithm, *h(n)* is always zero, meaning it does not try to be smart, hence less efficiency. It can be expected that the implementation of the two algorithms is based on the same structure, as shown in appendix A. The use of heuristics is problem-specific and might affect the convergence of the algorithm. To guarantee the convergence of A\* search algorithm, meaning always find a path, the heuristic calculation should be admissible [9]. A heuristic being admissible simply means it never overestimates the distance to the destination. If *h∗(n)* denotes the optimal cost to the destination from n, then *h(n)* is admissible if: *∀n, h(n) ≤ h∗(n)*.

## 2.3 Neural Networks

Neural network is a computational model dating back to the late 1950s, which was roughly inspired by biological neural systems. Deep neural networks have been applied in various fields not only those that were difficult to solve with traditional machine learning algorithms, image classification for instance but also relatively new fields such as bioinformatics and language modeling. The earliest implementation of neural networks had a different name, perceptron.

### 2.3.1 Perceptron

The perceptron was invented by Frank Rosenblatt in 1957 [10] as a simplified imitation of a biological neuron. Mathematically speaking, the perceptron algorithm approximated a binary classifier. Stated differently, it is a function that maps an input vector *x* to a binary output *y* such as the following: *y* = *f(x)* = 1 if *w · x + b* > 0, 0 otherwise where *w* is the vector of real-valued weights and *b* is the bias. This computation is essentially an affine transformation.

Diagram

Description automatically generated

Figure 1: An illustration of Perceptron.

Stated in geometric terms, *f(x)* is a linear decision boundary for its input vectors. However, if the training data is not linearly separable, then this algorithm is not guaranteed to converge.

### 2.3.2 non-Linearity

The non-linearity is introduced by making the output of each neuron nonlinear. This requires non-linear functions. These non-linear functions are often referred to as activation functions. However, there are certain traits of a non-linear activation function that are preferable when it comes to the training of a neural network. For instance, a non-linear function is preferred when it is continuous and differentiable [11]. The most widely used non-linear functions are hyperbolic tangent function (i.e., tanh), rectified linear unit (i.e., ReLU [12] and sigmoid function (i.e., logistic function)). The mathematical forms of the functions are the following:

Text, letter

Description automatically generated

Modern neural networks tend to have many layers. ResNet is proposed by K. He et al [13] that has a depth of up to 152 layers. This is eight times deeper than VGG nets [14]. Certain properties of non-linear functions such as sigmoid and hyperbolic tangent function make it impossible to be applied in modern neural network training. The numerical stability of ReLU makes it the dominant choice nowadays.

### 2.3.3 Loss-function

Neural networks can be applied as a supervised learning algorithm. The supervised training process requires both input data, denoted as *X* along with the ground truth (i.e., labels) that is denoted as *y*. The output of the neural network will be compared to the ground truth. In machine learning, it is preferred to know how wrong an algorithm is instead of measuring how correct it is. Even though these two are usually mutually derivable. How “wrong” the output is calculated by a certain loss function or cost function, which is the target function to be minimized.

*L(θ) = L (y, yˆ; θ)*

*θ* represents the parameters of target function, *y* is the true label and *yˆ* is the output. Mean Squared Error (abbr. MSE) and Cross Entropy (abbr. CE) are the most widely used loss functions.

### 2.3.4 Cross Entropy Loss

Cross Entropy Loss is commonly used in classification problems, which is computed as the following:

A picture containing text, clock, watch

Description automatically generated

In the context of information theory, cross entropy describes the difference between two probability distributions. When solving a classification problem, the output of the neural network is a probability distribution regarding all the classes presented in the training data. Each component within this distribution represents the confidence level of the neural network regarding the corresponding class. The goal of training a neural network classifier is to make the model as confident as possible for the true labels.

### 2.3.5 Backpropagation

The training of a neural network is usually a feed forward process, meaning the computation flow goes from input to output, one end to another, as shown in figure 2. There is no loop inside the architecture. When a feedback loop is formed, the neural network architecture is often referred to as Recurrent Neural Network [11].

Diagram

Description automatically generated

Figure 2: The Architecture of a fully connected feedforward network.

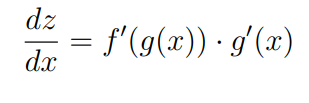
Intrinsically, the training of neural networks aims to find a set of neuron weights so that the corresponding loss functions are minimized. The loss, i.e., the error that describes how bad the network is doing, will be computed once the control flow reaches the end of the flow. An effective tool to numerically reflect how much the weights should be adjusted is backpropagation. As a short form of” backward propagation of error”, backpropagation was invented in the 1970s. It was until 1986 when Rumelhart, Hinton, and Williams [15] pointed out its importance regarding the training of neural networks, i.e., updating the weights of each neuron in a network. In the paper, D. Rumelhart et al. described how backpropagation could be applied to train neural networks not only more effectively but also more efficiently. The application of backpropagation made neural nets useful for solving previously insoluble problems. The backpropagation algorithm has been the core of training in neural networks nowadays. The feed forward process and backpropagation combined can be concluded as the following:

1. Take the input and go through the network until the output layer.

2. Compute the loss based on the loss function.

3. From the output layer backwards, adjust the weights of neurons on each intermediate layer until the first layer.

The effectiveness of backpropagation has been proven by many studies and experiments [16, 17, 18, 19]. There are two main handy tools available, partial derivative and chain rule in calculus to enable back propagation • partial derivative: for a multivariable function *z = f (x1, x2, …, xN)*, the partial derivative of each component, is denoted as:

A picture containing text, clock, watch

Description automatically generated

chain rule: define *z = f(g(x))*, the derivative of z regarding x is:

A picture containing graphical user interface

Description automatically generated

These two tools are essential to the training neural network, specifically to backpropagation algorithms. The way neural networks work is simple. Each neuron does an even simpler job. It takes the output of the neurons from the last layer and does an affine transformation as discussed previously. The value then is transformed by a non-linear function. The transformed value simply becomes the input of the next layer. This makes a neural network a giant composite function with all the weights and biases of its neurons as its variables. Partial derivative is needed to handle multivariate functions. Chain rule is crucial because of the nature of Neural Networks. Since a multilayer neural network is virtually a non-linear function, a one-layer neural network with non-linear activation approximates a non-linear function too. Thus, a multi-layer network is chained by many non-linear functions, i.e., each layer of the network. The earliest layer, as a non-linear function, is embedded the deepest, as shown in the formula:

A picture containing chart

Description automatically generated

with fi denotes the function that layer i computes. Clearly, the change of weights on early layers tends to have more impact than later layers. To propagate loss to the earliest layers, chain rule is essential since it takes all the adjustment that has been made on later layers and provides that information to early layers.

# 3. The Model

## 3.1 The Model

We model the network of the roads by a graph, the nodes correspond to bus stops and the edges corresponds to connecting roads. We characterize each bus stop by the intensity of visiting region near this bus stop (demand of the place), we give the weight of each edge according to the characteristics in the ends of the edge.

## 3.2 Data generation

The random graph is generated with some practical constraints such as each node has maximum 4 out neighbors and the edge weights are the multiplication of nodes Euclidean distance and the value given for the number of costumers. An example of such graph can be seen from figure 3. The graph can be generated as either directed or undirected (Fig.3).

A picture containing light, dog, red, palm

Description automatically generated

Figure 3: A random graph has 20 nodes.

The generated random graphs have no geographic information provided for the model and indeed there is no need for such information. To better approximate a road network and to better guide the search of the neural network model, certain assumptions must be made to the graph.

* Nodes are randomly given (x, y) Euclidean coordinates.
* Nodes are more likely to be connected when they are geographically closer.
* Each node has maximum 4 and minimum 2 out neighbors.
* Weights of edges are the multiplication of nodes Euclidean distance and the value given for the number of costumers.

The first two assumptions are very intuitive because the nodes that are geographically far away tend to use other intermediate nodes as hops to reach each other. For the third assumption, a driver has a limited number of choices when standing at a crossroad, as shown in Figure 4. More complex cases such as roundabouts might emerge. It is not much more complicated and can be adapted correctly. The last assumption might be confusing at the first glance. Firstly, it usually takes longer to travel through a long road segment. However, if such a road has a higher speed limit, it might not be the case. In urban or rural areas, the speed limit does not vary drastically.

Shape, arrow

Description automatically generated

Figure 4: An example shows the out degrees of a crossroad in a road network.

**Example:** Two graphs can be seen in Figure 5a and 5b. An observation is that the random graph with 1000 nodes presents certain small-world traits. Stated differently, if we treat this graph as the road map of a city or a province, there are communities (e.g., different zones of a city or different cities of a province) of nodes clustering together. These communities relate to each other by a few “important” edges. Those important edges are main roads of a city and highways between cities. As will be discussed shortly, the importance of roads can be described with centrality coefficients.

Chart

Description automatically generated with low confidence

(a) A random graph has 100 nodes. (b) A random graph has 1000 nodes.

Figure 5: Two examples show the output of random graph generation procedure.

Using this method, we generate data for the machine learning model to work with, after generating the data using this method, we feed it into our model to learn from it, the model learns how the demand of the stops has a role in building the optimal path.

## 3.3 The Machine Learning Model

The model is a fully connected neural network with three hidden layers. The first layer has 2048 neurons, the second layer has 1024 neurons, and the final layer has several neurons equal to the maximum degree of the input graph.

The activation function used in each hidden layer is ReLU, which is a common choice in deep learning. Dropout is also applied after each hidden layer, which randomly drops out a certain percentage of neurons to prevent overfitting.

The output layer uses SoftMax activation and is trained using cross-entropy loss. The model is built as a MXNet module and can be run on a GPU specified by the "context" parameter.

### 3.3.1 Fully Connected Layers

A fully connected layer, also known as a dense layer, is a type of neural network layer where every neuron in the layer is connected to every neuron in the previous layer. In other words, each neuron in the current layer receives input from every neuron in the previous layer.

The purpose of using fully connected layers in a neural network is to learn non-linear relationships between the input features. By connecting every input feature to every neuron in the next layer, the model can capture complex patterns and interactions between the features.

In the provided code, the model has three fully connected layers, each with a different number of neurons. The first layer has 2048 neurons, the second has 1024 neurons, and the final layer has several neurons equal to the maximum degree of the input graph.

Each layer applies an activation function (ReLU) to the output of the fully connected layer. This activation function introduces non-linearity into the model, making it possible to learn non-linear relationships between the input features and the output.

The purpose of using multiple fully connected layers with non-linear activation functions is to create a deep neural network that can learn complex, hierarchical representations of the input data. By stacking these layers, the model can learn increasingly abstract features at each layer, allowing it to make better predictions on complex tasks.

### 3.3.2 Activation function

Non-linear activation functions are mathematical functions applied to the output of each neuron in a neural network layer. They introduce non-linearity into the model, allowing it to learn and represent non-linear relationships between the input features and the output.

In contrast, a linear activation function would result in a linear transformation of the input data, making it difficult for the model to learn complex relationships and patterns in the data.

The most common non-linear activation functions used in deep learning models are:

Rectified Linear Unit (ReLU): ReLU sets negative values to zero and keeps positive values as is. This function is simple and computationally efficient and has been shown to work well in many deep learning models.

### 3.3.3 SoftMax:

The SoftMax layer is used in the output layer of a neural network for multi-class classification problems, producing a probability distribution over the classes. Non-linear activation functions enable the model to learn complex relationships and propagate errors during training. Cross-entropy loss is a commonly used loss function for multi-class classification, measuring the difference between predicted and true probability distributions. The SoftMax layer and cross-entropy loss are used in the provided code to train the model by adjusting its parameters using backpropagation and an optimization algorithm.

### 3.3.4 Dropout:

Dropout is a regularization technique used in deep learning to prevent overfitting, which occurs when the model learns the training data too well and does not generalize well to new data. The dropout layer is a type of layer that implements this technique.

During training, the dropout layer randomly sets a fraction of the outputs of the previous layer to zero, effectively "dropping out" those neurons. The fraction of neurons to be dropped out is a hyperparameter that can be tuned to achieve the desired level of regularization.

By dropping out some of the neurons, the model is forced to learn redundant representations of the data, which can improve its ability to generalize to new, unseen data. This is because the model cannot rely too heavily on any one neuron or set of neurons and must learn to distribute the information across multiple neurons.

During inference (i.e., when making predictions on new data), the dropout layer is usually turned off, and all neurons are used to make the prediction.

In the provided code, two dropout layers are used after the first and second fully connected layers, respectively. This helps to prevent overfitting and improve the generalization performance of the model on new data.

## 3.4 Evaluating the Model

The focus of this study is pathfinding using a neural network model. We need methods to evaluate both the effectiveness of the model and the quality of generated paths. We used cross entropy loss to train the model, therefore, one can argue that the model can also be evaluated with accuracy. The model accuracy in this work is interpreted as how likely a model makes the correct prediction on the next node. For greedy strategy, this metric is enough since it only cares about the next node. Higher accuracy means the model is learning this strategy better. However, for shortest path strategy, being able to correctly infer the next node does not directly mean that the model is learning the strategy well. Therefore, we will propose our own evaluation metrics. When it comes to comparing the neural network-generated path with the optimal path, two aspects are of great interest:

**1. Effectiveness:** can the model find a path? What is the probability that the model fails to find a path from A to B in a graph?

**2. Path quality:** are the paths generated by the model good? The path can have either fewer steps or smaller total edge weights.

### 3.4.1 Destination arrival rate

The effectiveness of a model indicates whether it can do its job. After the model is trained on a graph or a road network, it is interesting to know whether it can work. The method is direct: sample a pair of nodes and ask the model to give a path between them. If the model has a high chance to give a path to the queries, we can treat it as a functional model. The arrival

rate can be defined as the following:

Higher arrival rate means higher model effectiveness.

### 3.4.2 Path weight sum

People care about how fast they can reach their destinations. It is meaningful to know if the paths found by the model are good-enough, i.e., the path quality. Edge weight can be treated as the time spent to travel along a road. If we define the shortest path (i.e., benchmark) as weight by Dijkstra and the path taken by the model as weight by model. The time efficiency of the model is:

Smaller total edge weights indicate better path quality.

### 3.4.3 Path edge count

It might be annoying for a driver to frequently change to different roads when traveling. A lot of people would prefer traveling on as less roads as possible to reach their destinations. Therefore, it makes sense to evaluate how many edges the model must take to find the destination. We can define edge efficiency as the following:

Less edges count indicates smoother paths.

**Overall efficiency**

The most important factor is arrival rate therefore it got the highest weight, after that comes route quality and then the least important which is edge efficiency.

# 4. Research / Engineering Process

## 4.1 Process

After a wide search, we found out that the machine learning approach is slowly entering the public transportation field. This approach is being used in a pilot project based in Dubai from 2020, it is also being used in campus transportations in a few universities around the world. We assume that this approach is not fully usable yet due to the lack of data about the passengers' trips and the lack of available resources to run such models. Another factor is the problem itself; this problem requires a lot of heuristics such as traffic, number of passengers, passengers’ current station, passenger destination, weather, holidays, time of the year and much more, therefore making the calculating of the wights of the edges challenging, hence in this project we will take only some of these heuristics and we do not expect to get a fully accurate model.

To apply supervised learning algorithms, people first need to define the problems as either regression or classification. In our project we use the paths generated by shortest path algorithms as our training labels. Shortest path algorithms aim to find paths that minimize the total edge weights. It is reasonable to train neural networks aiming to minimize the sum of edge weights.

• **regression**: the target is to minimize sum of weights directly; therefore, mean squared error is opted.

• **classification**: cross entropy loss is usually preferred in this case; the model performance is indicated by the accuracy.

In our project, the problem is solved as classification for the following reasons:

**1. interpretability**. Mean squared error outputs a real number. In our case, it is not self-evident based on the output of how a model is performing. Whereas for a classification problem, a direct indicator is accuracy, which is directly interpretable. As will be shown, it takes more engineering effort to generate paths if we go with regression. From a programmer’s perspective, the output of the neural network model immediately indicates which edge should be picked when it is classification. It is not the case if the problem is treated as regression.

**2. consistency.** Initial step taken did show that regression was feasible. However, the later steps showed that classification was a better choice. Therefore, to reduce the engineering work and make the work more consistent, we treated it as a classification problem.

**After defining the problem, we can now proceed to generate the specific data for this problem, our idea is to solve the graph based on the bus stops demand, therefore we generate the specific data for this problem which is a graph with different bus stop demands, we then solve them using dijkstra algorithm for optimal paths , we then the proceed to train the model on this data , once we have a model trained we can now use it to predict future routes to solve the problem before it even occurs.**

## 4.2 Product

### 4.2.1 Algorithm for optimization

1. Generate a random graph that demonstrates a map of bus stops, for each node in the graph we generate a random number of passengers waiting at that node, for each passenger we randomly pick a destination node.
2. For each edge we calculate the weight using 2 parameters:

2.1 Check the number of passengers on both nodes connected by the edge that has the destination set to be the other node, for example edge (x, y) check how many passengers in x node and how many passengers in y node. More passengers result in lower edge weight.

2.2 Euclidean distance represents travel time.

2.3 *Edge weights formula =*

1. Now after the graph has been prepared, calculate the lightest path using the lightest path algorithm (Dijkstra) from random starting node A to random destination node B.
2. Add the path result to a dictionary.
3. Go back to step 3 until reaching the number of paths wanted.
4. Repeat steps 2-5 until we have sufficient graph data to train and evaluate our model.
5. Feed the graphs generated to the machine learning model and evaluate the model using the optimal paths found using the lightest path algorithm.
6. After reaching the desired accuracy, save the model to use it in the application.
7. Use the model in the application to predict new routes by giving the required data.

### 4.2.3 GUI

Our application starts from Startup window (Fig. 6). The user enters the source and destination of each route (numbers between 0 and 49) then clicks calculate routes. The user will be transferred to the next window (Fig. 7), the user now can click on the desired route to see the route calculated (Fig. 8). The user may click on the back button to calculate different 5 routes.

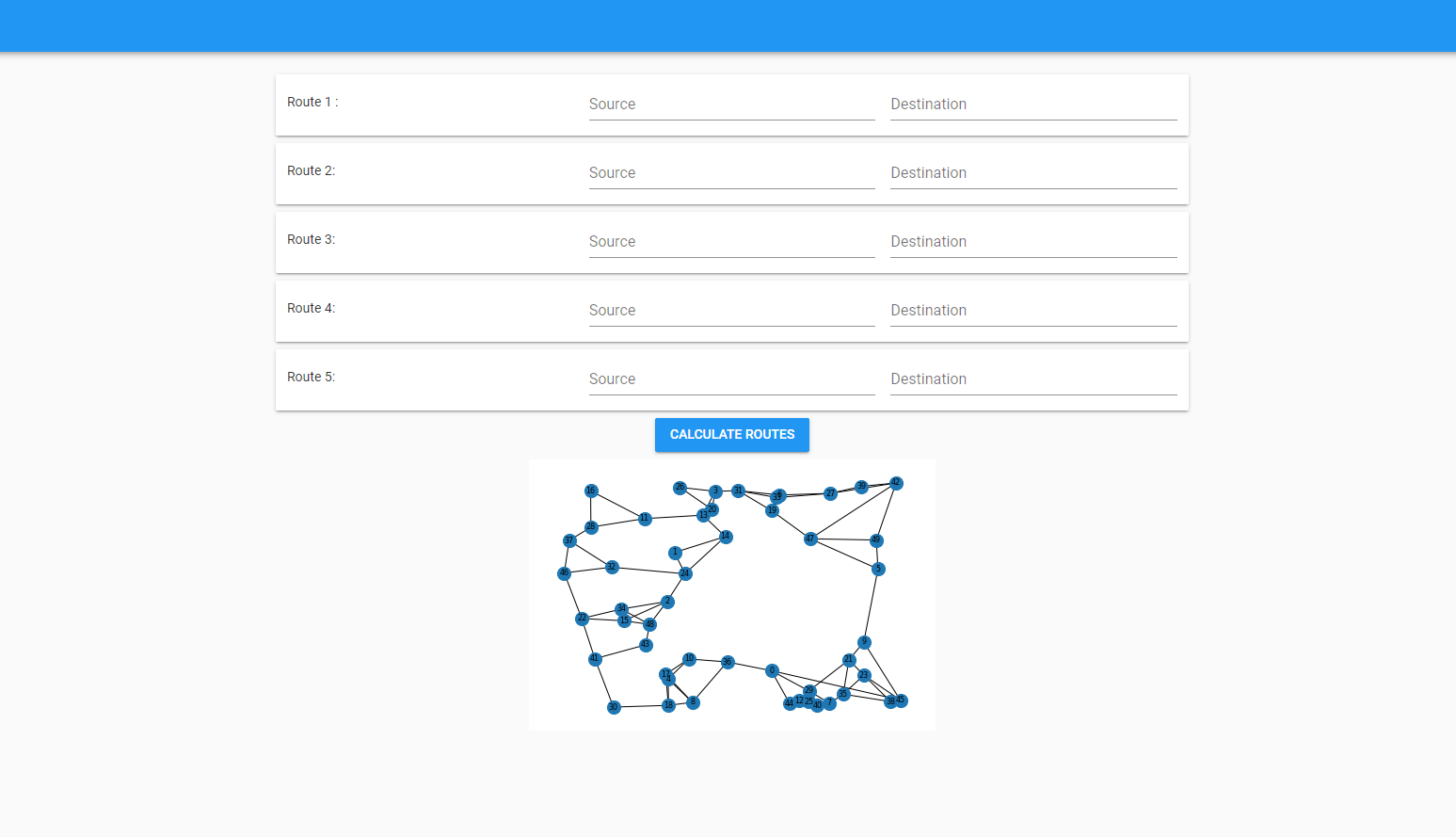


Fig. 6. Startup and Planning window

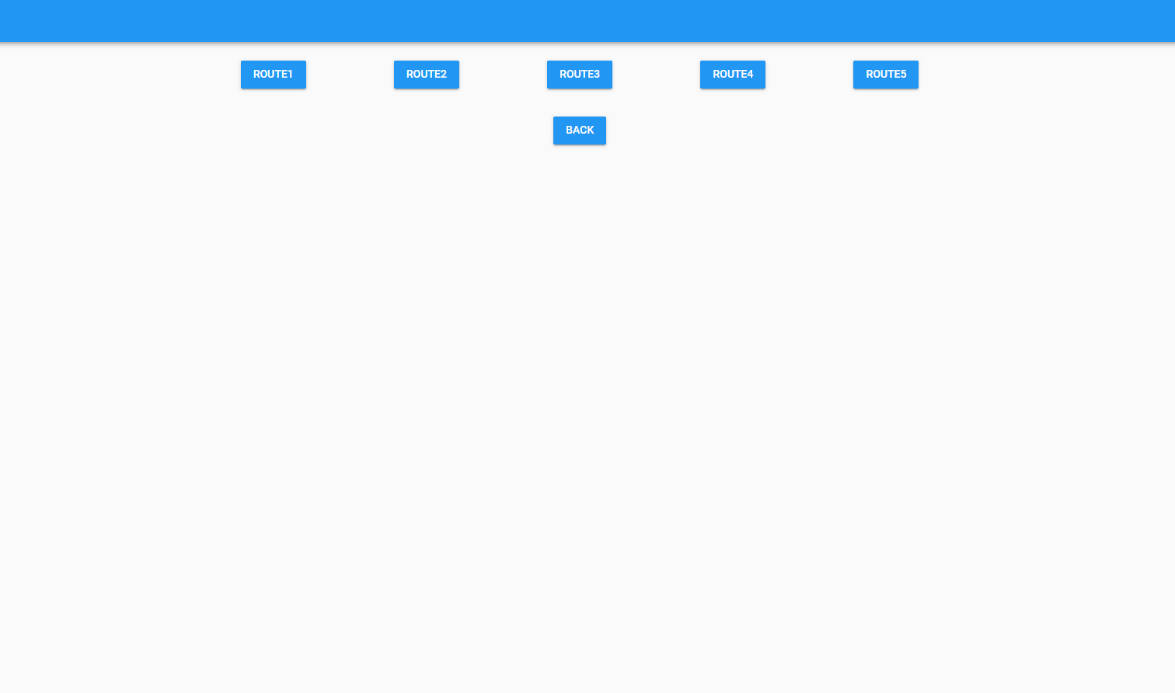


Fig. 7. The user can click on the route desired to see the calculated route.

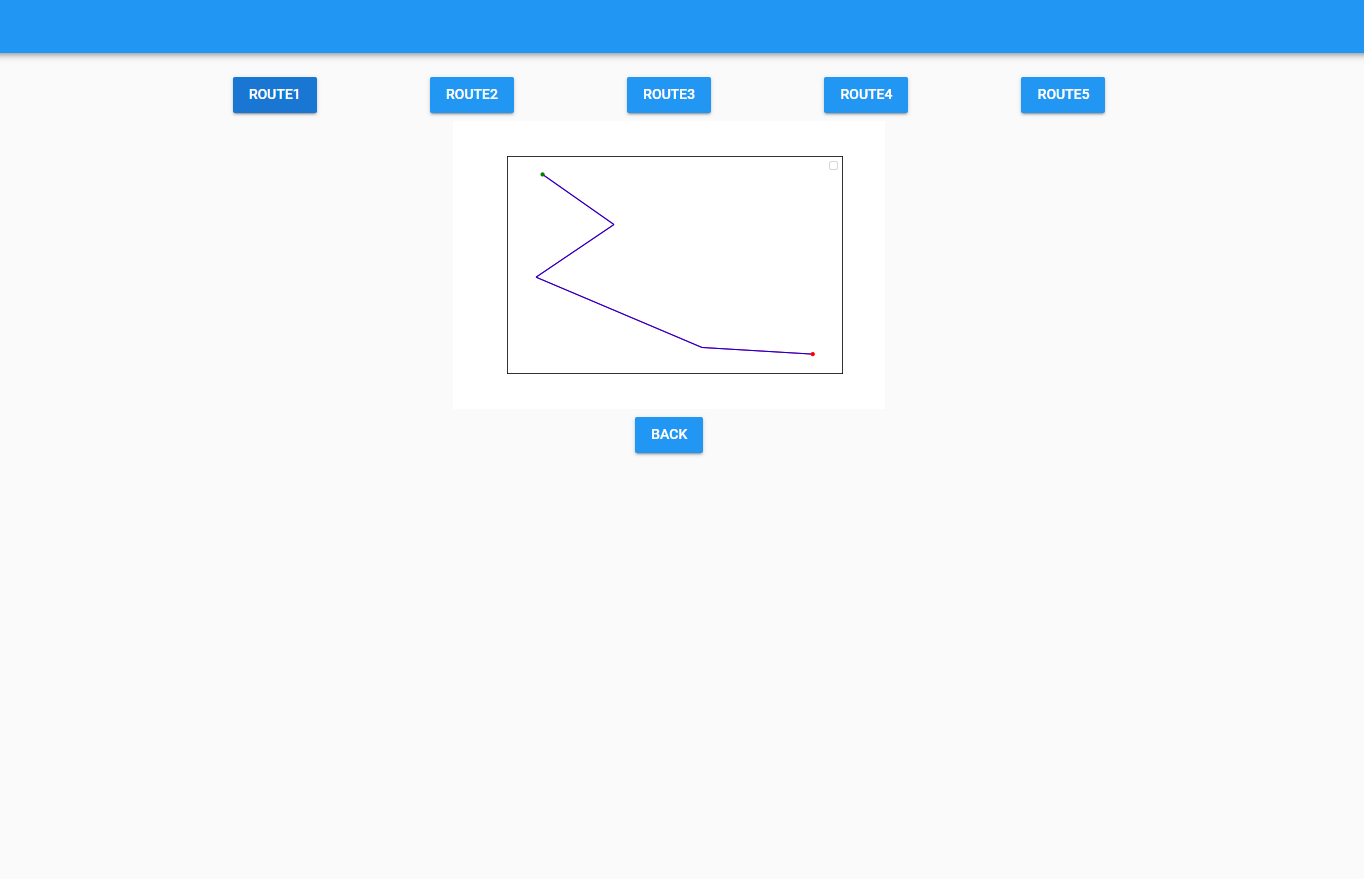
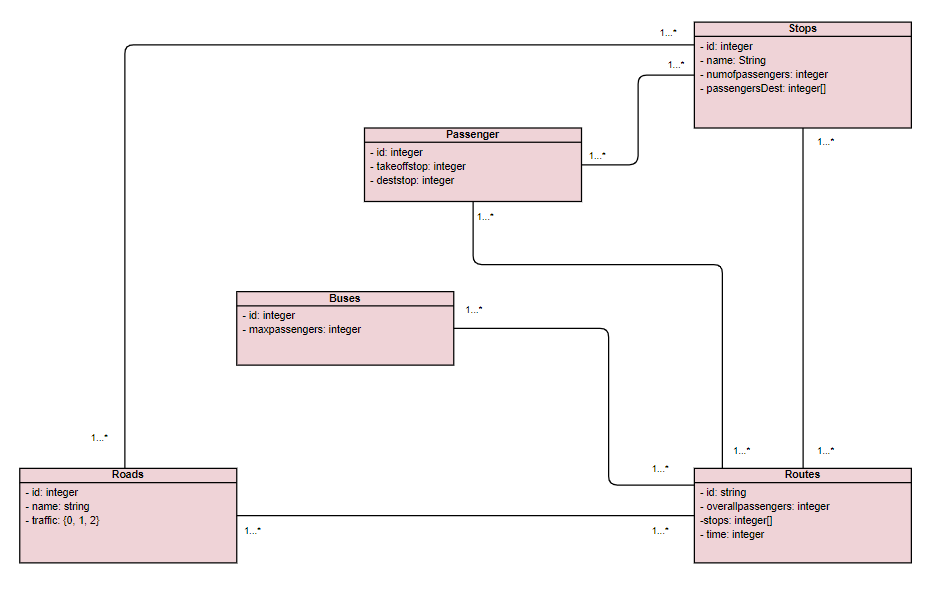
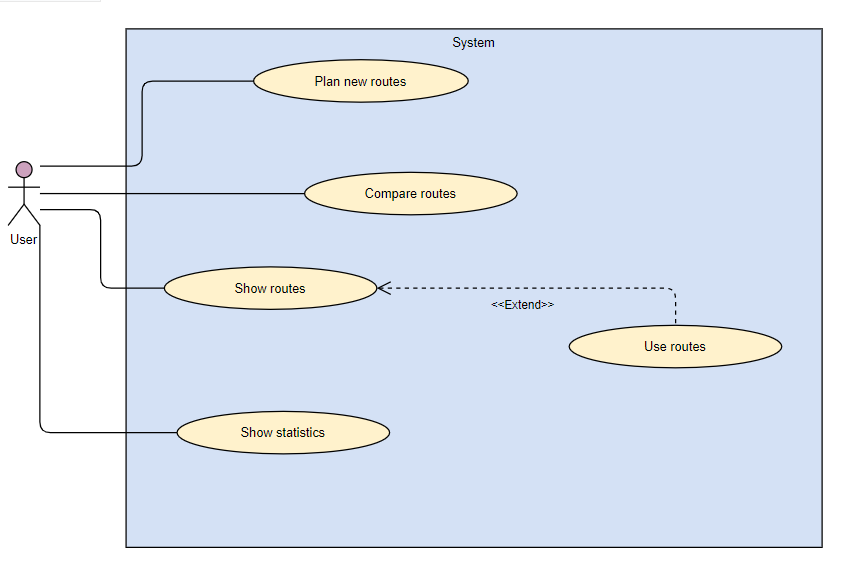


Fig. 8. The user clicked on route1 to see the first calculated route.

### 4.2.4 Class diagram



### 4.2.5 Use case diagram



### 4.2.6 Activity diagram

Diagram

Description automatically generated

# 5. Evaluation/Verification

|  |  |  |
| --- | --- | --- |
| **Test ID** | **Test Description** | **Expected Result** |
| 1 | The application calculates routes by clicking calculate route | Receive optimal route |
| 2 | Show route data by clicking on desired route | Show the desired route data |
| 3 | Check if data inserted is correct or not | If incorrect or missing data, show error message |
| 4 | Check if the paths are optimal | If path is not optimal, show error message |
| 5 | The model successfully processes raw data into a format suitable for training | Process graphs to get optimal paths |
| 6 | The model successfully trains on the preprocessed data to make predictions | The model is trained, and high accuracy achieved |
| 7 | The model performs efficiently without excessive resource usage | The model runs smoothly without using extra ram |

# 6. Results and analysis

After building the machine learning model and generating the data, we used the generated data to train the model, for the training phase we generated 10 graphs, from each graph we sampled 1000 routes for training and 100 routes for validation. We measure the model effectiveness using accuracy as mentioned in chapter 3.

6.1 Training results

We trained the model on 10 different graphs which is not highly ideal, although the results look promising, we have been able to achieve Train-accuracy **~ 0.92** and Validation-accuracy **~ 0.90** which is considered high in machine learning, we will later provide ideas for improvements which we believe can make the model more accurate.

Now that we have the model trained and the desired accuracy achieved, we can test the model on new generated data to see how it preforms, for this part we generated 50 graphs of size 50 nodes, 50 graphs of size 100 nodes, 50 graphs of size 150 nodes, for each graph we sampled 300 routes.

in Fig 9 we can see the size of each sample size (each sample is the 300 routes of the graph)

## 6.2 Testing results

We tested the model on a graph which consists of 50 nodes, we choose 200 pairs of nodes as our paths, the Accuracy achieved the test set ~0.87.

We then went further and tested the model on three different sizes of graphs and for each size we generated 50 graphs from each graph we picked 300 pairs of nodes as our paths.

Chart, scatter chart

Description automatically generated

Fig. 9 the sample size for each graph size, the sample size indicates how many nodes in the 300 routes of each graph, the indicates the average nodes amount for each graph size, the indicates the standard deviation.

## 6.3 Results

In Fig. 10 we can see the accuracy of the model using the graphs mentioned in the previous chapter 7.2, as we can see the model achieved average accuracy of **~0.90** on each graph size (50,100,150) with low standard deviation which indicated consistency.

Chart, scatter chart

Description automatically generated

Fig. 10 the model accuracy on the graphs, the indicates the average accuracy for each graph size, the indicates the standard deviation.

# 7. Conclusion and further work

As seen in the results section the model gives approximately 85% accuracy on the test set, the results are considered very good due to the simplicity of the model and data used, our data was generated randomly which makes it harder to find a pattern, therefore the predictions aren’t so accurate.

There is a lot to work on to make this model reliable, we suggest doing the following to get more accurate model which can be used in real life:

1. Using real life data: real life data contains patterns and shows how the customers used the buses for the past few years. The model can detect the changes of common destinations, the changes of customers amount in each area and so on. This change can make a huge impact on the accuracy of the model.
2. The model now uses only a few customers on each stop and the geographic location to make the predictions. We suggest adding more features such as each customer destination, date, and time of the customers’ trips since in real life travelling differs from day to day e.g., weekdays/weekends.
3. Another suggestion is using a constant graph which resembles a certain city. Currently we only generate few graphs to have sufficient data to train the model.
4. Make the application more user friendly. It can be deployed by adding a database and granting a token for each user.

# 8. User guide:

In this chapter, we explain how to use our program and how to maintain it, for more information check the file included Bus\_Router\_User\_guide.

## 8.1 Packages used:

Networkx

Apache MXnet

Numpy

sklearn

Anvil

Matplotlib

These packages must be installed prior starting, the user must make sure the packages versions support the other packages.

The code we provided makes sure the required packages are installed (until the due date of the assignment)

## 8.2 Training the model & statistics.

For training the model from the start, the user must run the code in Training-Bus\_Router.ipynb we provided on google colab, a few changes must be done:

1.The user must connect to their own google drive account

2.The user must change the paths to their own google drive account to save and load the model parameters

After those changes are done, the user now can run all the cells either by choosing runtime->run all from the nav bar of google colab, or by running each cell one by one.

Once all the cells finish running the user can see the results in the output box under each cell.

## 8.3 Using the model.

After training the model for at least one time, now we should have the model parameters saved in our google drive, the user can now run the cells without the cells containing the model training and result/tests.

## 8.4 Using anvil server.

After running all the cells ,the last cell is used for the anvil server ,once its running it runs for ever(or until the google colab runtime is disconnected), once this cell is running the user can head to the public URL <https://excitable-dry-mosquito.anvil.app> to use the anvil interface, the user can calculate 5 routes of their choice, the source and destination must be numbers between 0-49 ,once the user filled the five routes source and destination fields he then can click calculate routes to see the resulted routes, when clicking the route# button the route # is shown, the user can click back in order to calculate 5 different routes. **Refer to chapter 5.2.3 for visual explanation.**

# 9. References

[1] Tarjan R. Depth-first search and linear graph algorithms. SIAM journal on computing. 1972;1(2):146–160.

[2] Bundy A, Wallen L. Breadth-first search. In: Catalogue of Artificial Intelligence Tools. Springer; 1984. p. 13–13.

[3] Pearl J. Heuristics: intelligent search strategies for computer problem solving. 1984;.

[4] Korf RE. Artificial intelligence search algorithms. Chapman & Hall/CRC; 2010.

[5] Lanning DR, Harrell GK, Wang J. Dijkstra’s algorithm and Google maps. In: Proceedings of the 2014 ACM Southeast Regional Conference. ACM; 2014. p. 30. 60

[6] Floyd RW. Algorithm 97: shortest path. Communications of the ACM. 1962;5(6):345.

[7] Hougardy S. The Floyd–Warshall algorithm on graphs with negative cycles. Information Processing Letters. 2010;110(8-9):279–281.

[8] Cheng C, Riley R, Kumar SP, Garcia-Luna-Aceves JJ. A loop-free extended Bellman-Ford routing protocol without bouncing effect. In: ACM SIGCOMM Computer Communication Review. vol. 19. ACM; 1989. p. 224–236.

[9] Korf RE, Reid M. Complexity analysis of admissible heuristic search. In: AAAI/IAAI; 1998. p. 305–310.

[10] Rosenblatt F. The perceptron: a probabilistic model for information storage and organization in the brain. Psychological review. 1958;65(6):386.

[11] Goodfellow I, Bengio Y, Courville A. Deep Learning. MIT Press; 2016.

[12] Nair V, Hinton GE. Rectified linear units improve restricted boltzmann machines. In: Proceedings of the 27th international conference on machine learning (ICML-10); 2010. p. 807–814.

[13] He K, Zhang X, Ren S, Sun J. Deep residual learning for image recognition. In: Proceedings of the IEEE conference on computer vision and pattern recognition; 2016. p. 770–778.

[14] Simonyan K, Zisserman A. Very deep convolutional networks for largescale image recognition. arXiv preprint arXiv:14091556. 2014.

[15] Rumelhart DE, Hinton GE, Williams RJ. Learning representations by back-propagating errors. nature. 1986;323(6088):533.

[16] LeCun Y, Boser B, Denker JS, Henderson D, Howard RE, Hubbard W, et al. Backpropagation applied to handwritten zip code recognition. Neural computation. 1989;1(4):541–551.

[17] Hecht-Nielsen R. Theory of the backpropagation neural network. In: Neural networks for perception. Elsevier; 1992. p. 65–93. 61

[18] Werbos PJ. Backpropagation through time: what it does and how to do it. Proceedings of the IEEE. 1990;78(10):1550–1560.

[19] Riedmiller M, Braun H. A direct adaptive method for faster backpropagation learning: The RPROP algorithm. In: Neural Networks, 1993., IEEE International Conference on. IEEE; 1993. p. 586–591.

[20] https://www.liveabout.com/bus-routes-and-schedules-planning-2798726

[21] https://cities-today.com/machine-learning-helps-dubai-plan-better-bus-routes/