Optimization of Public Transport Routes by Means of AI Techniques

Daniel Filipe dos Reis Rodrigues
Computer Science and Engineering
Instituto Superior Técnico
Lisboa, Portugal
Email: daniel.f.rodrigues@tecnico.ulisboa.pt

Abstract—The bus transportation service is an important part of a city's infrastructure and, as transit keeps increasing, the need to optimize this service grows. The optimization of bus routes, which is a complex combinatorial problem, commonly known as the Bus Routing Problem, cannot be solved through exact mathematical methods in useful time. By creating methods that are able to design optimized bus routes, the overall public transportation service quality would improve greatly. We researched the literature for methods that had already been applied to other routing problems, looking to find a good candidate method for the Bus Routing problem. We developed a version of an Ant Colony System that was able to, given a network and the distribution of passengers, compute a bus route, using one bus. With the intention of expanding our algorithm to work with multiple buses, we improved our approach so that two buses could be used, achieving improved results.

I. INTRODUCTION

Population distribution within countries has been changing for some years now. As the opportunity for work decreases in the countryside, people end up moving to large cities so they can be closer to entertainment, health services and schools. Public transport infrastructure, which makes moving around the city convenient, is an essential component of any modern city. As more and more people choose to move to large cities, the population density increases. This means that the number of people using the bus public transportation service increases.

Public transport optimization is, therefore, crucial to keep up with the increased demand, while keeping low operating costs.

One of the most impactful optimizations that can be done to the bus transportation service is the optimization of bus routes. Buses share the road with other vehicles, contrary to the metro which usually benefits from a dedicated track. This means that a better route planning for each bus may have a big impact in the cost and quality of the service.

Lisbon is a large city that suffers from many of the previously mentioned issues. For this reason, the Lisbon municipality aims to optimize its public transport infrastructure.

In order to optimize the bus routes, cities can collect data regarding the number of people that enter/exit the bus in each stop/area, we call this the demand. Using this data and different Artificial Intelligence (AI) approaches we can try to arrive at better routing for the bus service. In this project we will be contemplating some AI approaches to this problem. These algorithms coupled with the data collected can produce results that will help us arrive at more efficient routes for the bus service.

The bus planning process can be seen as a sequence of levels in which every level heavily influences the quality of the solution in subsequent ones. As proposed in [1] these levels can be structured as follows:

A. Network Design
B. Setting Frequencies
C. Timetable Development
D. Bus Scheduling
E. Driver Scheduling

This project focuses on the main aspect of this planning process, the level A. Designing these bus networks is a difficult problem, so we hope that by means of AI methods we can reach efficient solutions. If better results were to be achieved on level A, it is reasonable to assume that the optimization of the whole process will contribute to the improvement of the overall public transport infrastructure, given the chain nature of the process.

II. BACKGROUND

A. Routing Problems

The Bus Routing Problem attempts to design an efficient bus network to provide coverage to the demand placed by the citizens onto the service. A bus networks is composed of groups of bus stops, where each group forms a bus route. Typically the optimization of the design is based on pre-existing bus stops and demand data for each bus stop.

The Travelling Salesman Problem is a well known routing optimisation problem in the literature. It consists of, given a set of locations, what is the shortest path that visits every location, and ends up in the starting location. Conceptually, the locations on this problem represent cities, villages, etc., and the objective is to optimize a route, so that a salesman can visit every location by travelling the less amount of distance possible. The TSP is considered in the literature as an NP-hard problem.

B. Algorithms

1) Genetic Algorithms: Genetic Algorithms are an optimization technique that belong to the larger class of evolutionary algorithms. Since these are one of the main methods
used to optimize bus routing problems, we will review them briefly.

These algorithms are inspired by biological evolution and they use operators such as reproduction, mutation and selection. The candidate solutions to the problem are represented as strings and a set of these strings constitute a population. The characteristics that make each solution, meaning the characters of the string, are called genes. The objective is to generate subsequent populations where the better genes of previous populations are more abundant. We will now go over three genetic operators, selection, crossover and mutation.

The selection operator is responsible for, given a population, choosing the strings that are going to reproduce according to the utility of the solution. One of the simplest ways to do this is by creating a weighted roulette wheel, commonly known as WRW, where strings are placed in different slot sizes. To do this the \( f_f \) value for each string is used to calculate a slot size and the string is placed on the WRW. The slot size corresponds to a reproduction probability and it is given by:

\[
p_r(i) = \frac{f_f(i)}{\sum_{i=1}^{n} f_f(i)},
\]

where,

- \( f_f(i) \) = fitness function value for solution \( i \)
- \( n \) = number of candidate solution/strings in the population
- \( p_r(i) \) = reproduction probability/slot size

This WRW is then used to copy strings, selected from the wheel by spinning it, to a mating pool that will then be used by the crossover operator. Spinning the WRW means selecting a string from the wheel, randomly, according to the probabilities represented by the slot sizes attributed to each string.

The crossover operator mates pairs of strings randomly from the mating pool created by the selection operator. The mating process consists of exchanging genetic information between the two strings being paired in order to generate new strings. In this case, for each mated pair two new strings are created where each one has information from both parent strings. The simple way to do this genetic information exchange is by using a simple arithmetic crossover [2]. This operation (Fig. 1) consists in selecting a cut point \( k \), between 1 and the length of the strings \( l \), randomly from an uniform distribution and generating two new strings by swapping genes on the parent strings between \( k \) and \( l \). An improved version of the simple crossover and the one used in [3] consists in selecting two cut points and swapping, between the two parent strings, the genes between both cut points (Fig. 2).

Finally, the mutation operator goes through the string space of the population and, with small probability, applies changes randomly to the strings. This prevents the loss of solution potential caused by the selection/crossover operators.

2) Ant Colony Algorithm: Ant colony algorithms are a group of algorithms based on techniques inspired from the behaviour of ant colonies, introduced by [4]. Ant colonies are able to find food and subsequently the shortest path to the food without the sense of sight.

Ants accomplish this feat by wandering randomly until they find a food source. Once a food source is found they return to the colony. During this whole process each ant leaves a constant amount of pheromone per unit of time, forming a pheromone trail. Once a wandering ant comes across a pheromone trail left by another ant, it has to decide whether to follow it or to choose another path. If the ant chooses to follow the pheromone trail it will leave its own pheromone, strengthening the trail. The stronger the trail (higher amounts of pheromone) the more appealing it is for ants so the higher the probability of other ants choosing that path.

When a new shortest path to the food source is discovered by an ant, that path starts being used by other ants. Because of the fact that the path is shorter than the previously considered shortest path, the round-trips from the colony to the food source are quicker, which allows the ants to make more round-trips per unit of time. Because of this, and because the ants drop a constant amount of pheromone per unit of time, the amount of pheromone in the trail of the new shortest path grows faster than the longer paths. Eventually this new shorter path will have more pheromone than the previously shorter path, becoming the main path taken by the ants. This logic, forms a positive feedback mechanism that allows the optimal path to the food source to be increasingly enhanced and the less optimal paths become less and less appealing because of pheromone evaporation.

This positive feedback mechanism is also what allows the ants to adapt to changes in the environment (Fig. 3). When an obstacle is placed on the trail that is being used by the ants, the trail gets split up. Now, because the pheromone trail is broken, when the ants reach the obstacle, both the ones coming from the colony and the ones returning from the food, they don’t know where to move in order to find the rest of the trail. Using the explained mechanism, the ants start to wander randomly form that point until they connect to the rest of the trail. Eventually, through the positive feedback mechanism, the
trail will be connected again through the shortest path.

III. SHORTEST PATH PROBLEM

Shortest Path problem consists in finding the path between two vertices that minimizes the weight sum of the edges. The problem can be defined in undirected, directed or mixed graphs.

Formally, the problem consists in, given:

- A graph \( G = (V, E, c) \), where \( V = (v_1, v_2, ..., v_n) \) is a set of vertices, \( E \) a set of edges, where \( e_{i,j} \) is an edge between the vertices \( i \) and \( j \), and \( c_{i,j} \) is the cost associated with that edge.
- Two vertices, the source vertex, \( s \), and the destination vertex \( d \).

Determine \( V' = (v_1', ..., v_n') \), where, \( V' \) is a subset of \( V \) and \( v_1 = s, v_n' = d \), such that \( \sum_{i=1}^{N} e_{i,i+1} \) is minimized.

Two of the most commonly used algorithms to solve this problem are Dijkstra algorithm and Bellman-Ford algorithm, for graphs with non-negative edge weights and graphs that may have negative edge weights, respectively.

IV. RELATED WORK

A. Bus Routing Problem

1) Multi-objective Function Optimization: In related work, multi-objective functions have already been defined and used in order to try and represent the interests of the stakeholders involved in the Bus Routing Problem. From these defined functions we call attention to the one representation presented in [5] consisting of two objective functions \( Z_1 \) and \( Z_2 \) which takes into account both passengers and operator interests. The functions are as follows:

\[
Z_1 = A \sum_{i,j=1}^{N} PH_{i,j} + B \sum_{i,j=1}^{N} WH_{i,j} + C \sum_{r \in R} EH_r \tag{2}
\]

\[
Z_2 = FS \tag{3}
\]

where,

- \( N \) is the number of nodes
- \( R \) is the Route set
- \( PH_{i,j} \) is the number of passengers riding time from node \( i \) to node \( j \) on an hourly basis
- \( WH_{i,j} \) is the number of passengers waiting and transfer time from node \( i \) to node \( j \) on an hourly basis
- \( EH_r \) is the number of unused seats in a bus on an hourly basis on route \( r \)
- \( A, B, C \) are the weighting coefficients that represent the importance of each objective
- \( FS \) is the number of buses required to offer all trips on the set of routes

In the first objective function \( Z_1 \) (eq. 2) we can isolate three terms. The first term, \( \sum_{i,j=1}^{N} PH_{i,j} \), represents the sum of the time each passenger spends riding, on an hourly basis. Lowering this value while maintaining the same demand coverage from the bus network is an improvement as it means that the passengers get faster to their destinations on average. The second term, \( \sum_{i,j=1}^{N} WH_{i,j} \), represents the sum of the time each passenger spends waiting for buses or transferring between buses before reaching the destination. It is also good to reduce this value as it implicates a faster travel between origin and destination on average. The third term, \( \sum_{r \in R} EH_r \), is the sum of the time each seat in every bus spends empty on an hourly basis. Lowering this sum, without compromising demand coverage, means that the number of seats that are not being used in hourly basis decreases which is good considering that it results in a more efficient bus network with lower costs for the operator.

A variation of the presented multi-objective function, also used in the literature, is the one presented in [6]. In this case the multi-objective function does not account for empty seat hours and instead has a different representation of trip time:

\[
Z = A \sum_{i,j=1}^{N} d_{i,j}p_{i,j} + B \sum_{i,j=1}^{N} d_{i,j}t_{i,j} \tag{4}
\]

where,

- \( d_{i,j} \) represents the transit demand from node \( x_i \) to node \( x_j \)
- \( p_{i,j} \) is the length of the shortest path between nodes \( x_i \) and \( x_j \) on the current route set
- \( t_{i,j} \) is the minimum number of transfers required to go from node \( x_i \) to node \( x_j \) using the shortest path on the current route set

This version of the multi-objective function was the one used in [7] on the proposed Public Transit Network Route
Generation Algorithm. They used an undirected graph to represent the problem, so the representation of the network is described by $G = (V,E)$ where $V$ is the set of nodes and $E$ is the set edges. They also used a matrix $D$ to represent the travel demands of the network, where $D_{i,j}$ indicates the number of passengers going from $i$ to $j$.

In their proposed solution, and because it is an optimization method, an initial candidate solution is needed. To generate their initial candidate solution they used edge usage statistics. To obtain said usage statistics they calculated all the shortest paths in the graph, then proceeded to calculate the deviation of edge scores for each edge’s score base on the total traffic going over it. Then they weighted the score of every edge by the inverse of its length and normalized the scores. They used these final normalized scores as a discrete probability for each edge. A function was then created in order to extent the route. This function would add the edges with the highest scores to the beginning or the end of the given route. By creating routes this way they got their first route set to work on using their Solution Modification Procedure.

Their Solution Modification Procedure consists in choosing a random route from the route set and applying one of three modifications:

1) Add node: A node is added to either the beginning or ending of the route and the route is extend
2) Delete node: Deletes a node from one end of the route and shortens the route
3) Insert node: Picks a random node that is not part of the route, adds it to the route and keeps it in the route if it produces better results. If the length of the route goes past the maximum defined length then the algorithm shortens the route using the delete operation

They tested their method on the Mandl’s Swiss Road Network [8] a 15 nodes, 20 edges network commonly used to test Bus Routing Problem solutions.

2) Genetic Algorithms for Bus Route Optimization: An example of an application of Genetic Algorithms to the Bus Routing Problem we present the solution in [3]. The authors proposed an application of a genetic algorithm to the Bus Routing Problem. The first challenge that arises is the representation. Because Genetic Algorithms work with strings, a binary string representation of the networks was used. In this representation each string of a population represents a network organization and every two bits of one string, a gene, represents one bus route. Given that the Genetic Algorithm is an optimization technique, the initial population is either generated randomly or is the network set we are trying to optimize.

The proposed solution in [3] can be divided into two parts, the Genetic Algorithm and the analysis of the network set. On the network set analysis there are three phases. The first being the assignment phase where values regarding network characteristics are assigned to each network set. The second phase, called the aggregation phase, aggregates the output from the assignment phase in order to get the performance indicators which are then used in the multicriteria analysis phase in order to classify the networks. This classification is done using the weighted sum of the performance indicators, the weights used are defined previous to running the algorithm. The result of this classification are the $ff$ values. These values are then fed into the Genetic Algorithm.

On the Genetic Algorithm part, the selection operator creates the weighted roulette wheel based on the $ff$ values of each network set and using the wheel generates the mating pool. Next, the crossover operator mates the networks from the mating pool creating a new network set and the mutation operator applies some changes to the networks in the network set in order to explore a broader solution space. After the new population has been generated it is fed into the network set analysis part of the overall algorithm so that it can be evaluated and assigned $ff$ values.

So, in conclusion the output from the Genetic Algorithm part of the method is fed into the network analysis part, which then feeds its analysis output back into the Genetic Algorithm. The algorithm continues this loop until a fixed set of iterations or until no improvement arrives from new generations.

B. Ant Colony Algorithm

1) Ant Colony Algorithm applied to the TSP: One of the common applications of the Ant Colony algorithm is the Travelling Salesman problem as it is able to produce near-optimal solutions to the problem. What makes the Ant Colony so interesting in this scenario is the fact that it is able to deal with real-time environment changes. One example of the Ant Colony Algorithm applied to the TSP is described in [4]. In this work, the authors took some principles obtained from the behaviour of real ant colonies and were able to, by making the necessary adaptations, use it to tackle the TSP. In their algorithm they use the concept of artificial ants which they compare to real ants. These artificial ants are agents that move from city to city in the graph of the problem. They choose the next city to move to using a probabilistic method that tries to recreate the decision making of real ants.

Ants prefer cities that are close-by and have higher amounts of pheromone in the edges that lead to it. The authors start off by having the artificial ants randomly start in different cities. At each time step, each ant moves to a new city and updates the pheromone value of the edge used, the authors call this local trail updating. When all the ants have completed a tour, the ant that has the shortest tour path updates the pheromone of the edges in the path. This is the global trail updating. The amount of pheromone added to each edge in this update is inversely proportional to the tour length of the ant.

The authors identify three real ant behaviours that they transferred into their artificial ant colony. These behaviours are: (i) the preference for paths with higher levels of pheromone, (ii) the faster growth rate of pheromone in shorter paths, and (iii) the trail is the communication medium between the ants.

Apart from these behaviours inspired in real ants, the authors also gave the artificial ants some capabilities that don’t have a natural correspondent, such as, the capability
to determine how far a city is, and a working memory that is emptied at the beginning of each new tour and is updated each time step in order to keep track of the visited cities.

In order to translate the capability (i) from the natural ants, the authors made the artificial ants choose the next city based on the following formula:

$$s = \begin{cases} \arg\max_{u \in M_k} \{ [\tau(r,u) \cdot n(r,u)]^\beta \} & \text{if } q \leq q_0 \\ S & \text{otherwise} \end{cases}$$

where \( \tau(r,u) \) represents the amount of pheromone in the edge \((r,u)\), \( \cdot \) is a function that outputs a value that is inversely proportional to the length of the edge between \( r \) and \( u \). \( \beta \) was introduced to regulate the importance of pheromone quantity and closeness. \( \hat{q} \) is a uniformly random number between 0 and 1. \( S \) is a random variable selected according to the following probability distribution:

$$p_k(r,s) = \begin{cases} \frac{[\tau(r,s)]^\beta [n(r,s)]^\beta}{\sum_{u \in M_k} [\tau(r,u)]^\beta [n(r,u)]^\beta} & \text{if } r,s \in M_k \\ 0 & \text{otherwise} \end{cases}$$

where \( p_k(r,s) \) represents the probability of ant \( k \) choosing \( s \) as the next city when standing on city \( r \).

As mentioned before, in this work pheromone trails are updated both globally and locally. Global pheromone updates are performed when all the artificial ants have completed their tours. The ant that performs the shorter tour deposits pheromone in every edge in the tour path. The amount of pheromone deposited is inversely proportional to the length of the tour and is given by the formula \( \tau(r,s) = (1-\alpha) \cdot \tau(r,s) + \alpha \cdot \Delta \tau(r,s) \), with \( \Delta \tau(r,s) \) being the shortest tour.

Local pheromone updates are performed every time an ant chooses an edge. The pheromone in the chosen edge is changed according to the formula \( \tau(r,s) = (1-\alpha) \cdot \tau(r,s) + \alpha \cdot \tau_0 \).

The authors applied this approach to TSP problems already solved in the literature by other methods. Comparing results to the literature, ACS applied to the TSP achieved results at least as good, and often better, than methods such as simulated annealing (SA), neural networks (NNs), self organizing map (SOM) and evolutionary computation (EC).

2) Ant Colony Algorithm applied to the Shortest Path Problem. The Shortest Path Problem is a well studied problem in the literature. Algorithms like the Dijkstra Algorithm and Bellman-Ford Algorithm are accepted as the best suited to solve this problem. But in the case of the work done in [9], the authors tackle a problem that involves more than the distance metric. The problems used by the authors are discrete artificial representations of geographical landscapes, where the best path from one location to another might not be the shortest. The objective is to minimise the energy expenditure of the path, and the shortest path is not always the most energy efficient path.

The authors also aimed to solving the problem, not by exhaustively searching all possible solutions, but by finding good solutions in a reduced amount of computation time.

Because of the reasons stated above, the authors opted on using an Ant Colony Optimization Algorithm to try and solve this version of the unique shortest path problem.

In order to apply an Ant Colony Algorithm to the Shortest Path problem, various modifications have to be made. We will now go over the main work done in [9], where the authors made the necessary adaptations to solve the Shortest Path Problem.

Each data set represents a terrain. A data set is composed of vertices and edges. Each vertex is represented by three coordinates, \( (x,y,z) \). The coordinates \( x \) and \( y \) are distributed on a grid while the coordinate \( z \), that represents the altitude, can assume any value, both positive and negative.

In the representation of the terrain, the \( z \) coordinate represents the altitude. The altitude is used to calculate the elevation angle \( \theta \), which is then used by the cost function in order to return a cost value. This cost function consisted of a simple operation on the angle of the terrain’s elevation, and returned a number representing the cost in the interval \([-0.2,1]\). The function is as follows:

$$f(\theta) = 1 - \frac{0.6}{90} \theta, \quad 0 \leq \theta \leq 180$$

Because the cost given by this function can be negative, to be used to attribute probabilities to edges the authors made a modification and used desirability instead of cost:

$$\text{Desirability} = \frac{1}{2 \cdot \text{cost}}$$

This function turns cost into desirability and the output from the function falls into the interval \([0.5,1.15]\). Edges with higher cost will have a lower desirability and vice versa.

The visibility component of the Ant Colony Algorithm was also adapted. In the TSP application of the algorithm, visibility only concerns the next vertex, where a vertex that is further away has a lower visibility. In the adaptation done by authors to apply the algorithm to the TSP, visibility is what keeps the ants moving in the general direction of the target. So visibility is the distance from the current position of the ant to the target vertex, relative to the distance from the possible next vertex to the target vertex, as we can see in figure 4. The equation that relates both distances and returns the visibility value is given by:

$$\text{Vis.} = \frac{\text{distance from current to target vertex}}{\text{distance from possible next vertex to target vertex}}$$
The authors decided to create the pheromone update and decay operations in such a way that the total amount of pheromone in the system stays constant. This way they avoid situations where the total amount of pheromone in the system decays or explodes with time. To start of a unit of pheromone is placed in every edge, so the total amount of pheromone in the system is equal to the number of edges. The pheromone decay is done after every iteration, the pheromone of every edge decays by a set percentage according to the following equation:

\[ \text{pheromone}_{ij} = (1 - \text{decay constant}) \times \text{pheromone}_{ij} \]

When it comes to the pheromone update, in this implementation of the algorithm, instead of waiting for an ant to finish its tour in order to place pheromone in the edges used according to the tour quality, the pheromone is placed every time an ant uses an edge. The update rule is given by:

If traversed: \( \text{pheromone}_{ij} = \text{pheromone}_{ij} \times \frac{\text{update constant}}{1 + \epsilon} \)

If the edge was traversed multiple times in one iteration (by multiple ants) the update is applied the number of times it was traversed.

In order for the system to achieve a constant amount of total pheromone, the total amount of pheromone placed by all the ants in one iteration must equal the amount that will evaporate in the pheromone decay update. To achieve this the following equation must be satisfied:

\[ \text{total system pheromone} = \frac{\text{(number of ant}}{\times \text{update constant)} \times \text{decay constant}} \]

When an ant needs to decide which vertex to traverse next, it will combine the different metrics into a single quality measure, which is then compared to the other edges. Probabilities are then attributed to each edge based on the comparison. The importance of each metric can also be adjusted by the use of scaling factors. The authors presented two ways of combining the edge metrics. The first, the Product Combination, is given by the following rule:

\[ \text{Prob}_{ij} = \frac{\text{pheromone}_{ij}^{\alpha} \times (\text{visibility})^\beta}{\sum_{k} \text{pheromone}_{jk}^{\alpha} \times (\text{visibility})^\beta} \]  

(7)

V. IMPLEMENTATION

A. Shortest Path Problem

Our ACO approach to the shortest path problem was based on the key aspects of an ACO system. These aspects being: Initial Pheromone, Pheromone Update rules, Pheromone Decay (evaporation) and edge selection.

To apply the algorithm to the shortest path problem, we start by using a number of ants equal to the number of nodes in the given graph G. The ants are all deployed on the source node with the objective of reaching the destination node.

In every iteration of the algorithm, each ant chooses an edge and uses it to move to the corresponding node. In order to choose the edge and navigate the graph the ants use an heuristic that takes into consideration node characteristics such as pheromone levels.

1) Initial Pheromone: In our system, the pheromone is stored in each edge. When starting the algorithm, a unit of pheromone is placed in every edge of the graph, so the total amount of pheromone in the system, at the beginning, is equal to the number of edges in the graph.

2) Pheromone Update: After an ant finishes a tour, the tour quality is evaluated in order to place pheromone accordingly. If the tour did not reach the destination node, which can happen if the ant ended up getting stuck in a dead end, the amount of pheromone deposited is 0. This prevents dead ends from getting to much pheromone and eventually becoming more and more attractive to ants.

If the ant finishes the tour by reaching the destination node, the quality of the tour is evaluated, and, pheromone is deposited in every edge that is part of the tour path accordingly. The higher the quality of the tour, meaning the shorter the tour, the larger the amount of pheromone that is deposited in the edges.

The amount of pheromone to be deposited is given by the following formula:

\[ \text{pheromone} = \frac{Q}{A \times B} \]  

(8)

Where, \( A \) and \( B \) are constants, \( l \) is the length of the ant tour and \( Q \), that is set to 1 by default, is a constant that can be used to boost specific path characteristics. For instance, if we want to encourage the ants to go through a specific node/edge, we can, after evaluating the tour, increase the value of \( Q \) which will increase the amount of pheromone deposited. This logic can also be applied if we are trying to avoid a node/edge by decreasing the value of \( Q \).

3) Choosing the next node/edge: In our approach to the problem, when an ant has to choose the next edge, it starts by listing the candidate edges. To be considered a candidate, an edge has to connect to a node that has not been visited, by the ant in question, in the current tour. Next, each candidate edge is attributed a probability of being chosen, which is calculated according to the pheromone levels, using the following formula:

\[ \text{Prob}_{a} = \frac{\text{pheromone}_{a}}{\sum_{c} \text{pheromone}_{c}} \]  

(9)

Where \( a \) is a candidate edge and \( C \) is the set of all candidate edges.

To prevent the algorithm from converging to local optimal solutions, we implemented a mechanism, typically used in Ant Colony Systems, that promotes exploration. To do this, we use a variable \( \epsilon \) to regulate how often the ants choose the next edge based on pheromone levels and when they choose randomly. So, in reality, the choice of the next edge is done based on the following formula:

\[ \text{Next edge} = \begin{cases} A & \text{if } q < (1 - \epsilon) \\ B & \text{otherwise} \end{cases} \]  

(10)
Where \( A \) represents the candidate edge with the most amount of pheromone, \( B \) is a random candidate edge, selected according to a uniform probability distribution. \( q \) is a variable chosen randomly according to a uniform probability in \([0,1]\), and finally, \( \epsilon \), \((0 \leq \epsilon \leq 1)\).

In order to detect if the algorithm has converged to a solution we introduced a stop condition. To do so, we monitor the current best solution at the end of each iteration. We then limit the maximum number of consecutive iterations that a path can be the best solution. Once this parameter is reached, the stop condition is triggered and the algorithm stops. When the algorithm stops by reaching this stop condition, we assume that the algorithm has converged.

4) Current best solution: We developed a function, called currentBestSolution (algorithm 1), that, given a graph, outputs the greedy path from the source node to the destination node according to the pheromone levels, meaning, the path from source node to destination node, choosing only the edges with the highest pheromone values. Throughout the execution of our algorithm, we use this function in two occasions. The first, is to check what is the current best solution, at the end of every iteration, so that we can monitor and enforce the stop condition. The second occasion is, after the algorithm has converged or has reached the maximum number of iterations, we apply the currentBestSolution function to the resulting graph, and consider the output the solution to the problem.

Algorithm 1: currentBestSolution

Result: Greedy path from source to destination node

1. solution = [source];
2. visited_nodes = [ ];
3. while current_node \# destination do
4.     edges = Graph.edges(current_node);
5.     options = every edge in edges and destination(edge) not in visited_nodes;
6.     next_edge = edge in options with highest pheromone;
7.     next_node = destination(next_edge);
8.     solution.append(next_node);
9.     visited_nodes.append(next_node);
10.    current_node = next_node;
4. end
12. return solution;

5) Ant Colony Algorithm function: Given the mechanics explained in the sections above, the overall flow of the Ant Colony function, algorithm 2, is as follows, in a given iteration, the ants are sent to perform their tours, after all ants finish, the pheromone is deposited in the edges according to the ant tours. Then, the evaporation update is applied to the pheromone level of every edge. Next, the ants are reset and the stop condition is checked, in case the stop condition has been reached the algorithm stops, otherwise, the algorithm proceeds to the next iteration.

Algorithm 2: AntColonyAlgorithm

Result: Graph with the resulting pheromone values for each edge

1. G = loadGraph();
2. for i in range(1, MAX_ITER) do
3.     for ant in ants do
4.         ant.tour();
5.     end
6.     G.depositPheromone();
7.     G.evaporatePheromone();
8.     for ant in ants do
9.         ant.reset;
10.    end
11.    if stopCondition(G) == True then
12.        break;
13.    end
14. end

6) Test Problem: Apart from some smaller test problems used, we also used a larger network to test our algorithm performance. This larger network, figure 5, was vaguely inspired in a network presented in [10]. we used this network with the intention of later comparing our results with the work done by the authors, but quickly the objectives of our work, restrictions and constraints, differed enough to make any kind of comparison senseless. The reason we continued using the network was the fact that it was larger then the test problems we had until then, but small enough to allow testing on it.

The network (fig. 5) is composed of 51 nodes and 73 edges. The shortest path problem consisted in finding the shortest path between the nodes A and AV, which have 657920 different simple paths between them. In order to evaluate the solutions produced by our algorithm, we first ran the Dijkstra algorithm on the to get the best solution to use as benchmark. Using the Dijkstra algorithm, we got a result of 18.65 km for the shortest path between nodes A and AV. We ran our implementation...
of the Ant Colony Algorithm on the network 100 times and achieved the same shortest path as the Dijkstra algorithm, 100% of the time. The algorithm did it with an average of 171 iterations, and, an average running time of 12 seconds.

B. Bus Routing Problem

In order to deploy the algorithm on the Bus Routing Problem, with the insertion of passengers and bus stops etc., some adaptations had to be made to the algorithm in order to be applied to this new problem context. We will now go over these adaptations.

1) Passengers: In order to simplify our work, we consider that passengers are dispersed over the nodes of the network and that every passenger has the same destination node. We also consider that, given the bus route of the bus, each passenger decides between two options, walking to the nearest bus stop that belongs to the bus route and taking the bus until the destination, or, walking from the current position to the destination node. The passengers decide between these two options, by choosing the one that yields the smaller amount of time to reach the destination node.

2) Quality of an ant tour: We consider that each ant tour is a possible bus route, so, we had to change the way we assess the quality of ant tours. Contrary to the Shortest Path version of the algorithm, where we considered that a shorter tour was always better than a longer tour, in this version, because the objective does not revolve around finding a shorter tour but instead transporting the passengers to their destination, we consider that the better tour is the one that minimises the average time the passengers take to reach their destination. We call this metric Average Passenger Time (APT).

In order to evaluate the tours, we created a function, calculateAvgPassengerTime (algorithm 3), that, given a bus route (ant tour), calculates the average time it takes for a passenger to reach the destination.

This function assumes that, given a bus route, passengers will choose between, walking to the destination, and, walking to the nearest bus stop and taking the bus, based on which option is faster.

When calculating the average time the passengers take to their destination, we assume that, the passengers walk at a pace of 5 km/h, while the bus has an average speed of 15 km/h, and, that the bus capacity is enough to carry all the passengers.

When calculating the average time the passengers take to their destination, we assume that, the passengers walk at a pace of 5 km/h, while the bus has an average speed of 15 km/h, and, that the bus capacity is enough to carry all the passengers.

3) Pheromone Updates: Using the described logic, when assessing the quality of an ant tour, the ants deposit pheromone according to the tour quality. The amount of pheromone deposited by the ants, in every edge of a tour $t$, is given by the equation:

$$pheromone = \frac{Q}{A \cdot B^{avg(t)}}$$  \hspace{1cm} (11)

Where $Q$, $A$, $B$ are constants, and, $avg(t)$ represents the average time passengers take to reach the destination in tour $t$.

Algorithm 3: calculateAvgPassengerTime

| Result: Average time passengers take to reach destination |
| total time = 0; |
| for node in Graph.nodes do |
| \hspace{1cm} distance to destination = Dijkstra(node, destination); |
| \hspace{1cm} time walking = (distance to destination * 60)/walking speed; |
| \hspace{1cm} closest stop = None; |
| \hspace{1cm} for stop in bus_path do |
| \hspace{2cm} distance to stop = Dijkstra(node, stop); |
| \hspace{2cm} if distance to stop ≤ closest stop then |
| \hspace{3cm} closest stop = stop; |
| \hspace{1cm} end |
| \hspace{1cm} time to bus stop = distance to stop*60/walking speed; |
| \hspace{1cm} bus stop to destination = distance to stop*60/bus speed; |
| \hspace{1cm} time using bus = time to bus stop + bus stop to destination; |
| \hspace{1cm} total time += min(time walking, time using bug)*node[number of passengers]; |
| \hspace{1cm} end |
| \hspace{1cm} average time = total time/number of passengers; |
| return average time; |

4) Multi-bus approach using the Ant Colony Algorithm: With the intention of extending our algorithm to be able to work with multiple buses, we created a different approach to the problem using the already implemented Ant Colony algorithm. To achieve this, we partitioned the network in a way that the source and destination nodes would be a part of both resulting networks, same for key nodes that, if removed, would make it impossible to reach the destination node.

We then deployed our algorithm in both resulting networks, achieving the optimal bus route for each network independently. We then applied the two optimal bus routes in the initial network (fig. 5), and calculated the average time for a passenger to reach the destination node, considering that both bus routes were available simultaneously. We did this using a slightly modified version of the calculateAvgPassengerTime function (algorithm 3), to accommodate the existence of two bus routes instead of one.

VI. RESULTS

A. Problem Description

To test our algorithm, we used the network 3 (fig.5), with passengers placed in various nodes. Using this network, the objective was to run the Ant Colony Algorithm, and achieve a bus route, composed of a set of bus stops (nodes), that would minimize the average time for a passenger to reach destination.
The bus would be leaving from node A, and the passengers' destination was node AV.

B. Single bus approach

First, we used the implementation of the Ant Colony algorithm applied to the Bus Routing Problem, described in V-B. After tuning the parameters, we ran the algorithm on the described problem 100 times. The objective of this test was to achieve the minimum value for the metric Average Passenger Time (APT). The algorithm achieved three different quality solutions relative to the APT:

<table>
<thead>
<tr>
<th>APT</th>
<th>#</th>
</tr>
</thead>
<tbody>
<tr>
<td>48.8788</td>
<td>85</td>
</tr>
<tr>
<td>49.4503</td>
<td>4</td>
</tr>
<tr>
<td>49.9528</td>
<td>11</td>
</tr>
</tbody>
</table>

**TABLE I**

AVERAGE PASSENGER TIME (APT) RESULTS ON NETWORK 3 (FIG. 5)

As shown in table I, the algorithm converges, approximately 85% of the time, to a solution that yields an APT of, approximately, 48.87 minutes. Given the nature of the problem, it is possible for different bus routes to yield the same APT. In these 100 runs, there were four distinct bus routes that yielded the best APT. These bus routes were:

<table>
<thead>
<tr>
<th>Bus Routes</th>
</tr>
</thead>
<tbody>
<tr>
<td>A, B, D, F, H, K, S, T, AA, AH, AL, AM, AN, AO, AW, AU, AV</td>
</tr>
<tr>
<td>A, C, D, F, H, K, S, T, AA, AH, AL, AM, AN, AO, AW, AU, AV</td>
</tr>
<tr>
<td>A, C, E, F, H, K, S, T, AA, AH, AL, AM, AN, AO, AW, AU, AV</td>
</tr>
</tbody>
</table>

**TABLE II**

BUS ROUTES THAT YIELD AN APT OF 48.87 MINUTES

We ran all the tests on a intel i7-2500k, 3.3Ghz, processor with 12 GB of DDR3-1333Mhz ram.

Our intention was to find solutions to the Bus Routing Problem in the literature in order to compare our results to, but given the very broad nature of the problem, we were unable to find a specific work that had a similar formulation and a similar set of restrictions when comparing to the problem we tackled. For that reason, we were unable to benchmark our results against other work done in the literature.

C. Multi-bus Approach

This method implies partitioning the network in two sub-networks. For this purpose, we used the same partitioning of the reference network, shown in V-B4, but this time including the passengers, which resulted in the sub-networks: sub-network 1, figure 6, and, sub-network2, figure 7.

The first step in this method is determining the best bus routes for each of the sub-networks individually. to achieve this, we ran our Ant Colony Algorithm on each sub-network independently. We got the best bus routes for each of the sub-networks. We then matched the solutions, using a combinatorial approach, in order to achieve the better match between a bus route from sub-network 1 and sub-network 2. The bus route combination, that yielded the best APT (39.62 minutes), is shown in table IV and figure 8.

Using this method, we were able to achieve an APT of 39.62 minutes.

VII. CONCLUSIONS

Despite all the methods described in the literature, the Bus Routing Problem, with its specific restrictions and constraints, still remains as a work in progress.
In our work, we started by going over the most common methods in the literature used to solve other routing problems, such as the School Bus Routing problem, the Shortest Path problem and the Travelling Salesman problem, with the objective of finding an algorithm that would demonstrate potential to be applied to the Bus Routing Problem.

A. Discussion

We tried to contribute to the literature by developing an Ant Colony System that was able to tackle the Bus Routing problem when operating with a single bus (bus route), with the objective of later expanding it to multiple buses. We made efforts to expand our approach to a multi-bus solution. We presented an algorithm that was able to consider two buses (bus routes) when solving the problem by partitioning the road network in two sub-networks, achieving, as expected, better results than the single bus variant.

Given the broad definition of the Bus Routing Problem, we were not able to find other work in the literature with enough similarities that could enable comparison between results. For this reason, we were not able to benchmark our results with other approaches.

B. Future Work

In the future, the scalability of the methods described in our work needs to be tested and evaluated, regarding the number of nodes in the road graph.

In order to better evaluate the quality of our solution, it would also be necessary to test it against other methods using the same reference problem. To better evaluate the quality of solutions, better metrics need to be considered, and, the algorithm needs to encompass constraints, such as, bus capacity and different passenger destination.

When it comes to, having a fully functional Ant Colony System, that is able to design close to optimal bus routes when given the distribution of passengers in a city, our work is still in the initial stages. As future work, finding ways for the Ant Colony Algorithm to operate multiple buses, or improving the method described in our work, would be a major step towards having an acceptable solution.

To improve our multi-bus approach, a good start would be to find an heuristic to partition the main network in a way that better results are produced.

REFERENCES