NTNU

Norwegian University of
Science and Technology

# The line planning with simultaneous optimisation of operational cost and travel time

Rosalia Rosalia

Master of Science in Mathematics
Submission date: June 2017
Supervisor: Markus Grasmair, IMF
Co-supervisor: Anton Evgrafov, MATH

Norwegian University of Science and Technology
Department of Mathematical Sciences

# Preface

This master thesis has been written as the final project of my study at the Department of Mathematical Sciences, Master degree program with specialisation in Applied Mathematics at the Norwegian University of Science and Technology (NTNU).

I would like to thank my supervisor Markus Grasmair for excellent guidance throughout my work with this thesis. I also thank my co-supervisor Anton Evagrof for feedbacks. And to my family, for cooperation and understanding that kept me motivated.

Rosalia,
June 1, 2017.

# Abstract

In this thesis, we discuss a method of line planning of a public transportation system on a city road network. The desired solution is the cheapest and fastest line network, which is a complex combination to be attacked at the same time. We want to address the fact that the optimisation problem is a hard problem since the feasible solutions are not clearly defined.

The method we use to approximate the solution is based on a heuristic method, the Local Search, and then we combine it with the Minimum Mean-weight Cycle on a graph. The local search method comes first to minimise the operational cost and then followed by the minimum mean-weight cycle algorithm to find some express lines to decrease the average value of travel time. We optimise the solution by performing both steps alternately.

At the end, we discuss the result from numerical experiments of the method with two small-scale city road networks, one with 10 stations and the other with 30 stations. The results show that the optimisation method produced reasonable solutions for the problem. The operational cost and average travel time decrease significantly compared to the initial guess.

# Contents

Chapter 1

# Introduction

Public transportation network systems have become a necessary requirement in an urban area. By transporting people collectively, the travelling cost per passenger is expected to be lower than using a private vehicle, and so is the environmental impact. To encourage more people to use a shared transportation mode instead of their own cars, a good planning and continuous improvement of the public transportation system is needed to be done. Travel time, line routes, cost, service quality, reliability, safety and comfort are among the important factors to be considered.

This thesis focuses on bus line planning on an existing road network in a city. We are given the characteristics of the road network, bus station locations, and demand in a certain period of time. Our goal is to find a bus line network with minimum operational cost while ensuring that the travel time is reasonable.

The many-to-many pattern of the travelling demand gives a huge variation of route in the network, thus it is difficult to choose the optimal combination among all possible lines. In general, most of the methods have been developed are based on heuristic approach with iterative process.

We will start with an overview of the researches that have been done. In chapter 2, we discuss the mathematical model formulation of the problem, and then describe the implementation of the model in more detail in chapters 3 and 4. The implementation is then followed by numerical experiments in the programming part to find an optimal solution. The result from the experiments will be discussed in chapter 5. Finally, we conclude the discussion in chapter 6 together with some suggestions

for the future work.

### Previous researches

Various methods have been developed to improve quality service of public transportation system. Since the late 1960s, researchers have extensively examined the public transportation network design problem. Kepaptsoglou and Karlaftis in [1] presented a list of over 60 studies published between 1967 and 2007 in a table. The list presents the major features of line network optimisation problem such as objective functions, decision variables, network structure, demand characteristics and the method to solve the problem.

In the stage of line planning, Schöbel underlined in [3] two main objectives that somehow contradict each other. A line network which is convenient for the passengers is costly. In the other hand, a line network which uses a small budget usually does not offer the service that customers would like to have. Schöbel then suggested to combine these two objectives and to identify Pareto solutions of such planning models.

According to Schöbel [3], there are some basic constraints must be considered in the line network optimisation problem. The constraints are budget, capacity of the network or a single vehicle, minimum or maximum bus frequencies on the road or station, and direct connections for every origin-destination pair.

Both Schöbel and Kepaptsoglou and Karlaftis reported that many of the optimisation methods with heuristic approaches are used to improve the result while reducing the need for computational power. Most of these methods consist of procedures for candidate line network generation and combinations of feasible lines. There are two distinct ways have been used to generate the line network.

The first approach assumes that a line pool — a set of all potential lines — is given. This information could be provided by, for example, a transportation company. Then we choose some lines in the line pool to create a line network by considering the shortest path procedure or driven by the passenger demand.

The second approach would construct a line network candidate within an optimisation process. The construction is done heuristically by considering some rules

or criteria regarding the line shapes.

In this thesis, we are interested to implement the second approach for defining the line network candidates. The optimisation process starts with a feasible set of lines, and then heuristically improved by testing different line network candidates. The construction of the line networks will be done during the optimisation process by modifying the temporary solution with respect to some rules.

Chapter 2

# Modelling of the problem

Consider a city as an area consisting of a number of given locations for the bus stations. A road network has been made in such a way to connect all the stations to facilitate the movement of the people throughout the city. On the road network, some interconnected bus lines will be established in such a way that each station in the city is visited by one or more bus lines. These interconnected bus lines form a *line network*, which is the decision variable of our problem.

Our intention is to find the cheapest and at the same time the fastest line network, which is impossible to do because the two objectives contradict each other. A less expensive line network tends to be inefficient in the passenger's point of view. While providing a line network that the passenger would like to have, in relation with short travel time, will increase the number of lines or frequencies and thus increase the operational cost. To accommodate these two concerns, we will follow the suggestion in [Schöbel] to combine the two objectives and then identify the *Pareto solutions* of our optimisation problem.

Our approach will start with an initial line network, and then search for some possible line networks in the neighbourhood to minimise the cost. This strategy is known as *local search method*, a heuristic based optimisation method for solving hard problems. The solution from the local search is a line network with minimum cost. Into this line network, we will add some express lines in addition to the existing lines such that the travel time could be reduced. Another local search will follow using the previous solution together with the additional express lines as the new

initial line network.

We will present the solutions obtained during the search as pairs of cost and time on a Cartesian diagram. Finally, we choose an approximation of the solution from the points lie on the Pareto front of this diagram.

## 2.1 The city road network

The road network in the city can be represented as a graph network with a bus station location as a *vertex* and a road segment between two bus stations as an *edge*. Let $V$ denote the vertex set and $E$ denote the edge set, then we have the graph $G = (V, E)$ representing the city road network. Since all the stations are interconnected with every other station via road segments in both directions, $G$ is thus a *complete directed graph* where the edge set $E$ consists of all pairs $(u, v)$ for $u, v \in V$.

The digraph $G$ is weighted by two different values according to travel time and cost. At each edge $e \in E$, we have $t_e$ for the travel time and $c_e$ for the cost of operating a single bus along edge $e$, respectively.

We also consider the transportation demand in the city. There will be different numbers of potential passengers that start from any point to their various destinations. Then for any $u$ and $v$ in $V$, positive constants $d_{u,v}$ indicate the passenger demand to travel from $u$ to $v$ per unit time.

Summary of the variables used to model the city road network:

| | |
|---|---|
| $V$ | vertex set, where $v \in V$ is a location for a bus station, |
| $E$ | edge set, where $e \in E$ is a direct connection roads between two stations, |
| $t_e$ | travel time on edge $e \in E$, |
| $c_e$ | operational cost on edge $e \in E$, |
| $d_{u,v}$ | demand for transportation from $u$ to $v$ in $V$ per unit time. |

## 2.2 The line network

To fulfil the transportation demand in the city, the buses will run regularly following some specific routes. Therefore, we require a line network consisting of several bus lines such that all the potential passengers in the city are served. This set of lines will be discussed in the next section. In this section we look first at the bus line's route, its properties and how to place it in the graph $G$.

Let $L$ be a bus line that visits a series of stations $v_0, ..., v_j$. To make sure that a bus is continuously used, we set $v_j = v_0$ such that it comes back to the first station where it departs and then start again for the next ride.

In the graph $G$, a line $L$ is a sequence of vertices $v_0$ — ... — $v_j$ with $v_0 = v_j$ which forms a *closed walk*. This can also be read as a sequence of edges $e_1$ — ... — $e_j$ where $e_i = (v_{i-1}, v_i)$ for $i = 1, ..., j$. In a walk, it is possible that a vertex or an edge occurs several times. Consequently, there are possibilities that a single line will visit some stations for more than once. The figures below show two basic types of possible line's shapes: the circular line and the back and forth line.

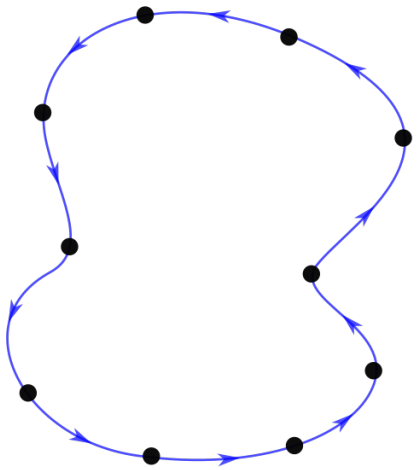

Figure 2.1: A circular line, visits each station exactly once.

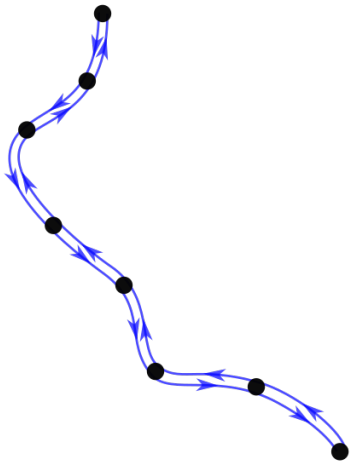

Figure 2.2: A back and forth line, traces the same route in both directions.

Different shapes of the lines can occur during the search. We do not restrict the shape of the closed walks for two reasons. First, it is difficult to determine which shapes would make a line better route. And the second reason is that we expect that these shapes lead to a better set up until we find an optimal network. The figures below show two possible combinations of both basic types.

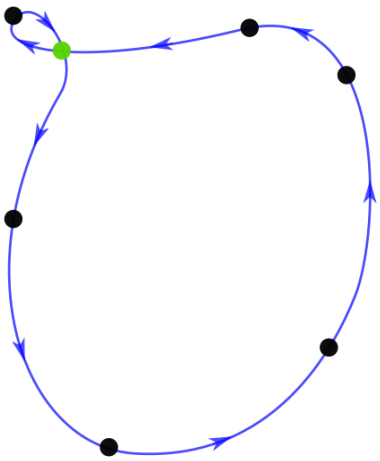

Figure 2.3: A line consists of two cycles that meet in the green station.

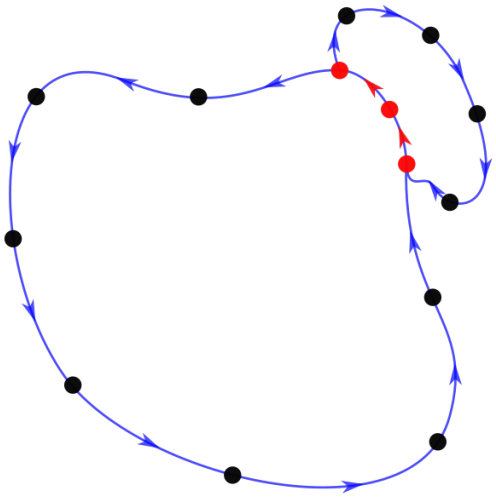

Figure 2.4: A line consists of two cycles that overlap each other on several consecutive stations.

Providing a public transportation system in a city requires a set of lines such that every station in the city is visited by one or more lines. Let $\mathcal{L} = \{L_1, ..., L_N\}$, $N \in \mathbb{N}$, be the set of $N$ lines operated in the line network. Then the edges in the lines $L_1, ..., L_N$ have to form a connected network graph $G' = (V, E')$ where $E' = \bigcup_{1,...,N} L_n$. As a consequence of $L_n \in \mathcal{L}$ being a closed walk, the connectivity of $G'$ assures that every vertex in the graph $G'$ is reachable from every other vertex. Hence the graph is *strongly connected*. Figure 2.5 presents a simple example of a line network consisting of three bus lines.

To transport all the potential passengers in a certain period of time, we need to define the frequency of buses for each line. Denote for each line $L_n$, a constant $f_n$ the number of vehicles operating per time unit. We call $f_n$ the frequency of line $L_n$. Collect this number for $n = 1, ..., N$ in a vector $f$, then together with the corresponding line network $\mathcal{L} = \{L_1, ..., L_N\}$ we have a pair $(\mathcal{L}, f)$ as a line

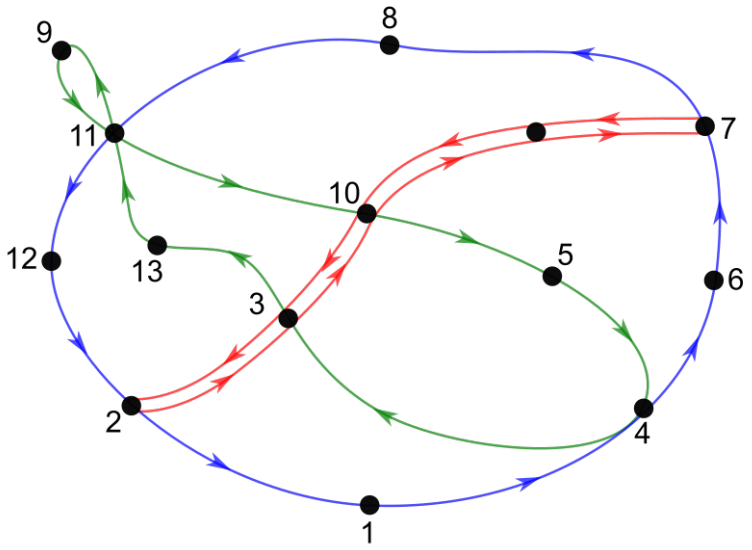


Figure 2.5: The graph of line network $\mathcal{L}$ consists of three lines: 1) A circular line in blue; 2) A back and forth line in red; 3) A line of two cycles in green.

configuration.

The pair $(\mathcal{L}, f)$ is the variable that determines the operational cost of the network, which we have to minimise at the end. We will discuss the definition of the operational cost later in section 2.5.

Here is a brief summary of the notation we have used in the line network:

| | |
|---|---|
| $\mathcal{L}$ | a line network, |
| $N$ | number of lines in line network $\mathcal{L}$, |
| $L_n$ | a bus line in line network $\mathcal{L}$, |
| $f_n$ | frequency of buses serving in line $L_n$, |
| $(\mathcal{L}, f)$ | a line network configuration. |

## 2.3 Passenger shortest path

A potential passenger who wants to travel from a location $u$ to another location $v$ in the city can choose from some available lines connecting $u$ and $v$. The journey starts from $u$ and goes to $v$ passing through some other stations, so the passenger route can be written as a sequence of stations $v_0$—$v_1$—...—$v_j$ where $v_0 = u$ and $v_j = v$. We assume that the best route is the fastest one among all possible routes. Therefore, information about the time for travelling on this path is necessary. This can be done by adding the travel time $t_e$ as a weighting function on the graph $G' = (V, E')$ where $E'$ is the edge set containing all the edges on the line network $\mathcal{L}$. We choose the path from $u$ to $v$ with the smallest weight. To be emphasized, the symbol $P_{u,v}$ is only for the shortest path between $u$ and $v$ within $\mathcal{L}$, not an arbitrary path from $u$ to $v$ in general.

The passenger shortest path is chosen within the available bus service, that is the line network $G'$, not the original network $G$. Thus, each edge in $P_{u,v}$ must be a part of some lines in $G'$. In other words, for all $u, v \in V$,

$$P_{u,v}(G') \subseteq \bigcup_n L_n. \tag{2.1}$$

Consider again the line network from figure 2.5, we add the travel times as shown in figure 2.6. There are several possible paths, for instance from station 8 to 3, such as $P_1$ = 8—11—12—2—3 with total travel time 19, and $P_2$ = 8—11—10—3 with total time 15. The second option is the shortest path, marked with a dotted line.

## 2.4 Meet the demand

The transportation requirement is that all passengers are transported to where they want to go. This means that the configuration of $(\mathcal{L}, f)$ has to satisfy the demand $d_{u,v}$ for any $u, v \in V$. We assume that all the passengers will always take the shortest connection $P_{u,v}$ in the line network $\mathcal{L}$.

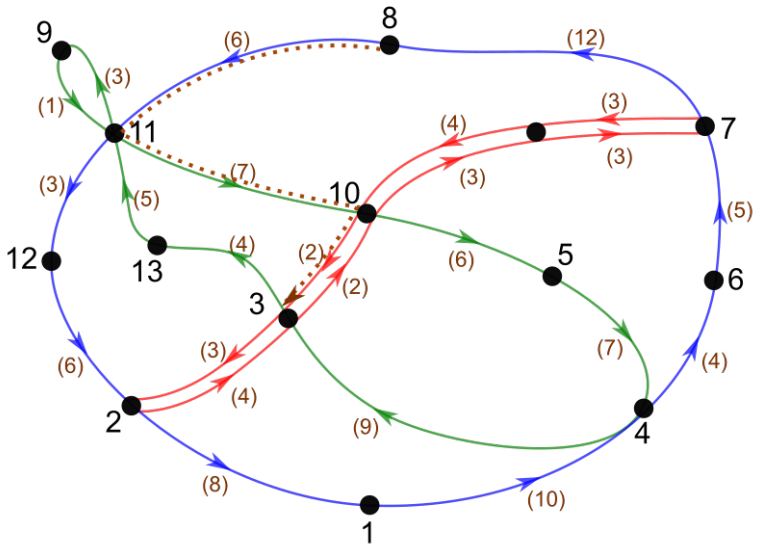


Figure 2.6: Graph $G'$ with travel time. The passenger shortest path from station 8 to station 3, $P_{8,3}$, is shown as the dotted line 8 — 11 — 10 — 3.

To solve this problem, we start by formulating some equations to calculate the following:

- The number of passengers on each edge over a unit time, and
- the capacity on each edge over the same time.

Then we can see whether the demand is satisfied in any edge by comparing the number of buses and number of passengers on the relevant edge. How to compute each of the values will be described in the following two subsections.

### 2.4.1 Demand on the edges

We are given a constant $d_{u,v}$, denoting the number of passengers who want to travel from a location $u$ to $v$ over a unit time. The passengers are assumed to be evenly distributed along the shortest path $P_{u,v}$. To find how many of them are currently on a certain edge $e$, we multiply with the travel time of this edge, $t_e$. The passengers travelling on a particular edge $e$ per unit time is thus

$$d_{u,v}\ t_e, \quad e \in P_{u,v}.$$

Note that this number is only for passengers following a particular route $P_{u,v}$. For finding all passengers travelling simultaneously on $e$, we add together all possible combination $(u, v) \in V^2$ and we get

$$d_e(u, v) = \sum_{\substack{u,v: \\ P_{u,v} \ni e}} d_{u,v}\ t_e. \tag{2.2}$$

This is the number of all passengers in the network who simultaneously travel on edge $e$. We want to check whether these passengers can all be transported by the buses. We will compute the number of available capacity on each edge in the next step.

### 2.4.2 Capacity on the edges

To find out how many buses are needed altogether on an edge $e$, first we need to know how many buses serving each line on this edge. The argumentation is similar as we did before for number of passengers. Assuming that the buses are evenly distributed along the trip, we multiply the frequency and travel time on edge $e$. The number of buses serving the line $L_n$ on this edge is then

$$f_n\ t_e, \tag{2.3}$$

where $f_n$ is the frequency of line $L_n$.

Consider now all lines that pass through edge $e$. Then the number of the buses altogether is the sum of (2.3) over all lines that serve $e$, that is,

$$b_e(\mathcal{L}, f) = \sum_{\substack{n: \\ L_n \ni e}} f_n\ t_e. \tag{2.4}$$

Assume that all the buses used in the network have a similar capacity $M$, then the demand $d_e(u, v)$ will all be transported along $e$ if

$$d_e(u, v) \leq M \cdot b_e(\mathcal{L}, f).$$

This gives us the following set of conditions to be satisfied: for all $e \in E'$,

$$\sum_{\substack{u,v: \\ P_{u,v} \ni e}} d_{u,v}\ t_e \quad \leq \quad M \sum_{\substack{n: \\ L_n \ni e}} f_n\ t_e. \tag{2.5}$$

## 2.5 Operational cost of the bus network

We are given a constant $c_e$ for each edge $e$ in the city network, which denotes the operational cost for a bus with a capacity $M$ on the edge. If an edge is served by $b_e(\mathcal{L}, f)$ buses as in equation (2.4), the operational cost on this edge is then $b_e(\mathcal{L}, f)\ c_e$. Summing the cost of all edges in the network gives us the total operational cost,

$$C(\mathcal{L}, f) = \sum_{e \in E'} \sum_{\substack{n: \\ L_n \ni e}} c_e\ f_n\ t_e. \tag{2.6}$$

From the economic point of view, it is very important that the bus network is cost efficient. Then the line configuration $(\mathcal{L}, f)$ is not only satisfying the conditions in (2.5), but also is the one with the lowest cost possible. This leads us to see the problem as an optimisation problem with respect to the cost. The cost function in (2.6) is the objective to be minimised, while the conditions in (2.5) are the constraints to be satisfied.

## 2.6 Summary of the problem

Our *line planning problem* can be written as an optimisation problem with respect to the line network $\mathcal{L} = \{L_1, ..., L_N\}$ and $f \in \mathbb{R}^N_{\geq 0}$,

$$\left.\begin{array}{l} \min\limits_{\mathcal{L}} \sum\limits_{e \in E'} \sum\limits_{\substack{n: \\ L_n \ni e}} c_e\ f_n\ t_e \end{array}\right\} \tag{2.7}$$

where each $L_n$ is a closed walk in $G' = (V, E')$ and $f_n$ for $n = 1, ..., N$ give the minimum cost for a fixed network $\mathcal{L}$,

$$\left.\begin{aligned} &\min_{f} \sum_{e \in E'} \sum_{\substack{n: \\ L_n \ni e}} c_e \, f_n \, t_e \\ &\text{such that } \sum_{\substack{u,v: \\ P_{u,v} \ni e}} d_{u,v} \, t_e \;\leq\; M \sum_{\substack{n: \\ L_n \ni e}} f_n \, t_e, \end{aligned}\right\} \tag{2.8}$$

where $P_{u,v}$ for all $u, v \in V$ solve optimisation problems

$$\left.\begin{aligned} &\min_{P} \sum_{e \in P} t_e \\ &\text{such that } \; P \subseteq \bigcup_{n} L_n \text{ for some } n \in \{1, 2, ..., N\}, \text{ and} \\ &\qquad\qquad P \text{ is a path from } u \text{ to } v \text{ in } G' = (V, E'), \; E' = \bigcup_{n} L_n. \end{aligned}\right\} \tag{2.9}$$

Our main problem, the line planning problem, clearly consists of these three nested optimisation problems. The desired solution is a line network configuration $(\mathcal{L}^*, f^*)$ with minimum operational cost. During the search of $\mathcal{L}^*$ in (2.7), we have to solve the second problem (2.8) that gives us a vector $f^*$ containing the optimal frequencies of the lines. Similarly, the computation of the second problem includes a set of passenger shortest paths $P_{u,v}$ for all $u, v \in V$ where each $P_{u,v}$ is the solution of the third problem in (2.9).

Beside being as cheap as possible, the line network configuration $(\mathcal{L}^*, f^*)$ should also be as fast as possible We will discuss a method to minimise the travel time later in chapter 4.

Chapter 3

# Implementation

The line planning problem as formulated in section 2.6 consists of three optimisation problems. The second problem is nested inside the first problem, as the third problem is also inside the second one. Thus we have to solve the third problem first before we can continue the computation for the second problem, and finally the main problem.

In order to find the cheapest network configuration $(\mathcal{L}^*, f^*)$, we evaluate different line networks $\mathcal{L}$ iteratively. At every iteration, we solve the nested problem stated in (2.9) to find the shortest paths $P_{u,v}$ within the current line network $\mathcal{L}$, and then use the solutions to solve the second problem in (2.8). The solution of the second problem is the optimal frequency set $f$ that minimises the cost of the current $\mathcal{L}$.

The step by step computation is briefly shown in the flowchart in figure 3.1. We will discuss the detail of each step in the sections of this chapter.

## 3.1 Initial line network

We start with the road network consisting the designed locations for the bus stations, and assume that every station can be reached from every other station. We are given the travel time $t_e$ between each pair $e = (u, v)$ of stations as input, as well as the cost $c_e$ for a bus to travel on that road segment. Furthermore, we assume that the demand for travelling from $u$ to $v$ is given for each $u, v \in V$.

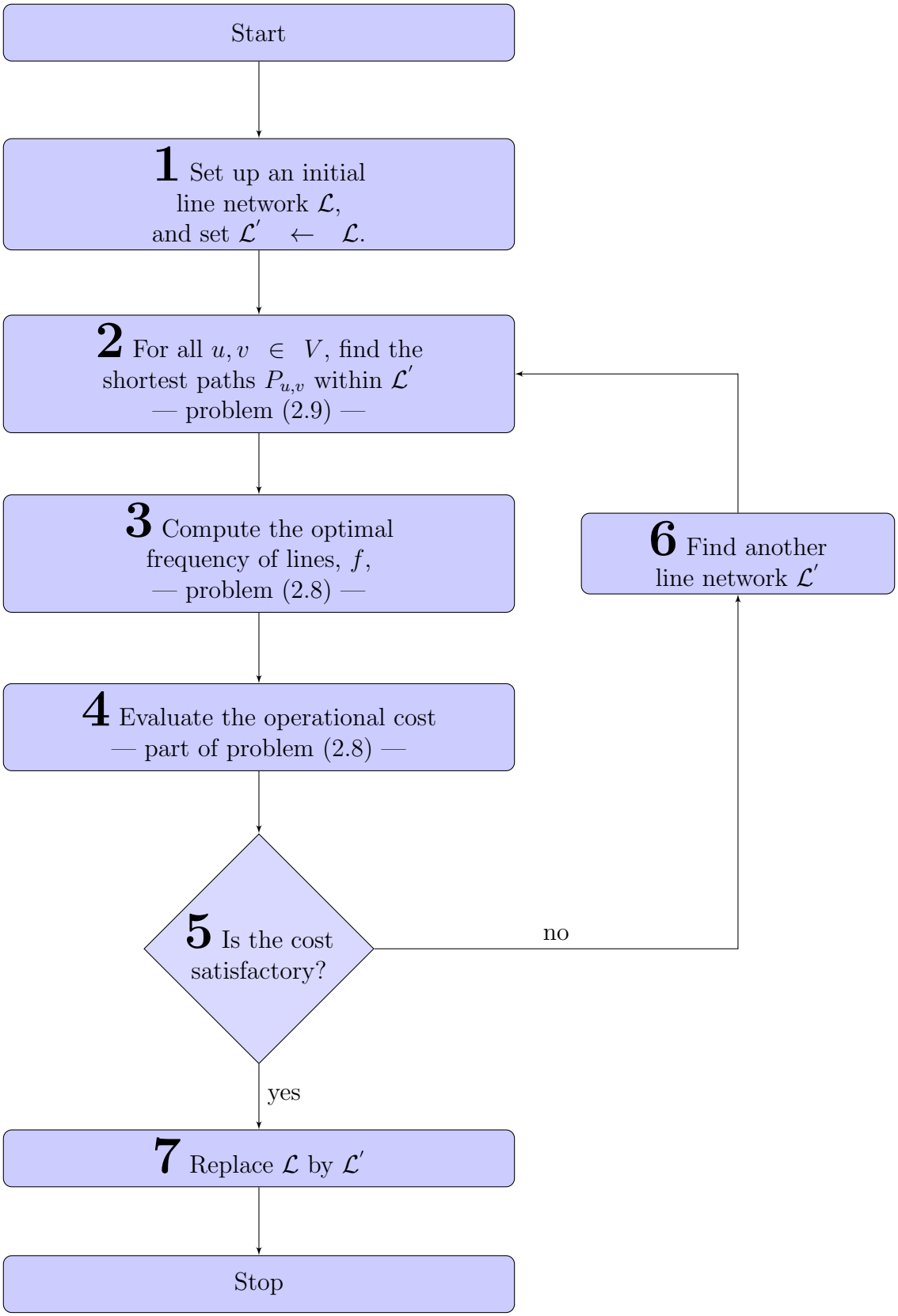


Figure 3.1: The computation steps for solving the line planning problem.

Before we start the computation, which is an iterative process testing different line networks, we set an estimated number of lines operating the transportation system.

### Limit the number of lines

The number of lines, $N$, naturally grows as a function with respect to the size of the network, $m$. It has to be a positive and monotonic increasing with a relatively slow rate function, say $N(m) = N_0 \log(m)$ for positive integers $m > 1$.

Let $N_1$ and $N_2$ be positive constants where $N_1 < N_2$, then we can define the number of lines, $N$, as a positive integer satisfying

$$N_1 \leq N \leq N_2. \tag{3.1}$$

We use this double inequality to bound the number of lines below and above. For the initial line network, we set the number in the middle between the lower and upper bounds,

$$N = \frac{(N_1 + N_2)}{2}.$$

### Constructing the initial lines

For the first guess of the line network, we think about how would potential passengers like to travel. We arrange a network in such a way that people benefit from it. The primary requirement is that the bus route is efficient, in the sense that the route takes as short as possible time to get to the destination. For a potential passenger who wants to travel from a station $u$ to $v$, it would be best to take the fastest possible route from $u$ to $v$, which is the shortest path $P_{u,v}(G)$ in the original graph $G = (V, E)$.

We list all the shortest paths for all pairs $(u, v)$ where $u \neq v$, and sort the list from the biggest to the smallest demand. We want to create $N$ lines, so we take the first $N$ shortest paths as the line candidates and check whether their routes connect all stations in the city. That is, the graph formed by the edges in the line network has to be a connected graph. This is the graph $G' = (V, E')$ where $E' = \cup_n L_n$, and

$n = 1, ..., N$ as we have discussed in section 2.2.

If some stations are not included in the first $N$ shortest paths, we include some shortest paths with lower demand into the line network candidate. By adding more shortest paths, the number of lines grow and there might be too many lines beyond the upper bound at the end. So we select a shortest path with lower demand only if it brings some excluded stations into $G^{'}$ or connect some components of the graph.

Once $G^{'}$ becomes a connected digraph with all the stations included, we create the lines by grouping the shortest paths into $N$ groups. Shortest paths with similar demand will be grouped and then attached one after another forming a longer path. Then we add a new edge to connect the last station with the first one to make a closed walk.

Suppose the shortest paths to be included in the first group are $P_1, P_2, ..., P_i$ where $P_1 = v_{1,0}$—...—$v_{1,k_1}$, $P_2 = v_{2,0}$—...—$v_{2,k_2}$, and $P_i = v_{i,0}$—...—$v_{i,k_i}$. The first line is then $L_1 = v_{1,0}$—...—$v_{1,k_1}$—$v_{2,0}$—...—$v_{2,k_2}$—$v_{i,0}$—...—$v_{i,k_i}$—$v_{1,0}$. Algorithm 1 describes the construction procedure of the initial line network.

## 3.2 Passenger shortest paths

Passengers who want to travel by public bus are assumed to take the shortest routes as discussed in section 2.3. These routes are the best, fastest routes according to the availability of bus lines. In the graph made by the line network $\mathcal{L}$, these are the shortest paths $P_{u,v}$ from each vertex $u$ to every other vertex $v$ in the vertex set $V$.

Recall from section 2.2, the graph of the line network $\mathcal{L}$ is given by $G^{'} = (V, E^{'})$ where $E^{'} = \cup_n L_n$ for $n = 1, ..., N$. Employ a weighting function $t : E^{'} \to \mathbb{R}$ to the graph, then we can compute the shortest paths $P_{u,v}(G^{'}) \subseteq E^{'}$ for all $u, v \in V$.

There are a number of well known algorithms for computing shortest paths, such as More, Dijkstra, Bellman and Ford, and Floyd and Warshall, according to Jungnickel, in chapter 3 in [4]. The first three algorithms compute the shortest path from a certain vertex to every other vertex in the graph with complexity equivalent to $O(m^2)$, while the last one, the algorithm of Floyd and Warshall with complexity $O(m^3)$ returns the shortest paths for all pairs of vertex.

**Algorithm 1:** initialLineNetwork

**Input:** demand $d$, number of lines $N$, shortest paths $P$, list of station $V$.
**Output:** initial line network $\mathcal{L}$.

$\mathcal{L} \leftarrow \{p_1, ..., p_N\}$
$P \leftarrow P \setminus \{p_1, ..., p_N\}$
$V_0 :=$ list of station in $\mathcal{L}_0$
$c :=$ number of connected component of the graph formed by $\mathcal{L}$
**while** $c > 1$ *or* $V_0 \neq V$ **do**
    $p_1 :=$ the first element in $P$
    $P \leftarrow P \setminus \{p_1\}$
    $\widetilde{\mathcal{L}} := \mathcal{L} \cup \{p_1\}$
    $\widetilde{V_0} := V_0 \cup \{\text{stations in } p_1\}$
    $\tilde{c} :=$ number of connected component of the graph formed by $\widetilde{\mathcal{L}}$
    **if** $\tilde{c} < c$ *or* $\widetilde{V_0} \supsetneq V_0$ **then**
        $\mathcal{L} \leftarrow \widetilde{\mathcal{L}}$
        $V_0 \leftarrow \widetilde{V_0}$
        $c \leftarrow \tilde{c}$
    **end**
**end**
Divide $\mathcal{L}$ into subsets $\mathcal{L}_n \subset \mathcal{L}$ for $n = 1, ..., N$
$\hat{\mathcal{L}} \leftarrow \emptyset$
**for** *all* $\mathcal{L}_n,\ n = 1, ..., N$ **do**
    $\mathcal{L} \leftarrow \mathcal{L} \setminus \mathcal{L}_n$
    $L :=$ connect all shortest paths in $\mathcal{L}_n$
    Add an edge connecting the last and the first vertex in $L$
    $\hat{\mathcal{L}} \leftarrow \hat{\mathcal{L}} \cup \{L\}$
**end**
$\mathcal{L} \leftarrow \hat{\mathcal{L}}$

In this thesis, we use the Dijkstra algorithm with a repetition such that all vertices becomes the starting vertex. Repeating the algorithm $m$ times increases the complexity to $O(m^3)$, but it can be done in a less expensive computation as follows: In the first iteration, the Dijkstra algorithm runs as usual to get shortest paths from a particular vertex $v_0 \in V$ to every other vertex. Thus we have $m-1$ shortest paths $P_{v_0,v_j}$ with $j = 1, ..., m-1$. A shortest path $P_{v_0,v_j}$ can written as a sequence of distinct vertices $P_{v_0,v_j} = v_0$—$v_1$—...—$v_{j-1}$—$v_j$ or a sequence of distinct edges $P_{v_0,v_j} = e_1$—$e_2$—...—$e_{j-1}$—$e_j$ where $e_j = (v_{j-1}, v_j)$.

By the Bellman's equation in section 3.5 in [4], a part of a shortest path is also a shortest path. Thus, we keep the sub paths of $P_{v_0,v_j}$ during the construction process. Since the algorithm searches the shortest paths progressively by adding the edges one by one, we have a possibility to keep the sub paths at every time a new edge has been added. The sub paths of the new shortest path are progressively computed from the last edge.

Suppose the newly discovered shortest path is $P_{v_1,v_\gamma}$, then we keep the sub paths $P_{v_2,v_\gamma}$,...,$P_{v_{\gamma-1},v_\gamma}$ such that we save some computational time at the later iterations. This procedure is presented in algorithm 2 by taking a weight function $t$ on the graph $G$ as an input. The outputs are the shortest paths $P$ and the weight of the shortest paths $t(P)$.

In our main computation for the line planning problem, the shortest paths algorithm runs every after a new line network has been made. Particularly in the first iteration, the line network is the initial guess described in the previous section.

Once we have the shortest paths for each pair of the stations, we assign demand to the network as we have discussed in section 2.4.1. It is more efficient to directly compute the demand on each edge during the shortest path computation. We add the demand $d$ as an additional input, and carry out an extra information in the output, the demand on edge $d_e$. The computation will be done by inserting three extra lines to algorithm 2. Between line 13 and 14, we compute the demand every time we get a shortest path as defined in formula (2.2). The additional lines are presented in algorithm 3.

**Algorithm 2:** ManyToManyShortestPaths

**Input:** $G,\ t$
**Output:** $P,\ t(P)$

```
1  for all (u, v) ∈ V × V do
2      t(u, v) ← ∞, c(u, v) ← 0, P(u, v) ← ∅
3  end
4  for s ∈ V do
5      t(s, s) ← 0,  c(s, s) ← 1,  P(u, v) ← s
6      Q ← V
7      while Q ≠ ∅ do
8          Find some u ∈ Q such that t(s, u) is minimal
9          for all η, φ vertices in P(s, u) do
10             if c(η, φ) = 0 then
11                 if P(η, φ) ⊆ P(s, u) then
12                     P(η, φ) := shortest path from η to φ
13                     t(η, φ) := distance from η to φ
14                     c(η, φ) ← 1
15                 end
16             end
17         end
18     end
19     Q ← Q \ {u}
20     for all v ∈ Q in the form (u, v) do
21         if c(s, v) = 0 then
22             if t(s, u) + w(u, v) < t(s, v) then
23                 t(s, v) ← t(s, u) + w(u, v)
24                 P(s, v) := P(s, u) ∪ {v}
25             end
26         end
27     end
28 end
```

**Algorithm 3:** Between line 13 and 14 in algorithm 2

```
for all edges e ∈ P(η, φ) do
    d_e := d_e + d(η, φ)t_e
end
```

The demand at each edge in the network are now assigned. To transport all the passengers, we will compute the number of buses on the relevant edge such that the total capacity is equal or larger than the demand. We have discussed the idea in section 2.4.2, and in the next section we will discuss the computation procedure.

## 3.3 Frequency of the lines

A bus network operates a different number of vehicles for each line depending on the demand. Assume that the demand is constant per time unit, then we can compute a set of constant $f = \{f_1, ..., f_N\}$, where each $f_n$ is the number of vehicle operating line $L_n \in \mathcal{L}$ per time unit.

These constants determine the operational cost of the network as formulated in equation (2.6). We want that this cost is the lowest one for the current line network. This brings us to solve the optimisation problem in (2.8), and so (2.9). We have discussed problem (2.9) in the previous section, so this section focuses on problem (2.8).

The problem in (2.8) is a linear optimisation problem as both objective function and all the constraints are linear, so it can be treated with the linear programming. The only issue in this part is the size of the problem. The number of inequality constraints to be satisfied is the same as the edges in the network, which can grow as fast as $O(m^2)$ for $m$ stations. To deal with this situation, we want to reduce the number of inequality constraints. Some of the inequalities will be removed if they are automatically satisfied by some other inequalities. This strategy will be described in the rest of this section.

We start by rewriting the problem in (2.8) in a more compact formula as

$$\min_{f_n \geq 0} \quad \sum_n f_n \; h_n \quad \text{such that} \quad \sum_{\substack{n: \\ L_n \ni e}} f_n \geq \mu_e \tag{3.2}$$

for all edges $e \in E' = \cup_n L_n$, where we use the notations

$$h_n = \sum_{e \in E'} c_e \, t_e, \quad \text{for the cost function of each } L_n \in \mathcal{L}, \text{ and}$$
$$\mu_e = \frac{1}{M} \sum_{\substack{u,v: \\ P_{u,v} \ni e}} d_{u,v}, \quad \text{the right hand side constants for all } e \in E', \ u, v \in V.$$

Now we want to remove some of the inequality constraints of (3.2). The idea is as follows:

*Step 1.*

Let $E^1$ be the set of all edges that are served by only one line. Then for all $n = 1, ..., N$, define $E_n^1$ as the set of edges in $E^1$ that are served by line $L_n$.
The multiple conditions

$$f_n \geq \mu_e, \quad \text{for all } e \in E_n^1$$

can be replaced by the single inequality

$$f_n \geq \mu_{e_n}$$

where $e_n \in E_n^1$ is such that $\mu_{e_n} = \max_e\{\mu_e; \ e \in E_n^1\}$ for $n = 1, ..., N$.
The solution of problem (3.2) if the constraints are restricted to the set $E^1$ is then $f_n^1$, where

$$f_n^1 = \begin{cases} \mu_{e_n} & \text{if } e_n \in L_n, \\ 0 & \text{otherwise.} \end{cases} \tag{3.3}$$

Hence the number of buses for each line $L_n$ is $f_n = f_n^1 + f_n'$, where $f_n' \geq 0$ will be computed on the next step.

*Step 2.*

Now we check the rest of the edges, that are served by more than one bus lines.
For $s > 1$, let $E^s \subset E$ be the set of edges where exactly $s$ lines being operated.
For any set $I \subset \{1, 2, ..., N\}$ of $s > 1$ distinct lines, denote by $E_I$ the set of edges

that are served by lines in $I$. In other words,

$$E_I = \{e \in E^{'}; e \text{ is served by the lines in } \mathcal{L}_I \subset \mathcal{L}\} \tag{3.4}$$

where $\mathcal{L}_I$ is a part of the line network with exactly lines in $I$.

For all $e \in E_I \neq \emptyset$ we have the inequality constraints

$$\sum_{i \in I} f_{n_i} \geq \mu_e,$$

which with the substitution $f_n = f_n^1 + f_n^{'}$ becomes

$$\sum_{i \in I} f_{n_i}^1 + f_{n_i}^{'} \geq \mu_e$$

or

$$\sum_{i \in I} f_{n_i}^{'} \geq \mu_e - \sum_{i=1}^{s} f_{n_i}^1. \tag{3.5}$$

Let $e_I$ be an edge in $E_I$ such that $\mu_{e_I} = \max_e\{\mu_e,\ \forall e \in E_I\}$, then we can replace the multiple constraints in (3.5) by the single inequality

$$\sum_{i \in I} f_{n_i}^{'} \geq \mu_{e_I} - \sum_{i \in I} f_{n_i}^1, \quad e \in E_I.$$

This single constraint can be removed as well whenever the constant at the right hand side is non-positive. This can be expected to result in fewer constraints.

*Simplified problem.*

Rewriting problem (3.2) after $f_n$ has been replaced by $f_n^1 + f_n^{'}$ and some of the constraints have been removed, we get

$$\left.\begin{array}{l} \min\limits_{f^{'} \geq 0} \sum\limits_{n} (f_n^1 + f_n^{'})\ h_n \quad \text{such that} \\ \sum\limits_{\substack{n: \\ L_n \ni e}} f_n^{'} \geq \mu_{e_I} - \sum\limits_{\substack{n: \\ L_n \ni e}} f_{n_i}^1, \quad \forall I \subset \{1, ..., N\} \end{array}\right\} \tag{3.6}$$

where $f_n^1$ are constants in (3.3) and $e_I$ is an edge in the edge set $E_I$ defined in (3.4).

Since $f_n^1$ are constants for all $n = 1, ..., N$, we can simplify problem (3.6),

$$\left.\begin{aligned} &\min_{f' \geq 0} \sum_n f_n' \, h_n \quad \text{such that} \\ &\sum_{\substack{n: \\ L_n \ni e}} f_n' \geq \mu_{e_I} - \sum_{\substack{n: \\ L_n \ni e}} f_{n_i}^1, \quad \forall I \subset \{1, ..., N\} \end{aligned}\right\} \tag{3.7}$$

The solution of the optimisation problem in (3.7) is a set of $N$ constants $f'$ that gives the frequency set in addition to $f^1$ we have found earlier. Hence the solution for the whole problem is $f = f^1 + f'$ corresponds to the line network $\mathcal{L}$.

The linear programming turns $f$ together with the optimal value of the objective function as the outputs. This value is the minimum operational cost defined in problem 2.7. On our chart in figure 3.1, this occurs at step 3. Then we continue to the next step to evaluate this value. If the cost is satisfactory, then we are done and the corresponding line network $\mathcal{L}$ is the optimal solution. Otherwise, we need to find another bus line network $\mathcal{L}'$ and evaluate it in the next iteration. How then, to find another line network to be evaluated, is the most challenging part of the thesis. This will be discussed in the following section and in the next chapter.

## 3.4 Search for new line networks

Searching of a different line network than $\mathcal{L}$ means that we are looking for another set of closed walks $\mathcal{L}'$ in the complete graph $G = (V, E)$ such that the edges in the closed walks form a connected graph $G' = (V, E')$ where $E' = \cup_n L_n$.

It is impossible to list all the combination of closed walks, since the number of closed walks increases exponentially with the number of vertices. Indeed, we can only evaluate the cost for a small number of possible line networks. There is no clear guidance on how to choose the networks, and thus we will use a heuristic method to approximate the solution. This section focuses on this approach, the so called *local search method*, one of the standard techniques for solving a hard optimisation problem.

### 3.4.1 The local search method

The local search method is a heuristic based method to find a solution "locally". The method searches for the optimal solution in the neighbourhood of the candidates. In other words, we have to choose a sufficiently good initial candidate that lies near the optimal solution. The initial line network that we have discussed in the first section of this chapter can be seen as a good line network since it is constructed by considering the shortest paths and demands from all to all locations. During the optimisation, the solution candidate will be replaced by the new one every time we find a better solution. Considering the modification of the solution candidate, two decisions must be made in the preparation step according to Korte and Vygen, section 21.3 in [5]

- Which modifications are allowed?
- When do we actually modify our solution?

For the first point, we will use four different methods of modifying the candidate solution which allow only small change in the line network. The details about these four methods come in the next sub section. For the second point, we will replace the temporary solution with a new one every time we find an improvement. That is when a new line network decreases the operational cost.

At every iteration, we choose any line network $\mathcal{L}'$ in the neighbourhood of $\mathcal{L}$. If the new line network gives a lower cost, then we use it to replace the previous solution. We stop the iteration when the cost is satisfactory or the maximum number of iteration is reached. We write this procedure in the algorithm below:

This algorithm summarise the computation steps we have discussed previously in the beginning of the chapter. Consider the chart in figure 3.1, it is clear that our computation steps in the chart follow the procedure in the local search algorithm in algorithm 4.

**Algorithm 4:** localSearch

**Input:** A complete digraph $G$, a weight function $c$, an initial line network $\mathcal{L}_0$, the desired cost $C$, and maximum number of iteration $m$.

**Output:** The optimal line network $\mathcal{L}$

```
1 i ← 1, L ← L_0, L' ← L_0
2 while c(L') > C and i < m do
3     L' := a new modification of L
4     if c(L') < c(L) then
5         L ← L'
6     end
7     i = i + 1
8 end
```

### 3.4.2 Find a new line network in the neighbourhood

Assume that our initial guess for the line network, $\mathcal{L}$, is good enough. Then we are close to the solution, and we can use the local search method to find some neighbours of $\mathcal{L}$ to be evaluated. A neighbour of a line network $\mathcal{L}$ is another line network $\mathcal{L}'$ that contains mostly the same lines as in $\mathcal{L}$, except a minor change in a few of the lines. Next, we will discuss four methods for making slight changes to the lines.

#### Flipping a segment of lines

The first method is by flipping a segment within a line. We randomly pick a few lines $L_{n_1}, L_{n_2}, ..., L_{n_i} \in \mathcal{L}$ and choose two stations $a_{n_i}, b_{n_i} \in L_{n_i}$ in each of these lines. The line segment between $a_{n_i}$ and $b_{n_i}$ will be in the reversed order in the new lines $L'_{n_i}$

Suppose we choose line $L_n = v_0$—...—$v_{k_1}$—...—$v_{k_2}$—...—$v_n$ where $v_0 = v_n$. Then by choosing $a_n = v_{k_1}$ and $b_n = v_{k_2}$, we flip the vertex sequence from $v_{k_1}$ to $v_{k_2}$. The resulting line is $L'_n = v_0$—...—$v_{k_1-1}, \boldsymbol{v_{k_2}}, \boldsymbol{v_{k_2-1}}$—**...**—$\boldsymbol{v_{k_1+1}}, \boldsymbol{v_{k_1}}, v_{k_2+1}$—...—$v_k$.

As an example, we consider the line $L_n$ = 1—2—3—4—5—6—7—8—9—10—1. Let $a_n = 3$ and $b_n = 7$ be two stations that are randomly chosen. Reversing the order of the stations starting from 3 until 7 gives us a new line $L'_n$ := 1—2—**7**—**6**—**5**—**4**—**3**—8—9—10—1 as illustrated in figures 3.2 and 3.3.

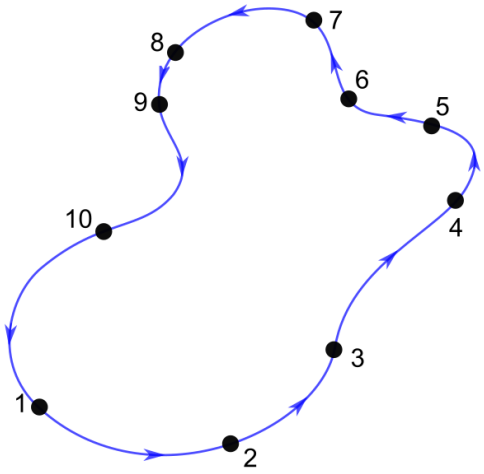


Figure 3.2: Line $L_n$

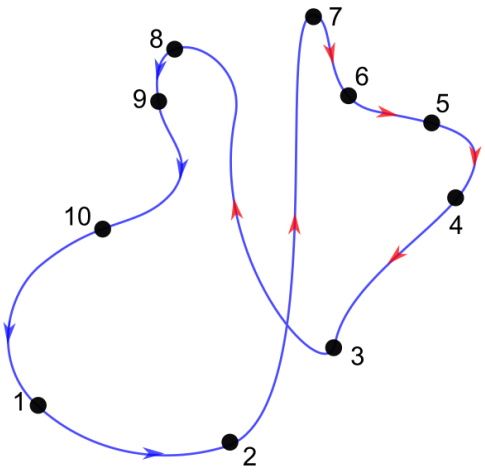


Figure 3.3: Line $L_n^{'}$

We emphasize that the change is done for multiple lines simultaneously. To make sure that the resulting line network is in the neighbourhood, we limit the number of lines can be changed to a relatively small number. Algorithm 5 shows the implementation of this idea.

**Algorithm 5:** lineChangeA

**Input:** $\mathcal{L}$
**Output:** $\mathcal{L}^{'}$

$\hat{\mathcal{L}}$ := randomly choose a small subset of $\mathcal{L}$
$\mathcal{L} := \mathcal{L} \setminus \hat{\mathcal{L}}$
**for** *all* $L \in \hat{\mathcal{L}}$ **do**
    $u, v$ := choose two random stations in $L$
    $l(u, v)$ := take the line segment between $u$ and $v$ in $L$
    $l^{'}(v, u)$ := reverse the order of $l(u, v)$
    $l \leftarrow l^{'}$
**end**
$\mathcal{L}^{'} := \mathcal{L} \cup \hat{\mathcal{L}}$

Flipping a segment in a line can however lead to some error in the network. It is possible that the resulting lines are already exist in the network, and hence we will loose some lines in the system. Another possible error is if the resulting lines contain some repeated stations that form self-loops. It is clear that a self-loop edge is unacceptable in a line's route. We will deal with the potential error of the line changes later at the end of this section.

### Exchange segments between lines

The second method of finding a neighbour line network $\mathcal{L}'$ is by exchanging line segments between a few different lines $L_{n_1}, ..., L_{n_i}$. We use the same idea of the line segment as in the previous method, but we exchange segments instead of flipping them. Suppose $L_{n_1}, ..., L_{n_i}$ are some lines in $\mathcal{L}$ that are randomly chosen, and this number should be relatively small compared to the number of lines in the network. At each $L_{n_k}$, we take a line segment starting from station $a_{n_k}$ to station $b_{n_k}$. Then we have $i$ line segments to be randomly exchanged between $L_{n_1}, ..., L_{n_i}$.

The following example illustrate the changing of three lines, before and after they exchange line segments. Suppose we choose two lines $L_{n_1}$=1—2—**3**—**4**—**5**—**6**—7—8—1 and $L_{n_2}$=16—**10**—**4**—**9**—**6**—**13**—12—11—14—15—16. The line segments 3—4—5—6 and 10—4—9—6—13 are chosen from $L_{n_1}$ and $L_{n_2}$, respectively. After the exchange, the resulting lines are $L'_{n_1}$ =1—2—**10**—**4**—**9**—**6**—**13**—7—8—1 and $L'_{n_2}$ =16—**3**—**4**—**5**—**6**—12—14—15—16, as shown in figures 3.4 and 3.5.

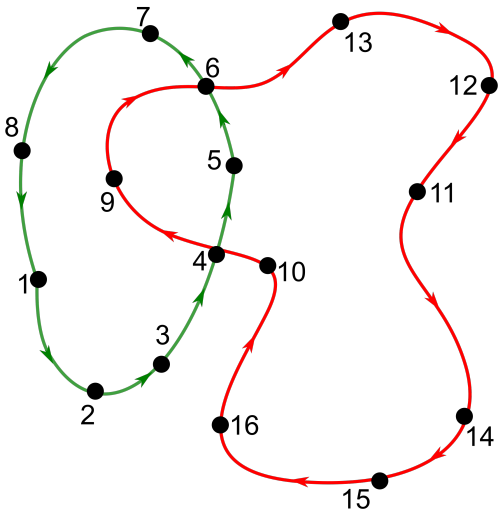


Figure 3.4: Line $L_{n_1}$ and $L_{n_2}$, before exchanging line segments.

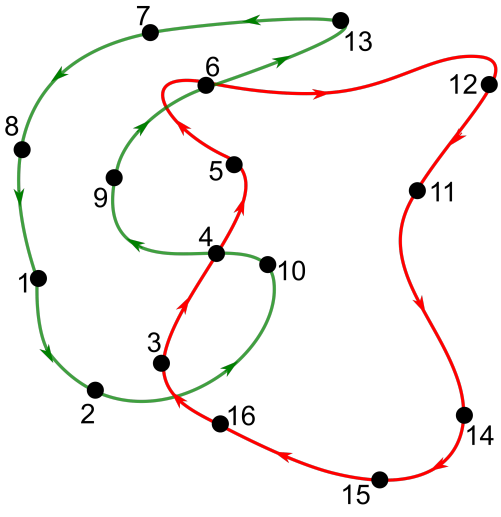


Figure 3.5: Line $L'_{n_1}$ and $L'_{n_2}$ after exchanging line segments.

The algorithm for exchange line segments between a few lines in a network is as follows,

**Algorithm 6:** lineChangeB

**Input:** $\mathcal{L}$
**Output:** $\mathcal{L}^{'}$

$\hat{\mathcal{L}}$ := randomly choose a small subset of $\mathcal{L}$
$\mathcal{L} := \mathcal{L} \setminus \hat{\mathcal{L}}$
$M \leftarrow \emptyset$.
**for** *all* $L \in \hat{\mathcal{L}}$ **do**
    $u, v$ := choose two random stations in $L$
    $l(u, v)$ := take the line segment between $u$ and $v$ in $L$
    $M := M \cup \{l(u, v)\}$
    $L^{'}$ := replace $l(u, v)$ in $L$ by zeros.
**end**
$M^{'}$ := shuffle the order of elements in $M$
**for** *all* $L^{'} \in \hat{\mathcal{L}}$ **do**
    $\mu$ := the first element in $M^{'}$
    $L^{'}$ := replace the zero element in $L^{'}$ by $\mu$
    $M^{'} := M^{'} \setminus \{\mu\}$
**end**
$\mathcal{L}^{'} := \mathcal{L} \cup \hat{\mathcal{L}}$.

The change done by exchanging segments between lines could also lead to similar errors as in the flipping method. There might be some stations that are visited several time consecutively, or the new lines might have existed in the network. Another error potentially occurs is that the whole network could become disconnected after a few lines have exchanged their parts. Thus we also need to check the connectivity of the network after a change has been made. We will discuss the method to check these potential errors after two more line-changing methods being described.

### Exchange stations within a line

The third method of bus line changes is by exchanging order of a few stations within the lines. Suppose we choose the line $L_n = v_0$—...—$v_{k_1}$—...—$v_{k_2}$—...—$v_{k_3}$—...—$v_n$ and $v_{k_1}, v_{k_2}$, and $v_{k_3}$ are the stations to be exchange positions, then the resulting line is $L_n^{'} = v_0, ..., v_{k_1-1}, \square, v_{k_1+1}, ..., v_{k_2-1}, \square, v_{k_2+1}, ..., v_{k_3-1}, \square, v_{k_3+1}, ..., v_n$ where the three boxes are places for $v_{k_1}, v_{k_2}$, and $v_{k_3}$ in random order. $L_n^{'}$ can be, for instance, $L_n^{'} = v_0, ..., v_{k_1-1}, \boldsymbol{v_{k_3}}, v_{k_1+1}, ..., v_{k_2-1}, \boldsymbol{v_{k_2}}, v_{k_2+1}, ..., v_{k_3-1}, \boldsymbol{v_{k_1}}, v_{k_3+1}, ..., v_n$

As an example, we choose three stations 3, 4, and 9 in a line $L_{n_i}$ = 1—2—**3**—**4**—5—6—7—8—**9**—10—11—1 as shown in figures 3.6. One of the possibilities of rearrangement of the line will be $L^{'}_{n_i}$ = 1—2—**4**—**9**—5—6—7—8—**3**—10—11—1, shown in figure 3.7.

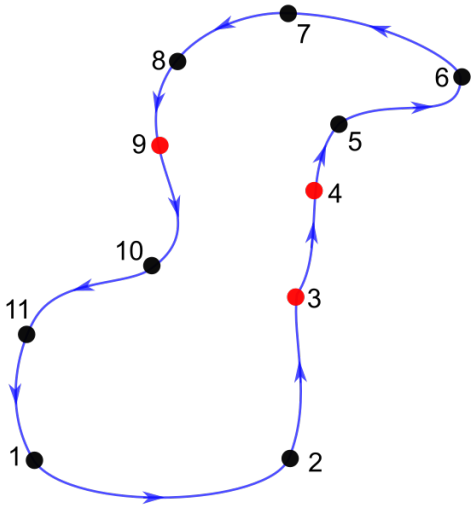


Figure 3.6: Line $L_{n_i}$ = 1—2—**3**—**4**—5—6—7—8—**9**—10—11—1

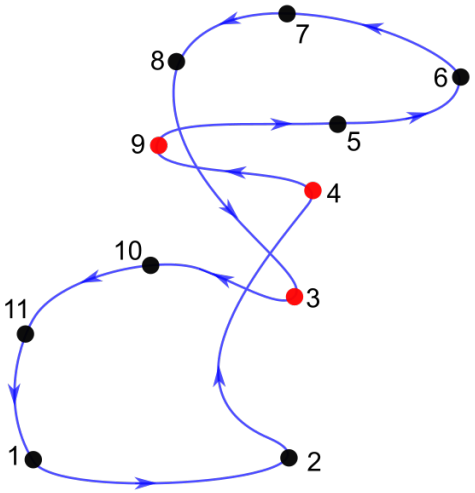


Figure 3.7: Line $L^{'}_{n_i}$ = 1—2—**4**—**9**—5—6—7—8—**3**—10—11—1.

As the two previous method, we do the station shuffling also for a few lines simultaneously. The algorithm is presented as in the following,

**Algorithm 7:** lineChangeC

**Input:** $\mathcal{L}$
**Output:** $\mathcal{L}^{'}$

$\hat{\mathcal{L}}$ := randomly choose a small subset of $\mathcal{L}$
$\mathcal{L} := \mathcal{L} \setminus \hat{\mathcal{L}}$
**for** *all* $L \in \hat{\mathcal{L}}$ **do**
    $\lambda$ := choose a random small number of stations in $L$
    $l := \{v_1, ..., v_\lambda\}$ randomly choose $\lambda$ stations in $L$
    $l^{'}$ := shuffle the order of elements in $l$
    $L^{'}$ := replace $l$ with $l^{'}$ in $L$
**end**
$\mathcal{L}^{'} := \mathcal{L} \cup \hat{\mathcal{L}}$.

This change, however, can also lead to some errors as the previous methods. Shuffling the stations can at some point cause repeated stations or some of the new lines are already exist in the network. We still have one more line-changing method before we discuss the procedure for handling the errors.

### Exchange stations between lines

The fourth method of changing a line network is described as shuffling some stations between a few lines simultaneously. Instead of shuffling the stations within the line in the previous method, here we choose a number of stations from different lines and then exchange the stations from one line to another. The order of the stations are also randomly changed. Thus, in this method, there are two shuffling process. First we do this to the order of the lines, and then to the order of the chosen stations in each line.

Suppose the chosen lines are $L_n = v_0$—...—$v_{n_k}$ —...—$v_n$ and $L_m = w_0$—...—$w_{m_k}$—...—$w_m$ and let the vertices $v_{n_k}$ and $w_{m_k}$ be the chosen stations from each line to be exchanged. The resulting lines after changing are $L_n^{'} = v_0$—...—$v_{n_k-1}$—$\boldsymbol{w_{m_k}}$—$v_{n_k+1}$—...—$v_n$ and $L_m^{'} = w_0$—...—$w_{m_k-1}$—$\boldsymbol{v_{n_k}}$—$w_{m_k+1}$—...—$w_m$.

Figures 3.8 and 3.9 below show an example before and after switching stations 6 and 13 in lines $L_n$ = 7—**6**—5—4—3—2—1—7 and $L_m$ = 14—**13**—12—11—10—9—8—14. After exchanging station number 6 and 13, we get the resulting lines $L_n^{'}$ := 7—**13**—5—4—3—2—1—7 and $L_m^{'}$ := 14—**6**—12—11—10—9—8—14.

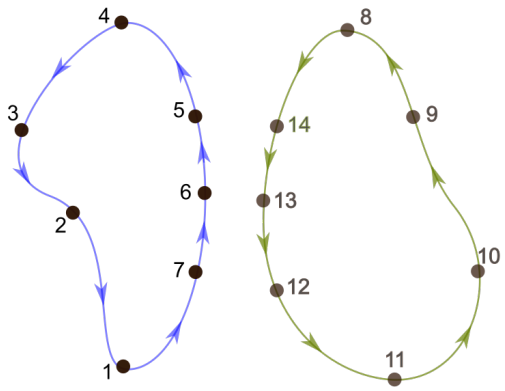


Figure 3.8: The lines $L_n$ and $L_m$ before exchanging stations

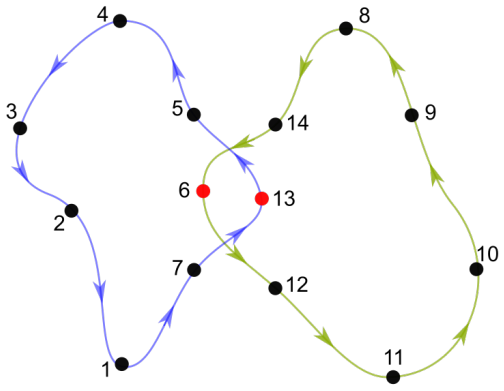


Figure 3.9: The lines $L_n^{'}$ and $L_m^{'}$, after exchanging stations

The algorithm for randomly exchange stations between lines is presented in the following,

**Algorithm 8:** lineChangeD

**Input:** $\mathcal{L}$
**Output:** $\mathcal{L}^{'}$

$\hat{\mathcal{L}}$ := randomly choose a small subset of $\mathcal{L}$
$\mathcal{L} := \mathcal{L} \setminus \hat{\mathcal{L}}$
$M \leftarrow \emptyset$.
**for** *all* $L \in \hat{\mathcal{L}}$ **do**
    $\lambda$ := choose a random small integer
    $l := \{v_1, ..., v_\lambda\}$ randomly choose $\lambda$ stations in $L$
    $l^{'}$ := shuffle the order of element in $l$ $M := M \cup \{l^{'}\}$
    $L^{'}$ := replace $l$ in $L$ by zero.
**end**
$M^{'}$ := shuffle the order of elements in $M$
**for** *all* $L^{'} \in \hat{\mathcal{L}}$ **do**
    $\mu$ := the first element in $M^{'}$
    $L^{'}$ := replace the zero element in $L^{'}$ by $\mu$
    $M^{'} := M^{'} \setminus \{\mu\}$
**end**
$\mathcal{L}^{'} := \mathcal{L} \cup \hat{\mathcal{L}}$.

After exchanging stations between different lines, there are the same potential errors as in exchanging segments in the second line-changing method: the line has existed in the network, forming of consecutive stations, or disconnection of the network may occur. We will discuss the method of checking and fixing the errors in the next subsection.

### Checking and fixing errors in the new line network

We have discussed four line-changing methods previously. Each method has some issues with potential error in the resulting lines. It is a good idea to always check the new lines before we decide to take the change. We will do the checking and fixing in a few steps:

The first check is the easiest one. We examine the resulting lines from the existence of self-loops. If there are consecutively repeated stations, then we fix it by

keeping only one of them and removing rest from the line. Suppose a new line $L_n = v_0$—...—$v_{n_k-1}$—$\boldsymbol{v_{n_k}}$—...—$\boldsymbol{v_{n_k}}$—$v_{n_k+1}$—...—$v_n$ has been made with consecutive repeating stations $v_{n_k}$—...—$v_{n_k}$. Then we remove the sequence and keep only one of the $v_{n_k}$. The result is $L_n^{'} = v_0$—...—$v_{n_k-1}$—$\boldsymbol{v_{n_k}}$—$v_{n_k+1}$—...—$v_n$.

The similar procedure applies when there are multiple such sequences exist in a line. We expect that the line has become shorter, and in some cases it could be too short. This might happen if, for example, after several times of modification, only a single station left in the line. This means that the line is no longer a closed walk, thus we remove it from the network.

Removing a line means that our line network has less lines as before and it can possibly lead to a disconnected network. So it is necessary to check the connectivity of the whole network. The procedure is run by the Breadth First Search algorithm, which is known as an efficient algorithm for finding shortest spanning tree in a graph. In the case that a new line network is disconnected, then we cancel the change and try to make another change instead.

Beside the line with a single station, we also remove the line if it is already exist in the network. This also reduce the line number, $N$. The complete checking and fixing steps is presented in algorithm 9.

**Algorithm 9:** fixLines

**Input:** $\mathcal{L}$, $\hat{\mathcal{L}}$
**Output:** $\mathcal{L}'$

$\mathcal{L} \leftarrow \mathcal{L} \setminus \hat{\mathcal{L}}$
% remove consecutive repeated stations
**for** *all* $L \in \hat{\mathcal{L}}$ **do**
    $l :=$ find sequences of repeating stations in $L$.
    **for** *all* $l(v_k) \in l$ *where* $v_k$ *is a station with self-loop* **do**
        $v_k \leftarrow l(v_k)$
    **end**
**end**
% remove lines that consist of a single station
**for** *all* $L \in \hat{\mathcal{L}}'$ **do**
    **if** $L$ *contains only one station* **then**
        $\hat{\mathcal{L}}' \leftarrow \hat{\mathcal{L}}' \setminus \{L\}$
    **end**
**end**
% checking connectivity of the network
$\mathcal{L} \leftarrow \mathcal{L} \cup \hat{\mathcal{L}}$
$d :=$ BFS($\mathcal{L}$), distances from an arbitrary station to every other stations
**if** $d$ *contains an infinite element* **then**
    return
**end**
% checking the existence of the new lines in the network
**for** *any* $L_i, L_j \in \hat{\mathcal{L}}, i \neq j$, **do**
    **if** $L_i = L_j$ **then**
        $\hat{\mathcal{L}} \leftarrow \hat{\mathcal{L}} \setminus \{L_i\}$
    **end**
**end**

Among the four methods we have discussed, the local search will choose one at every iteration to be implemented. We set weights to every method to let the local search choose any of these methods by the probability based on the weights. We set positive constants $w_1, w_2, w_3$, and $w_4$ such that $w_1 + w_2 + w_3 + w_4 = 1$.

At every iteration, the local search method generates a random number between 0 and 1 to decide which method to be used to change the line network. Suppose we set $w_1 = w_2 = w_3 = w_4 = 0.25$, then each method has similar probability to be chosen.

By performing the line-changing methods, we expect that the local search approaches the optimal solution with minimum cost. We still have another variable, the average travel time, to be minimised during the computation. In the next chapter, we will discuss a method to find express lines in order to reduce the average travel time per passenger.

Chapter 4

# Express bus lines

The method for finding neighbourhood of the line networks we discussed in section 3.4 intended to reduce the operational cost during the iterative computation, but there is no guarantee that the average travel time would be decreased at the same time. In this chapter, we want to introduce a method to find another line network that can be expected to reduce the travel time. Instead of making small change to the network, we perform a relatively large change in the next iteration.

The method we choose adds a few new lines into the network. We call these new lines *express lines*, that is, lines that transport passengers faster as their routes are intended to visit only certain stations and skip the rest in between. By this approach, we expect that the travel time can be reduced, and hence also the average travel time in the network.

However, we are aware of the consequent that adding new lines tends to increase the cost at the same time. The chance to reduce the cost after the new lines involved is tiny. We can still consider the network as a candidate of the solution if the cost increment is reasonable for the time reduction.

## 4.1 Adding direct connections to a line network

As indicated above, the goal of this chapter is to introduce express lines which save time by transporting passengers between two stations without intermediate stops. To that end, we first compute for each pair of stations $(u, v)$ how much time we can

save by adding a direct connection from $u$ to $v$ to the current network.

Recall from section 2.3, that the shortest path from $u$ to $v$ is denoted by a sequence of distinct vertices $P_{u,v} = v_0 — ... — v_j$ where $v_0 = u$ and $v_j = v$, or distinct edges $P_{u,v} := e_1 — ... — e_j$ where $e_i = (v_{i-1}, v_i)$.

Let $t(P_{u,v}) = \sum_{i=1}^{j} t_{e_i}$ be the travel time through the path $P_{u,v}$ and $t_{u,v}$ be the travel time directly on the edge $(u, v)$. Then the saved time by using the direct connection is

$$\tilde{\tau}_{u,v} = t(P_{u,v}) - t_{u,v}. \tag{4.1}$$

where $\tilde{\tau}_{u,v}$ is non-negative, and $\tilde{\tau}_{u,v} = 0$ if $P_{u,v}$ is already a direct connection.

The value of $\tilde{\tau}_{u,v}$ is the time saved per passenger who takes the direct connection instead of going through the original path. To compute the total time saved for all the passengers on the direct connection from $u$ to $v$, we first need to compute the number of passengers in this part.

Assume that the passengers who travel from $u$ to $v$ are all moved to the direct route. Then we need to sum up the demands $d_{i,j}$ for all $i, j$ in the network for which $P_{u,v}$ is a part of the shortest paths from $i$ to $j$. The number of passengers on the direct connection $(u, v)$ is thus

$$d(P_{u,v}) \;=\; \sum_{\substack{(i,j):\\ P_{i,j} \supseteq P_{u,v}}} d_{i,j}, \tag{4.2}$$

and the total saved time for all of them is

$$\tau(u, v) \;=\; d(P_{u,v}) \;\; \tilde{\tau}_{u,v}. \tag{4.3}$$

Applying this computation to all pairs $(u, v)$ in the network, we obtain a new weight function $\tau$ on the complete directed graph $G = (V, E)$ over $V$. This is the graph we defined for the city road network in the first section of chapter 2. On this complete digraph, we use $\tau$ as a weighting function, and then construct closed walks within the graph for the new, faster lines.

In order to save as much time as possible, the closed walks are required to have

large weights. To construct such walks, we have to choose edges with first the large weight and then progressively add another edge with biggest possible weight until we get a closed walk with as large as possible weight.

But then, this method leads us to some difficulties. As the closed walks may visit all vertices, adding as many direct connection as possible is not a good idea to save time. Adding more edges might decrease the average time saved per passenger. Without question, we want to avoid such computation in solving this particular problem.

Instead of computing the total weight of the walks, we propose to take the average weight. An advantage of this method is that there exists an efficient algorithm for solving the problem, namely the *minimum mean cycle algorithm*. This algorithm computes the closed walk with the smallest mean weight. More detail about the algorithm will be discussed in the next section.

## 4.2 Minimum mean-weight cycle

As indicated above, we want to find a closed walk in the network $(G, \tau)$ such that the average weight is as large as possible. Since we want to use the algorithm for finding the minimum mean-weight cycle, the resulting cycle will be the opposite of our goal. To use the algorithm in this particular problem, we need to modify the weighting function in the edge set $E$ such that the minimum weight indicates the maximum saved time. We will use the weighting function $-\tau$ instead of $\tau$, and hence the minimum mean-weight cycle in $(G, -\tau)$ is the maximum in $(G, \tau)$.

In the network graph $(G, -\tau)$, suppose $W = e_1 — ... — e_k$ is a closed walk. The average weight of the edges in $W$ is

$$-\overline{\tau}(W) = \frac{\sum_{i=1}^{k}(-\tau_{e_i})}{k}.$$

We now want to find a closed walk $W^*$ with smallest value $-\overline{\tau}(W^*)$. In [2], the algorithm of minimum mean-cycle is introduced for a strongly connected graph. If the graph is not strongly connected, then we can find a minimum mean-weight

cycle of each strong component of the graph. In our case, $G$ is always a strongly connected graph, thus we skip checking for the strong connectivity.

Let $s$ be any vertex in $G$, then every other vertex in $G$ is reachable from $s$ because of the strong connectivity. For all $v \in V$ and non-negative integers $k$, define $F_k(v)$ to be the minimum weight of a walk of length $k$ from $s$ to the vertex $v$. If no such walk exists, then we define $F_k(v) := \infty$.

As an example, we consider the graph with five vertices in figure 4.1. The walks of lengths $k = 0, 1, 2, 3, 4$ from $s$ to $v$ with minimum weights are shown on the graph below.

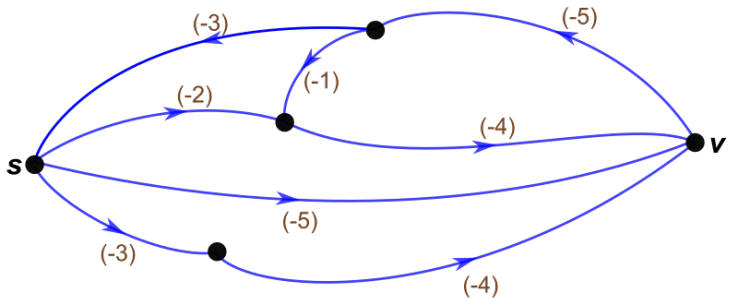


Figure 4.1: A directed graph with five vertices

$$
\begin{aligned}
k = 0: &\quad F_k(v) = \infty, \\
k = 1: &\quad F_k(v) = -5, \\
k = 2: &\quad F_k(v) = \min\{-2-4\ ,\ -3-4\} = -7, \\
k = 3: &\quad F_k(v) = \infty, \\
k = 4: &\quad F_k(v) = \min\{-5-5-1-4\ , -5-5-3-5\} = -18.
\end{aligned}
$$

We use the values of $F_k(v)$ for all $v \in V$ to find the minimum weight of all possible closed walks as introduced in the theorem below.

**Theorem 4.1.** *In the strongly connected graph $G$ with $n$ vertices, the minimum cycle mean is*

$$\lambda^* = \min_{v \in V} \max_{0 \le k \le n-1} \left[ \frac{F_n(v) - F_k(v)}{n-k} \right] \tag{4.4}$$

*Proof.* The proof of this theorem can be found in [2]. □

Beside the minimum mean-weight computed in (4.4), we are also interested in producing the cycle whose mean-weight is $\lambda^*$. korte and Vygen in section 7.3 in [5] suggest the Minimum mean cycle algorithm that constructs a closed walk $C$ with minimal average weight $\lambda^*(C)$, or decides that the graph is acyclic.

Our graph $G$ is a complete digraph which is strongly connected, thus we skip the computation of the connectivity. With the weighting function $\tau$ as an input, the algorithm gives outputs the minimum mean cycle $C$ and the minimum average weight $\lambda^* = -\overline{\tau}^*(C)$. The algorithm is presented below,

**Algorithm 10:** Minimum mean-weight closed walk

**Input:** $G, \tau$
**Output:** $C$

Add to $G$ a vertex $s$ and edges $(s, v)$
$\tau((s, v)) \leftarrow 0$ for all $v \in V$
$n \leftarrow$ number of vertices, $p \leftarrow \emptyset$
$F_0(s) \leftarrow 0$, $F_0(v) \leftarrow \infty$ for all $v \in V \setminus \{s\}$.
**for** $k = 1$ *to* $n$ **do**
    **for** *all* $v \in V$ **do**
        $F_k(v) \leftarrow \infty$.
        **for** *all edges of the form* $(w, v)$ **do**
            **if** $F_{k-1}(w) - \tau((w, v)) < F_k(v)$ **then**
                $F_k(v) := F_{k-1}(w) - \tau((w, v))$ and $p_k(v) := w$.
            **end**
        **end**
    **end**
**end**
$v :=$ a vertex for which $\max\limits_{\substack{0 \le k \le n-1 \\ F_k(v) < \infty}} \dfrac{F_n(v) - F_k(v)}{n - k}$ is minimum.
$C := p_1(p_2(...(p_n(v))))$—...—$p_{n-1}(p_n(v))$—$p_n(v)$—$v$, a closed walk.

The algorithm in the previous section returns a closed walk $C$ with the smallest mean-weight. As a walk is allowed to visit a vertex multiple times, $C$ might include multiple loops on the same routes. We keep one of the loops and remove the rest from $C$. Suppose algorithm 10 gives us the minimum mean-weight closed walk: $v_0$—...—$\boldsymbol{v_{k_1}}$—**...**—$\boldsymbol{v_{k_j}}$—$\boldsymbol{v_{k_1}}$—**...**—$\boldsymbol{v_{k_j}}$—$\boldsymbol{v_{k_1}}$—...—$v_n$—$v_0$, then we remove one of the sequences in the form $v_{k_1}$—...—$v_{k_j}$. Our express line is then $v_0$—...—$\boldsymbol{v_{k_1}}$—**...**—$\boldsymbol{v_{k_j}}$—$\boldsymbol{v_{k_1}}$—...—$v_n$—$v_0$.

## 4.3 Express lines in the line network

Together with the express line from the computation above, the new line network $\mathcal{L}$ is expected to provide a lower travel time. In a larger network, it is a good idea to add more than one express line. By including the first express line into the line network $\mathcal{L}$, we can compute again another express line. The computation steps follow the similar procedure as we construct the first express line. We repeat the process until the number of maximum express lines is reached or the total saved time is no longer significant. The algorithm below describes the repeating process,

**Algorithm 11:** Express lines

**Input:** Complete graph $G$, line network $\mathcal{L}$, minimum saved time $\epsilon$, number of express line $\ell$

**Output:** $\mathcal{L}_{ex}$

$\mathcal{L}_{ex} \leftarrow \emptyset,\ \tau \leftarrow \infty$
**while** $\tau > \epsilon$ *and* $|\mathcal{L}_{ex}| < \ell$ **do**
    $P :=$ passenger shortest paths
    $t(P) :=$ travel time in $\mathcal{L} \cup \mathcal{L}_{ex}$
    Add direct connections into $\mathcal{L} \cup \mathcal{L}_{ex}$
    $\tau :=$ saved time by using direct connections
    $C :=$ Minimum mean-weight closed walk $(G, \tau)$
    $\mathcal{L}_{ex} \leftarrow \mathcal{L}_{ex} \cup \{C\}$
    $l :=$ number of element in $\mathcal{L}_{ex}$
**end**

When a set of express line $\mathcal{L}_{ex}$ is established and added into the line network $\mathcal{L}$, we compute the operational cost and average travel time for the whole line network $\mathcal{L} \cup \mathcal{L}_{ex}$. The cost might increase because the number of lines has increased, but the travel time should be lower than before. This is the main goal of adding the express lines, that some of the passengers are transported faster.

We call this procedure *line-optimisation cycle*, a cycle consisting of a number of local search iteration and a step of express lines inclusion. If the tavel time is satisfactory, then the computation is done and we choose our optimal solution among the solutions on the Pareto front. Otherwise, we will try to improve our solution by performing another line-optimisation cycle.

In the second cycle, we perform again the local search to minimise the operational cost over $\mathcal{L} \cup \mathcal{L}_{ex}$. During the search, we keep the express lines in $\mathcal{L}_{ex}$ fixed while $\mathcal{L}$ is regularly replaced by the other one in its neighbourhood. The solution is a line network $\mathcal{L}$ such that $\mathcal{L} \cup \mathcal{L}_{ex}$ gives minimum operational cost. If the average travel time is satisfactory then we stop the computation. Otherwise, we try again to improve the solution in the next cycle.

For the third cycle, we want to test a different express line set $\mathcal{L}'_{ex}$. First, we remove the existing express lines $\mathcal{L}_{ex}$ from the line network and re-compute the cost and travel time. Then we construct new express lines based on the line network solution. We use the new express lines together with our current solution as an initial guess for the local search. During the computation, the express lines are kept fixed as we did previously. Then the same steps follow the local search until the end of the cycle, and we can repeat this computation cycle several times until we are satisfied with the average travel time, or a maximum number of cycle has reached.

An extra attention must be paid in the replacement of the express lines. It is possible that the network becomes disconnected or some stations are missing when the old express lines are removed, and the new express lines do not connect the components back. Thus, we check the connectivity of the line network $\mathcal{L} \cup \mathcal{L}_{ex}$, where $\mathcal{L}_{ex}$ is the new express line set. If the network is disconnected or there are some stations missing, then we need to include back one or more of the old express lines until all stations are connected to each other. It can be done as long as the number of lines does not exceed the upper bound $N_2$.

We modify algorithm 11 by taking the old express lines as additional input variable $\mathcal{O}_{ex}$ and add a few lines at the end of the algorithm:

---

1 **if** $\mathcal{L} \cup \mathcal{L}_{ex}$ *makes a disconnected network* **then**
2  take a subset $\tilde{\mathcal{O}}_{ex} \subset \mathcal{O}_{ex}$ such that $\mathcal{L} \cup \mathcal{L}_{ex} \cup \tilde{\mathcal{O}}_{ex}$ makes a connected network
3 **end**
4 $\mathcal{L}_{ex} \leftarrow \mathcal{L}_{ex} \cup \tilde{\mathcal{O}}_{ex}$

---

At the end, we expect to have several good solution candidates; therefore, it is important to identify the solutions with relatively low cost and low travel time. We

will choose one of the the possible solutions lie on the Pareto front on the diagram showing all solutions in the form $(time, cost)$.

### Summary

In this chapter we have discussed about the express lines, how to produce and include them in the line network. With the express lines in the network, the potential passengers have more options to travel faster, hence the average travel time in the network is expected to be lower than before.

Even though replacing express lines in the beginning of the third line-optimisation cycle does not guarantee to give a lower travel time than in the first two cycles, we would expect some reductions in a longer optimisation process. We will see the long term behaviour in the numerical experiment in the next chapter.

We end this chapter with a summary of our method for solving the line planning problem. After the inclusion of express lines, we add a few more steps to our chart in figure 3.1. The complete computation steps are presented in figure 4.2.

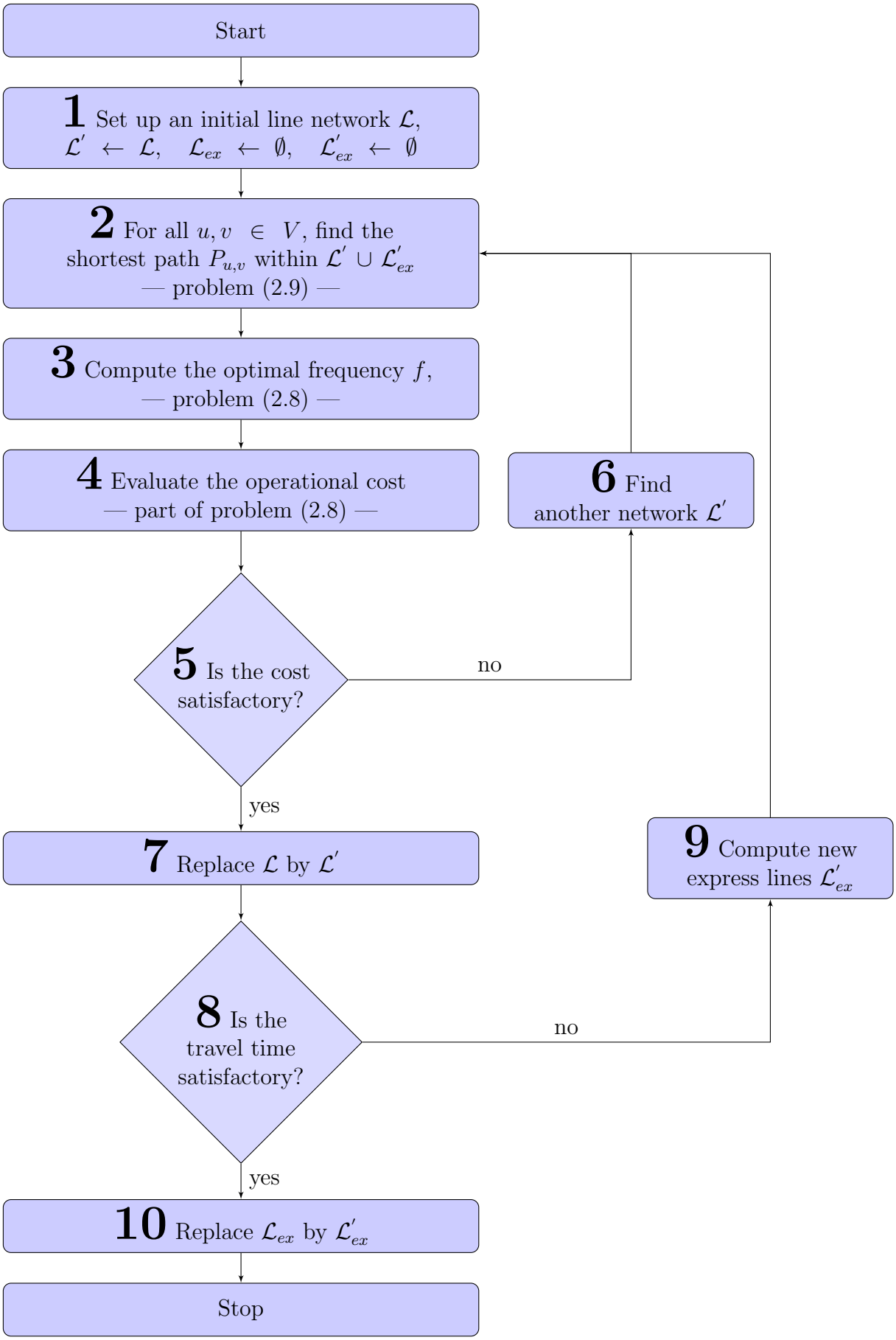


Figure 4.2: The computation steps of bus routing optimisation

Chapter 5

# Numerical experiments and discussions

In this chapter, we will discuss the numerical experiments while implementing the ideas we have discussed previously. We tested our computation with two networks of different size: 10 and 30 stations that are made as follows:

First, we create a road network based on a real map. We choose a special area with its bus station locations, and then generate some data for our input. In this experiment, we use Google interactive map and restrict the area for Trondheim municipality. For the travel time between each pair stations, we use the direct driving time provided by the map, and then add a constant to each of it as the boarding and alighting time at the bus stations. Since Google map gives us the travel time in minute, then we add the stopping time also in the same time unit. Assume that the buses need in average 30 seconds stopping time at each station, then we add 0.5 minutes to every connection.

The value of the travel cost between stations are not provided by the map. Therefore, we take the relevant distance that is given simultaneously with the travel time by the map and scale it by some constants. For the same reason as we added extra stopping time, we also add a constant, in this case 0.25.

For the demand, we do not have any real or relevant scaled data. So we create the demand values by generating random numbers between 20 and 40 from every station to every other station, except one particular station that we set as a center point of the area. This can be, for example, the city center having a higher demand. The demand from everywhere else to this station is set to be random numbers between

40 and 60. We expect that the result will show that this particular station is visited by more lines. In this experiment, this special station is named as *Station-7*.

Denote the 10-stations network by $\mathcal{N}_{10}$ and the 30-stations by $\mathcal{N}_{30}$. We will do the multiple line-optimisation cycle to each network, starting with $\mathcal{N}_{10}$.

## 5.1 Line network optimisation in $\mathcal{N}_{10}$

We set our first experiment for 10 computation cycles and discuss the solutions in three stages: 1) the solution obtained from the first local search computation; 2) the solution at the end of the first cycle, i.e. after we added an express line; and 3) the complete long term computation.

At each cycle except the first one, we use the express lines together with the other lines as initial guess in the local search part, and we end the cycle by replacing the express lines as we have discussed in section 4.3.

### 5.1.1 Local search only

We start our experiment with the local search method to optimise only the operational cost. The target cost is set to be very low, 10% of the initial cost because we want to let the local search iterates until the maximum number of iteration is reached. The maximum number of iteration $i_{max}$ is set to be 10,000, including the computation of cost and travel time of the initial guess. We set 3 initial lines following the procedure in section 3.1 and set the lower and upper bounds $N_1 = 2$ and $N_2 = 4$, respectively. In this stage, however, we expect the same number of lines or decrease by relatively small number at the end of the computation. There is no possibility to have a new line during our local search method.

During the computation, we keep the solution whenever it gives a lower cost. Figure 5.1 shows the operational cost decreased significantly in the first 725 iterations. After that, it slightly decreased until the iteration of 3,346. The local search did not find a better line network in the later iterations. We recorded the same iteration numbers as the cost decreased to see the travel time behaviour. Figure 5.2 shows that the travel time fluctuated during the first 415 iterations before it

consistently decreased.

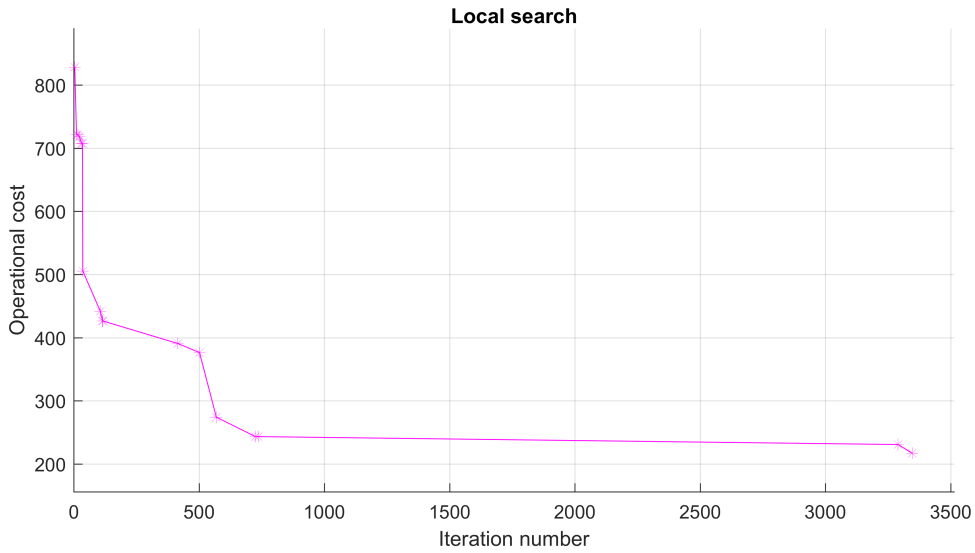


Figure 5.1: Operational cost vs. iteration number for $\mathcal{N}_{10}$. The data points are recorded when the solutions give lower operational cost. The cost decreased significantly in the first 725 iterations, and decreased more at iteration 3,290 and 3,346.

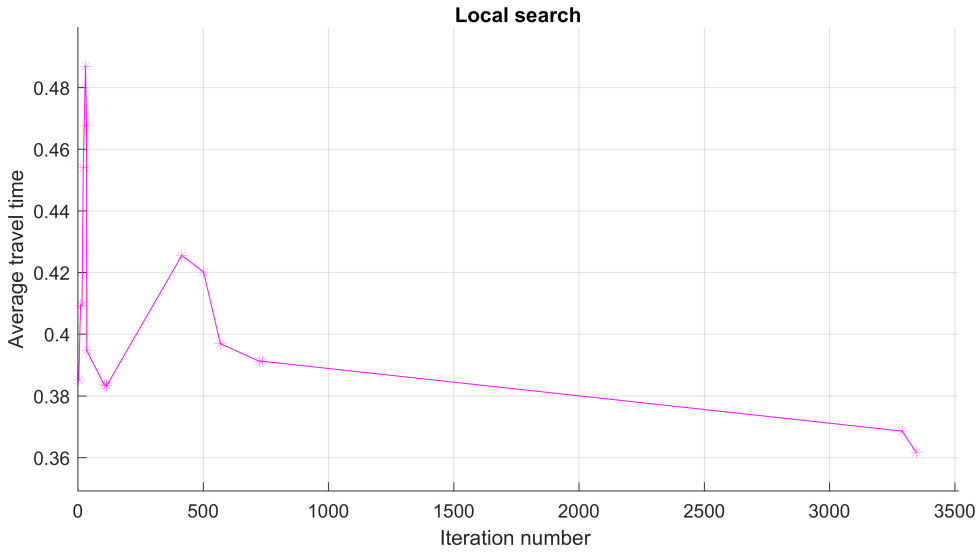


Figure 5.2: Average travel time vs. iteration number for $\mathcal{N}_{10}$. The data points are the same iteration numbers as in the previous figure, recorded when the solutions give a lower cost. The average travel time fluctuated during the first 415 iterations and then constantly decreased as the cost decreased.

It is also interesting to see how the local search behaves. The dots on figure 5.3 represent all solutions during the local search iterations. The more condensed dot

population lie in the direction approaching the solution, marked by a green circle. This can be interpreted as a good behaviour of the local search. To be precise, the line-changing methods described in section 3.4.2 are good enough to produce the neighbour line networks.

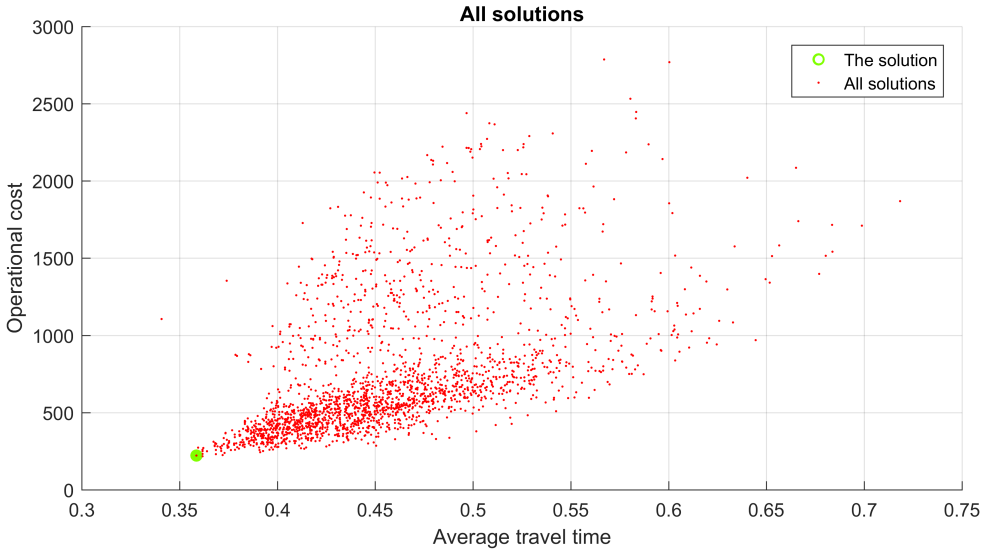


Figure 5.3: Operational cost vs. average travel time for the 10-stations network.

The best solution is found at iteration 3,346 with operational cost 217.0751 unit and average travel time 0.3617 unit. It is a line network consisting of three lines, described in table 5.1 and illustrated in figure 5.4.

| Line number | Route | Frequency |
|---|---|---|
| $L_1$ | 10—6—7—10 | 1.5876 |
| $L_2$ | 3—8—5—1—7—3 | 2.1533 |
| $L_3$ | 3—2—9—3—7—4—3 | 1.1021 |

Table 5.1: Three lines in the optimal line network in $\mathcal{N}_{10}$.

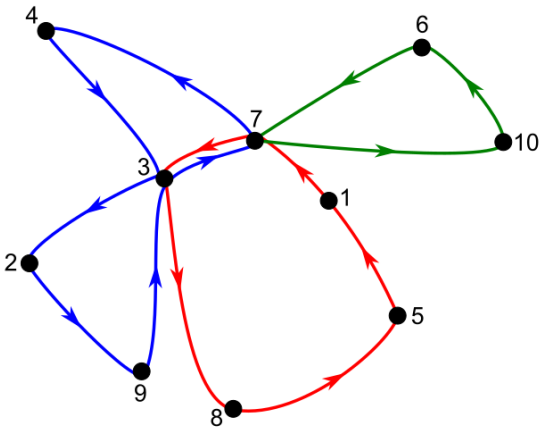


Figure 5.4: The optimal line network on $\mathcal{N}_{10}$, searched by the local search method. $L_1$ is coloured green, $L_2$ is coloured red, and $L_3$ is coloured blue.

### 5.1.2 Adding express lines to the local search solution

In the second stage of our discussion, we will compare the last solution with the new one after an express line has been included. We added only one express line because the network is small and adding an express line makes our line network reaches the upper bound $N_2 = 4$.

The method used for finding the express line, the minimum mean-weight cycle algorithm, is designed in such a way that the travel time must be reduced. Thus, we have a guarantee that once we add the express line to the network, the average travel time must be reduced. While the operational cost could be increased at this stage. Figure 5.5 and 5.6 show the the change right after an express line has been added, marked with in the blue dots.

While the average travel time significantly decreased, the operational cost has a light increment. This solution appears in the Pareto front as one of the best solutions. Figure 5.7 shows the Pareto front together with some of the other solutions. The Pareto solutions are the three points marked with green circles. We could choose the one with blue dot inside, since it has much lower travel time than the other two candidates. This is the solution founded at the last iteration, when we added the express line to the network.

Once we decided the optimal solution to be taken, we could find the lines in this line network solution as presented in table 5.8 and figure 5.2.

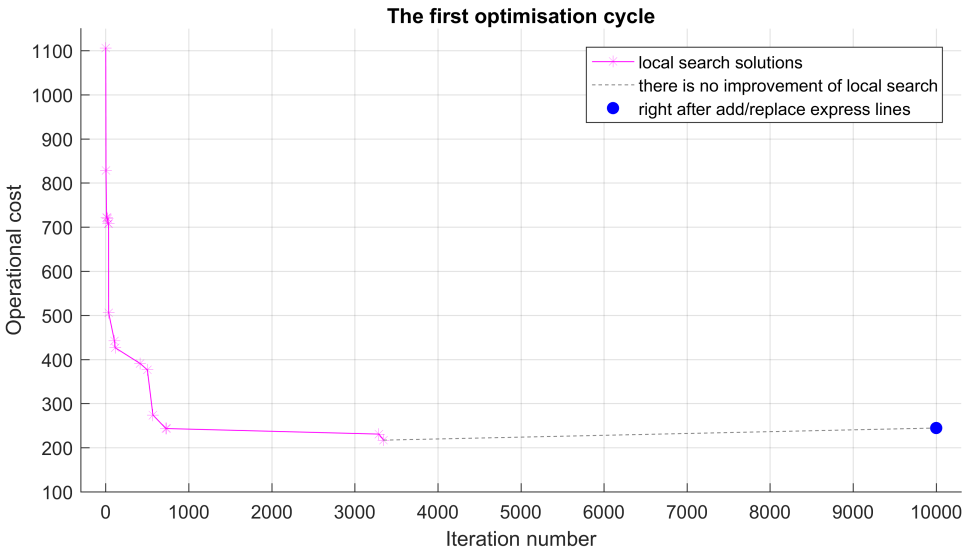


Figure 5.5: The first optimisation cycle on $\mathcal{N}_{10}$. The local search solution was founded at iteration 3,346. At the end of the local search, an express line was added into the optimal line network, causing a slight increment to the operational cost.

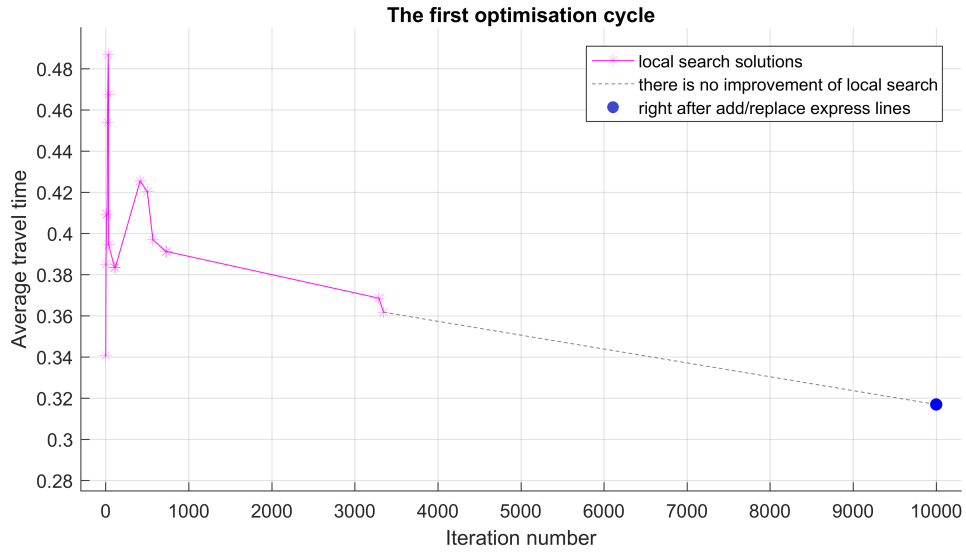


Figure 5.6: The first optimisation cycle on $\mathcal{N}_{10}$. The points were recorded at the same time when the cost decreased. At the end of the local search, an express line was added into the optimal line network, significantly lowering the average travel time.

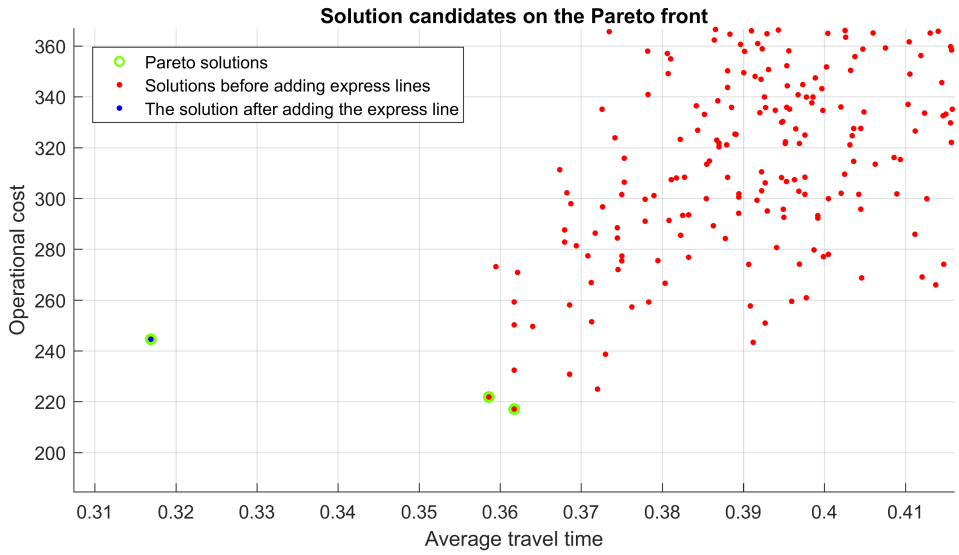


Figure 5.7: The Pareto solutions right after the addition of the express line

| Line number | Route | Frequency |
|---|---|---|
| $L_1$ | 10—6—7—10 | 0.9200 |
| $L_2$ | 3—8—5—1—7—3 | 2.5597 |
| $L_3$ | 3—2—9—3—7—4—3 | 1.1021 |
| $L_{ex}$ | 6—5—10—5—10—3—6 | 0.6125 |

Table 5.2: Lines in the line network with the lowest travel time

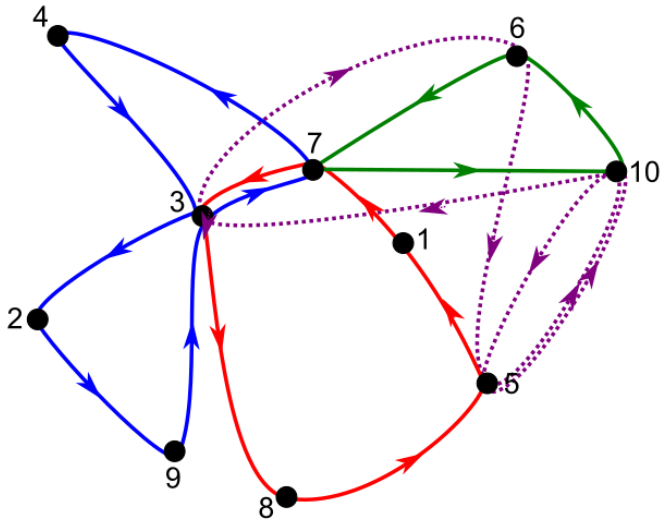


Figure 5.8: The line network in $\mathcal{N}_{10}$ with an express line in the purple dotted line.

The operational cost for this line network configuration is 244.5719, which is 12.68% higher than the operational cost before adding the express line. In the other hand, the travel time decreased 12.38% from 0.3617 to 0.3169 unit time. Hence, the express line constructed by the minimum mean cycle algorithm works. However, we want to continue the computation with the rest 9 cycles to see the long term behaviour.

### 5.1.3 The complete computation

Now we continue the computation by repeating the same procedure as in the first cycle. The difference is that we have an express line in the line network. During the local search, we keep the express line fixed while the other lines could be modified at every iteration.

In the second cycle, the local search reduced the cost twice, while the travel time within the associate cost reduced once and then remain in the same level. This is shown in figures 5.9 and 5.10 by the pink stars and lines. We end the second cycle by replacing the express line. First we removed the existing express line.

Based on the remaining lines in the line network, we compute again the travel time and then construct a new express line using the same procedure as in the previous express line construction. Our data in figure 5.10 shows that the new express line did not lower the travel time. This means that the old express lines did better in reducing time. This possibility has been discussed in chapter 4.

We let the new one to be included in the line network, and use them together as an initial guess for the next computation cycle. The line network solution from the last local search will be the initial guess together with the new express lines.

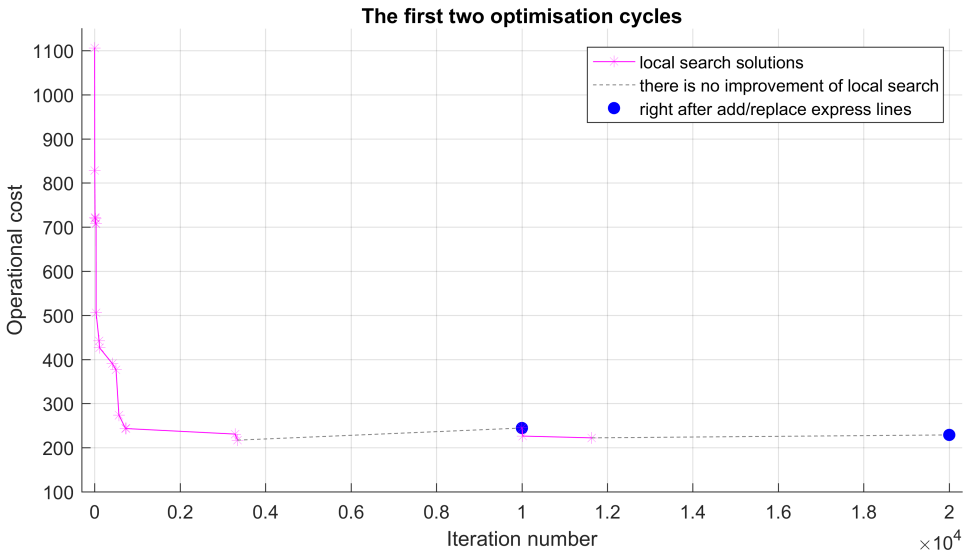


Figure 5.9: The first two computation cycle. At the end of the second cycle, a new express line has replaced the old one.

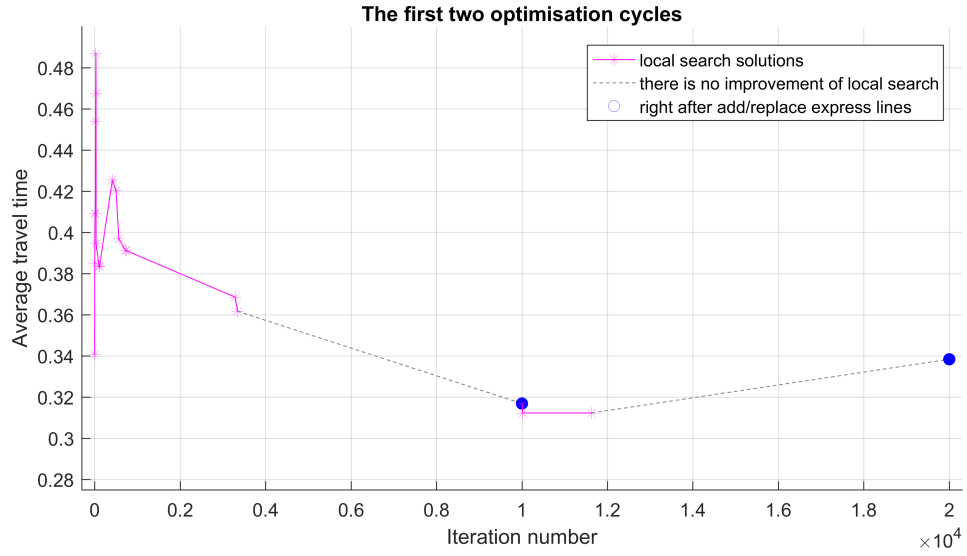


Figure 5.10: The first two computation cycle. At the end of the second cycle, a new express line has replaced the old one.

We continue the computation to the third cycle. At the end of this cycle, we replace again the express line with a new one. This time, we have travel time reduction in the contrary to the previous cycle while the cost remain increased. At the end of computation, 10 optimisation cycles have been made, and we present the operational cost and associate travel time for the complete computation process in figures 5.11 and 5.12.

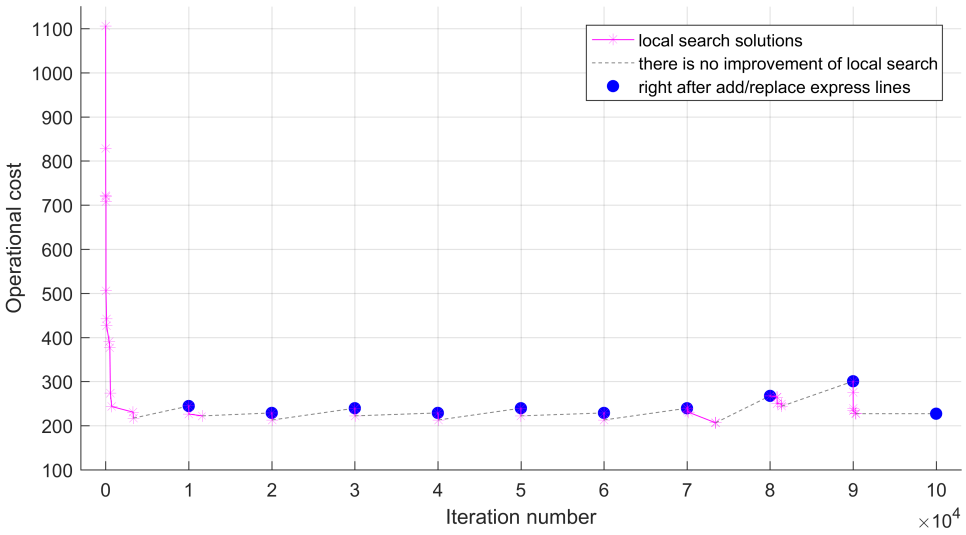


Figure 5.11: The operational cost slightly increased every time the express line being replaced and lowered again, but there is no good improvement of the solution.

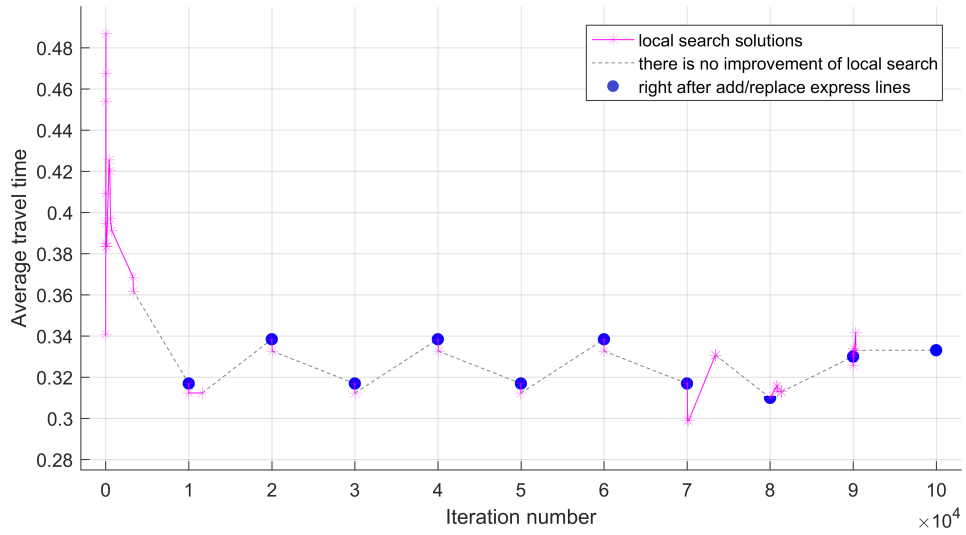


Figure 5.12: The travel time oscillated by the express line replacements.

Figure 5.12 shows that the travel time decreased when we added an express line for the first time. In the later 9 cycles, replacing express line caused 5 times reduction and 4 times increment of travel time. While the cost has always increased at every time we added or replace the express line, as shown in figure 5.12.

We plot the solutions on the Pareto front together with some other solutions that lie nearby. The red dots indicate the solutions from the first cycle, before we added

the express line. The blue dots indicate the solutions after we added or replaced the express line. It is clear that the existence of express line is very important to reduce the travel time.

We choose one of the best cost-time combination solutions lie on the Pareto front in figure 5.13. There are five available solutions from the complete computation process, with the best cost-time combination as presented in table 5.3.

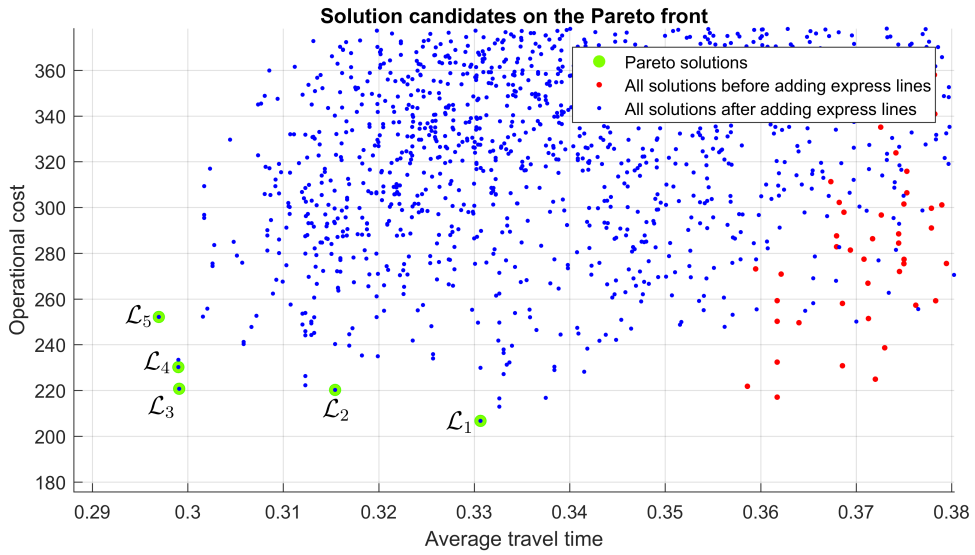


Figure 5.13: The five Pareto solutions together with some of the other solutions.

| Solution candidate | Operational cost | Average travel time |
|---|---|---|
| $\mathcal{L}_1$ | 206.7021 | 0.3306 |
| $\mathcal{L}_2$ | 220.2366 | 0.3154 |
| $\mathcal{L}_3$ | 220.7350 | 0.2991 |
| $\mathcal{L}_4$ | 230.2126 | 0.2990 |
| $\mathcal{L}_5$ | 252.1360 | 0.2970 |

Table 5.3: Solution candidates on the Pareto front from the complete 10 cycles optimisation process.

Suppose we choose line network $\mathcal{L}_3$ as the optimal solution, The table and the sketched graph in the following show the idea of our optimal line network.

| Line number | Route | Frequency |
|---|---|---|
| $L_1$ | 10—6—7—10 | 1.1979 |
| $L_2$ | 8—3—7—1—5—8 | 2.1313 |
| $L_3$ | 3—2—9—3—4—3 | 1.1021 |
| $L_{ex}$ | 6—5—10—5—10—3—6 | 0.7178 |

Table 5.4: Lines in the optimal line network $\mathcal{L}_3$

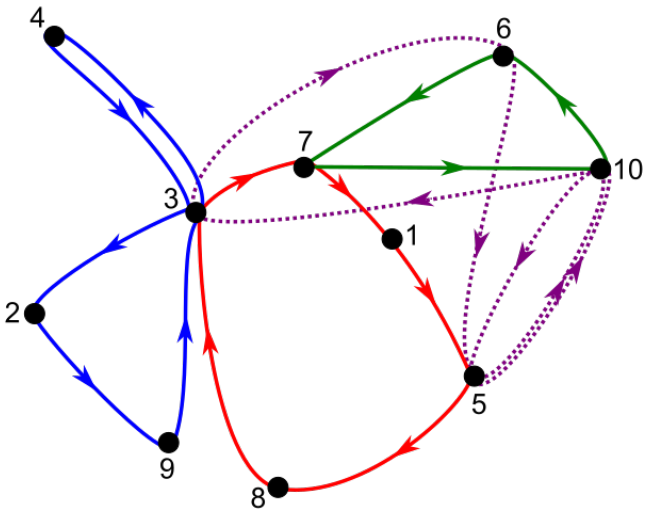


Figure 5.14: The optimal line network for the 10-stations network $\mathcal{N}_{10}$.

We have discussed the result from our experiment with 10 stations network. In short, the summary of the results is presented below:

| **Procedure** | **Operational cost** | **Average travel time** |
|---|---|---|
| The local search only | 217.0751 | 0.3617 |
| local search solution plus express line ($1^{st}$ cycle) | 244.5719 (increased by 12.67%) | 0.3169 (decreased by 12.39%) |
| Complete computation | 220.7350 (decreased by 9.75%) | 0.2991 (decreased by 5.63%) |

## 5.2 Line network optimisation in $\mathcal{N}_{30}$

In this section, we will briefly present the results from the experiment with the 30-stations network $\mathcal{N}_{30}$. The maximum number of iteration in local search is 2,500, and the maximum computation cycle is 4.

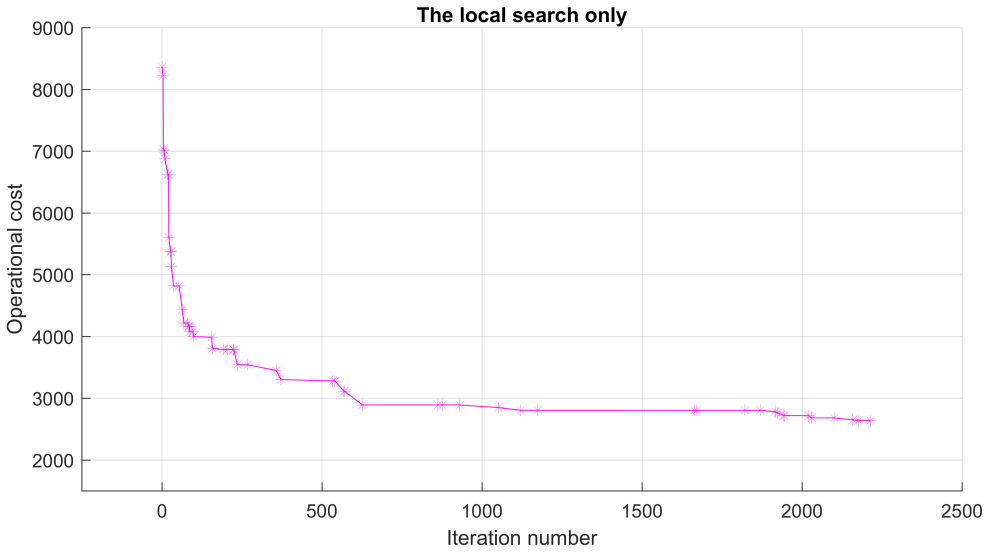


Figure 5.15: Local search procedure on the network $\mathcal{N}_{30}$. The operational cost decreased very fast in the first 627 iteration, then slower down until itration 2,213. This gives the lowest cost 2,636.5589 unit.

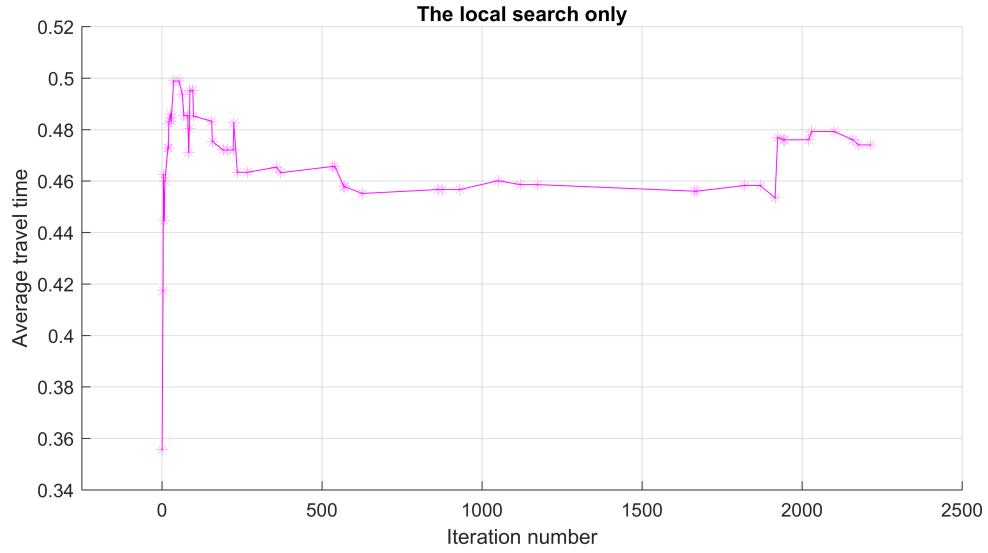


Figure 5.16: Local search procedure on the network $\mathcal{N}_{30}$. The average travel time tended to increase while the associate cost decreased. At the point when the cost is lowest, the average travel time is 0.4740.

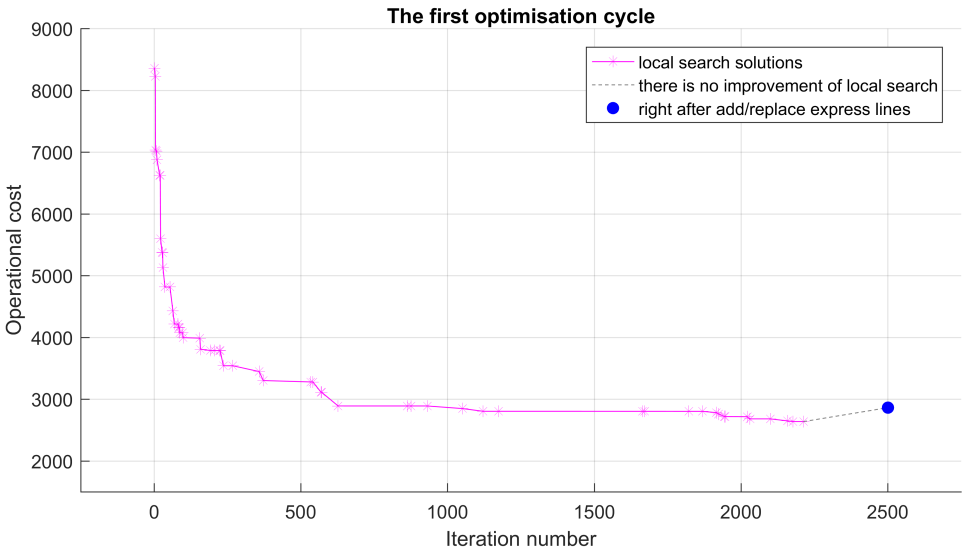


Figure 5.17: The first optimisation cycle. The cost slightly increased by 8.56% to 2,862.3387 unit when the express lines included.

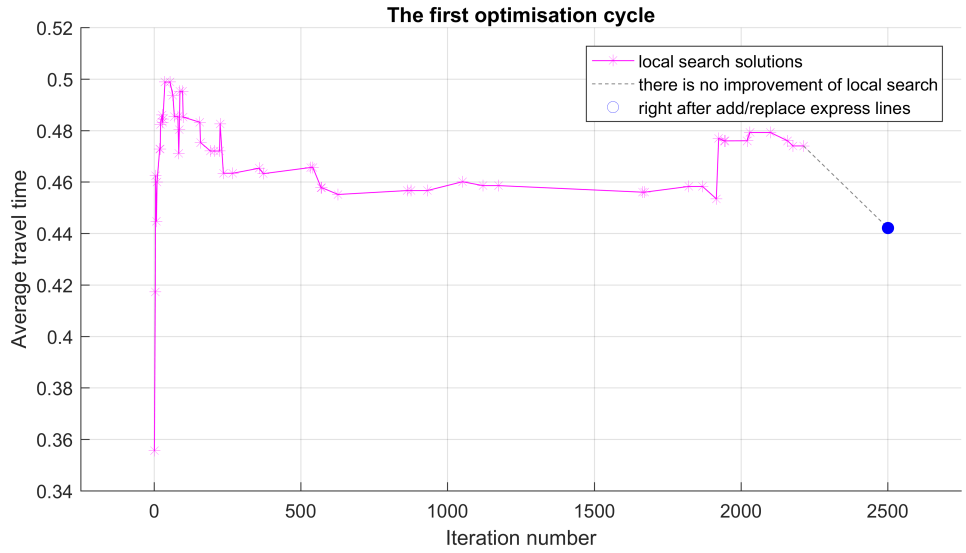


Figure 5.18: The first optimisation cycle. The travel time decreased by 6.73% to 0.4421 unit from the previous solution.

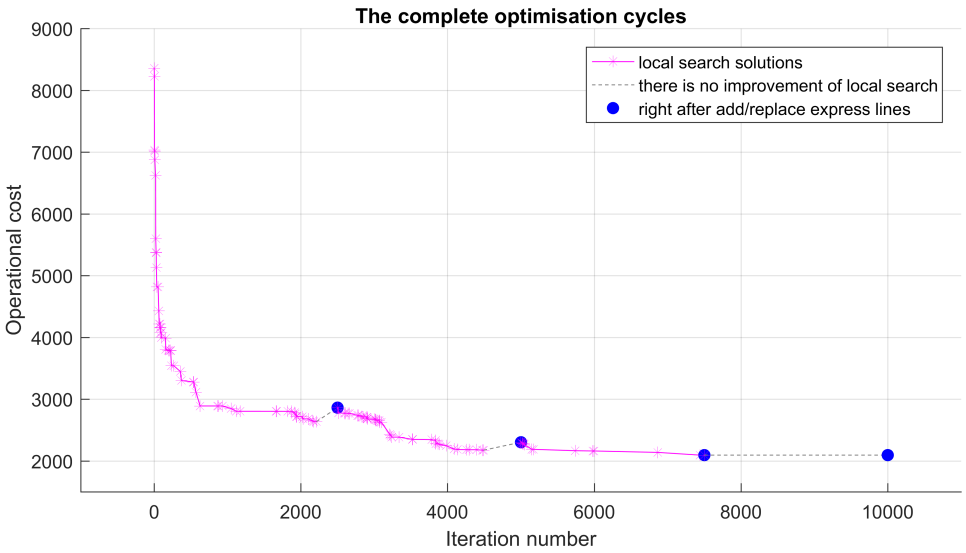


Figure 5.19: The operational cost during the complete computation. The local search improved the solution in almost every computation cycles. Only in the last computation cycles, the local search did not improve solution. Adding and replacing the express lines, however, never decrease the cost.

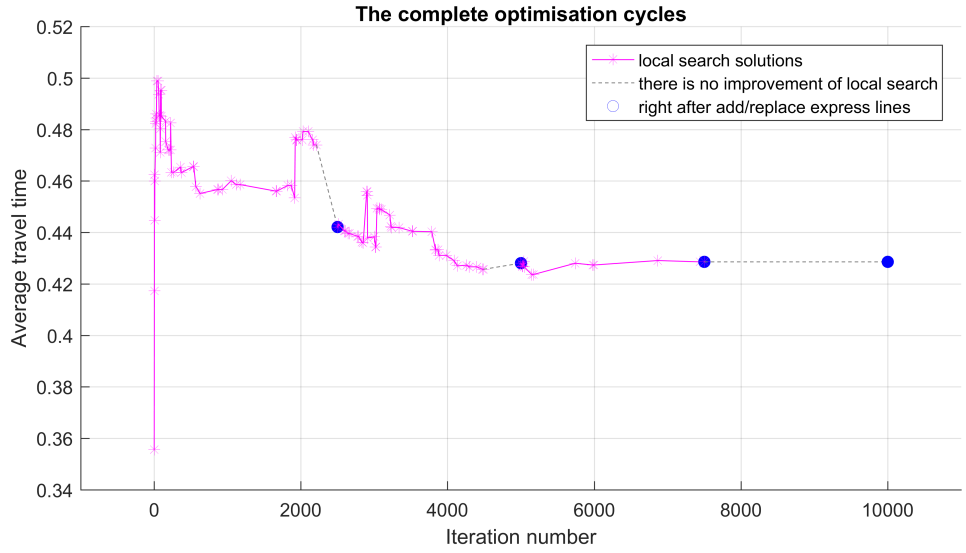


Figure 5.20: The average travel time during the complete computation. The points recorded as the same time when the associate operational cost improved or in addition or replacement of express lines.

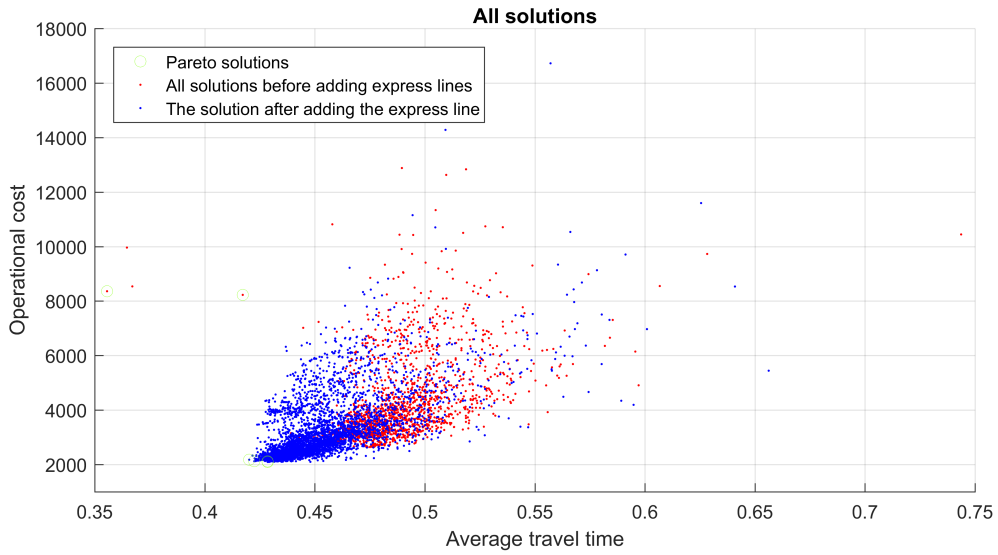


Figure 5.21: The plot of all solutions in the form of *(cost,time)*. The most dense area is in the nearby of some of the Pareto solutions.

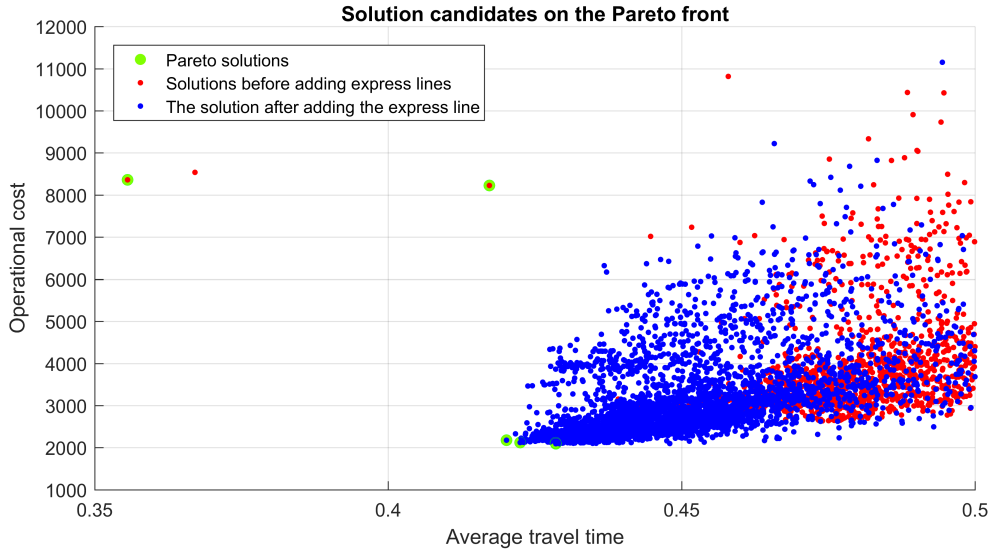


Figure 5.22: The solutions on the Pareto front together with some of the other solutions nearby. There are several red dots lie on the Pareto front, can be interpreted as: some line network without express lines could transport faster. This is possible because the way of constructing the initial guess was considering the shortest path of the passengers.

Among the Pareto solutions in figure 5.22, we could choose the most left hand side with blue dot inside because it gives extremely lower cost than the other two points with red dots inside, and lowest travel time among the blue dots. This point lies in the coordinate (2,177.7166 , 0.42016). This was recorded at iteration 9,170.

The line network associated with this point is as described in table 5.5.

| Line number | Route | Frequency |
|---|---|---|
| $L_1$ | 20—7—18—20 | 5.4753 |
| $L_2$ | 2—19—25—9—30—2 | 4.3249 |
| $L_3$ | 23—11—30—7—14—3—7—30—23 | 13.3765 |
| $L_4$ | 8—3—27—7—8 | 2.8654 |
| $L_5$ | 1—29—5—17—7—1 | 11.2668 |
| $L_6$ | 7—4—7—28—7 | 2.9037 |
| $L_7$ | 6—26—7—6 | 5.7947 |
| $L_8$ | 10—21—7—21—10 | 3.7021 |
| $L_9$ | 15—12—16—7—22—13—24—15 | 6.3937 |
| $L_{ex}$ | 7—24—7 | 9.3129 |

Table 5.5: Lines in the optimal line network.

The results from our experiment with 30 stations network are presented in the summary below,

| **Procedure** | **Operational cost** | **Average travel time** |
|---|---|---|
| The local search only | 2,636.5589 | 0.4740 |
| local search solution plus express line ($1^{st}$ cycle) | 2,862.3387 (increased by 8.56%) | 0.4421 (decreased by 6.73%) |
| Complete computation | 2,177.7166 (decreased by 31.44%) | 0.42016 (decreased by 4.96%) |

Chapter 6

# Conclusion and further work

We have defined a line planning problem which is an optimisation problem with two objectives, cost and time. We proposed a heuristic-based method to find a line network on a given city road network. The solution is an optimal line network with relatively low operational cost and average travel time.

We proposed a combination of the local search method, a heuristic method and the minimum mean weight cycle algorithm as an optimisation tool for a line network planning.

The objective of the local search is to optimise the operational cost. The initial line network is created using the many-to-may shortest paths method, based on the Dijkstra algorithm. We proposed a method for the local search to find a new line network in the neighbourhood of the initial solution. To further optimise the travel time, we applied the minimum mean-weight cycle algorithm to construct an express line  a special line visiting certain stations and skipping the rest in between.

We have applied our optimisation method for two study cases, each consisting of 10 and 30 stations, respectively. In both cases, the local search method could find significantly lower cost solutions compared to the initial solutions. Adding an express line for the first time to the line network decreased the average travel time. However, it increased the operational cost.

Performing further optimisation cycle showed that for the 10-stations case, the local search did not find any better solution. But, for the 30-stations case, the local search found some better solutions. Express line replacement did not always reduce

the average travel time. On the other hand, it always the operational cost, which is a logical consequence of adding a new line. Finally, we plotted all the solutions on a chart, to find the best combination of operational cost and average travel time solutions. The Pareto solutions can be further evaluated by considering the trade-off between the lowest cost and the lowest average travel time solutions.

**Further work**

We propose some possible improvements can be done in the future:

- We could add some more constraints into the problem, such as limiting the number of line transfer.
- We could vary the vehicle capacity.
- It is possible to optimise with respect to the station locations.
- After a line network is established, we could generate a time table such that the actual travel time can be computed including the waiting time.

# List of Symbols

| | |
|---|---|
| $v$ | a vertex represents a bus station location |
| $e$ | an edge represents the direct connection between two bus station locations |
| $V$ | vertex set, where $v \in V$ is a location for a bus station |
| $E$ | edge set, where $e \in E$ is a direct connection roads between two stations |
| $G$ | complete graph of the city road network |
| $t_e$ | travel time on edge $e \in E$ |
| $c_e$ | operational cost on edge $e \in E$ |
| $d_{u,v}$ | demand for transportation from $u$ to $v$ in $V$ per unit time. |
| $d_e$ | total demand on edge $e$ per unit time. |
| $N$ | number of lines in a line network |
| $N_1$ | lower bound for $N$ |
| $N_2$ | upper bound for $N$ |
| $M$ | capacity of a bus |
| $\mathcal{L}$ | a line network |
| $L_n$ | a bus line in line network $\mathcal{L}$ |
| $f_n$ | frequency of buses serving in line $L_n$ |
| $f$ | frequency set for line network $\mathcal{L}$ |
| $(\mathcal{L}, f)$ | a line network configuration |
| $\mathcal{L}^*$ | the optimal line network |
| $f*$ | frequency set of the optimal line network $\mathcal{L}^*$ |

| | |
|---|---|
| $E'$ | edge set of the line network $\mathcal{L}$ |
| $G'$ | graph of a line network $\mathcal{L}$ |
| $P$ | path in a graph |
| $t(P)$ | total travel time on path $P$ |
| $P_{u,v}$ | passenger shortest path from $u$ to $v$ |
| $d_e(u,v)$ | total demand on edge $e$ |
| $b_e(\mathcal{L},f)$ | number of buses on edge $e$ |
| $C(\mathcal{L},f)$ | total operational cost of line network $\mathcal{L}$ |
| $w_i$ | probability of line-changing methods, $i=1,2,3,4$ |
| $\mathcal{L}_{ex}$ | set of express lines |
| $\tilde{\tau}$ | saved time by using direct connection instead of shortest path |
| $d(P_{u,v})$ | number of passenger per unit time on the direct connection $(u,v)$ |
| $\tau$ | total saved time by all passenger per unit time |
| $F_k(v)$ | walk with minimum weight of length $k$ from a source vertex to $v$ |
| $\lambda^*$ | minimum cycle mean |
| $\mathcal{N}_{10}$ | road network with 10 stations |
| $\mathcal{N}_{30}$ | road network with 30 stations |

# Bibliography

# Bibliography

[1] Kepaptsoglou, Konstantinos and Karlaftis, Matthew G. *Transit Route Network Design Problem: Review.* Journal of Transportation Engineering 135(8):491-505, August 2009

[2] KARP, Richard M. *A characterization of the minimum cycle mean in a digraph.* Discrete Mathematics, 23:309-311, North-Holland Publishing Company, 1978.

[3] Schöbel, Anita. Line planning in public transportation models and methods. OR Spectrum 34(3):491-510, 2012

[4] Jungnickel, Dieter. *Graphs, Networks and Algorithms, $4^{th}$ edition.* Springer Heidelberg New York Dordrecht London, 2013

[5] Korte, Bernhard and Vygen, Jens. *Combinatorial optimization, theory and algorithms, $5^{th}$ edition.* [On Algorithms and Combinatorics (21)]. Springer Heidelberg Dordrecht London New York, 2012