Real-Time Scheduling for a Variable-Route Bus System

Real-Time Scheduling For a Variable-Route Bus System

FeiYang Zhu
Department of Applied Mathematics
Harvard University

Supervisor
Margo S. Levine

In partial fulfillment of the honors requirements
Bachelor of Arts in Applied Mathematics
March 29, 2019

Acknowledgements

I would like to thank my advisor, Professor Margo Levine, for her support and guidance throughout this endeavor. Without her, this thesis could not have been possible.

I would also like to thank Professor Scott Kominers for providing helpful feedback on the overall framework and soon-to-be Prof. Haifeng Xu for his guidance in the theoretical model.

Last but not least, I would like to thank my family and friends for their support and friendship throughout this process.

Abstract

Traditional buses operate on an outdated fixed-route system that is often characterized by inefficiencies such as trips to empty bus stops and long waiting time at the station. However, the prevalence of personal mobile devices provides more opportunities than ever for passengers to interact with the buses, giving rise to the possibility of personalized travel routines. In this thesis, we propose a variable-route system that can customize and optimize bus routes in real time based on the passenger requests and their characteristics. Using a computer simulated example, we demonstrate that the variable routing system reduce the total waiting time on bus stop and the total travel time on the buses by a significant margin. Furthermore, we show that by leveraging genetic algorithm, we can reduce the computation time by as much as 90% while still maintaining a significant reduction in waiting time and traveling time. This thesis provides an initial performance check for the optimal bus route scheduler and highlights an implementation that can achieve significant time reduction with much less computation time.

Contents

Chapter 1

Introduction

Bus is a key mode of transportation for city residents and visitors. According to American Transportation Association, nearly 6 million bus trips were taken in the U.S. in 2017 alone. Ridership concentrates further for metropolitan city such as New York, Boston, and Chicago. In these regions, 20% of the adults use public transportation on a daily or weekly basis. [1]

For city dwellers, buses provide cheap and convenient alternatives to driving. For local government, buses provide a versatile form of public transportation with the flexibility to serve a variety of access needs and an unlimited range of locations throughout a metropolitan area. Infrastructure investments needed to support bus service can be substantially lower than the capital costs required for subway systems. As a result, bus service can be implemented cost-effectively on routes where ridership may not be sufficient or where the capital investment may not be available to implement subway systems.

Despite the inherent advantages of bus service in terms of flexibility and low capital cost, the public frequently finds the quality of bus service provided in urban centers to be lacking. Bus routes in major cities were designed around the cities as they once were, with lines connecting to streetcars, busy wharves,

and big businesses of the time. As the cities evolved in their layout and demographic distribution, their bus routes remain the same. Riders often experience sluggish service delayed by unnecessary detours to empty bus stops and frequent transfer between two popular destinations that should have been in the same route. Moreover, compared to subway systems, the advantageous flexibility and decentralization of bus operations also result in a lack of system visibility and permanence that contributes to public perceptions of unreliability and disorganization.

The goal of this thesis is to propose a bus-scheduling system where city governments can maintain their current infrastructure while creating more intelligent and efficient bus routes that reduce the overall travel time and wait time. Here, we introduce a variable-route system that can customize and optimize bus routes in real time based on the incoming requests and their respective characteristics. Using a computer simulated example, we demonstrate that the variable routing system reduce both the total wait time on bus stop and the total travel time on the buses by a significant margin. Furthermore, we show that by leveraging genetic algorithm, we can reduce the computation time by as much as 90% while still maintaining a significant reduction in wait time and travel time. This thesis provides an initial performance check for the optimal bus route scheduler and highlights an implementation that can achieve significant time reduction with much less computation time.

1.1 Motivations

Traditionally, buses operate on a fixed-route system where they pass through a series of predetermined stops. Thus, determining where to put the bus stops and how frequent they should be visited are the central concerns for the route

planners. Planners gather information on population influx and outflux at each region throughout the day and track the origins of the incoming population and destination of outgoing population. In addition, planners predict the demographics of the metropolitan area in twenty to thirty years (population, employment, density, traffic congestion etc) using complex modeling software that begin to operate forward from the present using different baseline scenarios. With those information, route planners would decide on the number of bus stops they would place in each region, how frequently the bus stops would be visited and which bus stops would belong to the same route. From those information, a network of fixed bus routes is designed.

The main problem with today's fixed-route bus system is that it's inflexible to the changing demand throughout the day. For example, a stop might have a lot of passengers who are taking the bus to work in the morning, but the same stop might have a lot passengers who are taking the bus to lunch or grocery shopping in the afternoon. In another word, the popular destination of the stop changes throughout the day. In that case, a fixed-route system would decide either to accommodate both spots for work and lunch in the same route or to maintain two separate lines for going to lunch and to work. This leads to a trade-off between travel time and the number of transfers. The less stops that a bus serves, the shorter is its travel time but the more likely its passenger would need another transfer to get to his/her destination, which leads to longer wait time at the station. With a fixed-route system, there is no satisfying solution that can mitigate this trade-off between travel time and wait time as a result of the transfers.

This thesis aims to develop a new bus-routing system that can reduce the travel time, the wait time and the amount of transfers the passenger need to take without incurring a significant renovation cost for the government. There are

several features that we want the system to incorporate. First, it should avoid sending buses to stops that are either empty or far away from where the buses are. Second, it should be flexible enough to accommodate the shift in the nature of the requests throughout the day. Furthermore, it should be able to pick up and drop off the passenger in one seating (i.e. no transfer). Lastly, it should provide a clear time interval for pick up and drop off for the passengers. If accomplished, this system can drastically improve the convenience and reliability of buses while minimizing the cost of renovation for the local government.

1.2 Literature Review

Different form of routing problem has been studied extensively. Form the classic traveling salesman problem where the objective is to minimize total costs while visiting a finite number of places, to the more modern vehicle routing problem where the objective is to deliver as many items as possible in the city under a capacity constraints, routing problem has always been a central focus for scholars who are interested in operation research. After the dazzling success of ride-sharing platform such as Uber and Lyft, the interest in the field grows even bigger. Yet, while everybody tries to develop the next algorithm to optimize Uber by another one over gazillion second, little has been done on the side of public transportation. This paper attempts to apply some of the existing optimization techniques to public bus system and propose a more flexible bus system that is better suited to the needs of the riders.

The particular form of the routing problem that the paper considers is called the pickup and delivery problem (PDP). In this routing problem, vehicles pickup certain loads at a set of pick-up locations and deliver the loads at their respective drop-off locations. The goal is to schedule the minimum costs routes for the

vehicles. In addition, the requests can specify certain requirements as load and time windows.

Similar to other routing problems, PDP is NP-hard. While small to medium sized instances can be solved with exact algorithm, heuristics would be needed for a larger problem. Studies considering exact solution methods for the pickup and delivery problem centers around branch and cut or column generation. (Parragh et al., 2008b) while studies considering heuristics remain large inconclusive, with methods ranging from large neighbor-hood search to greedy randomized adaptive search (Parragh et al., 2008b).

Two main approaches for solving the pickup and delivery problem are branch-and-price and branch-and-cut. Cordeau and Iori M.(2010) proposed a branch-and-cut algorithm for the single vehicle pickup and delivery problem. Using a combination of new formulation and a tabu heuristic, they were able to solve an instances up to 17 requests within a reasonable computing time. However, the method is limited to single vehicle scheduling and not sufficient to solve our multi-vehicle routing problem. Similarly, Ropke and Cordeau (2008)'s branch-and-price algorithm is limited to single vehicle. Though it significantly outperforms its predecessor by considering instances of 100 and 500 requests in their paper, the efficiency of the algorithm is non-translatable to the problem for multiple vehicles. In addition, Hernandes-Perez and Salazar-Gonzalez (2004) proposed a 0-1 integer linear model for the single vehicle PDP with the goal of minimize total travel costs, which they improved on subsequently in 2007. and Renaud et al. (1996) considers the single vehicle problem with a double insertion construction heuristic improved by deletion and reinsertion. For the most parts, expertise have been focused on solving single vehicle problem due to the complexity of the a multiple-vehicle dynamic problem.

1.3 Overview of the Thesis

The remainder of the thesis is as follow: Chapter 2 provides the specification of the problem and discusses important details in modeling requests and buses. Chapter 3 discuss the construction of a fixed-route bus system. Chapter 4 provides a theoretical framework to solve the routing problem with mixed integer linear programming. Chapter 5 discusses an approximation technique called genetic algorithm that can generate close to optimal solution while using a fraction of the time that's required of MILP. Chapter 6 compares the performance of the MILP model and heuristics to a fixed-route bus system on a computer simulation. Chapter 7 offers some concluding thoughts.

Chapter 2

Problem Specification

Just like Uber and Lyft, we want to design a routing system that allows all potential riders to request buses that can customize their routes based on riders' pick-up and drop-off locations. Before heading to the bus stop, a potential rider would first place a request on a mobile device to specify the his/her pick-up and drop-off bus stops. The rider can choose to be pick-up now or at a later time that he/she specifies. Then, the mobile app would return the bus assignment information along with estimated arrival time and estimated trip time. Once a rider is assigned to a bus, we assume such assignment is final and immutable.

In the duration of the trip, a bus may change its course based on more incoming requests, but must adhere to the constraints for the pick-up and drop-off time for the already assigned riders. A bus must start from one of the depots and end at one of the depots where the starting and ending depot do not have to be the same. Finally, a bus may not exceed its seat capacity and gasoline reserve at any point of the trip and every bus has the same seat and gasoline capacity.

Below, we will provide a specification for the problem and discusses each component in details in the following section.

The routing problem is consisted of 3 components G, R, B where:

- $G = (V, E, T)$ represents the graph in which the buses will travel on. V corresponds to the set of potential stopping locations. E corresponds to the set of arcs between each pair of node. T represents the set of time to traverse each arc. This will be expanded in section 2.2
- $R \ni r =< p_r, d_r, [e_r^p, l_r^p], [e_r^d, l_r^d] >$ represents the set of requests, including pick-up location, drop-off location, time interval for pick-up and time interval for drop-off. This will be expanded in section 2.3
- $B \ni b =< a_b, T_b, Q_b, R_b >$ represents the set of buses, including the starting location, maximum trip time, seat capacity and targets assigned. This will be expanded in section 2.4.

2.1 Modeling Graph

We model the request network with a directed graph $G(V, E, T)$. Each potential stopping location is denoted by a vertex, $v \in V$. The shortest arc connecting vertex i to j is denoted by an edge $(i, j) \in E$. Finally, each arc (i, j) is associated with a time of traverse $t_{ij} \in T$.

The set of vertices V is consisted of the locations of the depots (where the buses begin and end its daily service) , the requests' pick-up locations and the requests' drop-off locations and the starting locations of the buses. Since we only need to optimize the route based on the set of unreached stops, we can further reduce the set of nodes V where $V = V_s \cup V_g \cup \{a_b | b \in B\}$. V_s denotes the set of locations that have not be reached by the buses yet, V_g denotes the set of depots, and $\{a_b | b \in B\}$ denotes the set of starting locations for the buses, which will explain more in the next section. This drastically reduces the size of the original graph and improves the efficiency of real-time route optimization.

Furthermore, we constraint the edges so that there is no self-connecting node and the head of any edge must not be a depot. This further reduces the size of the problem. Note also that the shortest path definition fixates the route that the buses can take between any pair of vertices. Given the complexity of the problem, we are willing to trade the flexibility of bus route for efficiency.

2.2 Modeling Requests

Let R denotes the class of requests. For our purpose, we would only need to consider the set of active requests (i.e. excluding historical requests that have already been served). Each $r \in R$ consists of a tuple of six parameters $< p_r, d_r, [e_r^p, l_r^p], [e_r^d, l_r^d] >$. For $r \in R$, $p_r \in V$ denotes the request's pick-up location and $d_r \in V$ denotes the drop-off location. We assume the pick-up location and drop-off location is chosen from a set of bus stops, S. Note that S is different from the set of bus stops waiting to be served, V_s. Whereas each element in S represents a unique bus stop location, V_s represents the set of bus stops that are waiting to be pick up and can have repeated locations. $[e_r^p, l_r^p]$ denotes the possible range of pick-up time for the request, where e_r^p denotes the earliest possible pick-up time and l_r^p denotes the latest possible time. Unless the request is placed for a later time, this is usually determined by adding the time at which the request is placed to a fixed range, such as (time of request generation, time of request generation + 10 minutes). Similarly, $[e_r^d, l_r^d]$ denotes the possible range of drop-off time for the request.

2.3 Modeling Buses

Let B denotes the class of buses, Each $b \in B$ is consisted of a tuple of four parameters $< a_b, T_b, Q_b, R_b >$. a_b denotes the location of bus b at the beginning

of every optimization period. T_b denotes the maximum time that the bus can travel without needing a refuel, Q_b denotes the maximum capacity in bus b, and R_b denotes the set of requests that are assigned to the bus. In particular, $R_b = R_b^n \cup R_b^s$, where R_b^n is the set of requests that have been assigned but haven't been picked up, and R_b^s is the set of requests that have been assigned and picked up. The set of requests R_0 denotes the requests that have not been assigned.

Chapter 3

Fixed-Route Model

Because we use computer simulation instead of real world data in this paper, we can not leverage the real world bus routes to formulate a fixed-route model for performance comparison. As such, we need to define our own fixed-route model given a simulated set of bus stops and depots.

Given the set of bus stops of size $|S|$ and depots of size $|V_g|$, we would like to generate a set of bus routes that are representative of the bus network in the real world. By observing bus networks of major cities like New York, Boston and Chicago, we observed three main characteristics regarding the buses:

1. Routes tend to begin and end at the same depot and do not traverse any other depots en-route

2. Routes tend to localize to a particular region of the city

3. With transfer, there is a way to travel from a bus stop to any other stop in the map

We should aim to develop the network of fixed bus routes such that it fits these three observations. Observation 1 can be easily satisfy by ensuring the buses to begin and end at the same depot. To satisfy observation 2, we utilize the idea from k-means clustering. k-means clustering states:

Given a set of objects, k-means algorithm partitions the objects into a predefined number of k clusters such to minimize the within-cluster sum of squares.

In plain words, this means that the algorithm groups the closest objects together in k clusters. Intuitively, the group of stops that form a bus route should be close to each other so that the time it takes to finish a route is reasonably short. Assume that we have k routes to plan for. the first objective is to find a way to partition the bus stops into k groups such that the sum of the distance within each group is minimized. This is exactly the purpose of the k-mean algorithm. Assuming that each depot is corresponded to one and only one route, we will apply the k-mean algorithm to partition the bus stops into $|V_g|$ clusters.

Furthermore, we want to satisfy observation 3. To ensure a way to travel from any stop to any other stop, we simply need to ensure that there is at least one stop for every pair of clusters to serve as the "transfer" stop between the pair. Transfer stop should be as close to both clusters as possible to minimize the detours that we need to add to the system. Thus, for each pair of clusters, A and B, we want to find a pair of stops (a, b) where $a \in A$ and $b \in B$ such that the distance between (a, b) is the shortest $\forall a \in A, \forall b \in B$. Subsequently, we would add a to cluster B, and b to cluster A. We do the same for every pair of clusters.

Now that we have our clusters of bus stops, we can form the routes. For each cluster, we assign it to the closest depot that's not already taken by another route. Starting and ending with the depot, the route is constructed iteratively such that the closet stop to the tail is appended to the route. For example, given that the first stop in the route is the depot, the second stop would be the closest stop to the depot that has not been assigned. Similarly, the third stop would be the closest stop to the second stop that's yet been assigned. The process goes on until we assign every stop in the clusters to the route. The last stop in the

route will then be joined with the depot, forming a cycle that characterized the fix route.

Chapter 4

Mixed-Integer Linear Programming Model

In this chapter, we will model the problem with mixed-integer linear programming (MILP). First, we will define the variables, the objective function and the constraints of the model for the static routing problem. Next, we will discuss adaptations that would need to be made to match the dynamic problem.

4.1 Variables

We define x_{ij}^b to indicate whether bus b would travel along the direct edge ij from i to j.

$$x_{ij}^b = \begin{cases} 1, & \text{if bus b plans to traverse (ij)} \in E' \\ 0, & \text{otherwise} \end{cases}$$

We define y_r^b to indicate whether request r is assigned to b.

$$y_r^b = \begin{cases} 1, & \text{if rider r is assigned to bus b} \\ 0, & \text{otherwise} \end{cases}$$

We further define g_i^b to indicate variable whether bus b plans to end its service at depot $i \in V_g$.

$$\mathrm{g}_i^b = \begin{cases} 1, & \text{if bus b plans to refuel at the depot at vertex i} \in V_g \\ 0, & \text{otherwise} \end{cases}$$

We also define t_i^b to denote the expected time that bus b will arrive at vertex i, q_i^b to denotes the number of passengers in bus b when it's at vertex i.

4.2 Objective Function

To achieve an optimal system of bus route, we aim to reduce the overall trip costs across all bus routes as much as possible while accommodating as many riders as possible. We capture this object with the objective function below. λ is a balancing constant.

$$\min \sum_{i,j \in V', b \in B} t_{ij} x_{ij}^b - \lambda \sum_{r \in R, b \in B} y_r^b \tag{4.1}$$

4.3 Request Assignment Constraints

To ensure the behavior of the model follows in accordance to a system of buses, we define the following constraints. First, if a rider has been assigned to bus b, then bus b needs to pick up the passenger at vertex p_r and drop him/her off at vertex d_r. Graphically, this implies that there must be an outgoing edge from p_r

and an incoming edge to d_r. We formulate the constraints below:

$$\sum_{j \in V} x^b_{p_r j} = y^b_r, \forall r \in R^n_b \cup R_0, \forall b \in B \tag{4.2}$$

$$\sum_{j \in V} x^b_{j d_r} = y^b_r, \forall r \in R, \forall b \in B \tag{4.3}$$

4.4 Route Constraints

Next, we want to constrain the edges so that a viable route can be formed. Except for the starting point and the ending point of the bus route, each incoming edge to vertex must match with an outgoing edge from that vertex. For the starting point, a_b, the net outflow (the number of outgoing edges minus the the number of incoming edges) equals to 1 if bus b has been assigned with some requests and equals to 0 otherwise. Similarly, for the ending point, the net inflow (the number of incoming edges minus the the number of outgoing edges) equals to 1 if bus b has been assigned with some request and equals to 0 otherwise. The constraints are as follow:

$$\sum_{j \in V} x^b_{a_b j} - \sum_{j \in V} x^b_{j a_b} = \sum_{r \in R} y^b_r, \forall b \in B \tag{4.4}$$

$$\sum_{j \in V} x^b_{ji} - \sum_{j \in V} x^b_{ij} \leq \sum_{r \in R} y^b_r, \forall i \in V_g, \forall b \in B \tag{4.5}$$

$$\sum_{j \in V} x^b_{ji} - \sum_{j \in V} x^b_{ij} = 0, \forall i \in V \setminus V_g \cup \{a_b | b \in B\}, \forall b \in B \tag{4.6}$$

Constraint (4) (5) used $\sum_{r \in R} y^b_r$ instead of 1 to better account for the case that bus b is not assigned. If bus b is not assigned to any route, $\sum_{r \in R} y^b_r = 0$,

forcing net outflow in (4) and net outflow in (5) to 0. In addition, we want to ensure that a bus will end at a depot if it's been assigned to some requests.

$$\frac{1}{M}\sum_{r\in R} y_r^b \leq \sum_{i\in V_g} g_i^b \leq 1, \forall b \in B \tag{4.7}$$

$$\sum_{j\in V} x_{ji}^b = g_i^b, \forall i \in V_g, \forall b \in B \tag{4.8}$$

Constraint (7) ensures that an assigned bus will end at exactly one depot. Constraint (8) ensures that the assignment of depot will translate to an assignment of an incoming edge to that depot. Together with constraint (5), constraints (7) and (8) ensure proper vehicle behavior at the depot.

4.5 Time Constraints

Next, we need to establish the time constraints. Intuitively, if edge (i, j) will be traverse by some bus b, then the the arrival time at vertex j must be no earlier than the arrival time at vertex i plus the time to traverse the edge (i, j). The arrival time for each starting point a_b is 0 by construction.

$$t_{a_b}^b = 0, \forall b \in B \tag{4.9}$$

$$t_j^b \geq t_i^b + t_{ij} - M(1 - x_{ij}^b), \forall i, j \in V \setminus \{a_b | b \in B\}, \forall b \in B \tag{4.10}$$

To ensure a reasonable amount of waiting time, the bus must arrive at the pick up location within the pick up time interval. Similarly, the bus must arrive

at the drop off location within the drop off time interval.

$$e_r^p - M(1 - y_r^b) \leq t_{p_r}^b \leq l_r^p + M(1 - y_r^b), \forall r \in R_b^n \cup R_0, \forall b \in B \tag{4.11}$$

$$e_r^d - M(1 - y_r^b) \leq t_{p_r}^b \leq l_r^d + M(1 - y_r^b), \forall r \in R, \forall b \in B \tag{4.12}$$

4.6 Fuel and Capacity Constraints

Finally, we need to ensure that all buses will meet the gas and seat capacity constraints at all times. For any assigned location, the arrival time can not exceed the time threshold permitted by the available gasoline, T_b.

$$0 \leq t_i^b \leq T_b, \forall i \in V, \forall b \in B \tag{4.13}$$

Similarly, the number of riders can not exceed the seating capacity Q_b.

$$q_i^b \leq Q_b, \forall i \in V, \forall b \in B \tag{4.14}$$

Next, we need to model the inflow and outflow of riders at each bus stop. At bus stop j, the number of passengers on the bus should equal to the number of passengers at the most recently visited location plus the number of incoming passengers and minus the number of outgoing passengers.

$$q_j^b = q_i^b + \sum_{\{r \in R | p_r = j\}} y_r^b - \sum_{\{r \in R | d_r = j\}} y_r^b - M(1 - x_{ij}^b), \forall ij \in V, \forall b \in B \tag{4.15}$$

Lastly, we want to ensure that the bus has no more passengers when it reaches

the depot.

$$q_i^b \leq M(1 - g_i^b), \forall i \in V_g, \forall b \in B \tag{4.16}$$

When $g_i^b = 1$, RHS of the inequality (16) drops to 0, thus forcing the number of passengers to 0.

4.7 Model Summary

Our model can be summarized as follow:

$$\text{minimize} \sum_{i,j \in V', b \in B} t_{ij} x_{ij}^b - \lambda \sum_{r \in R, b \in B} y_r^b$$

$$\text{subject to constraints (4.1) - (4.16)}$$

$$\text{over } \mathrm{x}_{ij}^b \in \{0,1\}, y_r^b \in \{0,1\}, g_i^b \in \{0,1\}, t_i^b \in \mathbb{R}^+, q_i^b \in \mathbb{Z}^+$$

Recall that in every re-optimization period, the assignment of bus request must be immutable. To account for this, we will fixate the current period value of y_r^b for the next period. During implementation, we can achieve a similar effect by exporting the value of y_r^b after every optimization period, re-importing them in the next period as parameter z_r^b and setting the next period y_r^b equals to the available z_r^b. We will add the constrain below:

$$y_r^b \geq z_r^b, \forall r \in R, \forall b \in B \tag{4.17}$$

Note that the greater and equal sign ensures that only z_r^b with values of 1 will get assigned during re-optimization. This constraint is crucial in that it keeps track of previous assignments and enables us to apply our model to dynamic problem such as this one.

Our final model becomes:

$$\text{minimize} \sum_{i,j \in V', b \in B} t_{ij} x_{ij}^b - \lambda \sum_{r \in R, b \in B} y_r^b$$

subject to constraints (4.1) - (4.17)

over $\mathrm{x}_{ij}^b \in \{0,1\}, y_r^b \in \{0,1\}, g_i^b \in \{0,1\}, t_i^b \in^+, q_i^b \in^+$

Chapter 5

Genetic Algorithm

In theory, the previously proposed MILP model should yield the optimal bus-routes assignments. In practice, however, solving such model with exact algorithm is known to be NP-hard [2] and computationally intensive when the problem size is large. If we want to apply the model in a real city to serve thousands of active request at any given time, it's crucial for us to reduce the computational complexity of the problem, even if it meant to sacrifice some performance. In this chapter, we consider a popular approximation algorithm, genetic algorithm, to reduce the run time of the program. First, we will provide a high level overview of the ideas behind the algorithm, then we will describe the adaptation to the problem.

5.1 Genetic Algorithm Overview

Genetic algorithm (GA) is an optimization technique that iteratively improve the sub-optimal solutions based on ideas from reproduction and natural selection. In nature, organisms with better genetic traits are able to survive longer and pass down the favorable traits to their offspring. Their children, who posses similar

genetic traits as they are, are also likely to survive longer and pass down the genetic traits to the grandchildren. Over many generations, this favorable genetic materials propagates to an increasing number of individuals through natural selection.

GA simulates this process of biological reproduction and natural selection on a population of feasible solutions to a constrained maximization problem. At the beginning of the algorithm, an initial population of feasible solution is generated. Since the solutions do not have to be optimal, the process of generation would be relatively quick. Each feasible solution is represented as a **chromosome** with a certain **fitness** value computed by a **fitness function**. After an initial population is generated, the algorithm will engage in a series of steps to try to manufacture solutions with higher fitness scores. The process ends once the algorithm can not generate any solution with higher fitness score than the highest score in the current population. Below, we will describe the improvement process in detail.

First, the algorithm selects the feasible solutions for breeding and matches them in pairs in the selection phase. The highly fitted chromosomes will have a higher chance of being selected, imitating the natural phenomenon that fitted individuals are more likely to win in the competition for mating. Next, selected individuals breed within their pairs in the crossover phase. Each pair produces two children. For each pair, the individuals have a certain probability, p1, of engaging in an operation called **crossover**. If the pair engages in crossover, each of its child will contain a mixture of each parent's gene. Otherwise, one child will completely inherit one of its parent's gene, and the other will completely inherit that of the other parent. I.e. the children will simply be duplicates of their parents. The probability of crossover is called the crossover rate. After the crossover phase, each child undergoes mutation with a small probability, p2. The mutation changes the genetic setup of the child for better or worse. Finally, the

children are integrated with the old population to generate a new generation, and the improvement process repeats until the algorithm can no longer see significant improvement in the subsequent children.

In the follow sections, we will adapt GA to approximate the optimal solution to our MILP model. Since our MILP model is a dynamic program as a result of constraint (3.17), the approximated solution of GA on MILP model would also be dynamic and does not need to be otherwise adjusted.

5.2 Initial Population

We will generate the initial population in two steps for clarity. Given the set of active requests, R and available buses, B, we will first assign the number of requests each vehicle will ahve without specifying requests themselves. Figure 5.1 shows an example of the assignment for a problem with 8 active requests, 6 buses and 2 depots. Based on the naming convention aforementioned in Chapter 2, the 2 depots are $< -1, -2 >$ and the 8 active requests are $< (1, 9), (2, 10), (3, 11), (4, 12), (5, 13), (6, 14), (7, 15), (8, 16) >$ where the first number in each tuple denotes the pick-up location and second number in each tuple denotes the drop-off location.

Buses	B1	B2	B3	B4	B5	B6
Requests	3	2	0	0	2	1

Figure 5.1: An example of step 1 in initial population generation

Next, we want to specify the requests each bus was assigned to. The quickest approach would be to find a feasible solution to the MILP model that also satisfies the bus assignment in figure 5.1 Thus, let n_b denotes the number of requests that bus b is assigned to in the assignment above, we can write the constraints that

specifies the above assignment:

$$\sum_{r \in R} y_r^b = n_b, \forall b \in B \tag{5.1}$$

This constraint dictates the number of requests each bus must assign to. Given constraint (4.1) and the previous MILP model, we can generate a feasible solution. Again, run-time should not be an issue since we are only solving for a feasible solution instead of an optimal solution. Figure 5.2 shows an example of such feasible solution.

B1	a_{b1}	1	9	6	2	10	14	-1
B2	a_{b2}	3	8	11	16	-1		
B3	a_{b3}	-1						
B4	a_{b4}	-2						
B5	a_{b5}	7	5	13	15	-1		
B6	a_{b6}	4	12	-2				

Figure 5.2: An example of step 2 in initial population generation

We repeat this process N times to generate the initial population.

5.3 Selection Phase

After generating a set of N initial feasible solutions, we will choose a subset of the solutions for breeding based on their fitness. Intuitively, the best solution is the one that minimize total trip time while serving as many requests as possible. This condition is precisely captured by the minimization of objective function in the MILP model. Thus, leveraging the objective function in the MILP model, we can evaluate the fitness of a feasible solution based on how small its objective value

is. For simplicity of notation, we define the fitness function $f_k(x_{ij}^b, y_r^b)$ where:

$$f_k(x_{ij}^b, y_r^b) = -\sum_{i,j \in V', b \in B} t_{ij} x_{ij}^b + \lambda \sum_{r \in R, b \in B} y_r^b \forall k \in \{0, N-1\} \tag{5.2}$$

The RHS is a negation of the objective function in MILP. We need to flip the sign here because the fitness function is a maximization problem but the objective function from MILP model is a minimization problem.

Given a feasible solution, $f_k(x_{ij}^b, y_r^b)$ completely specify the fitness score for solution k. Since the goal is to find a solution the has the maximum fitness score, the most fitted model with the highest probability of being chosen for breeding is the one that has the highest fitness score. We can define the probability function $P_k(f_k|F) = \frac{f_k - min(F)}{max(F) - min(F)}$, where $F = \{f_b \mid \forall i \in 0, N-1\}$ is the set of fitness score for all N feasible solution.

The probability function is a normalization of the fitness score that bounds it in the interval 0,1. Using the probability of admit for each solution, we run a Bernoulli trial for each to determine a list of solutions to breed. Noted that since we require an even number of solutions to form pairs with each other, if the list of admitted solution has an odd number of element, we will randomly remove 1 element. Finally, we randomly shuffle the list and take consecutive elements to form the list of breeding pairs.

5.4 Crossover Phase

In the crossover phase, we want to perform a crossover between parts of the solutions within each pair with some probability , p1. The idea is to mix and match the strong solution candidates to produce a "super-breed" that's extremely close to the optimal solution. Given the parent solutions pair, s1 and s2, if the Bernoulli trial for crossover failed, then the children solutions, c1 and c2, will

simply be a replica of s1 and s1. However, if the Bernoulli trial succeed, the children will be produced by the following crossover procedure:

For children 1, inherit 2 randomly selected bus-routes from parent 1. To illustrate this operator, consider the following example:

Parent 1

B1	a_{b1}	1	9	6	2	10	14	-1
B2	a_{b2}	3	8	11	16	-1		
B3	a_{b3}	-1						
B4	a_{b4}	-2						
B5	a_{b5}	7	5	13	15	-1		
B6	a_{b6}	4	12	-2				

Parent 2

B1	a_{b1}	-2						
B2	a_{b2}	**5**	**13**	4	12	3	11	-1
B3	a_{b3}	**6**	**14**	-1				
B4	a_{b4}	**7**	**15**	-1				
B5	a_{b5}	**1**	**2**	**9**	**10**	-2		
B6	a_{b6}	8	16	-1				

Assume that B2 and B6 are the two random routes selected from parent 1, they will be added to child 1 as its B2 and B6 route.

Child 1

B2	a_{b2}	3	8	11	16	-1
B6	a_{b6}	4	12	-2		

The remaining requests that have not been included in Child 1 are shown in bold in Parent 2. From the four routes that are bold, Child 1 will select two bus routes to add to its network in the order of importance.

First, the route must not be taken in Child 1's graph. In another word, Child 1 can not choose route B2 or B6 from parent 2 since those were already assigned. That leaves us with B3, B4 and B5 to choose from. Next, choose two routes that have the most bold requests from the remaining routes of parent 2. From each of the 2 routes, add the subset of bold requests into child 1's solution. For example, if child 1 were to added B2 from parent 2, it can only integrate the subset ¡(5,13)¿ instead of the whole string. If there is a tie, choose one randomly. This leads us to choose B5, and randomly select between B3 and B4. Assume that we randomly choose B3, child 1 becomes the following.

Child 1

B2	a_{b2}	3	8	11	16	-1
B3	a_{b3}	6	14	-1		
B5	a_{b5}	1	2	9	10	-2
B6	a_{b6}	4	12	-2		

Lastly, we have two unassigned requests $< (5, 13), (7, 15) >$ and 2 unassigned buses B1 and B2. We leverage the method that we use to generate the initial population to solve for a feasible assignment of the 2 request to the 2 remaining buses. The feasible result is to assign all of the remaining two requests to bus 1 in the order of $< 7, 15, 5, 13 >$. An example of the result is shown below.

Child 1

B1	a_{b1}	7	15	5	13	-1
B2	a_{b2}	3	8	11	16	-1
B3	a_{b3}	6	14	-1		
B4	a_{b4}	-1				
B5	a_{b5}	1	2	9	10	-2
B6	a_{b6}	4	12	-2		

We do the same thing for child 2 as well to arrive at its solution.

5.5 Mutation Phase

After a child is generated, the child will undergo mutation with a small probability of p2. Here, a mutation occurs by re-assigning the requests of 2 buses. The idea is to introduce non-negligible perturbation to the network with small probability while maintaining part of the previous assignment. For example, assume the child vehicle consisted of the following buses.

Now assume that buses B2 and B3 were selected for re-arranging. Keeping all other assignment constant, the requests belonging to them will now be reassigned according to the constraints in chapter 4.

Requests that need to be reassigned: $< (3, 11), (8, 16), (6, 14) >$

Child 1

B1	a_{b1}	7	15	5	13	-1
B2	a_{b2}	3	8	11	16	-1
B3	a_{b3}	6	14	-1		
B4	a_{b4}	-1				
B5	a_{b5}	1	2	9	10	-2
B6	a_{b6}	4	12	-2		

Buses that are available: $< B2, B3 >$

Using the subset of requests and buses above, we find a feasible solution to constraints (4.1) - (4.17). Again, the run-time should be acceptable given the relatively small size of the problem. The final solution is constructed (example given below):

Child 1 (after mutation)

B1	a_{b1}	7	15	5	13	-1
B2	a_{b2}	11	16	-1		
B3	a_{b3}	3	6	8	14	-1
B4	a_{b4}	-1				
B5	a_{b5}	1	2	9	10	-2
B6	a_{b6}	4	12	-2		

5.6 Integration Phase

After all selected parent pairs go through crossover phase and mutation phase, the newly generated children will be evaluated in terms of their fitness score. Since the higher score indicates a better performance, if the highest fitness score among the children is lower than that of the older generation, the algorithm will terminate because it can no longer improve the solution space. Otherwise, if there's improvement, the children solution will be integrated into the population to form the new generation, and the algorithm will run again on the new generation.

Chapter 6

Experimental Results

In this chapter, we implement three bus models (fixed-route, MILP and GA) on a computer simulated city with dynamic requests. We compare their performances on three different metrics: total travel time, total waiting time and total run time. Intuitively, we would want the program to have a minimal total travel time and total waiting time to minimize the inconvenience of the passenger while maintaining a reasonable run-time to allow for real-time optimization.

6.1 Simulation Design

To model the real world as accurately as possible, we design the simulation with the following steps in mind. Below is a flow chart that shows a condensed version of our simulation design

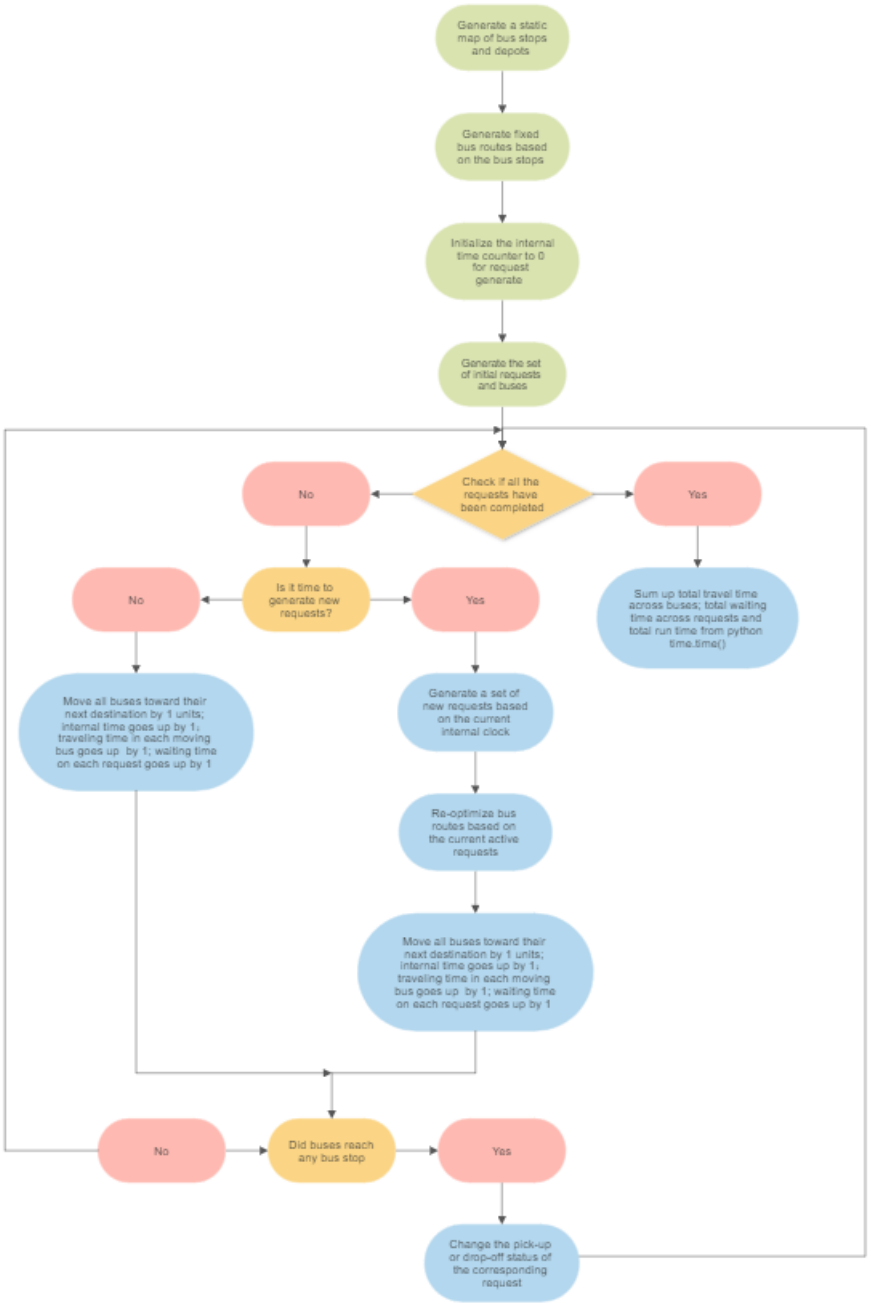

Figure 6.1: Simulation Design

Green blocks correspond to actions in static phase before the buses begin to run. Blue blocks corresponds to action in dynamic phase while the buses run. Yellow blocks correspond to the major check-points in various part of the simulation and red blocks indicate the direction in which the program will go after the checkpoints. Note that the only difference between the three models is at the 2nd blue block in the middle lane from the top, where they need to re-optimized based on the set of active requests. While the fixed-route system simply direct each bus to its next stop in the route, MILP and GA re-solve the whole problem to arrive at a potentially new set of bus routes. In the section below, we will discuss the static phase and dynamic phase of the simulation in detail.

6.1.1 Static Phase

To simulate map of possible locations, we begin by generating a set of bus stops of size S and a set of depots of size D. The coordinates of the bus stops are generated at random within a rectangular box that represents the boundary of the city. Within the rectangular box, a smaller concentric rectangular box represents the city center, making the outer boundary the outskirts of the city. Realistically, bus stops tend to cluster in the city center because of the higher population density and depots tend to locate at the outskirt of the city because of lower rent. Furthermore, depots tend to be far apart from each other to maximize efficiency. To simulate this distribution of locations, we generate 70% of the bus stops within the city center, 30% within the outskirts and one depot at each corner of the simulation, which essentially sets the size of depot D to 4. An illustration of the bus map are give below.

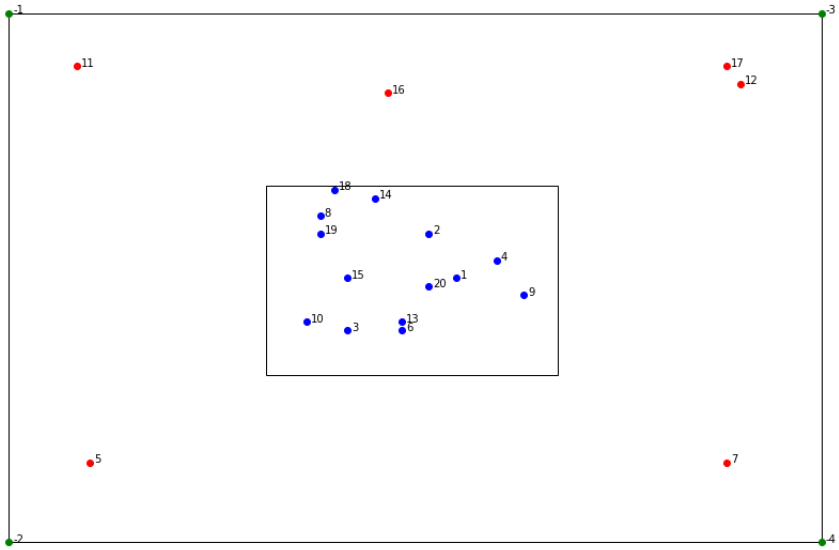

Figure 6.2: An example of the bus routes

The larger rectangle represents the boundary of the city while the smaller one represents the boundary of the city center. The green dots represent the depots of the network, the red dots represent the bus stops that are outside of the city center and the blue stops represent the bus stops that are inside the city center. For each pair of points i, j in the map, we compute the Euclidean distance between them d_{ij}. Here we assume that buses are traveling in Euclidean distance instead of using a more sophisticated distance measure that accounts for factors like road conditions and traffics for simplicity. We also assume that buses are traveling at a constant speed of 1 unit of distance per unit of time. Thus the time it takes to travel from point i to j, t_{ij}, is equal to d_{ij}.

Using method of constructing a fixed route from Chapter 4, we can construct a network of bus routes for the given requests:

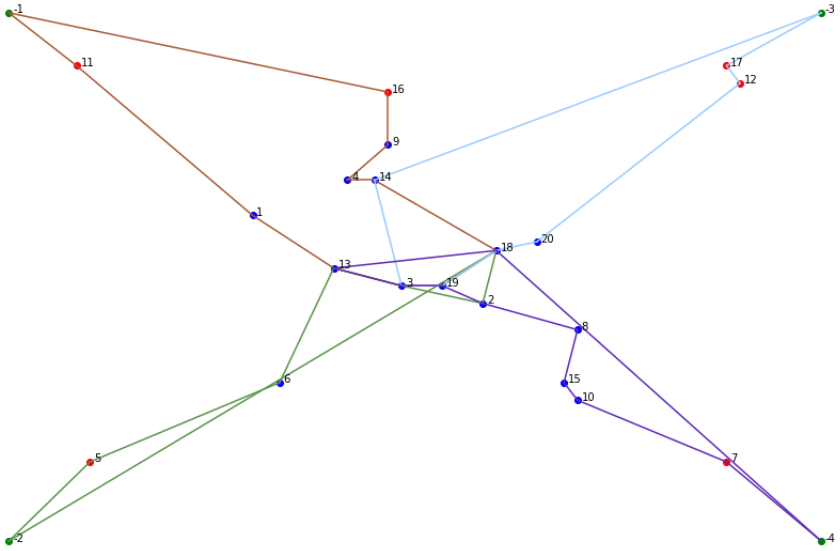

Figure 6.3: An example of the bus routes

$R =< p_r, d_r, [e_r^p, l_r^p], [e_r^d, l_r^d] >$ represents the set of requests, including pick-up location, drop-off location, time interval for pick-up and time interval for drop-off. This will be expanded in section 2.3

$B =< a_b, T_b, Q_b, R_b >$ represents the set of buses, including the starting location, maximum trip time, seat capacity and targets assigned. This will be expanded in section 2.4.

Next we want to generate the set of requests of size R and and the set of buses of size B. For each bus, $b =< a_b, T_b, Q_b, R_b >$, we need to specify its initial starting location a_b, its seating capacity, Q_b, and fuel capacity, T_b and initialize its requests assignment to None. Buses are randomly initialized at one of the four depots. For simplicity, we assume the buses are standardize so they have the same seating capacity and fuel capacity. Each bus is generated along with a travel time counter initialized to 0. In the dynamic phase, the counter will go up by 1 each time the bus moves.

For each request, $r =< p_r, d_r, [e_r^p, l_r^p], [e_r^d, l_r^d] >$ we need to specify its pick-up

location p_r, its drop-off location d_r, its earliest and latest pick-up time $[e_r^p, l_r^p]$, and its earliest and latest drop-off time $[e_r^d, l_r^d]$

Each request's pick up locations is generated at random among the bus stops, but its drop-off location is generated in a way that the drop-off location is more likely to be at the same route as the pick-up location. This reflects the optimizing effects achieved by the traditional fixed-route bus system. Since the public planners spent a considerable amount of effort designing the bus routes, we would assume that they are somewhat predictive of the request demand. For concreteness, we assume that it's 50% more likely for the two bus stops to be in the same route than not. For a network with 4 bus routes, the probability of the pick-up stop and drop-off stop being in the same bus route equals to $\frac{1}{1.5+1+1+1} * 1.5$, which is approximately 33.3%.

To initialize the time interval for pick-up and drop-off, we define an internal clock to keep track of the internal time elapsed during the simulation. At this static phase, time is initialized to zero. As a request is generated, it will take the current time of the simulation as input to form its time interval for pick-up and drop-off.

For each request, the interval for pick-up is $(e_r, e_r + 50)$ and the interval for drop-off is $(e_r + t_{p_r,d_r}, e_r + 50 + t_{p_r,d_r} * 2)$. e_r is the time that the request is generated, and t_{p_r,d_r} is given by the time it takes to travel from the edge connecting the pick-up and drop-off locations, which equals to the distance that it takes to travel between the two locations under our assumption that buses travel at 1 unit of distance per unit of time. Intuitively, a request can not be picked up before it's clocked in the system, so the earliest pick-up time e_r would be the time that it's generated. We allow 50 time units for the vehicle to pick up the requests since it's generated, so the latest pick-up time would be $e_r + 50$. The earliest drop-off occurs when the bus picks up the customer at the earliest

pick-up time at e_r and traverse directly to the drop-off location in time t_{p_r,d_r}. The latest drop-off occurs when the bus picks up the customer at the latest pick-up time and take a small detour to pick-up/drop-off other requests before dropping the passenger off. Here, we assume the detour can not take twice as long as a direct trip between pick-up and drop-off. All of these are captured by the time interval given above.

6.1.2 Dynamic Phase

In the dynamic phase, the program will periodic receive new requests within the service hours that are predetermined by us. In this simulation, I keep the service hours to 400 time units for ease of computation. Furthermore, we assume a set of requests of size 2 is generated every 50 time units during the service hour. After the service hour, the buses will not receive any more requests, and they will simply finish its assigned requests and return to the depot.

The program terminates when the internal clock is after 400 time units and all the requests have been full-filled. If that's the case, the program will exit and return the total travel time across buses, total waiting time across requests and total run time.

If the termination conditions have not been achieved, the bus scheduler will do one of two things at each step. If there are new requests, the scheduler will first re-optimize its route based on the incoming requests (for fixed route, it has a do-nothing re-optimization function). Then, based on the new routes, it will communicate to the buses and let them know of their next destination. If there are no incoming requests, the scheduler will simply tell the buses their next destination. With the information on their next destinations, the buses will toward the destination for 1 unite of time. In that interval, the travel time of any moving bus goes up by 1, the waiting time of any requests that have not been

picked up goes up by 1, and the internal clock also goes up by 1. This is the only time elapse that we clock in the simulation because we assume that the time it takes to re-optimize the route should be minimal, in theory, compared to the time it takes for the bus to travel from one point to another. Furthermore, this helps to isolate the effects of traveling time and waiting time, independent of the scheduling time.

Since the buses move 1 unit at a time, it may go over the next destination that it's designated for. Thus, we set up a reaching condition such that if the bus is within 1 unit of the location, we would consider that it has reached the location. Whenever the buses reach the a location, the scheduler will check whether it's a corresponding location for a pick-up or drop-off. If so, the scheduler will change the request status to indicate that it has been picked up or dropped off. Once a request has been picked up, its clock for waiting time stops.

Finally, the program checks whether the terminating conditions have been reached. If not, it will repeat this dynamic phase again.

6.2 Simulation Results

In the simulations, we run the fixed-route, MILP and GA model on a few different simulations to compare their performance. Our hypotheses are that **1). MILP and GA will significantly outperform fixed route in terms of travel time and waiting time 2). GA will drastically improve the program run time of the MILP model**

We keep the same setup across all simulations and only vary the size of initial requests to achieve a controlled experiment. Keeping the number of vehicles to 4, the service hour to 400 time units and size of the incoming requests constant at 2 per every 50 units of time, we ran the three models on an initial requests

of different sizes ranging from 5 to 11. For each size of the initial request, we ran 3 iterations of the model and take the average performance as our metrics to smooth out any irregularities. Ideally, a higher number of iterations would be even better at smoothing out the performance metrics, but due to the computation complexity of the task where the longest run-time for a single model is 16 hours per iteration, we decided to settle for a smaller number of iterations for the scope of this thesis.

6.2.1 Travel Time

Figure below shows the total travel time required of the three program versus the size of the initial requests.

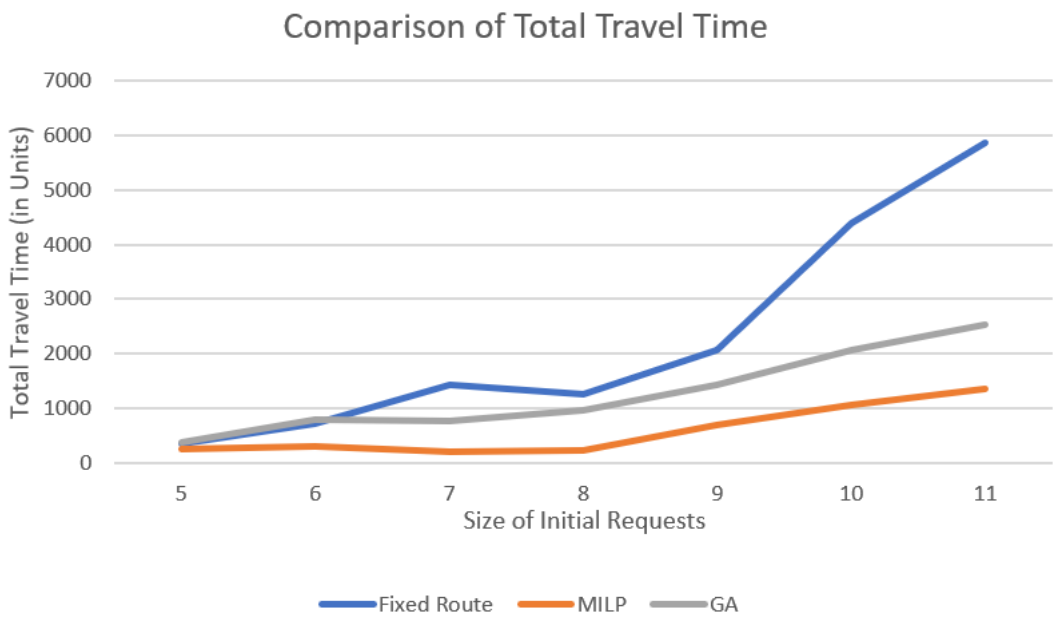

Figure 6.4: Total travel time vs. size of initial requests

First, we compare the performance of MILP/GA versus fixed-route. Although there are some idiosyncratic fluctuations at the cases with the case of smaller initial request size, as the number of initial requests becomes bigger, we can see

a clear trend that MILP and GA are outperforming the fixed route bus schedule by a huge margin. This confirms our hypothesis that as the routing problem becomes more complex, the more inefficient that the fixed route model becomes, and thus the more improvement MILP/GA can bring. In the most complex case with 11 initial requests, MILP improves the performance of a fixed-route model by 76% and GA improves the performance by 57%

Next, we compare the performance of GA versus MILP. Although GA perform worse than MILP in every case, the performance gap between MILP and GA seem to be relatively constant even as the program becomes more complex. This gives us the confirmation that GA could potential be a good approximation for MILP solution when the problem becomes too complex to run the MILP model.

6.2.2 Total Waiting Time

Figure below shows the total waiting time required of the three program versus the size of the initial requests.

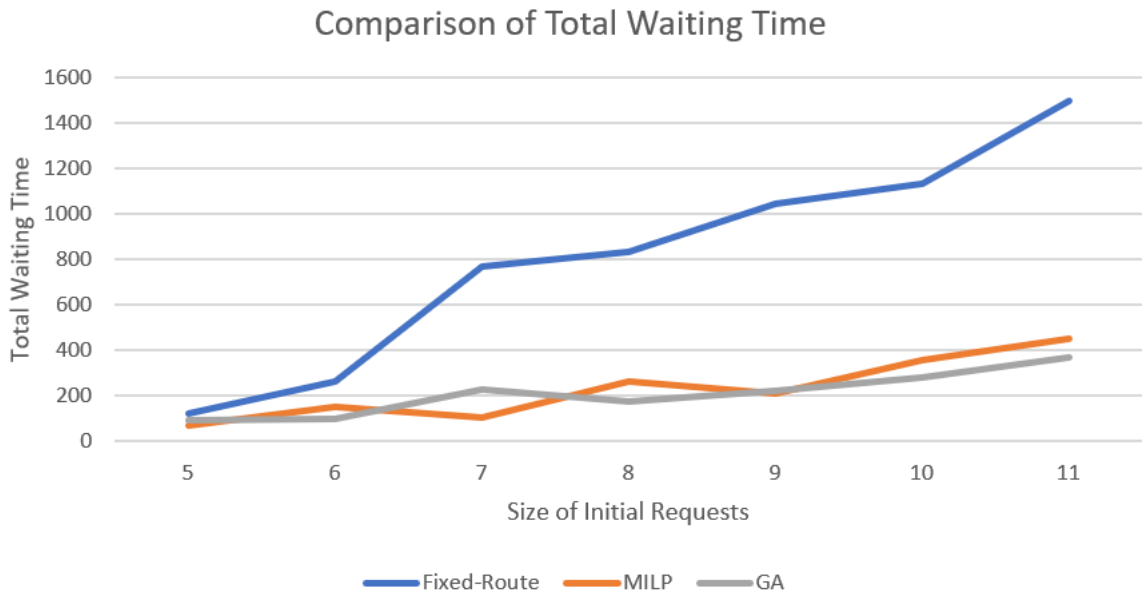

Figure 6.5: Total travel time vs. size of initial requests

We compare the performance of MILP/GA versus fixed route. For every initial request size, both MILP and GA have a shorter waiting time than fixed route model. However, unlike travel time, as the size of the problem increase, MILP/GA does not appear to be exponentially better than the fixed route. This is likely because MILP/GA did not explicitly optimized for wait time as they did for travel time, but only place a bound on the waiting time to be 50 time units. In addition, since the nature of the fixed-route bus essentially guarantees an arrival within the time it takes to traverse the route, its waiting should also display a constant increase in the size of the problem. Nonetheless, in the most complex case with 11 initial requests, MILP reduces the waiting time of a fixed-route model by 70% and GA reduces the waiting time by 75%

6.2.3 Run Time

Figure below shows the total run time required of the three program versus the size of the initial requests.

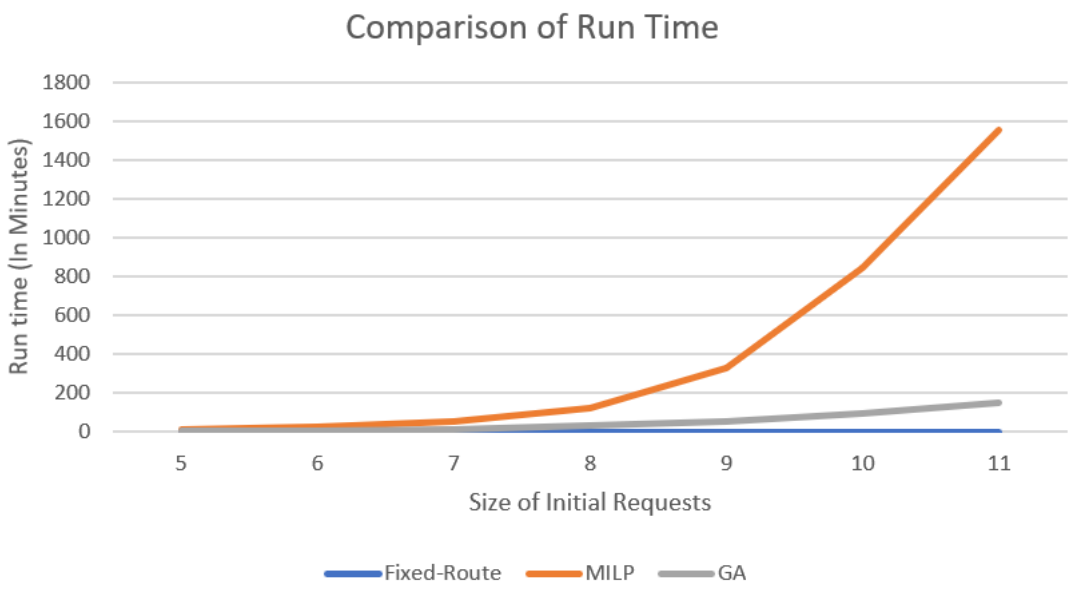

Figure 6.6: Total travel time vs. size of initial requests

Increasing complexity has a minimal impact on the run time of the fixed route model because the model does not require re-computation during the request optimization phase. However, increasing complexity lead to an exponentially higher run time for both MILP and GA. In the case where the initial requests size is 11, MILP reached a run time of around 16 hours, making it extremely difficult to adapt it to a problem same size as the real world routing problem, even with the aid of a super computer. GA demonstrated a 90% reduction in run time compared to MILP, but still took around 2.5 hours for the case with 11 initial requests. The result suggests that simply relying on GA for approximation is still far from enough to solve the city's routing problem. As a result, other approximation algorithms would be needed to enhance the performance.

Chapter 7

Conclusions

Traditional bus system suffers from an outdated and inflexible design. As a result, riders often experience sluggish service delayed by unnecessary detours to empty bus stops and frequent transfers between two popular destinations that should have been in the same route. The cost and the inconvenience of renovation prevents local government from taking imminent actions to renovate their current bus system, further delaying and exacerbating the problem.

This thesis is proposes a bus-scheduling system where city governments can maintain their current infrastructure while creating more intelligent and efficient bus routes. We introduce a variable-route system built with Mixed-Integer Linear Programming and Genetic Algorithm that can customize and optimize bus routes in real time based on the incoming requests and their respective characteristics. Using a computer simulated example, we demonstrate that the variable routing system reduce both the total waiting time on bus stop and the total travel time on the buses by a significant margin. Furthermore, we show that by leveraging genetic algorithm, we can reduce the computation time by as much as 90% while still maintaining a significant reduction in waiting time and traveling time. This thesis provides an initial performance check for the optimal bus route scheduler and highlight an implementations that can achieve significant time reduction with

much less computation time.

This thesis lays the ground work for future researchers who are interested in applying techniques from solving vehicle routing problem in public transportation. As the paper demonstrates, even though genetic algorithm achieves a drastic time reduction compare to the MILP model, its computation complexity makes it still far from enough to solve the city's routing problem. Further research could be done on applying other approximation algorithms to the problem.

Other possible extensions include improving the design of the variable-route system that the paper proposed. To accommodate individuals who are not as tech savvy and are not used to calling a ride from their phones, public planner can consider an integrated system of fixed-route and variable-route buses that address the needs of the less tech-savy individuals but still improve the overall efficiency of the system. Another possible improvement is a system of localized variable-route bus. In this system, every bus is only allowed to served the bus stops that are within its district and every stop is connected to another one through a transfer stop.

References

[1] Anderson, "Who relies on public transit in the U.S.," 2016. 1

[2] Karp, "Reducibility Among Combinatorial Problems," 1972. 21

[3] J.-F. Cordeau and L. G. Salazar Gonzalz J.J. Iori M., "A branch-and-cut algorithm for the pickup and delivery traveling salesman problem with lifo loading," pp. Networks, 46 – 59, 2010.

[4] K. D. S. N. Parragh and R. Hartl., "A survey on pickup and delivery problems: Part ii: Transportation between pickup and delivery locations. Journal fuer Betriebswirtschaft," pp. Journal fuer Betriebswirtschaft, 58: 81 – 117, 2008.

[5] S. Ropke and J.-F. Cordeau., "Branch-and-cut-and-price for the pickup and delivery problem with time windows," pp. Transportation Science, 40(5):455–472, 2008.

[6] H. Hernandes-Perez and J.-J. Salazar-Gonzalez., "A branch-and-cut algorithm for a traveling salesman problem with pickup and delivery," pp. Discrete Applied Mathematics, 145:126–139, 2004.

[7] H. Hernandes-Perez and J.-J. Salazar-Gonzalez., "The one-commodity pickup-anddelivery traveling salesman problem: Inequalities and algorithms.," pp. Networks, 50(4):258 – 272, 2007.

[8] F. B. J. Renaud and G. Laporte., "Perturbation heuristics for the pickup and delivery traveling salesman problem.," pp. Computers Operational Research, 29: 1129 – 1141, 1996.

[9]