# Aggregate waiting time reduction on public transportation networks

**An application of multilayer networks**

Frederique A.E. Akse

Linacre College

University of Oxford

A thesis submitted in partial fulfilment for the degree of

*MSc Mathematical Modelling and Scientific Computing*

2013 - 2014

TO MY GRANDFATHER – whom I was never fortunate enough to meet anywhere else than the problem solving sessions in my dreams and in the sudden moments of clarity in times of need (read exams).

## Acknowledgements

I first want to thank my supervisors Mason Porter and Mikko Kivelä. During the writing of this dissertation they have given me the freedom to explore and discover the field of networks while providing me with enough guidance to navigate myself through an ever expanding network of interesting papers and data sets. You have let me experience what it is like to do (real) research.

I want to express my gratitude towards Riccardo Gallotti and Raghav Pant for their help during this project and to the Infrastructure Transitions Research Consortium (ITRC) for supplying data on the British rail network.

I also want to thank my parents with whom I first came to Oxford and who will luckily both be there to drive me back for their endless support. A big thank you also to my brother who spend countless hours of his vacation in the MMSC office just so we could have dinner together in the evening. I hope Atomic Burger was worth it.

I want to express my gratitude to professor Jan Bouwe van den Berg from the Vrije Universiteit Amsterdam for his guidance and help during my undergraduate studies.

Then there are a few people from the MMSC I would like to thank; Professor Alain Goriely for being my mentor during the first term, for introducing me to the wondrous world of solid mechanics, but most of all for his support during a difficult time; Katrhyn Gillow who has mastered the organization of this course; and my fellow students, some of whom have persuaded me to divert from bread and cheese and try a proper sandwich, that have made the (slightly depressing) office feel like home and who have become my friends.

# Abstract

We will study the possibility of reducing the aggregate waiting time for public transportation system using a ‘multilayer’ network representation of these systems. A multilayer network can be interpreted as a set of graphs that are connected by edges. To estimate the number of passengers who travel between any pair of stations in the network, we use a model for passenger mobility called the doubly constraint gravity model. If we assume that all passengers travel along the path that minimizes travel time, then we can use the graph-theoretical concept of shortest paths to find these paths. We show that the waiting times along a path can be obtained from the shortest paths on a multilayer network representation, as we distinguish between travel, transfer, and stopping time using different edges types. However, an algorithm for finding shortest paths on multilayer networks did not yet exist. We we show that Dijkstra’s algorithm which find the shortest paths on graphs can also be used to find the shortest paths on a multilayer network. Furthermore, we show that an adaptation of Dijkstra’s algorithm, when applied to multilayer networks, can reduce the number of iterations required to find all shortest paths on the network. By combining the shortest path and passenger volume information, we can estimate the aggregate waiting time on a network. We show that the aggregate waiting time of a multilayer network representation of a small network can be improved by perturbing the existing schedule. Naturally, the next step would be to optimize the schedule of a real transportation network. This will be pursued in future work.

# Contents

# Chapter 1

# Introduction

The roots of graph theory are linked to the proof of the Königburg bridge problem, which was constructed by mathematician Leonard Euler in 1735 [3]. The problem asks if a path exists that crosses each of the seven bridges that connect the four parts of the city of Königburg exactly once. By representing each part of the city as a node and each bridge as an edge between two nodes, Euler showed that no such path exists. The graph that was constructed is an example of a transportation network. In the twentieth century, researchers started using graph representations of transportation systems to study the properties of these systems [29]. In these graphs, geographical locations (e.g. cities, stations, airports, etc.) are represented as nodes and the edges in the network represent the routes between them (e.g. roads, tracks, flightpaths, etc.) [29].

However, a graph is a simplified representation of a transportation network. For instance, graphs are unable to represent the time-dependent nature of networks; if a route between two locations exists at some given time, then only will the respective nodes be connected by an edge. In transportation networks, this time-dependency is a crucial feature. This is because connections only exist at certain points in time; for example, a train runs from Oxford to London Paddington only four times in one hour and so the edge representing the connection between the stations will only exist four times every hour.

The time-dependency can be incorporated if a transportation system is represented as a multilayer network. In this case, a multilayer network can be thought of as a series of graphs, where each graph (i.e. layer) represents the network at a specific moment in time. The edges used to connect nodes to their counterparts in other time-layers can now be interpreted as waiting time. Edges between two different stations in different time-layers represent travel time.

Moreover, a multilayer network allows to segregate a network on the basis of node characteristics. For example, in a rail network connections between stations are part of different routes [39]. By segregating the nodes in the graph on the basis of the routes that are used to connect them to other nodes, we can include the notion of travel, transfer and stopping time in the network. For example, an edge between a node and its counterpart in another route layer can be interpreted as the required transfer time at the station represented by the node.

In contrast to those in a graph, edges in a multilayer network can thus have different interpretations. This allows for a more realistic representation of a network. For example, we can more accurately define a path on a network using a multilayer network representation. For each journey on the network, we can, in addition to the stations that are traversed, specify the times at which they are traversed and the routes that are used to go between stations.

A multilayer network that has both routes and time as layers can be used to represent a transportation schedule. By studying the network, we can, for example, deduce information about possible travel patterns of the users of the transportation network. This is especially useful when accurate data on passenger behaviour is not available for the network that is studied. An understanding of the travel patterns of passengers is essential for the construction and adjustment of schedules on a network, as it allows operators to respond to fluctuations in demand by adjusting their existing schedule (i.e. changing the number and/or distribution of vehicles on a network) or by introducing price fluctuations [26].

Using a network representation of a transportation system, we can estimate passenger flows on a system by finding the shortest paths. Depending on the network, the definition of a shortest path may vary. For example, a shortest path between an origin and a destination node can be the path with the smallest travel time, the path that requires the least number of interchanges or the path that has the smallest travel distance. We can find shortest paths on a graph using Dijkstra's algorithm [12]. However, an algorithm for finding shortest paths on multilayer networks did not yet exist. We will show that Dijkstra's algorithm can also be used to find the shortest paths on multilayer networks. Furthermore, we propose an adjusted version of Dijkstra's algorithm which can reduce the number of iterations required to find all shortest paths on a multilayer network. Together with information on the number of passengers that travel between each pair of nodes in a network, we can estimate passenger volumes on the network.

The construction of actual data on passenger mobility is difficult, as it requires the almost constant monitoring of the position of individuals [16]. Additionally, the collection of spatio-temporal information of users in a network can pose a serious threat to their privacy [6]. When data on passenger mobility patterns is available, it can be used to create a better understanding of passenger preferences on a network. For example, using smart card transaction data obtained for the rail network in Japan, it was found that in addition to travel time passengers choose to minimize the total waiting time along their journey, while also trying to minimize the number of transfers when faced with a choice of train [25].

This behaviour can be explained by considering the value of travel time [45]. The value of travel time can be determined based on the mode of travel that is used (e.g. walk, car, train, etc.). One way in which the value of travel time can be interpreted, is that a passenger would be willing to spend more to avoid a travel mode for which the value of travel is high (e.g. buy a more expensive train ticket that reduces the waiting time along the journey). Consequently, the time spent in travel modes for which the productivity (e.g. ability to do work) of a passenger is high is valued less than time spent in travel modes for which the productivity of a passenger is low. For example, the value of time that is spent walking or waiting is likely to be valued more than in-vehicle time [45]. By measuring the value of transfer and waiting time with respect to the value of in-vehicle time, it was concluded that passengers value transfer and waiting time at 2 and 2.5 times the value of in-vehicle time, respectively [45]. One way to increase the usage of public transport is for policy makers to focus on the reduction of the aggregate waiting time on the network.

It is found that the time a passenger spends waiting when using public transportation is dependent on the level of synchronization of the schedule and on the punctuality with which the schedule is executed [44, 19, 9]. Research on the punctuality of the schedule focusses on methods to recomputing schedules in case of delays [22, 43], or on the construction of delay resistant schedules [27].

Attention has been increasingly paid to methods that improve the synchronization between the schedules from different transportation modes [11, 8, 46]. Nair et al. suggest that the aggregate waiting time on a multi-modal transportation network can be reduced by perturbing the existing schedules [10]. They show that the aggregate waiting time on the Washington D.C. public transportation network can be reduced by 26.38%, affecting approximately 15% of the users on the network [10]. The advantage of this method is that its implementation cost is relatively low, because only minor changes to the schedule are introduced.

We apply this method to a single-modal transportation networks on which different operating companies are responsible for passenger transportation. To this end, this thesis will use the formalism of multilayer networks to develop a framework that can potentially reduce the aggregate waiting time on a transportation network through a small perturbation of the existing schedule.

## 1.1 Thesis outline

Chapter 2 provides a general description of the British rail network and introduces basic terminology and data sets that we use. In Chapter 3, we construct a graph representation of the British rail network and discuss its characteristics using standard network diagnostics. In Chapter 4, we introduce the concept of multilayer networks and introduce the procedure for the construction of multilayer network for general public transportation networks. The chapter concludes with a discussion of the network properties of the constructed network. in Chapter 5, we introduce methods to obtain information on passenger flow in a network. In Chapter 6, we introduce the schedule optimization method and illustrate it using a small scale example. We provide concluding remarks in Chapter 7 as well as an overview of possible directions for future work.

# Chapter 2

# Preliminaries

This chapter provides a background on Britain's rail infrastructure, introduces the terminology used for describing rail infrastructures, and gives a description of the data sets used in this dissertation.

## 2.1 Britain's rail infrastructure

Britain's rail infrastructure, regulated by Network Rail since its privatization in October 2002, is responsible for 1.5 billion passenger journeys each year and is considered the fastest growing rail network in Europe [36]. The 2,519 stations contained in the network are connected using approximately 20,000 miles of track. All stations are owned by Network Rail and leased to train operating companies (TOCs) with the exception of 19 critical stations, which remain under the direct control of Network Rail. These critical stations include Paddington, Glasgow Central, London Bridge, Birmingham New Street, and St Pancras International [39]. The licensing of the other stations is controlled by the Office of Rail Regulation (ORR) in order to enforce the existing domestic competition laws [31]. The ORR is also responsible for the regulation of track access by the train operating companies [31]. The assignment of track access among the different operators is motivated by the commercial needs of rail operator companies, the flexibility needs of Network Rail with respect to the optimization of the network capacity, and the existing maintenance requirements to ensure passenger safety on the network [31]. Although the ORR is the independent regulator of Britain's rail network, it works in close collaboration with both the Scottish and British government on the development, funding, and execution of major rail projects [32].

A total of 29 TOCs are responsible for passenger transportation. Freight transportation is controlled by 8 freight operating companies (FOCs) [39]. TOCs operating on Britain's rail network include Arriva Trains Wales, First Great Western, Southern, and ScotRail. The Association of Train Operating Companies (ATOC) is the umbrella organization for TOCs [34]. ATOC's goal is to optimize passenger satisfaction across operators. This has, for example, lead to the introduction of the Rail Settlement Plan and the National Rail Enquiries (NRE). The former allows passengers to buy tickets for journeys that might require the use of two or more different TOCs on the network and the latter provides passengers with up-to-date information about the train times, fares, reservations and trains disruptions across the United Kingdom [34].

## 2.2 Terminology

In this section, we introduce the terminology that we will throughout this dissertation. A *route* is defined as an ordered set of stations that are serviced by a single train (e.g. {Birmingham New Street, Wolverhampton, Penkridge}) and is operated by one TOC. A *trip* is a realization of a route at a specific time (e.g. the train that departs from Birmingham New Street at 9.42 am, calls at Wolverhampton at 9.59 am and terminates at Penkridge at 10.09 am). Each trip can be mapped to at most one route, whereas a route can be mapped to at least one trip. A *journey* is the sequence of trips required to travel from the selected departure station to the target station; one can usually find a journey online using a journey planner. The *time headway* is defined as the minimum time between two subsequent departures at a station and is implemented for the safety of passengers and personnel in case of an incident. The *minimum transfer time* at a station provides an indication of the time that is needed to interchange at that station.

## 2.3 Data

We obtained the data used for this thesis from [21]. It is structured following the General Transit Feed Specification (GTFS) format. The GTFS format was originally developed by Google for Google Maps, but it has since then been adopted by various public transit agencies [13, 20]. See Appendix A for a description of the contents of GTFS data files.

Raw data for the British rail network is published weekly by ATOC, Network Rail, the Department for Transport [40, 38, 30]. The GTFS data files for the British rail network were constructed by Nathan Johnson from data published by ATOC, Network Rail, the Department for Transport for the period 17–24 August 2013. The geographic data included in the GTFS files was obtained from OpenStreetMap. Unfortunately, the procedure that was used for the construction of the data was unavailable. For future work, it might be worthwhile to format the raw data provided by [40, 38, 30]. This could reduce the time needed for the preprocessing of the data.

We obtained information about the the passenger flow through each station in Great Britain's rail network from a data set constructed by the Infrastructure Transitions Research Consortium (ITRC) for the study of inter-dependent infrastructure failures resulting from extreme national hazards [37]. As the passenger flow information is for the period 2011–2012, there is a discrepancy with the schedule data described above. Using the measured growth in passenger volume in the period 2011-2013, the 2011-2012 passenger flow data can be used to approximate the passenger flow in 2013.

Additionally, we obtained the travel times between stations situated in London, which are connected through the London Underground from [14, 15]. We hypothesize that this information is essential when studying Britain's rail network, as transfers between rail stations via the London Underground are common. We show the complete list of connections in Appendix B.

# Chapter 3

# Graphs

This chapter provides an introduction to the main graph-theoretical concepts that we will use in this thesis.

## 3.1 Graph formalism

A graph is defined as a tuple $G = (V, E)$, where $V$ is a set of nodes and $E$ is a set of edges that connect pairs of nodes. *Nodes* are an abstract representation of the entities contained in the system that is being modelled. When modelling rail networks, nodes represent the stations and an edge between a pair of nodes specifies the existence of a track (edge) between two stations (nodes).

If the edges in a graph have a specified direction, the network is *directed.* Conversely, if edges do not have specified direction, the graph is *undirected.* A graph can be represented by an *adjacency matrix.* For an unweighted, directed graph with $N$ nodes, the adjacency matrix $A$ is an $N \times N$ matrix, whose elements are given by [29]:

$$A_{ij} = \begin{cases} 1, & \text{if there is an edge from node } j \text{ to node } i \\ 0, & \text{otherwise} \end{cases} .$$

For undirected networks, the adjacency matrix is symmetric (i.e. $A_{ij} = A_{ji}$) [29].

Edge weights can be used to specify the strength of a connection between nodes. In a *weighted* graph, the edge weights might represent geographical distance, travel time, or passenger volume. A weighted, directed graph can be represented by a *weighted adjacency matrix* $w_{ij}$, whose elements are:

$$W_{ij} = \begin{cases} w(i,j), & \text{if there is an edge from node } j \text{ to node } i \\ 0, & \text{otherwise} \end{cases},$$

where $w(i,j)$ is the weight of the edge from node $j$ and $i$ [29]. For a graph to be undirected, nodes have to be connected by two edges with equal edge weights, but in opposite direction [29].

A *path* on a graph is defined as the sequence of nodes and edges that are traversed between an origin and a destination node [29]. The associated *path length* is the sum of the edge weights of the edges along the path. Depending on the choice of edge weights, the path length can have different interpretations. For example, if the edges are weighted by the travel time between nodes, then the path length gives the total time required to travel from an origin to a destination node. For an unweighted graph, the path length gives the number of edges that are traversed between the origin and destination node [29]. For nodes that are not connected the path length is infinite. The *shortest path* or (geodesic path) between two nodes is the path with the smallest path length. We will introduce Dijkstra's algorithm which can be used to find the shortest paths on a graph in Chapter 5.

A graph is *connected* if there exists a path between all pairs of nodes in the graph [3]. For directed graphs, we can distinguish between strongly connected and weakly connected graphs. A directed graph is *strongly connected* if there exists a directed path between all pairs of nodes in the graph [29]. If a path between every pair of nodes only exists when all edges are assumed to be undirected, the graph is *weakly connected.* A graph is *disconnected* if there exists at least one pair of nodes that are not connected by a path. In an disconnected graph, one can identify connected components. A *connected component* of a graph is the largest possible subset of nodes such that each pair of nodes in this set can be connected by a path [29]. For a transportation network, it is essential that the network is strongly connected, as preferably passengers should be able to travel between any two locations in the network without having to use an alternate form of transportation. For a graphical representation of different network types, see Appendix C.

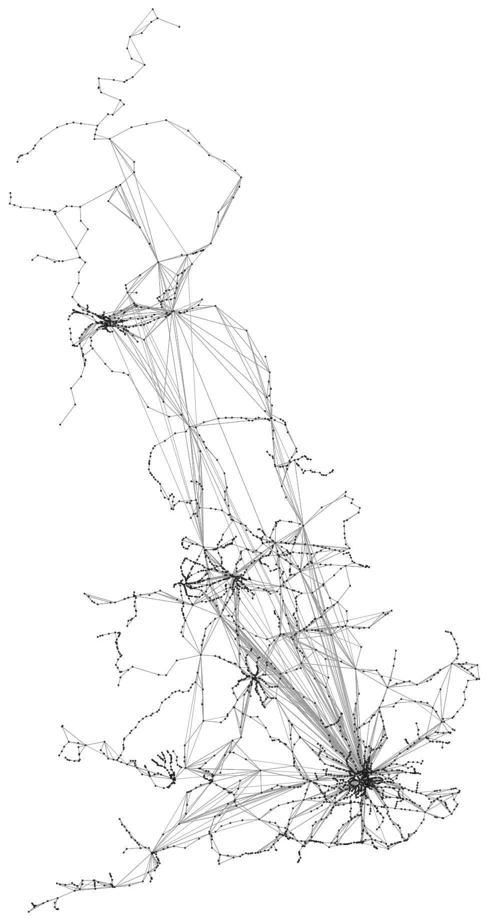


Figure 3.1: Graph of the British rail network. For clarity, the direction of the edges is not included.

## 3.2 Graph representation of the British rail system

We construct two graph representations of the British rail system from the data described in Section 2.3. Both graphs are directed and have edges that are weighted by the travel time. The first graph does not contain connections between London stations provided by the London Underground. The second graph does include these connections. To construct both graphs, we perform the following steps for each station in each trip listed in the British rail data:

1. We determine the departure time ($t_{dd}$) at the departure station ($s_d$) in min-

utes.

2. We determine the arrival time ($t_{aa}$) at the arrival station ($s_a$) in minutes.
3. We compute the travel time ($t_{\text{travel}}$) between the departure and arrival station as $t_{\text{travel}} = t_{aa} - t_{dd}$.
4. We construct an edge from $s_d$ to $s_a$ with edge weight $t_{\text{travel}}$.

For the graph that includes the London Underground, we extend the graph with the connections listed in Appendix B using the procedure described above. We show a graphical representation of the British rail network that includes the London Underground in Figure 3.1.

## 3.3 Centrality measures

Now that we have constructed two suitable network representation for the British rail network, we can study their properties using different graph measures. For example, centrality measures can be used to quantify the importance a node or edge in a network [29]. The concept of importance is prescribed by the centrality measure that is used. For example, if one uses degree centrality, a node is considered important if it has a large number of neighbours in comparison to other nodes in the network. In this section, we will discuss three centrality measures (i.e. degree centrality, node betweenness centrality and edge betweenness centrality). For a thorough review of other centrality measures for graphs, see [29].

### 3.3.1 Degree centrality

Degree (aka degree centrality) is a simple measure that can be used to determine central nodes in the network. It assumes that central nodes are high degree nodes. The *incoming degree* $k_i^{\text{in}}$ of a node $i$ is given by the number of incoming edges of the node. Similarly, the *outgoing degree* $k_i^{\text{out}}$ is defined as the number of outgoing edges of node $i$. One can compute both the incoming and outgoing degree from the corresponding adjacency matrix:

$$k_i^{\text{in}} = \sum_{j=1}^{N} A_{ij}, \quad k_i^{\text{out}} = \sum_{j=1}^{N} A_{ji}. \tag{3.1}$$

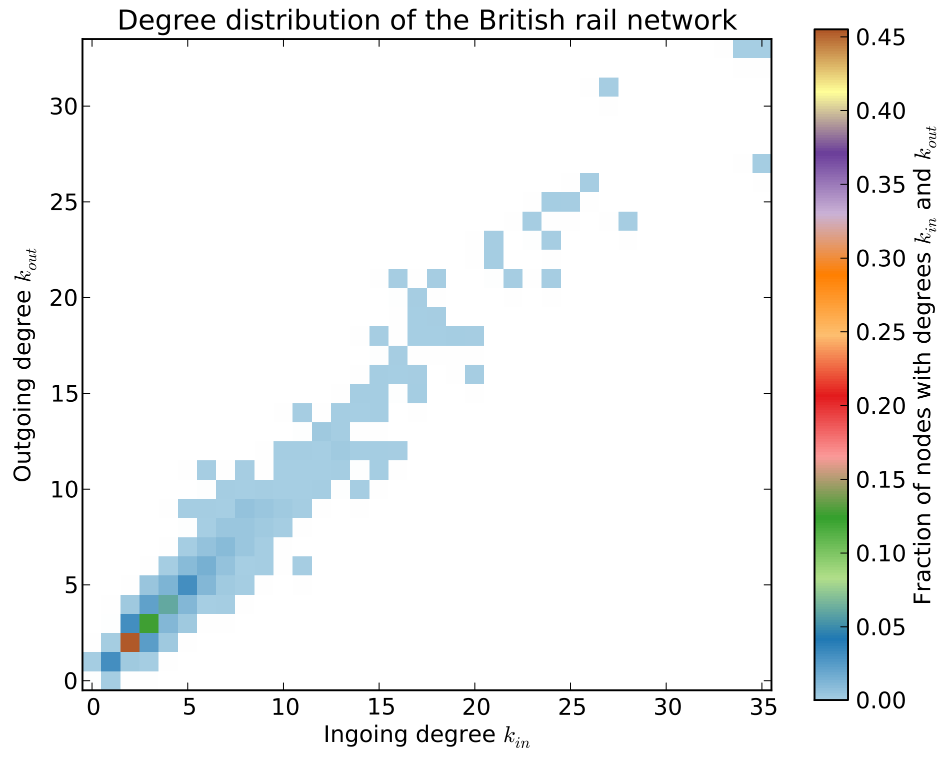


Figure 3.2: Degree distribution of the directed, weighted network representation of the British rail system (including the London Underground) as a density plot.

Note that for an undirected network the incoming and outgoing degree are equal. The *degree centrality* for a node $i$ is determined from the total degree of a node:

$$k_i = k_i^{\text{in}} + k_i^{\text{out}}. \tag{3.2}$$

In transportation networks, high-degree nodes are likely to be essential for connecting different parts of the network. For example, for a directed, weighted representation of the British rail system, we find the station Birmingham New Street in the top 10 of high degree nodes presented in Table 3.1. This is in agreement with our expectation, as Birmingham New Street station has a central position geographically and is an essential transfer station in going between the North, East, South, and West of the United Kingdom.

In Figure 3.2, we show the density plot for the ingoing and outgoing degrees of

| Rank | **Degree Centrality (excluding LU)** | **Degree Centrality (including LU)** |
|---|---|---|
| 1 | Birmingham New Street (0.026) | London Bridge (0.027) |
| 2 | Manchestor Piccadilly (0.024) | Birmingham New Street (0.026) |
| 3 | London Bridge (0.024) | Manchestor Piccadilly (0.024) |
| 4 | Clapham Junction (0.022) | Clapham Junction (0.022) |
| 5 | Reading (0.020) | Reading (0.020) |
| 6 | Leeds (0.020) | Leeds (0.020) |
| 7 | Crewe (0.019) | London Euston (0.020) |
| 8 | Glasgow Central (0.019) | Crewe (0.019) |
| 9 | Preston (0.018) | Glasgow Central (0.019) |
| 10 | Nottingham (0.018) | London Liverpool Street (0.019) |

Table 3.1: Stations with the highest degree centrality for a directed, weighted network representation of the British rail system that excludes the London Underground (LU) and a graph representation that includes the London Underground. The numbers in brackets give the normalized degree centrality of the station. The number of nodes in the network $N$ was used as the normalization factor.

the nodes in the British rail system. We find that for most nodes the number of ingoing and outgoing edges is approximately equal. Furthermore, we see that the approximately half of nodes in the network have 3 ingoing and 3 outgoing edges. Only a small fraction of nodes in the network have a degree $k$ that exceeds 40. These nodes are known as "hubs" and include the nodes listed in Table 3.1.

## 3.3.2 Geodesic node betweenness centrality

Geodesic betweenness centrality provides a measure for the relative importance of a node in the network on the basis of the fraction of shortest paths that go through this node. Geodesic betweenness centrality for directed networks, which can be either weighted or unweighted, is given by [29]

$$x_i = \frac{1}{N^2} \sum_{s=1}^{N} \sum_{t=1}^{N} \frac{n_{st}^i}{n_{st}}, \tag{3.3}$$

where $n_{st}^i$ is the number of shortest paths from node $s$ to node $t$ that traverse node $i$, $n_{st}$ is the total number of shortest paths from $s$ to $t$, and $N$ is the number of nodes in the network [29]. In the case that both $n_{st}^i$ and $n_{st}$ are 0, we set $n_{st}^i/n_{st} = 0$.

| **Rank** | **Number of Interchanges** | **Betweenness centrality (excluding LU)** | **Betweenness centrality (including LU)** |
|---|---|---|---|
| 1 | Clapham Junction | Birmingham New Street (0.237) | London Euston (0.403) |
| 2 | London Waterloo | Crewe (0.177) | London Victoria (0.311) |
| 3 | London Victoria | Reading (0.171) | Preston (0.169) |
| 4 | London Bridge | London Euston (0.160) | Crewe (0.144) |
| 5 | East Croydon | Preston (0.150) | London Kings Cross (0.120) |
| 6 | Birmingham New Street | Leicester (0.149) | London Paddington (0.0116) |
| 7 | London Euston | Peterborough (0.132) | London Liverpool Street (0.115) |
| 8 | Manchester Piccadilly | Leamington Spa (0.131) | Birmingham New Street (0.113) |
| 9 | St. Pancras International | Derby (0.117) | Reading (0.103) |
| 10 | London Kings Cross | Cheltenham Spa (0.112) | Cardiff Central (0.091) |

Table 3.2: Stations with the highest geodesic betweenness centrality for a graph representation of the British rail system that excludes the London Underground (LU) and the graph representation that includes the London Underground. The stations that have the highest number of interchanges [33].

A high betweenness centrality can be an indication that a node is essential in connecting different parts of a network. See Table 3.2 for the 10 stations with the highest betweenness centrality for the two monoplex network representations of the British rail system.

### 3.3.3 Geodesic edge betweenness centrality

Edge betweenness centrality can be used as a measure of the importance of an edge in a network, as it determines the fraction of shortest paths that traverse this edge. The geodesic edge betweenness centrality $e_i$ for an edge $i$ is given by

$$e_i = \frac{1}{N^2} \sum_{s=1}^{N} \sum_{t=1}^{N} \frac{e^i_{st}}{e_{st}}, \tag{3.4}$$

where $e^i_{st}$ is the number of shortest paths from $s$ to $t$ that traverse edge $i$ and $e_{st}$ is the total number of shortest paths between $s$ and $t$ [29]. For a transportation network, geodesic edge betweenness centrality can provide information on the likelihood of congestion between nodes or along a path in the network. Note, however, that edge betweenness centrality does not necessarily give an indication of the importance with respect to passenger volume along an edge. To measure the importance of an edge in a network, its betweenness centrality should be combined with information of the number of passengers that travel between all pairs of stations in the network.

| Rank | Origin station | Destination station | Edge betweenness centrality |
|---|---|---|---|
| 1 | London Victoria | London Euston | 0.115 |
| 2 | London Euston | London Victoria | 0.105 |
| 3 | London Euston | Preston | 0.055 |
| 4 | London Paddington | London Victoria | 0.056 |
| 5 | London Victoria | London Liverpool Street | 0.054 |
| 6 | Crewe | Shrewsbury | 0.054 |
| 7 | London Liverpool Street | London Victoria | 0.045 |
| 8 | Newport (Wales) | Cardiff Central | 0.045 |
| 9 | London Kings Cross | London Euston | 0.044 |
| 10 | Edinburgh | London Euston | 0.038 |

Table 3.3: Edges with the highest geodesic edge betweenness centrality for the graph representation of the British rail network including the London Underground.

## 3.4 The British rail network

Table 3.2 lists the 10 stations that have the largest number of interchanges, and the ten stations that have the largest shortest path betweenness centralities. We find a significant difference in the betweenness centralities in the two networks. For example, only one London station is among the 10 stations with the largest betweenness centrality when excluding the London Underground connections, but five London stations occur in the top ten when the London Underground connections are included. This implies that shortest paths going through London often include tube connections. This conclusion is supported by data that ranks the stations with the largest number of transfers as can also be found in Table 3.2. London stations with a high betweenness centrality are also the stations with the most passengers transfers according to data obtained from [33]. If a majority of passengers travel along the shortest paths in a network, then this is an expected result, as betweenness centrality gives an indication of the number of shortest (optimal) paths going that traverse a station. This implies not only that the London Underground is of importance when travelling through London, but also that London itself has a critical position in the network. For example, we find that 40% of the shortest paths in the network traverse the station London Euston.

The importance of the London Underground is also emphasized by the the geodesic edge betweenness centralities (see Table 3.3). The majority of connections with a large edge betweenness centralities are either tube connections between two London stations or connections between a London station and another important interchange station outside of London. An example of the former is the connection between London Victoria and London Euston. The connection between London Euston and Preston is an example of the latter. Only two edges within the 10 edges with the largest betweenness centrality do not have a London station as either its starting or ending point. One of these is the connection from Newport (Wales) to Cardiff Central, which is critical for connecting Wales to the rest of the United Kingdom.

# Chapter 4

# Multilayer networks

In graphs, all edges are assumed to be of the same type. However, one can often identify different types of relations within a system. Consider, for example, a network that describes relations between individuals on social media. When represented as a graph, each edge (possibly weighted and/or directed) indicates that a connection exists on social media without specifying the connection type. As, for example, connections on LinkedIn have a very different nature than connections on Facebook, important information about the network might be lost because of the graph representation. A multilayer network representation allows the identification of different relation types through the construction of different layers [24]. In the example of social media, each layer could represent a social medium (e.g. Facebook, Twitter, LinkedIn etc.) although other segregations are also possible. Edges within layers and edges between layers also have different interpretations depending on the segregation structure. This will be discussed further in Section 4.1.

A multilayer network might also restrict the set of choices of an agent that traverses it. Consider a rail network. In a graph representation, the existence of an edge implies that there exists a connection between two nodes (i.e. a train service is run between two stations). However, it does not provide information about the time(s) that this service is available, which is an important feature in a rail network (e.g. in identifying the shortest paths). See the discussion in Chapter 5. One can include temporal using a multilayer network in which each time instance corresponds to a layer. The possible edges that can be chosen at a particular node no longer depend purely on the existence of the edge, but also on the current availability of the edge. In relation to multilayer networks, the traditional graph is referred to as a monoplex network.

Recall from Chapter 3 that a monoplex network (i.e. a traditional graph) is a tuple $G = (V, E)$, where $V$ is a set of nodes and $E$ is the set of edges that connect pairs of nodes. Multilayer networks have, in addition to nodes and edges, a set of layers $L$. One can visualize a 'one-dimensional' multilayer network as a stack of monoplex networks that are connected to each other using a particular type of edge. Higher dimensional structures are also possible. Following the terminology introduced in [24], one dimension of a multilayer network will subsequently be called an *aspect*. For example, a one-aspect multilayer representation of a rail network could segregate the network with respect to the TOCs. A two-aspect multilayer network could include both different TOCs and time.

Each aspect has a set of *elementary layers*. For example, the elementary layers of an aspect that represents the TOCs, would be the individual TOCs (e.g. Arriva Trains Wales, First Great Western, Southern etc.). For an aspect that describes time, an elementary layer could contain information on the network for a specific period in time, i.e. Monday morning, Wednesday 13.53 or $t = 743$. A specific combination of elementary layers from different aspects are called *layers*. The combination of a node and a layer is defined as a *node-layer tuple*. Continuing the example of a multilayer network with TOCs and time as the two aspects, a layer in the network could be the tuple (Arriva Trains Wales, Wednesday 13.53). A possible node-layer tuple is (Birmingham New Street, Arriva Trains Wales, Wednesday 13.53). In the most general definition of a multilayer network, each node can exist in any subset of layers and each edge is allowed to pairwise connect all possible node-layer tuples.

## 4.1 Multilayer network formalism

Let $L_a$ be the set of elementary layers associated with an aspect $a$ [24]. We define $\mathbf{L} = \{L_a\}_{a=1}^{d}$ as the sequence of sets of elementary layers, where $d$ is the number of aspects in the network [24]. The set of all possible layers in the network can be defined as the Cartesian product of all sets of elementary layers in a network

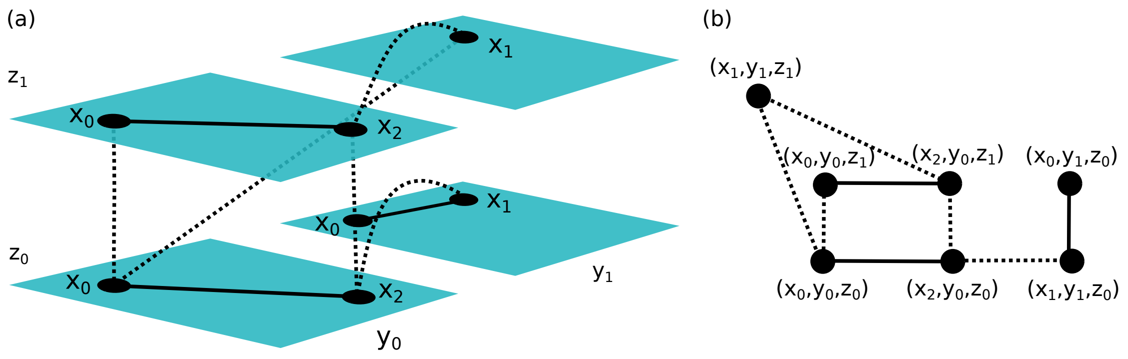


Figure 4.1: (a) A graphical representation of a general type of multilayer network. The network has three nodes (i.e. V=$\{x_0,\ x_1,\ x_2\}$), and two aspects. The set of elementary layers corresponding to each aspect are given $L_1 = \{y_0, y_1\}$ and $L_2 = \{z_0, z_1\}$. There are four possible layers: $(y_0, z_0)$, $(y_0, z_1)$, $(y_1, z_0)$ and $(y_1, z_1)$. The set of node-layer tuples is $V_M = \{(x_0, y_0, z_0), (x_0, y_0, z_1), (x_0, y_1, z_0), (x_1, y_1, z_0), (x_2, y_0, z_0), (x_2, y_0, z_1)\} \subset V \times L_1 \times L_2$. Intra-layer edges are drawn as solid lines. Inter-layer edges are drawn as dashed lines. (b) The underlying graph of the multilayer network given in (a). Again, intra-layer edges are drawn as solid lines and inter-layer edges as dashed lines.

$L_1 \times L_2 \times \cdots \times L_d$ [24]. All possible node-layer tuples are given by the Cartesian product $V \times L_1 \times \cdots \times L_d$. We also want to allow the construction of multilayer networks in which a node only exists in a subset of layers. The set of node-layer tuples that exist in the network is denoted by $V_M \subseteq V \times L_1 \times \cdots \times L_d$ [24]. We will use the term *node-layer tuple* to describe node-layer tuples in the set $V_M$. The set of edges that connect node-layer tuples in a network is denoted by $E_M \subset V_M \times V_M$ [24]. A multilayer network $M$ can thus be defined as the quadruple $M = (V_M, E_M, V, \mathbf{L})$ [24]. See Figure 4.1 for a graphical representation of a multilayer network.

It is important to note that a multilayer network can be represented by a monoplex network in which each node is labelled by a node-layer tuple. We will call the monoplex representation of a multilayer network the underlying graph $G_M = (V_M, E_M)$ (see Figure 4.1).

The monoplex representation of a multilayer network allows natural generalizations of several concepts defined for monoplex networks [24]. A *weighted* multilayer network can, for example, be constructed by assigning weights to the edges in the underlying graph. Similarly, a multilayer network is considered *directed* if the underlying graph is directed.

An important difference with monoplex networks is that one can distinguish between different edge types in the network [24]. We define *intra-layer* edges as edges that exist within a layer, while *inter-layer* edges connect nodes between layers. The full set of edges $E_M$ in the network can thus be described as the union of the intra- and inter-layer edges.

However, the generalization of the concept of a path to multilayer network is not straight forward. For example, one has to establish whether changes between layers are included in a path [24]. If changes between layers are included, then a path is defined as the sequence of node-layer tuples and edges that are traversed when going from an origin node-layer tuple to a destination node-layer tuple [24].

In order to compute the associated path length, one should consider whether there are cost associated with changing layers in the network [24]. For example, consider a multilayer network representation of a rail network in which each layer represents a route in the network and edges are weighted by travel time. Changing layers in this network implies that a passenger needs to change trains. As it was found that transfer and waiting time are valued twice more than in-vehicle time, one could weigh the weights of inter-layer edges twice more than the weights of intra-layer edges [45].

One also has to determine whether edges within layers have the same interpretation for all layers in the network [24]. For example, one could construct a network for which all intra-layer edges have the same length. Alternatively, intra-layer edges could be of the same type, but be weighted differently depending on the layer they are in. An example of the latter is a multilayer transportation network that has a layer representing the travel mode walking and another layer representing the travel mode bus. In both layers the intra-layer edges are weighted by travel distance. If one has to travel between two nodes either by foot or by bus, then the nodes in the bus layer can be considered 'closer' than in the walking layer. Intra-layer edges in the bus layer could, thus, be weighted more than the intra-layer edges in the walk layer. For a thorough review of multilayer networks and multilayer network diagnostics, see [24].

## 4.2 Multilayer network construction

We construct two weighted, directed single-aspect multilayer networks using data for the British rail system on Monday 19 August 2013. In the first network, we use time as the aspect. Each layer in this network represents 1 minute. The network represents the British rail system for a single day (and 2 extra hours) from 00.00 to 02.00 the next day (i.e. $t \in [0, 1560]$). For the second network, we use TOCs as the aspect. Both networks are constructed from the data introduced in Section 2.3 (using the Python package PLEXMATH [23]).

We will now explicitly describe the steps that are taken in the construction of directed, weighted 2-aspect multilayer networks that have routes and time as aspects. This type of multilayer network will used in Chapter 6.

1. We extract the arrival ($t_{da}$), departure ($t_{dd}$) and minimum transfer ($t_{dt}$) time and route ($r$) for the departure node ($s_d$) and convert the times to minutes.
2. We extract the arrival time ($t_{aa}$) for the arrival node ($s_a$) are and convert the times to minutes.
3. We construct an edge from $(s_d, r, t_{da})$ to $(s_d, r, t_{dd})$ with weight $t_{aa} - t_{dd}$.
4. We construct an edge from the departure tuple $(s_d, r, t_{dd})$ to the arrival tuple $(s_a, r, t_{aa})$ with weight $t_{aa} - t_{dd}$.
5. We construct a set $\hat{R}$ that contains all routes $\hat{r}$ that stop at the departure node.
6. For all routes $\hat{r}$ in $\hat{R} \backslash \{r\}$, we determine the next departure time $t^{+}_{dd}$ of the departure node $s_d$ in route $\hat{r}$. If $t^{+}_{dd} - t_{da} > t_{dt}$, we place an edge from $(s_d, r, t_{da})$ to $(s_d, \hat{r}, t^{+}_{dd})$ with weight $t^{+}_{dd} - t_{da}$

The constructed network only contains inter-layer edges. Note that the interpretation of the inter-layer edges constructed in the each steps is different. For example, in step (3) we construct edges between nodes and their counterparts that lie in the same elementary route layer, but in different elementary time layers. These edges represent the stop time of a train along a route at a particular

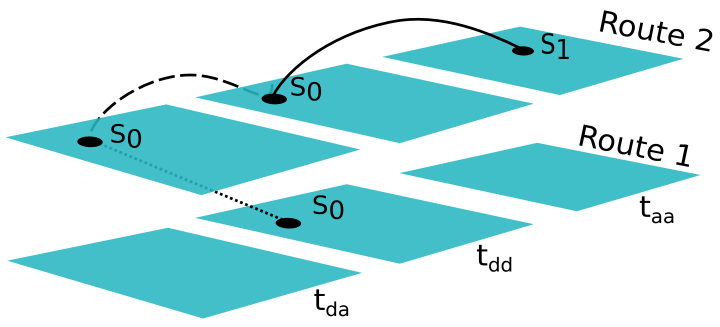


Figure 4.2: A two-aspect multilayer network with aspects routes and time. The construction of the edges for a station S1 is illustrated. The solid line is an edge that represents travel time. The dashed line indicates stop time. The dotted line indicated a transfer to another route.

station. The edges constructed in step (4) connect two nodes that lie in the same elementary route layer, but in different elementary time layers and can be interpreted as the travel time between two stations in the network. Finally, in step (6), we construct edges that connect nodes to their counterpart that lies in different elementary route and elementary time layers. These edges naturally represent the transfer time between routes at a particular station. As we can associate different costs with different types of inter-layer edges, future work could consider the construction of a network in which, for example, the waiting time is valued more than the in-vehicle time [45].

In case of the British rail system, the resulting two-aspect multilayer network would have 6,054 elementary route layers and 1,577 elementary time layers. Each layer contains at most 2,551 active nodes. So there are $24 * 10^9$ possible node-layer tuples. However, the actual multilayer network contains only 742,744 node-layer tuples. The resulting network is, thus, very sparse.

## 4.3 Multilayer network properties

Figure 4.1 suggests that most stations appear in at most a single elementary operator layer. Figure 4.2 shows the number of nodes in each elementary time layer. We find that the number of trains that arrive and depart is significantly lower for Monday between $t = [0, 200]$ and $t = [1500, 1560]$. This confirms the intuition that

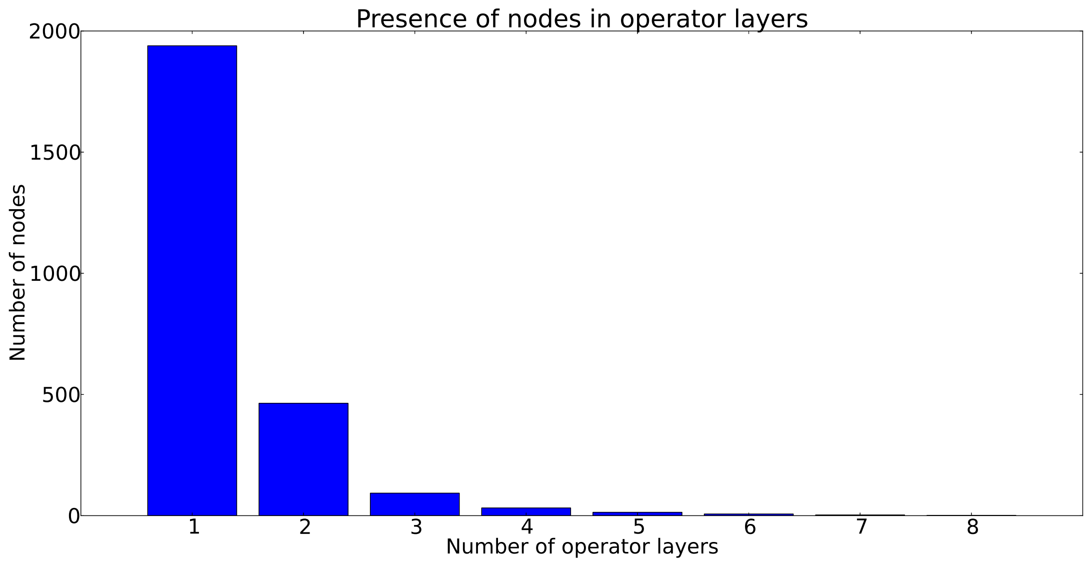


Figure 4.3: Number of nodes plotted with respect to the number of TOC layers they appear in.

few or no trains are running at night. As expected, the highest level of activity on the network occurs during the day (i.e. between 6 am ($t = 360$) and 7 pm ($t = 1140$)).

When we look at the number of nodes for each TOC in a TOC-aspect multilayer network (see Figure 4.3), we find that Northern Rail has the highest number of nodes, followed by ScotRail and Southern. TOCs with the least number of nodes include Island Line, Heathrow Connect and Heathrow Express. We note that these TOCs are very localized within the network, as they are used to provide a connection to a specific location (e.g. Heathrow).

A visualization of the node similarity for all TOC layers in the network can be found in Figure 4.4. We find that First Transpennine Express and Northern Rail share the largest number of stations, followed by the pairs First Capital Connect and Southeastern, and First Great Western and CrossCountry. The majority of operator pairs share fewer than 10 stations. Together with Figure 4.1, this observations leads to the conclusion that, from an TOC perspective, the network is highly segregated as the majority of stations are serviced by a single operator. One can obtain further insight about the importance of operator coordination

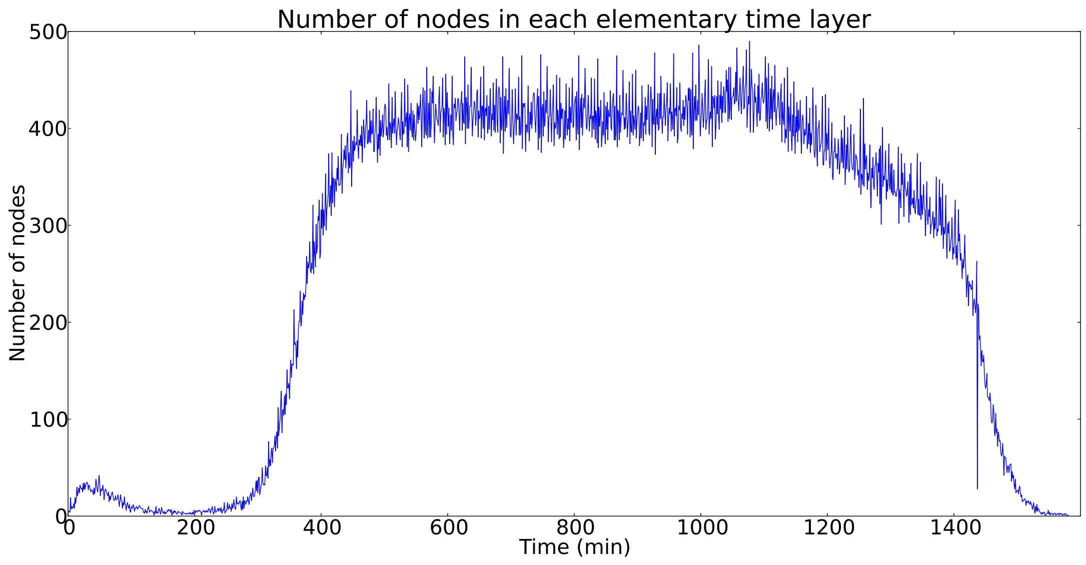


Figure 4.4: The number of nodes in each elementary operator layer obtained from the operator-aspect multilayer network.

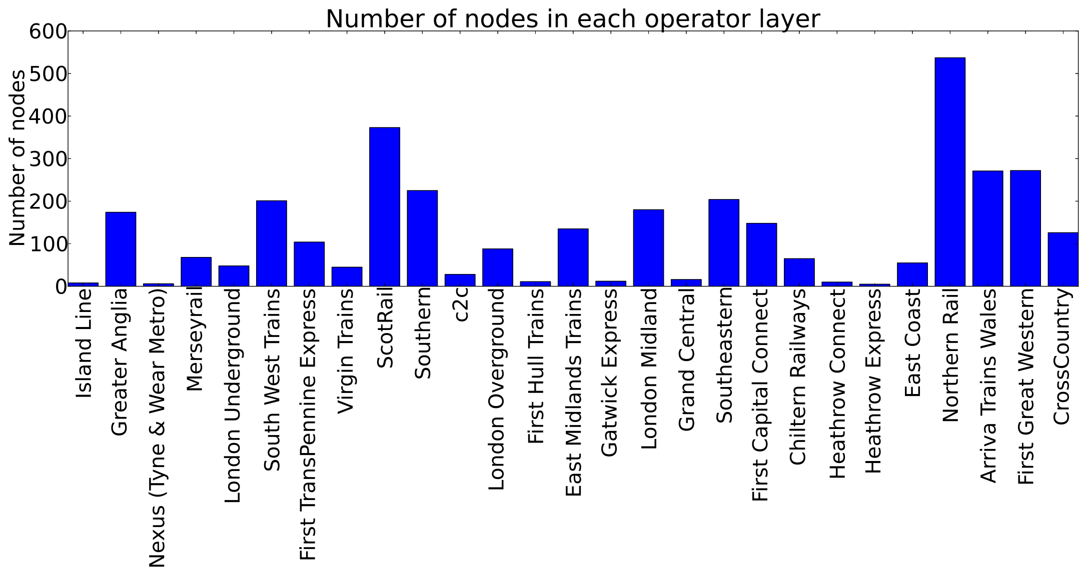


Figure 4.5: The number of nodes in each elementary time layer obtained from time-aspect multilayer network.

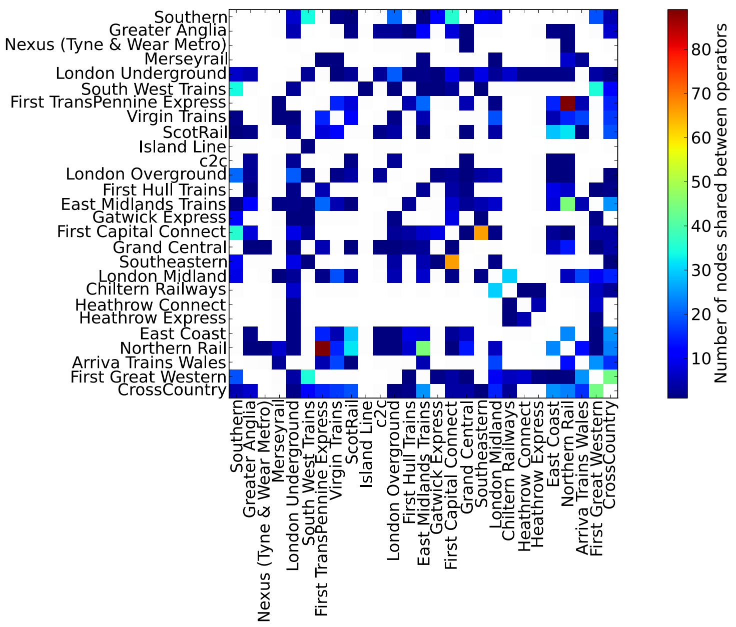


Figure 4.6: Number of stations that are shared by every pair of operators in the network represented as a density plot.

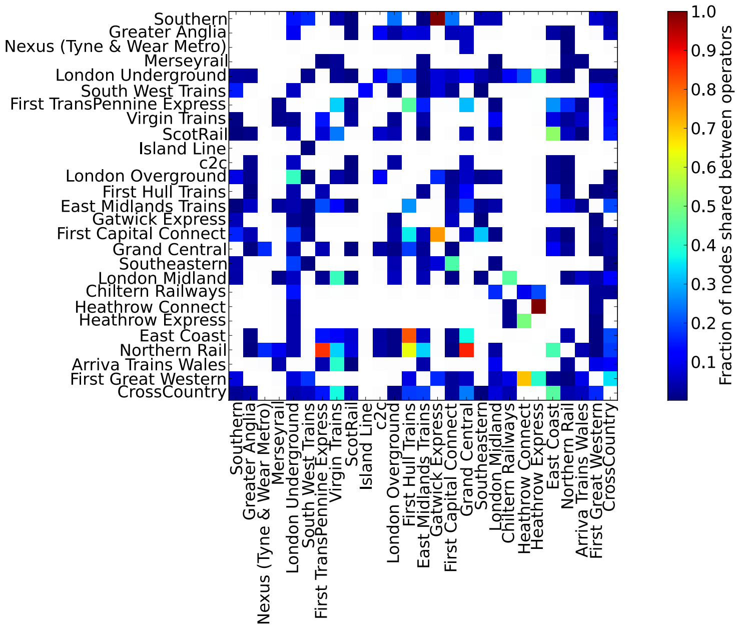


Figure 4.7: The fraction of nodes that are shared by every pair of operators in the network presented as a density plot. The number of stations for the operators along the horizontal-axis are used for the denominator.

on the network from examining the fraction of nodes that are shared by each operator pair relative the the number of stations serviced by either one of the operators. In contrast to Figure 4.4, we find that Figure 4.5 is asymmetric, as the number of stations shared between each operator pair is normalized by the total number of stations serviced by the operator along the horizontal axis. We find that localized operators such as Heathrow and Gatwick Express tend to share all of their stations with at least one other operator. We also find that First Great Pennine Express shares the majority of the stations it services with Northern Rail. This information could, for example, be used to evaluate the level of coordination between the two operators. If shortest paths transferring from First Transpennine Express to Northern Rail spend on average more time at a transfer station than shortest paths transferring at the same station between trains of the same operator, this might provide an indication that there exists a lack of coordination between these two operators. In general, we might be able to obtain information about the level of coordination between operators by computing the fraction of aggregate waiting time spend while transferring between two TOCs and aggregate waiting time on the network.

# Chapter 5

# Estimating passenger flow

Our aim is to reduce the total waiting time for passengers who traverse a rail network. To do this, we first need to measure the current total waiting time $T_{tot}$ of the network. Let $w_{pqs}$ be the time spent at station $s$ waiting for a transfer from trip $p$ to trip $q$, and let $C_{pqs}$ be the number of passengers waiting at station $s$ for a transfer from trip $p$ to trip $q$. The total waiting time $T_{tot}$ for the network is then

$$T_{tot} = \sum_{p,q \in Q} \sum_{s \in S} w_{pqs} C_{pqs}, \tag{5.1}$$

where $Q$ is the set of all trips in the network and $S$ is the set of all stations in the network. When $q$ and $p$ describe the same trip or are part of the same route, then $C_{pqs} = 0$. One can estimate the number of passengers that transfer at station $s$ from individual tickets sale data as this contains information on the origin and target station for a subset of all journeys undertaken on the British rail network. Ticket sale information for the British rail network is collected by a system called LENNON and is used by Office of Rail Regulation (ORR) to publish quarterly statistics on the use of the rail network [18]. As the raw data collected by LENNON is not publicly available, we need to pursue a different course. Section 5.1 will, therefore, discuss a human mobility model know as the gravity model that can be used to estimate the number of passengers who travel between any pair of stations in the network.

Once information on passenger volume in the network has been obtained either by estimation or by actual data, we need to determine the waiting time that is experienced by the passengers when travelling to their destination. This requires knowledge of the specific journey that is undertaken by each passenger, where it is essential that we can distinguish between waiting and travel time. As smart-card data usually does not record the intermediate stations visited by a passenger, we need to investigate the paths or set of paths that a passenger is most likely to take between any pair of stations in the network [25]. This will be the subject of Section 5.2.

## 5.1 Origin-Destination matrices

Origin-Destination (OD) matrices contain information about the travel demand in a network. For example, in case of a rail network, the elements of an origin-destination matrix $T_{ij}$ would describe the number of passengers that travel from station $i$ to station $j$ [1]. OD matrices are traditionally constructed from data obtained through direct measurement (i.e. by counting) and/or by surveying users of a transportation system [2]. As data collection can take years – during which the network itself – the policies to which the network is subjected or user preferences might change, this process is extremely costly and often inaccurate [28]. Once an OD matrix has been obtained using traditional methods, this matrix can be updated from available passenger counts [2]. The relation between an OD matrix and the number of passengers that traverse an edge $e$ in the network is given by,

$$v_e = \sum_{ij} p_{ij}^e T_{ij}, \qquad e \in E, \tag{5.2}$$

where $v_e$ is the number of passengers that traverse the edge $e$, the variable $p_{ij}^e$ is the proportion of trips between node $i$ and $j$ that use edge $e$ and $T_{ij}$ is the OD matrix with origin $i$ and destination $j$ [1]. Finding an OD matrix is thus an example of an inverse problem in which $T_{ij}$ is estimated from known values $v_e$ and $p_{ij}$ [1]. The values for $p_{ij}$ are usually estimated using an assignment model that describes the route choice mechanisms on a network [7].

See [1], for a review of different methods used to construct OD matrix on the basis of available OD matrices and passenger volume counts.

### 5.1.1 The gravity model

When no prior OD matrix is available, one can use human mobility models to estimate the passenger distribution on the network. One example of a frequently used human mobility model is the gravity model. The model, which is obtained by analogy from Newton's law of gravity, assumes that the passenger flow between two regions is directly related to the population size in each regions and inversely related to the distance between them [4]. One can use different measures for the distance between two regions. Examples include travel distance, travel time, or travel cost. One can also choose different functions to relate the distance between two regions to the flow between them [4].

The gravity model in its most basic form is given by

$$T_{ij} = K\frac{X_i Y_j}{d_{ij}^{\mu}} \tag{5.3}$$

where $K$ is a normalization factor, $\mu$ is a model parameter that one can determine (e.g. by multiple regression analysis) and $d_{ij}$ is Euclidean distance between a region $j$ to the region $i$ [4]. The variables $X_i$ and $Y_j$ are generally defined as the production ability of a zone $i$ and the attraction ability of a zone $j$, respectively [4]. In the context of a rail network, we can interpret $X_i$ as the number of passengers that depart from station $i$, and $Y_j$ as the number of passengers that arrive at station $j$. Their values can be determined from passenger counts at individual stations. The total number of passengers that enter and exit a station $i$, as provided by the data described in Section 2.3, are given by $s_i^{\text{exit}}$ and $s_i^{\text{enter}}$ respectively.

For a realistic approximation of the number of passengers travelling between any pair of stations in a transportation network, we require that the number of passengers that exit a station $i$ as predicted by the gravity model has to be equal to the number of passengers that exit a station $i$ as found from passenger counts [4]:

$$\sum_{k=1}^{N} T_{ik} = s_i^{exit}. \tag{5.4}$$

The same requirement is imposed for the number of passengers that enter a station

$i$ [4]:

$$\sum_{l=1}^{N} T_{lj} = s_j^{enter}. \tag{5.5}$$

Substituting the expression for $T_{ij}$ given in Equation (5.3) into Equation (5.4), we find

$$KY_j \sum_{l=1}^{N} X_l d_{lj}^{\mu} = s_j^{enter}. \tag{5.6}$$

Rewriting this expression for $Y_i$ yields

$$Y_j = \frac{s_j^{\text{enter}}}{K \sum_{l=1}^{N} X_l / d_{lj}^{\mu}} \tag{5.7}$$

Similarly, we find

$$X_i = \frac{s_i^{\text{exit}}}{K \sum_{k=1}^{N} Y_k / d_{ik}^{\mu}} \tag{5.8}$$

The resulting gravity model, also know as the doubly constraint gravity model, is now given as [4]

$$T_{ij} = \frac{1}{K} \frac{s_i^{exit} s_j^{enter} / d_{ij}^{\mu}}{\left(\sum_l X_l / d_{lj}^{\mu}\right) \left(\sum_k Y_k / d_{ik}^{\mu}\right)}, \tag{5.9}$$

The advantage of the gravity model is that it is easy to understand and apply. However, it fails to accurately represent the influence of the trip purpose in the assignment of passenger volume. It was, for example, found that $\mu = 1/2$ is most appropriate for describing commuting flows, while for social and shopping trips the appropriate values are $\mu = 3$ and $\mu = 2$ respectively [5].

Extensions of the gravity model have been developed to appropriately account for different trip purposes using the concept of travel-time factors. As we will use the doubly constrained gravity model for the estimation of the OD matrix, we refer to [4, 5] for a discussion on improved versions of the gravity model. In future work, we could explore the effects of the estimated OD matrix on the solution to the optimization problem defined in Chapter6. Other human mobility models that can be used to estimate passenger volumes on a network include the radiation model, the intervening opportunity model, the direct demand model, and the growth factor model [4, 5].

## 5.2 Shortest paths

At this point, we assume that the number of passengers that travel between each pair of stations in the network is available either from direct measurement or from a human mobility model, such as the gravity model. We now want to determine which path passengers are likely to take when travelling between all pairs of stations in the network.

To do this, we model passengers as a group of identical agents who make the rational choice to always take the 'shortest' path from their origin to target station. Recall from Section 3.3.2 that the shortest path (or geodesic path) between two nodes $i$ and $j$ in a network is the path with the smallest sum of edge weights [29]. In this section we describe Dijkstra's algorithm which can be used to find the shortest paths on a monoplex network. We then show that Dijkstra's algorithm can also be used to find the shortest paths on multilayer networks. Finally, we propose an adaptation of Dijkstra's algorithm for multilayer networks which can potentially reduce the number of iterations that is required to find all shortest paths.

### 5.2.1 Monoplex network: Dijkstra's Algorithm

A variety of methods exist for finding a single or all shortest paths in a network. The simplest method is the breath first search (BFS) algorithm that finds the shortest path for unweighted graphs. When considering a weighted network, Dijkstra's algorithm (or adaptations thereof) are popular as it is the fastest known single-source shortest path algorithm [12]. For each node in the network, both methods store the distance to the source node and the ancestor of the node. An *ancestor* (or *predecessor*) of a node is the node that precedes this node on a path from the source node to this node [12]. A node can have multiple ancestors, as there can be multiple paths connecting the node and the source node. Dijkstra's algorithm begins with the initialization of a network: for each node $i$ in the network, the distance $d(i)$ to the source node is set to infinity and the ancestor $a(i)$ is given as 'Unknown'. Subsequently, the distance $d(s)$ of the source node $s$ is set to 0. Two sets are constructed: the set $M$ contains the nodes that have been visited and is initially empty and the set $U$ contains the nodes that have been visited and initially equals the set $V$. At each iteration, the node $u$ with the smallest distance to the source node is selected (e.g. using a heap queue algorithm), removed from the set $U$, and added to the set $M$. For each neighbour $v$ of the node $u$, the distance and ancestor $a(v)$ for node are only updated if

$$d(u) + w(u, v) \leq d(v), \tag{5.10}$$

where $w(u, v)$ is the edge weight between nodes $u$ and $v$. The updated distance and ancestry of node $v$ are now given by

$$d(v) = d(u) + w(u, v), \quad a(v) = u. \tag{5.11}$$

In the above, we assume that we find a single shortest path between every two nodes in the network. This can be generalized to find all shortest paths by storing all ancestors of a node. A new node $u$ is selected in each subsequent iteration as long as the set $U$ is not empty. When $U$ is empty, the algorithm terminates. A graphical representation of Dijkstra's algorithm in a monoplex network can be found in Appendix D. We give the pseudo-code for Dijkstra's algorithm in Algorithm 1 in Appendix E.

When Dijkstra's algorithm terminates, the distance $d(i)$ for every node in the network gives the smallest edge sum when going from the source node $s$ to node $i$. One can deduce the shortest path from the ancestor information stored for each node. We first find the ancestor of the target node $i$, then the ancestor of the ancestor of node $i$ an so on, until we find a node whose ancestor is the source node. The set of nodes visited during this procedure are the nodes that lie on the shortest path from the source to the target node. For a proof of Dijkstra's algorithm, see [12].

### 5.2.2 Multilayer networks: Dijkstra's Algorithm

In some cases, more information about the network is given than can be represented in a monoplex network. For example, the connections in a railway network are subject to the existing schedules applied to this network. Edges between nodes therefore only exist at specific moments in time. Networks in which the temporal aspect is incorporated either through the use of timestamps or the construction of a multilayer network, might therefore provide additional and/or more accurate information about the dynamics in the modelled system than when using a monoplex representation.

As the path taken by passengers on the British rail network depends on the availability of connections between their departure and arrival time, we will use a multilayer network representation to obtain the shortest paths in our network. As no shortest-path algorithm has been developed for multilayer networks, we will attempt to adapt Dijkstra's algorithm for multilayer networks.

We assume that a path on a network in this case includes inter-layer edges. The shortest path for a multilayer network is, thus, the sequence of node-layer pairs that have been traversed to get from a source node-layer to a target node-layer. As we can represent a multilayer network by its underlying graph, we can map each tuple in the multilayer network to a node in a monoplex network. We can then find the shortest paths on a multilayer network using Dijkstra's algorithm on the monoplex representation (see Figure 5.2).

For monoplex networks, a shortest path between two nodes depends only the topology of the network and the type of edge weights that are used to construct the network (see Section 4.1). Shortest paths in monoplex networks are therefore independent of any characteristics that the nodes in the network might possess. Each entity in a system is represented exactly once in its network representation. A node can thus be considered unique within the network. One interpretation of a multilayer network is that it segregates the nodes in a monoplex network on the basis of their characteristics. For example, in a network representation of the British rail system, nodes could be segregated based on the TOCs that service them and the times that they are active (i.e. a train arrives or departs). Instead of the node, the combination of a node and its elementary layers is now unique (e.g. the tuple (Birmingham New Street, Arriva Trains Wales, Wednesday 15.53)). The node itself (Birmingham New Street) has multiple representations in the network, as it can appear in multiple node-layer tuples.

When applying Dijkstra's algorithm to multilayer networks, we find the shortest paths between uniquely defined entities in a network (i.e. nodes and tuples). However, in a multilayer network one can also find shortest paths between entities

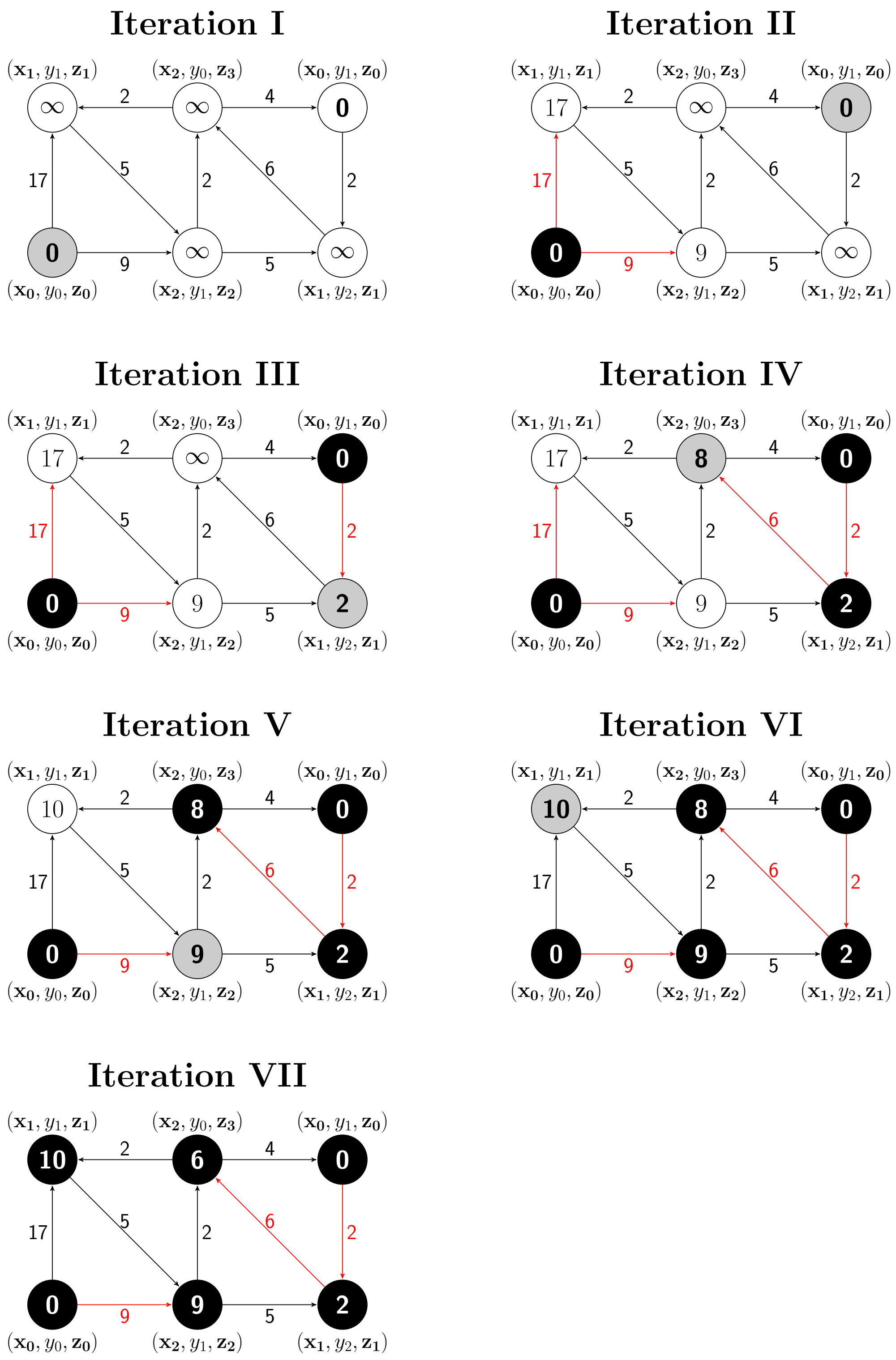

Iteration I
Iteration II
Iteration III
Iteration IV
Iteration V
Iteration VI
Iteration VII
$(\mathbf{x}_1, y_1, \mathbf{z}_1)$
$(\mathbf{x}_2, y_0, \mathbf{z}_3)$
$(\mathbf{x}_0, y_1, \mathbf{z}_0)$
$(\mathbf{x}_0, y_0, \mathbf{z}_0)$
$(\mathbf{x}_2, y_1, \mathbf{z}_2)$
$(\mathbf{x}_1, y_2, \mathbf{z}_1)$

Figure 5.1: A graphical representation for Dijkstra's algorithm on the monoplex representation of a multilayer network. We exclude the aspect $Y$ from the aggregated tuple. The tuple reference is given above or below each node in the network, where the elementary layers that are included are represented using bold face. The node values correspond to the current shortest distance from the aggregated source tuple (lower left tuple and upper right). White node represent tuples that are unvisited. Black nodes indicate that a tuple has been visited. A grey node is the source tuple currently being evaluated. The red vertices give the current shortest paths in the network.

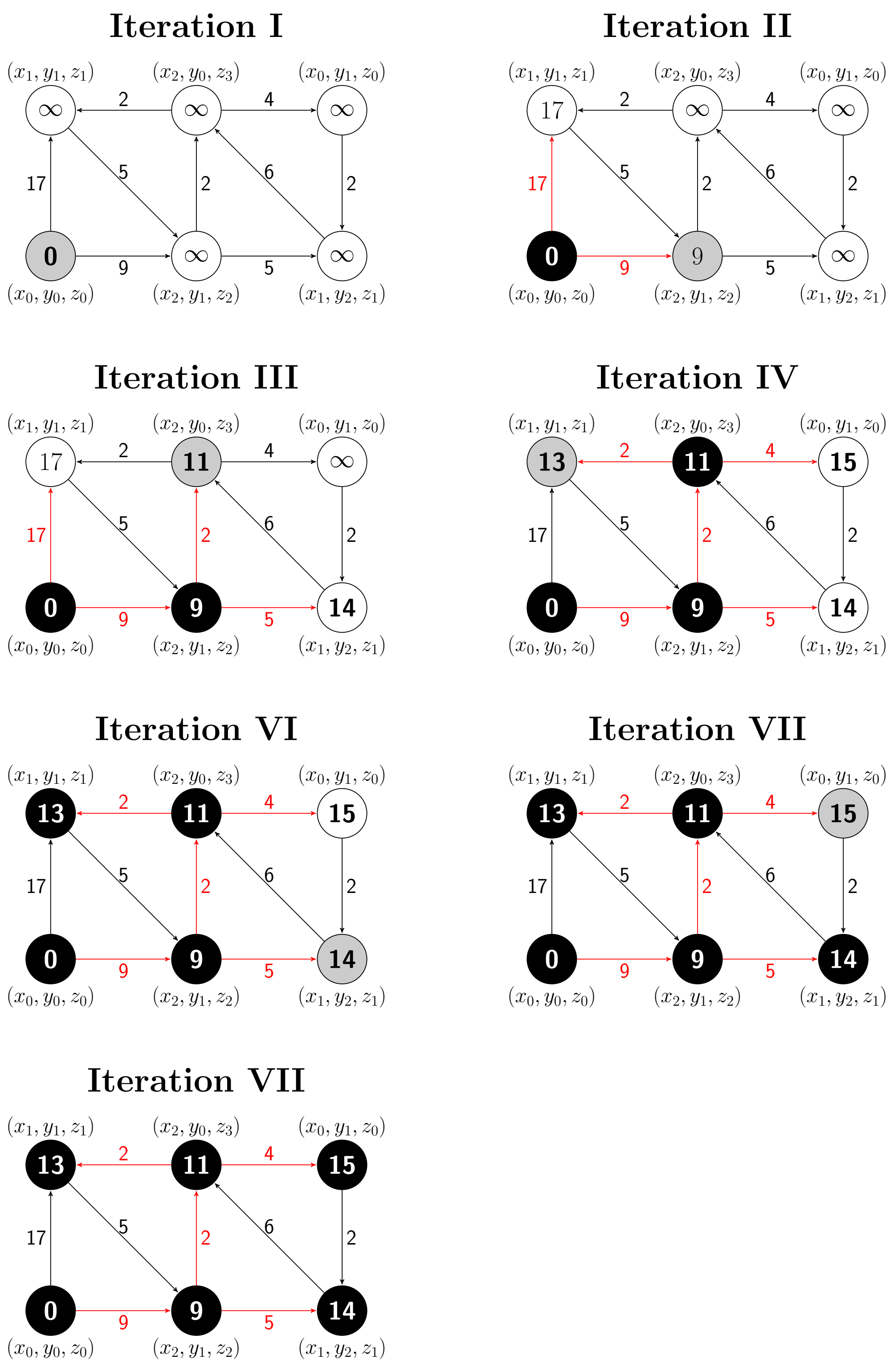


Figure 5.2: A graphical representation for Dijkstra's algorithm on the monoplex representation of a multilayer network. The tuple reference is given above or below each node in the network. The node values correspond to the current shortest distance from the source tuple (lower left tuple). White nodes represent tuples that are unvisited. Black nodes indicate that a tuple has been visited. A grey node is the source tuple currently being evaluated. The red vertices give the current shortest paths in the network.

that are not uniquely defined. For example, consider a 2-aspect multilayer network with a set of nodes $X = \{x_0, x_1, x_2\}$ and two aspects $Y$ and $Z$. The set of elementary layers of aspect $Y$ is $L_y = \{y_0, y_1, y_2\}$. Similarly, the set of elementary layers for aspect $Z$ is $L_z = \{z_0, z_1, z_2, z_3\}$ respectively. The set of node-layer tuples in the network is $V_M = \{(x_0, y_0, z_0), (x_0, y_1, z_0), (x_1, y_1, z_1),$
$(x_1, y_2, z_1), (x_2, y_0, z_3), (x_2, y_1, z_2)\}$. See Figure 5.1, for a graphical representation of this network.

We now define a set of *aggregated node-layer tuples*. An aggregated node-layer tuple is a representation of a node-layer tuple in which the elementary layer information of one or more aspects is excluded. In the case of the example described above, we could exclude the aspect $Y$ and define a set of aggregated node-layer tuples $\hat{V}_M = \{(x_0, z_0), (x_1, z_1), (x_2, z_3), (x_2, z_2)\}$. Once the aspect that is excluded has been defined, we can map each aggregated node-layer tuple to at least one, but generally more, node-layer tuples. Conversely, a node-layer tuple can be mapped to only one aggregated node-layer tuple. For instance, the aggregated node-layer tuple $(x_0, z_0)$ can be mapped to two node-layer tuples (i.e. $(x_0, y_0, z_0)$ and $(x_0, y_1, z_0)$.

Instead of finding the shortest path between node-layer tuples, we can now find the shortest paths from an aggregated node-layer tuple to either node-layer tuples or aggregated node-layer tuples. One can use aggregated node-layer tuples to find the shortest paths on the network if the choice for a shortest path is independent of one or multiple aspects. For example, when finding the shortest paths on a rail network with aspects defined as routes and time, passengers might be assumed to be indifferent to the routes they use as long as these routes lie on a shortest path.

From this point, we will refer to node-layer tuples as tuples and aggregated node-layer tuples as aggregated tuples. To find the shortest paths using aggregated tuples, we again map our multilayer network to the underlying graph. The main difference is that in the case of aggregated source tuples we can have multiple source nodes. From Figure 5.1, we see that the node-layer tuples $(x_0, y_0, z_0)$ and $(x_0, y_1, z_0)$ corresponding to the aggregated tuple $(x_0, z_0)$ are source nodes. When the source nodes have been initialized, we can use Dijkstra's algorithm on the network. The distance $d$ now gives the distance of each tuple to the aggregrated source tuple. For example, the distance from the aggregated source tuple $(x_0, z_0)$

to the tuple $(x_1, y_1, z_1)$ is 10. Similarly, the distance from the aggregated source tuple to the tuple $(x_1, y_2, z_1)$ is 2.

Additionally, we can find the distance between aggregated tuples. The distance from an aggregated source tuple to an aggregated tuple is given by the minimum of the distances from the aggregated source tuple to tuples that can be mapped to this aggregated tuple. For instance, the distance from the aggregated source tuple $(x_0, z_0)$ to the aggregated tuple $(x_1, z_1)$ would be given by the minimum of the distances from the aggregated source tuple $(x_0, z_0)$ to the tuples $(x_1, y_1, z_1)$ and $(x_1, y_2, z_1)$. We, thus, find that the distance between the two aggregate tuples is given by $\min(10, 2) = 2$. To find the shortest paths between aggregated source tuples and aggregated tuples, we can thus use Dijkstra's algorithm followed by a post-processing step. The post-processing step consist of finding the shortest paths to all tuples that can be mapped to the same aggregated tuple and then selecting the path that has the smallest path length.

### 5.2.3 Adaptation of Dijkstra's algorithm

However, if we want to find the shortest paths between two aggregated tuples, the number of iterations required for Dijkstra's algorithm can potentially be reduced. We will now describe an adaptation of Dijkstra's algorithm that can be used to find the shortest paths between aggregated tuples.

For each tuple $\kappa$ in the network, we set the distance $d(\kappa)$ to infinity and the ancestor $a(\kappa)$ is specified as 'Unknown'. We construct the sets $M$ and $U$ containing the visited and unvisited tuples respectively. The set $M$ is empty upon construction, while the set $U$ contains the tuples in the set $V_M$. The aggregate tuple associated with the tuple $\kappa$ from which one or more aspects are excluded is denoted $\hat{\kappa}$. For each aggregated tuple $\hat{\kappa}$, we set the distance $\hat{d}(\hat{\kappa})$ to infinity and the ancestor $\hat{a}(\hat{\kappa})$ to 'Unknown', where $\hat{d}(\hat{\kappa})$ gives the distance from the aggregated tuple $\hat{\kappa}$ to the aggregated source tuple and $\hat{a}(\hat{\kappa})$ stores the ancestor tuple that minimizes the path length to the aggregated source tuple. We construct a set $\hat{M}$ that contains the aggregated tuples that have been visited and a set $\hat{U}$ that contains the aggregated tuples that have not been visited. The set $\hat{M}$ is empty upon construction. The set $\hat{U}$ is initialized to contain the aggregated tuples contained in the set $\hat{V}_M$.

Let $\hat{\gamma}$ be the aggregated source tuple. The distance is specified as $\hat{d}(\gamma) = 0$. For all tuples $\gamma$ that can be mapped to the aggregated source tuple $\hat{\gamma}$, the distance is $d(\gamma) = 0$

At each iteration, a tuple $\alpha$ with the smallest distance to the aggregate source tuple $\hat{\gamma}$ is selected, removed from the set $U$ and added to the set $M$. Additionally, if the the corresponding aggregated tuple $\hat{\alpha}$ is still in the set $\hat{U}$, then the aggregated tuple is removed from the set $\hat{U}$ and added to the set $\hat{M}$. For each neighbour tuple $\beta$ of the tuple $\alpha$, the distance $d(\beta)$ and ancestor $a(\beta)$ are updated if

$$d(\alpha) + w(\alpha, \beta) \leq d(\beta), \tag{5.12}$$

where $w(\alpha, \beta)$ is the weight of the edge that connects tuple $\alpha$ to tuple $\beta$. If this condition is satisfied, then,

$$d(\beta) = d(\alpha) + w(\alpha, \beta), \quad a(\beta) = \alpha. \tag{5.13}$$

Furthermore, we check the condition

$$d(\beta) \leq \hat{d}(\hat{\beta}). \tag{5.14}$$

If this condition is satisfied, then we update the distance $\hat{d}(\hat{\beta})$ and ancestor $\hat{a}(\hat{\beta})$:

$$\hat{d}(\hat{\beta}) = d(\beta), \quad \hat{a}(\hat{\beta}) = \alpha \tag{5.15}$$

A new tuple $\alpha$ whose distance to the aggregated tuple is the smallest, is selected in each subsequent iteration. In comparison with Dijkstra's algorithm described in Section 2.2.1, we can now impose a different stopping condition; the algorithm terminates when the set $\hat{U}$ is empty. Note that if the set $U$ is empty, then the set $\hat{U}$ is empty. The shortest distance from the aggregated source tuple to an aggregated tuple $\hat{\delta}$ is now given by $\hat{d}(\hat{\delta})$. The shortest path between the two aggregated tuples can be found from the ancestor information given in $a(\delta)$ for the tuple stored in $\hat{a}(\hat{\delta})$.

As the number of aggregated tuples in $\hat{U}$ is always smaller or equal to the number of tuples in $U$, the use of the adjusted version of Dijkstra's algorithm reduces the number of iterations required to find the shortest paths between two aggregated tuples. For example, in Figure 5.1 the algorithm would have terminated after Iteration VII as at that point the shortest path to all aggregate tuples in the set $\hat{U}$ is found. In future work, the effect of the adjusted version of Dijkstra's algorithm on a variety of multilayer networks could be studied. The pseudo code for the adapted version of Dijkstra's algorithm can be found in Appendix E.

# Chapter 6

# Schedule Optimization

In this section we introduce a method that can be used to decrease the aggregate waiting time on a public transportation network by perturbing the existing schedule. We follow the methodology described in [11].

## 6.1 Problem formulation

Let $R$ be the set containing all routes in a network. Let $r \in R$ denote one particular route in this network. The stations visited by route $r$ are contained in the set $S_r \subseteq S$, where $S$ is the set containing all stations in the network. The set $P$ contains all the trips associated with route $r$. Let $p \in P$ be one trip along route $r$. We define the time headway $h_p$ as the average inter-departure time of trips along route $r$. The time interval spent at station $s \in S_r$ during trip $p$ is given by $t_{ps} = [t_{ps}^a, t_{ps}^d]$, where $t_{ps}^a$ and $t_{ps}^d$ are the arrival and departure time at station $s$ for trip $p$ respectively. Station $s$ can also be serviced by other routes and their associated trips. The set of $Q$ contains the trips that are not included in route $r$. The union of the sets $Q$ and $P$ contains all trips on the network.

For each station $s$, the minimum time needed to transfer between trips is denoted by $t_s^t$. A transfer from trip $p$ to trip $q$ exists if the sum of the arrival time of trip $p$ and the transfer time at station $s$ is smaller or equal to the departure time of trip $q$ at station $s$, i.e. $t_{ps}^a + t_s^t \leq t_{qs}^d$. Similarly, one can transfer from trip $q$ to trip $p$ if $t_{qs}^a + t_s^t \leq t_{ps}^d$. The ability to transfer between trips $p$ and $q$ is determined by two indicator variables:

$$\xi^-_{pqs} = \begin{cases} 1, & \text{if } t^a_{qs} + t^t_s \le t^d_{ps} \\ 0, & \text{otherwise} \end{cases}, \qquad \xi^+_{pqs} = \begin{cases} 1, & \text{if } t^a_{ps} + t^t_s \le t^d_{qs} \\ 0, & \text{otherwise} \end{cases}. \tag{6.1}$$

We seek to estimate the number of passengers who transfer from trip $q$ to trip $p$ and vice versa. Let $F$ be the set that contains the shortest paths between each pair of stations in the system obtained using the adapted version of Dijkstra's algorith for multilayer networks. We use the adapted version, as we assume that passengers are indifferent to the routes they use to get to their destination. Let $f \in F$ be a vector containing the tuples with elements $(s, r_i, t)$, where $s$ is the station, $r_i$ denotes the elementary route layer in which trip $i$ lies and $t$ gives the elementary time. If $[(s, r_q, t^a_{qs}), (s, r_p, t^d_{ps})]$ is in $f$, then the number of people that transfer from trip $q$ to trip $p$ on their journey from station $s^{start}_f$ to $s^{end}_f$ is given by the entry ($s^{start}_f$, $s^{end}_f$) of the OD matrix. The total numbers of passengers leaving $C^-_{pqi}$ and entering $C^+_{pqi}$ trip $p$ at station $s$ are, thus, given by

$$C^-_{pqs} = \textstyle\sum_{w \in W} \phi^-_{pqw} T(s^{start}_w, s^{end}_w), \qquad C^+_{pqs} = \textstyle\sum_{w \in W} \phi^+_{pqw} T(s^{start}_w, s^{end}_w), \tag{6.2}$$

where $T$ is the OD matrix and $\phi^-_{pqw}$ and $\phi^+_{pqi}$ are indicator functions:

$$\begin{aligned} \phi^-_{pqf} &= \begin{cases} 1, & \text{if } [(s, r_p, t^a_{ps}), (s, r_q, t^d_{qs})] \in f \\ 0, & \text{otherwise} \end{cases}, \\ \phi^+_{pqf} &= \begin{cases} 1, & \text{if } [(s, r_q, t^a_{qs}), (s, r_p, t^d_{ps})] \in f \\ 0, & \text{otherwise} \end{cases}. \end{aligned} \tag{6.3}$$

Our aim is to shift the existing schedule for each trip $p$ by an amount $\delta_p$ such that the waiting times along route $r$ are minimized while all available transfers between trips remain feasible. To do this, we define a variable $\omega_{pqs}$ that gives the sum of the individual waiting times for passengers who transfer from trip $p$ to trip $q$ at station $s$ (and vice versa) if we shift the schedule by $\delta_p$:

$$\omega_{pqs} = (t^d_{qs} - (t^a_{ps} + \delta_p))C^-_{pqs} + ((t^d_{ps} + \delta_p) - t^a_{qs})C^+_{pqs}, \quad \forall p \in P,\ \forall q \in Q,\ \forall s \in S_r. \tag{6.4}$$

This yields rise to the following optimization problem:

$$\min_{\delta_p} \sum_{p \in P} \sum_{s \in S} \sum_{q \in Q} \omega_{pqs} \tag{6.5}$$

subject to the constraints

$$-h_p/2 \leq \delta_p \leq h_p/2 \qquad \forall p \in P, \tag{6.6}$$

$$\xi^-_{pqs}(t^d_{qs} - (t^a_{ps} + \delta_p)) \geq t^t_s \qquad \forall p \in P,\ \forall q \in Q,\ \forall s \in S_r, \tag{6.7}$$

$$\xi^+_{pqs}(t^d_{ps} + \delta_p) - t^a_{ps}) \geq t^t_s \qquad \forall p \in P,\ \forall q \in Q,\ \forall s \in S_r, \tag{6.8}$$

$$\delta_p \in \mathbb{R} \quad \omega_{pqs} \in \mathbb{R}^+ \qquad \forall p \in P,\ \forall q \in Q,\ \forall s \in S_r. \tag{6.9}$$

Equation (6.6) ensures that the maximum shift of a trip in the schedule does not exceed half the time headway (see Section 2.2 for the definition). Equations (6.7) and (6.8) guarantee that all existing connections remain feasible.

Because we attempt to optimize the schedule by solving equation (6.5) for each route $r \in R$ using the (updated) schedule, the algorithm used for our optimization problem is an example of a greedy algorithm. A greedy algorithm chooses the locally optimal solution in the hope of finding the global optimal solution [12]. Although greedy algorithms can yield the optimal solution in a wide range of problems, we do not have any guarantee that this is the case for our problem. Finding an optimal solution is, however, not our main objective, as we are only looking for a significant improvement of the existing schedule.

Even if the new schedule is optimal and we see a significant decrease in the total waiting time of the network, it might not be a desirable solution. As we shift each trip $p$ individually, the periodicity of a route might be disrupted and the schedule could even become highly irregular. An irregular schedule is generally no concern in systems for which the demand for and the frequency of connections are high. In such systems passengers tend to arrive at their departure station at 'random' as the expected waiting times at the departure station and all subsequent transfer stations are low. Consequently, the total waiting time in such a system is relatively small even when passengers do not follow a preplanned schedule. In systems with a low departure frequency, however, the use of an irregular schedule has been found to encourage passengers to either learn the schedule or use a journey planner to attempt to limit the negative consequences of an irregular timetable [41]. Journeys on the British rail network are subject to specific rules that limit the routes passengers are allowed to take as the result of the dynamic pricing policies and the different TOCs [35]. ATOC therefore advices passengers to consult a journey planner before departing [35]. An irregular schedule is therefore a reasonable solution to the optimization problem when considering the British railway system. Nevertheless, the irregularity of the optimized schedule can be controlled by adjusting constrains in the optimization problem [11].

## 6.2 Implementation of optimization problem

For the construction of the optimization problem we use the Python Optimization Modelling Object (Pyomo) open-source software package [17]. We solve the optimization problem using the Network-Enabled OPtimization System (NEOS) optimization server using the COIN Branch and Cut (CBC) solver [42]. The CBC solver has been developed by the COmputational INfrastructure for Operations Research (COIN-OR) project and uses an algorithm based on the simplex algorithm developed by George Dantzig for solving linear optimization problems [42].

To solve the optimization problem (6.5), we first construct a 2-aspect multilayer network with routes and times as aspects and find the shortest paths in this network. We then initialize the system such that the arrival and departure times of the trips $p$ corresponding to the route $r$ and the set of intersecting trips $q$ at every station along route $r$ in the system is known. We determine the values for $\xi^{-}_{pqs}$ and $\xi^{+}_{pqs}$ from equation (6.1)and we determine the transfer volumes $C^{-}_{pqs}$ and $C^{+}_{pqs}$ using equation (6.2). We then optimize the schedule for route $r$.

As the shift of each trip $p$ in route $r$ is restricted by the constraint that all existing transfers given in $\xi^{-}_{pqs}$ and $\xi^{+}_{pqs}$ must remain feasible, we make the assumption that the combination of trips that describe each shortest path remains unchanged. This is a strong assumption, as it is likely that shifts in the schedule allow new transfer possibilities that yield shorter paths than are currently considered. The construction of a new multilayer network using the optimized schedule that results from shifting the route $r$, which can be used to construct a new set of shortest paths, is a possible solution, but it is computationally expensive. Consequently, we keep the transfer volumes $C^{-}_{pqs}$ and $C^{+}_{pqs}$ constant throughout the optimization of the routes in the network. Note that we update the indicator functions $\xi^{-}_{pqs}$ and $\xi^{+}_{pqs}$ to ensure that new transfer connections remain feasible in subsequent route optimizations.

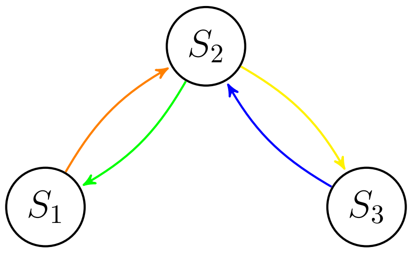


Figure 6.1: Monoplex representation of the model network. Each node represents a station in the network. Each edge represents a route. The edge colors indicate the TOC that is associated with each route (i.e. orange=Arriva Trains Wales, green=First Great Western, yellow=Southern and blue=ScotRail).

The existing schedule can now be optimized by perturbing a route $r$ in the network. When the waiting time along the route remains constant after optimization, but a shift is suggested, we choose not to update the schedule. Otherwise, the arrival and departure times of the trips along route $r$ are updated, i.e.

$$t_{ps} = \left[t^a_{ps} + \delta_p, t^d_{ps} + \delta_p\right], \quad \forall p \in P_r, \ \forall s \in S_r. \tag{6.10}$$

Note that we make the assumption that all arrival and departure times at all stations serviced by trip $p$ are shifted by $\delta_p$. We thus do not lengthen or shorten the time spent at individual stations along the route. In the subsequent iteration, a new route is selected, the values for the indicator variables are determined, and the schedule is optimized. We repeat this process for all routes in the network. We expected that the schedule that we obtain using this procedure has a lower aggregate waiting time than the original schedule.

## 6.3 Example

To illustrate the optimization process we consider the monoplex network given in Figure 6.1. Each edge in the figure represents a route. The edge colors indicates the TOC that is associated with each route (i.e. orange=Route 1, green=Route 2, yellow=Route 3 and blue=Route 4). The nodes represent the stations at which two or more routes intersect. The schedule that is run on the network is given in Figure 6.2. Each rectangle presents a trip in the schedule. The color of each rectangle indicates which route the trip belong and can be matched with Figure

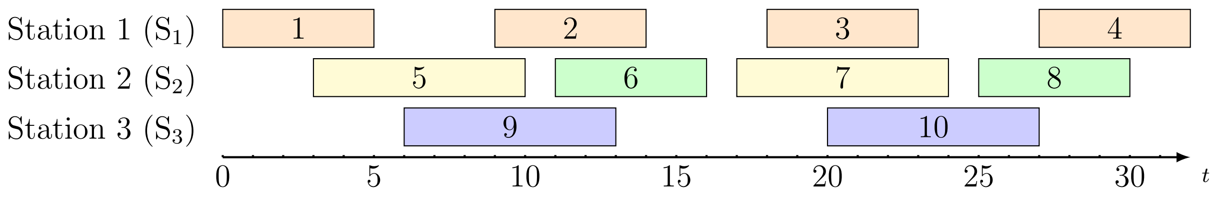


Figure 6.2: The schedule associated with the network given in 6.1. Each rectangle presents a trip in the schedule. The color of each rectangle indicates which route the trip belong and can be matched with Figure 6.1. The length of the rectangle is prescribed by the length of the trip and the position of the rectangle by the departure of the trip. For example, trip 1 is part of Route 1, departs from Station 1 at $t = 0$ and arrives at Station 2 at $t = 5$.

6.1. The length of the rectangle is prescribed by the length of the trip and the position of the rectangle by the departure of the trip. For example, trip 1 is part of Route 1, departs from Station 1 at $t = 0$ and arrives at Station 2 at $t = 5$. Similarly, trip 5 is runs along Route 2, departs from Station 2 at $t = 11$ and arrives at Station 1 at $t = 16$. From Figure 6.2, one can determine the transfers that are available to the passengers in the network. A transfer from trip $i$ to trip $j$ is possible if the rectangles corresponding to trip $i$ and trip $j$ do not overlap and the arrival station of trip $i$ and the departure station of trip $j$ are identical. The transfer from trip 1 to trip 5, for example, is not possible as trip 5 has departed from Station 2 before the arrival trip 1 at this station. The transfer from trip 6 to trip 10 is also not possible as the arrival station of trip 6 is Station 1 while the departure station of trip 10 is Station 3. Examples of possible transfers are trip 1 to trip 7, and trip 5 to trip 10.

From the model network and the associated schedule, we can construct a weighted, directed 2-aspect multilayer network (see Figure 6.3). For the construction of the multilayer network, we have followed the procedure described in Section 4.2. We assume that the stop time at a station is less than one minute. Consequently, there are no edges connection a node to its counterpart in the same route layer. Furthermore, we assume that the minimum transfer time at all stations is one minute. The average headway for each route is given in Table 6.1. Solid lines are used for edges that connect two node-layer pairs that lie in the same route layer. Dashed lines are used for edges that connect node-layer pairs

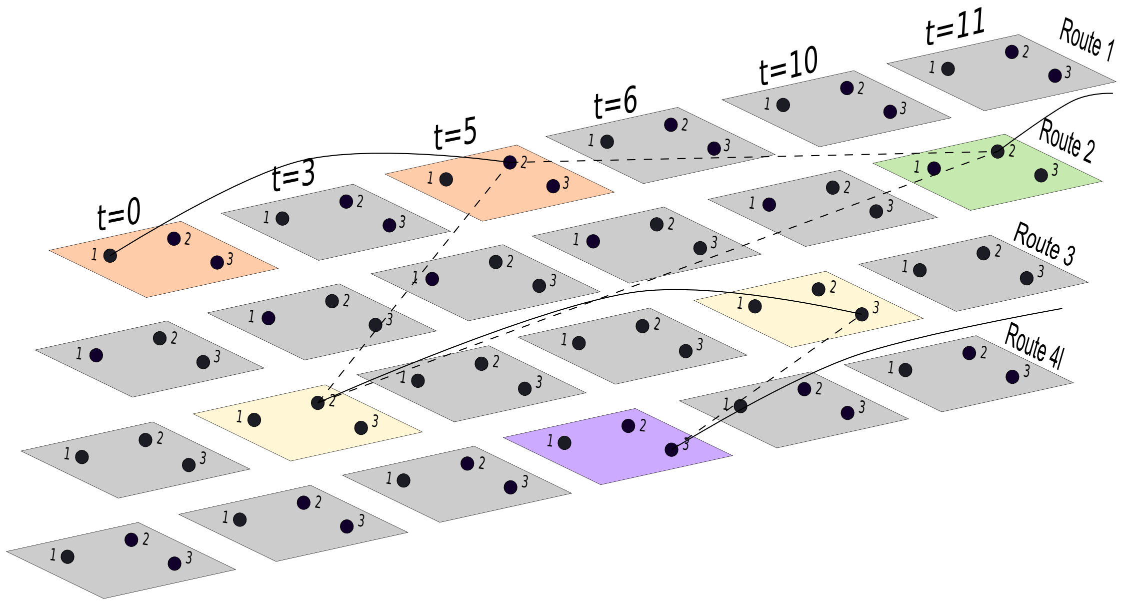


Figure 6.3: Multilayer network representation for $t = 0$ to $t = 11$. The vertical axis represents the different TOCs. The horizontal axis represent time. Only the first trip of each route is represented. A layer is coloured if it has at least one tuple with either an ingoing or outgoing edge. A layer is grey if the tuples have degree zero on the underlying graph. Elementary time layers in which there are no tuples with a degree greater or equal to one in the underlying graph are not represented in the network. For instance, $t = 1$ and $t = 2$ are excluded from the network presented in Figure 6.3. Solid lines are used for edges that connect two node-layer pairs that lie in the same route layer. Dashed lines are used for edges that connect node-layer pairs that differ both in route and time layer.

that differ both in route and time layer.

We want to find a schedule for which the aggregated wait time is lower than for the schedule given in Figure 6.3. To do this, we have constructed the an OD-matrix for the network:

$$OD = \begin{bmatrix} 0 & 1436 & 66 \\ 1436 & 0 & 8 \\ 66 & 8 & 0 \end{bmatrix} \tag{6.11}$$

Using adapted version of Dijkstra's algorithm, we find the shortest paths on the network. For example, we find that there is a shortest paths from Station 1 at $t = 0$ to Station 3 is given by

**Shortest path 1: Station 1 at $t = 0$ to Station 3**
(Station 1, Route 1, 0) $\rightarrow$ (Station 2, Route 1, 5) $\rightarrow$ (Station 2, Route 3, 17) $\rightarrow$ (Station 3, Route 3, 24)

Passengers on Shortest path 1 have to transfer at Station 2, where they have to wait for 12 minutes for the next departure. Another shortest path on the network describes the path between Station 1 and Station 3 that departs from Station 1 at $t = 9$:

**Shortest path 2: Station 1 at $t = 9$ to Station 3**
(Station 1, Route 1, 9) $\rightarrow$ (Station 2, Route 1, 14) $\rightarrow$ (Station 2, Route 3, 17) $\rightarrow$ (Station 3, Route 3, 24)

This path has a waiting time of 3 minutes. In both Shortest path 1 and Shortest path 2, passengers transfer from the Route 1 to the Route 2. We find that there are not shortest paths in the network for which passengers from another route transfer to Route 1, as this would return them to a previously visited station. For example, it is possible to transfer from trip 6 to trip 3 at Station 1. However, trip 6 departs from station 2 and trip 3 goes to station 2. A transfer to the Route 1 would, thus, naturally lead to a cycle. Consequently, there is no shortest path in which passengers transfer to a trip in Route 1.

We can now determine the aggregate wait time on Route 1. To do this, we find the total number of passengers that travel between Station 1 and Station 3 from the OD-matrix. From (6.11), we find that the total number of passengers that travel from Station 1 to Station 3 is 66. We assume that the passengers are homogeneously spread over Shortest path 1 and Shortest path 3. The aggregate waiting time of passengers on Route 1 is now $33 * 12 + 33 * 3 = 495$ minutes. The aggregate waiting time on the network is defined as the sum of the aggregate wait time of each route in the network. This is not the exact aggregate wait time on the network, but an upper bound as the wait time from a trip $p$ to a trip $q$ is included in the wait time for the route that includes the trip $p$ and the route that includes the trip $q$. The aggregate waiting time of passengers for the schedule 6.2 is found to be 1782 minutes.

We start the optimization process by finding a shift of the trips along the Route 1 that will reduce the aggregate wait time on the network. The initial waiting time along the Route 1 is computed using PYOMO by setting $\delta_p = 0$ for all $p \in P_{\text{Route 1}}$ and is found to be 495 minutes. Upon optimization of Route 1, we find that the new schedule reduces the waiting time along the route from 495 to 297 minutes, while the aggregate waiting time on the network is reduced from 1782 to 1386 minutes. In Figure 6.4, we label each trip following the notation that is introduced in (6.5); a trip $p$ is part of the route that is being optimized; the other trips are denoted by $q$. The optimized schedule is given in Figure 6.4(b).

We see that the shift of trip $p_1$ is restricted by constraint (6.6) as the waiting time for transfer to trip $q_3$ could have been further reduced by a larger positive shift. The shift of trip $p_2$ is restricted by constraint (6.7). A further shift would have made the transfer from trip $p_2$ to trip $q_3$ impossible as the minimum transfer time at Station 2 is 1 minute. We see that the trips $p_3$ and $p_4$ have been shifted in the new schedule. This does not effect the aggregate waiting time along the route as the wait time associated with both trips was 0 in the original schedule. We could have obtained a different schedule by shifting routes $p_3$ and $p_4$ within the limits of condition (6.6) for which the reduced aggregate wait time along the route would have been the same. The solution to the optimization problem is thus not unique.

Moreover, we observe that the departures along the Route 1 which were initially periodic, leaving every 10 minutes, have become more irregular leaving either 6 or 7 minutes after each other. We also note that the inter-departure times have decreased in the new schedule. Important is, however, that we do not change the number of trips along a route. The decrease in inter-departure time is, therefore, not an indication of an increase in the departure frequency along the route.

The subsequent optimization of Route 2 produces the schedule that can be found in Figure 6.4(c). The waiting time along the route remains constant and the aggregate waiting time of the network is not reduced. Subsequent optimization of Routes 3 and 4 yield reductions in waiting times along each route from 396 to 330 and 330 to 265 minutes respectively. For the final schedule given in Figure 6.4(e), we find that the network has an aggregated wait time of 1254 minutes.

| **Route** | **Intersecting trips** | **Headway** | **Waiting time along route** | **Minimum waiting time network** |
|---|---|---|---|---|
| Route 1 | 4 | 8 | 495 | 1254 |
| Route 2 | 8 | 10 | 495 | 1452 |
| Route 3 | 8 | 9 | 396 | 1452 |
| Route 4 | 4 | 11 | 396 | 1254 |

Table 6.1: Comparison of number of intersecting routes, headway and initial waiting time along the route. The final column describes the minimum overall waiting time on the network that can be obtained if the route is optimized first.

The aggregated waiting time on the network has been reduced by approximately is 30%.

As we are using a greedy optimization methods, there might be another schedule that provides a larger reduction in the aggregated wait time on the network. We now investigate whether the order in which the routes are optimized has an influence on the aggregate waiting time of the final schedule that is obtained. Different optimization strategies can be applied as routes can, for example, be optimized in order of increasing/decreasing overall waiting time, increasing/decreasing number of intersecting routes, or increasing/decreasing average headway time.

For our sample network, we are able to test every route ordering as the number of possible combination is only 4! = 24. We find that the total waiting time in the network is reduced from 1782 to 1650, 1452 or 1254 minutes depending on the ordering of the routes suggesting that the problem has two local and one global minimum. We also find that the global minimum is never reached when we start the optimization with either the Route 2 and Route 3. We reach the global minimum of 1254 minutes for 8 out of the 24 possible orderings. Similarly, we find the local minima of 1452 and 1650 for 12 and 4 orderings respectively. Table 6.1 gives the intuition that optimization of routes that are intersected by the smallest number of trips will give a larger reduction in the waiting time than routes that are intersected by a larger number of trips. However, any conclusions about the optimal strategy for the optimization of a transportation schedule will require either the study of larger and more realistic sample networks or the study of a set of real transportation schedules. This could be a direction for future work.

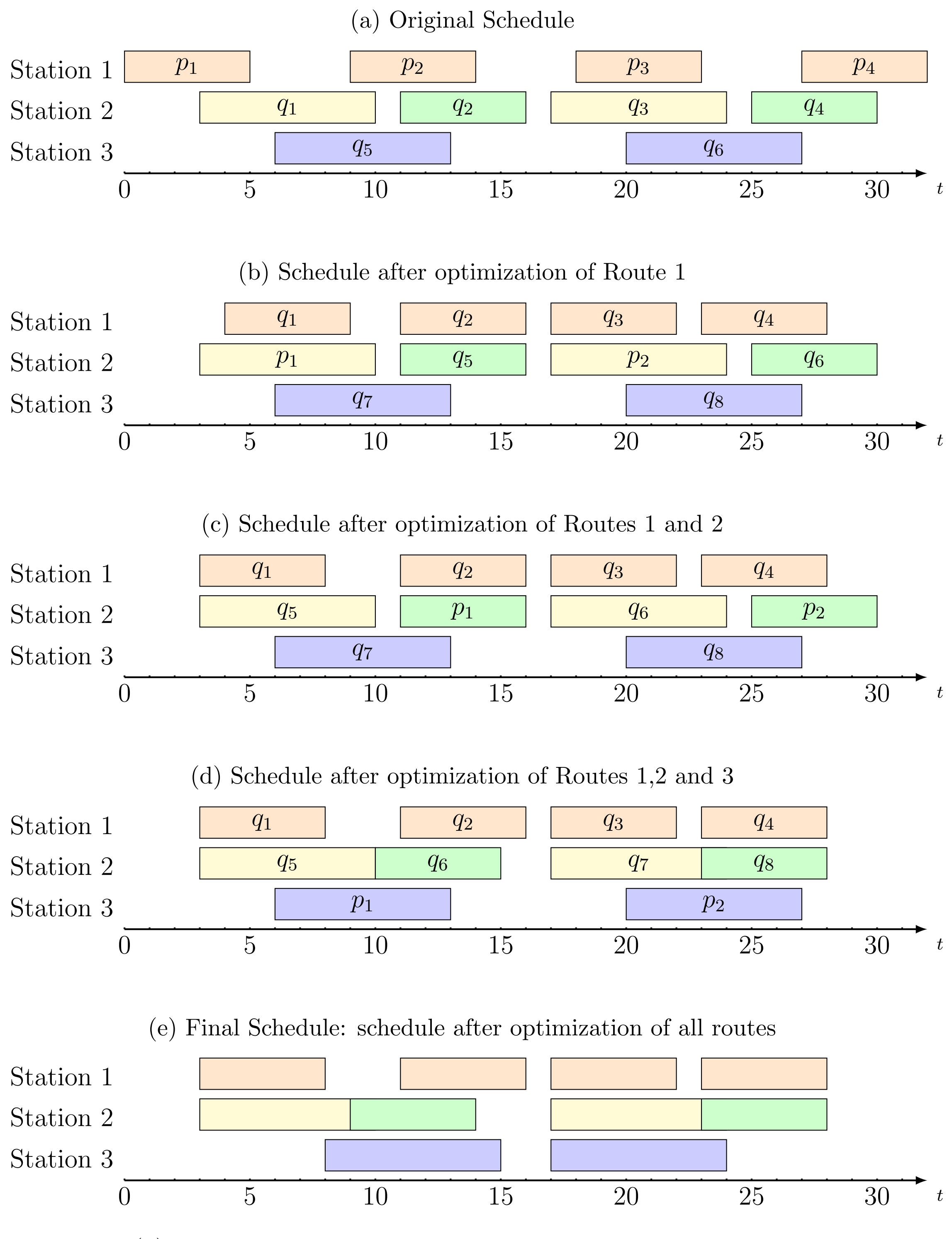


Figure 6.4: (a) The original schedule of the network given in Figure 6.1. The route that is currently being optimized is indicated by $p$. The other routes are denoted by $q$. (b)-(e) The schedule after each iteration of the optimization process. Again, $p$ is the trip that is currently optimized. The other routes are denoted by $q$. The result of the optimization of the route is given in the next figure. For example, the schedule after the optimization of the Route 1 in (a) is given in (b). (f) The optimized schedule. The trips labels are omitted as the optimization process has been completed.

# Chapter 7

# Conclusion

In this thesis, we have used a graph-theoretical framework to develop a method with which we can obtain estimates for the aggregate waiting times on public transportation networks. We needed these estimates for the construction of an optimization problem that attempts to reduce the aggregate waiting time by perturbing the trips in the network.

We first constructed two weighted, directed monoplex network representations of the British rail network (i.e. including and excluding the London Underground) and used centrality measures to obtain the central nodes in the network. We found that the stations with the highest degree and betweenness centrality are stations that have a high number of passengers interchanges. We also found that the London Underground is of critical importance for the British rail network, as London rail stations become more central when London Underground connections are included in the network.

We, then, constructed two single-aspect multilayer networks for the British rail system using TOCs and time as aspects. The time-aspect multilayer network describes the network for a single day. As expected, we found that the activity on the network is highest during the day. We also found that the network is highly segregated with respect to the TOC, as most stations are only serviced by one TOC. Additional analysis could be done on these networks. One could, for example, find the the number of operators that use the same track (edge) on the network. Furthermore, we could investigate the properties of the network on different days of the week.

To estimate the number of passengers that travel between each pair of stations, we construct an OD matrix using the doubly constrained gravity model. We discuss how shortest paths can be found on multilayer networks. An important observation is that one can find the shortest paths on any multilayer network by using Dijkstra's algorithm on its underlying graph. Additionally, we found that the shortest paths on a multilayer network are not only dependent on the topology of the network, but also on the specification of the aspects on the path. Using the concept of aggregated tuples, we found a potentially faster algorithm for multilayer networks in which the shortest path is assumed to be independent of one or multiple aspects. Future work could include a study of the possible advantages of this method.

Finally, we investigate how the aggregate waiting time on a public transportation network can be reduced. Following [11], we try to decrease the aggregate waiting time on a network by perturbing the existing schedule. Using a small sample network, we show that the aggregate waiting time on this network is reduced by introducing a perturbation to the original schedule. We also note that the final solution to the optimization problem is dependent on the order in which the routes are optimized.

As the sample network is an extremely simplistic, the next step would be to use the method described in this thesis to the data of the British rail network. This would allow us to see what the effects of small perturbations to an existing schedule would be for a real public transportation networks. Furthermore, we could, for example, compare the results of the optimization of the British rail network to a rail transportation network that has fewer TOCs (e.g. the Dutch rail network). This could provide insight into the effects of the use of different TOCs on the waiting time experienced by passengers on the network.

We could further improve the optimization procedure by recomputing the shortest paths for each schedule that is found after the optimization of one route. Additionally, we could investigate if a specific ordering of the routes can be found that will lead to the optimal schedule.

We could also obtain more information on the waiting times along different routes of the British rail network using the shortest paths that are found using Dijkstra's algorithm. Furthermore, the construction of a 2-aspect multilayer network in which waiting time is weighted twice more than in-vehicle time, could give a more realistic indication of the paths that passengers are most likely to take. Alternatively, this can be done by the construction of a set of 'desirable' paths in which shortest paths, but also paths with other desirable properties (e.g. lower ticket cost or fewer transfers), are included.

In conclusion, we believe that the application of the method to a real public transportation network is the first step to be taken in further research. Furthermore, we find that there are a variety of interesting directions that can be pursued in future work.

# Appendix A

# GTFS data description

Description of data contained in a General Transit Feed Specification file.

| [b!] **File** | **Field Name** | **Details** |
|---|---|---|
| agency.txt | agency_id | Unique identification for train operating company |
| | agency_name | Full name of train operating company |
| stops.txt | stop_id | Unique identification for rail station |
| | stop_name | Full name of rail station |
| | stop_lat | Latitudinal position of rail station |
| | stop_lon | Longitudinal position of rail station |
| routes.txt | route_id | Unique identification for route in the network |
| | agency_id | See agency.txt |
| calendar.txt | service_id | Unique identification for a set of days when a route is operated. |
| | | Note: multiple routes can be assigned the same service_id |
| | monday | Indicator of availability of service for all Mondays in date range |
| | tuesday | Indicator of availability of service for all Tuesdays in date range |
| | wednesday | Indicator of availability of service for all Wednesdays in date range |
| | thursday | Indicator of availability of service for all Thurdays in date range |
| | friday | Indicator of availability of service for all Fridays in date range |
| | saturday | Indicator of availability of service for all Saturdays in date range |
| | sunday | Indicator of availability of service for all Sundays in date range |
| | start_date | Start date for the service given in the format YYYYMMDD |
| | end_date | End date for the service given in the format YYYYMMDD |
| trips.txt | route_id | See route.txt |
| | service_id | See calendar.txt |
| | trip_id | Unique identification of a trip in the network. |
| | | Note: a trip_id is associated with at most 1 route_id |
| stop_times.txt | trip_id | See trip.txt |
| | arrival_time | Arrival time of a train along a specific trip on a route |
| | departure_time | Departure time of a train along a specific trip on a route |
| | stop_id | See stops.txt |

# Appendix B

# London underground connections

Table containing the edge information for connections between London stations using the London Underground. The edges are assumed to be undirected.

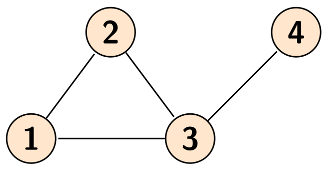
| **Station** | **Station** | **Time** |
|---|---|---|
| Amersham | Chalfont and Latimer | 3 |
| Brixton | Vauxhall | 4 |
| Chalfont and Latimer | Harrow-on-the-Hill | 20 |
| Ealing Broadway | Shepherdś Bush | 15 |
| Elephant & Castle | London Bridge | 3 |
| Farringdon | Moorgate | 4 |
| Finsbury Park | Seven Sisters | 3 |
| Greenford | Shepherdś Bush | 16 |
| Harrow-on-the-Hill | Farringdon | 31 |
| Highbury & Islington | Finsbury Park | 2 |
| Kentish Town | London Euston | 5 |
| London Liverpool Street | Stratford | 8 |
| London Bridge | Old Street | 5 |
| London Charing Cross | London Waterloo | 2 |
| London Euston | Charing Cross | 7 |
| London Euston | London Kings Cross | 1 |
| London Euston | London St Pancras International | 1 |
| London Kings Cross | Highbury & Islington | 2 |
| London Kings Cross | London Euston | 1 |
| London Liverpool Street | West Ham | 13 |
| London Paddington | London Victoria | 14 |
| London Paddington | Marylebone | 3 |
| London St Pancras International | Highbury & Islington | 2 |

| **Station** | **Station** | **Time** |
|---|---|---|
| London St Pancras International | London Euston | 1 |
| London Victoria | London Cannon Street | 12 |
| London Victoria | London Liverpool Street | 18 |
| London Victoria | London Euston | 33 |
| London Waterloo | Elephant & Castle | 4 |
| London Waterloo | Balham | 17 |
| London Waterloo | London Bridge | 2 |
| Old Street | London St Pancras International | 5 |
| Old Street | London Kings Cross | 5 |
| Marylebone | London Charing Cross | 9 |
| Moorgate | London Liverpool Street | 2 |
| Seven Sisters | Tottenham Hale | 2 |
| Shepherdś Bush | London Liverpool Street | 27 |
| South Ruislip | Greenford | 4 |
| Stratford | West Ham | 3 |
| Tottenham Hale | Walthamstow Central | 5 |
| Vauxhall | London Victoria | 28 |
| Wembley Central | London Paddington | 22 |
| West Ham | London Bridge | 13 |
| West Hampstead | London Waterloo | 14 |
| West Ruislip | South Ruislip | 3 |

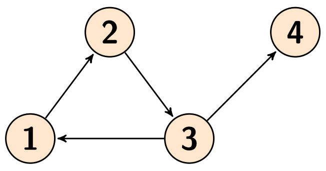
# Appendix C

# Monoplex network types

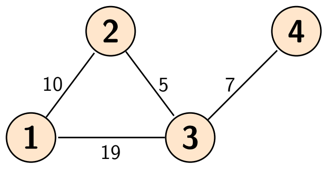
### Undirected



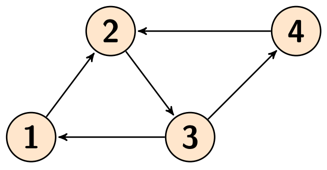
### Directed



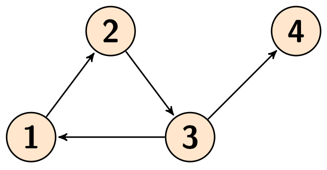
### Weighted (undirected)



### Strongly connected



### Weakly connected

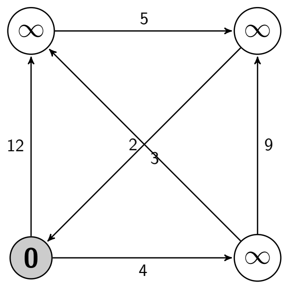


# Appendix D

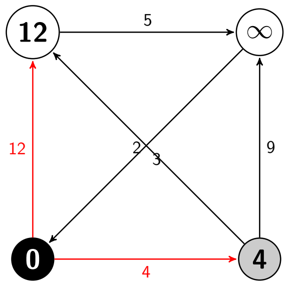
# Dijkstra's algorithm visualization

A visualization of Dijkstra's algorithm is provided for a monoplex network. White nodes are unvisited. Black nodes have been visited. A grey node is the current source node, red vertices give the current shortest paths in the network. The node values correspond to the current shortest distance from the source node.

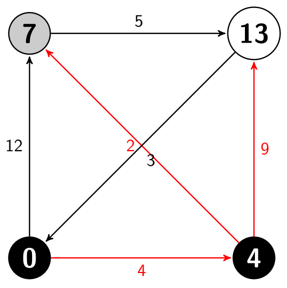
**Iteration I**



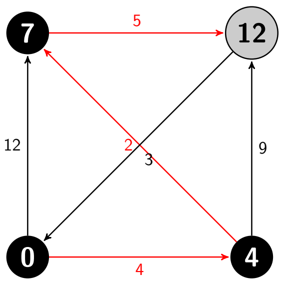
**Iteration II**



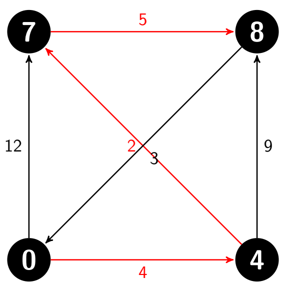
**Iteration III**



**Iteration IV**



**Iteration IV**



# Appendix E

# Dijkstra's algorithms

**Algorithm 1** Dijkstra's algorithm [12]

```
Input: Network $G$ and source node $s$
Output: Dictionary $d$ with the distance to the source node of all nodes in the
    network and a dictionary $a$ containing the ancestor information
 1: for each vertex $i$ in V do
 2:     $d(i) = \infty$      % initializes distance to source node
 3:     $a(i) = -1$      % initializes ancestor
 4: end for
 5: $d(s) = 0$      % initializes distance for source node
 6: $S = \emptyset$      % set of visited nodes
 7: $Q$=$V$      % set of unvisited nodes
 8: while $Q \neq \emptyset$ do
 8:     $u$ = EXTRACT-MIN(Q)      % extracts node with the smallest distance to
        the source node.
 8:     $S = S \cup \{u\}$
 9:     for each vertex $v$ in G.Adj[u] do
10:         if $d(v) > d(u) + w(u, v)$ then
10:             $d(v) = d(u) + w(u, v)$ $quad$
10:             $a(v) = u$
11:         end if
12:     end for
13: end while
14: return d, a
```

**Algorithm 2** Dijkstra's algorithm on underlying graph

**Input:** Network $G$ and source tuple $\gamma$

**Output:** Dictionary $d$ with the distance to the source tuple of all tuples in the network and a dictionary $a$ containing the ancestor information

```
 1: for each tuple $\kappa$ in $V_M$ do
 2:     $d(\kappa) = \infty$     % initializes distance to source node
 3:     $a(\kappa) = -1$     % initializes ancestor
 4: end for
 5: $d(\gamma) = 0$     % initializes distance for source tuple
 6: $S = \emptyset$     % set of visited tuples
 7: $Q$=$V_M$     % set of unvisited tuples
 8: while $Q \neq \emptyset$ do
 8:     ($\alpha$) = EXTRACT-MIN(Q)     % extracts tuple with the smallest distance
        to the source tuple.
 8:     $S = S \cup \{\alpha\}$
 9:     for each tuple $\beta \in$ G.Adj[$\alpha$] do
10:         if $d(\beta) > d(\alpha) + w(\alpha, \beta)$ then
10:             $d(\beta) = d(\alpha) + w(\alpha, \beta)$
10:             $a(\beta) = \alpha$
11:         end if
12:     end for
13: end while
14: return  d, a
```

**Algorithm 3** Adjusted Dijkstra's algorithm for underlying graph

```
Input: Network $G$ and source tuple $\gamma$
Output: A dictionary $\hat{d}$ with the distance from the aggregated tuple to all aggregated tuples in the network, a dictionary $\hat{a}$ containing the elementary layer(s) that provide the shortest paths, a dictionary $d$ with the distance to the source tuple(s) of all tuples in the network, and a dictionary $a$ containing the ancestor information
  for each tuple $\kappa$ in $V_M$ do
    $d(\kappa) = \infty$    % initializes distance to source node
    $a(\kappa) = -1$    % initializes ancestor
    $\hat{d}(\hat{\kappa}) = \infty$
    $\hat{a}(\hat{\kappa}) = -1$
  end for
  $d(\gamma) = 0$    % initializes distance for source tuple that can be mapped to $\hat{\gamma}$
  $\hat{d}(\hat{\gamma}) = 0$
  $M = \emptyset$    % set of visited tuples
  $U$=$V_M$    % set of unvisited tuples
  $\hat{M} = \emptyset$    % set of visited aggregated tuples
  $\hat{U}$=$\hat{V}_M$    % set of unvisited aggregated tuples
  while $\hat{U} \neq \emptyset$ do
    $\alpha$ = EXTRACT-MIN(Q)    % extracts tuple with the smallest distance to the source tuple.
    $S = S \cup \{\alpha\}$
    for each tuple $\beta \in$ G.Adj[$\alpha$] do
      if $d(\beta) > d(\alpha) + w(\alpha, \beta)$ then
        $d(\beta) = d(\alpha) + w(\alpha, \beta)$
        $a(\beta) = \alpha$
        if $d(\beta) < \hat{d}(\hat{\beta})$ then
          $\hat{d}(\hat{\beta}) = d(\beta)$    % $\hat{\beta}$ is the aggregated tuple
          $\hat{a}(\hat{\beta}) = \alpha$
        end if
      end if
    end for
  end while
  return $\hat{d}$, $\hat{a}$, $d$, $a$
```

# References

[1] T. Abrahamsson. Estimation of origin-destination matrices using traffic counta literature survey. In *Interim Report IR98-021, International institute for Applied System Analysis, Laxenburg*, pages 87–94. Computer Science Press, 1998.

[2] T. Abrahamsson. *Travel Behaviour Research: Updating the State of Play*, chapter Estimation of Origin-Destination Matrices Using Traffic Counts: An application to Stockholm, Sweden, pages 199–220. Elsevier, 1998.

[3] A. Barabasi, D.J. Watts, and M. Newman. *The Structure and Dynamics of Networks*. Princeton University Press, 2006.

[4] P.H.L. Bovy, M.C.J. Bliemer, and R. van Nes. Transportation modeling. `http://ocw.tudelft.nl/fileadmin/ocw/courses/TransportationandSpatialModelling/res00018/transportation4801.pdf`, August 2006. Accessed: 22 August 2014.

[5] M.J. Bruton. *Introduction to transportation planning*. Hutchinson Technical Education, 1970.

[6] J. Candia, M.C. Gonz'alez, P. Wang, T. Schoenharl, G. Madey, and A. Barab'asi. Unconvering individual and collective human dynamics from mobile phone records. *Journal of Physics A: Mathematical and Theoretical*, 41, 2008.

[7] E. Cascetta. Estimation of trip matrices from traffic counts and survey data: A generalized least squares estimator. *Transportation Research Part B: Methodological*, 18:289–299, August-October 1984.

[8] L. Castelli, R. Pesenti, and W. Ukovich. Scheduling multimodal transportation systems. *European Journal of Operational Research*, 155:603–615, June 2004.

[9] A. Ceder, B. Golany, and O. Tal. Creating bus timetables with maximal synchronization. *Transportation Research par A*, 35:913–928, 2001.

[10] C. Coffey, R. Nair, F. Pinelli, A. Pozdnoukhow, and F. Calabrese. Large-scale transit schedule coordination baed on journey planner requests. In *Annual Meeting of the Transportation Research Board*, pages 48–58, 2012.

[11] C. Coffey, R. Nair, F. Pinelli, A. Pozdnoukhow, and F. Calabrese. Missed connections: Quantifying and optimizing multi-modal interconnectivity in cities. In *IWCTS'12 Proceedings of the 5th ACM SIGSPATIAL International Workshop on Computational Transportation Science*, pages 26–32, 2012.

[12] T.H. Cormen, C.E. Leiserson, R.L. Rivest, and C. Stein. *Introduction to Algorithms*. McGraw-Hill Higher Education, 2nd edition, 2011.

[13] Google Developers. What is gtfs? `https://developers.google.com/transit/gtfs/`, 2012. Accessed: 27 July 2014.

[14] Transport for London. Plan a journey. `http://www.tfl.gov.uk/plan-a-journey/`. Accessed: 18 August 2014.

[15] Transport for London. Tube map. `http://www.tfl.gov.uk/cdn/static/cms/documents/standard-tube-map.pdf`, May 2014. Accessed: 18 August 2014.

[16] M.C. Gonzales, C.A. Hidalgo, and A. Barabasi. Understanding individual human mobility patterns. *Nature*, 453:779–782, 2008.

[17] W.E. Hart, J. Watson, and D.L. Woodruff. Pyomo: modeling and solving mathematical programs in python. *Mathematical Programming and Computation*, 3(3):219–260, 2011.

[18] D. Hibbs. What you need to know about lennon: an acorp briefing sheet. `http://www.acorp.uk.com/Assets%20intranet%20page/Briefing%20Sheets/12%20-%20lennon.pdf`, June 2010. Accessed: 27 July 2014.

[19] A. Higgins and E. Kozan. Modeling train delays in urban networks. *Transportation Science*, 32:346–357, 1998.

[20] J. Hughes. Proposal: remove “google” from the name of gtfs. `https://groups.google.com/forum/#!topic/gtfs-changes/ob_7MIOvOxU`, 2009. Accessed: 27 July 2014.

[21] N. Johnson. gbrail. `http://www.gbrail.info/gtfs`, 2014. Accessed: 7 July 2014.

[22] A. Kauppi, J. Wikström, B. Sandblad, and A. Andersson. Future train traffic control: control by replanning. *Cognition, Technology Work*, 8:50–56, 2006.

[23] M. Kivelä. Multilayer networks library [software — plexmath project webpage], note = Available at http://www.plexmath.eu,.

[24] M. Kivelä, A. Arenas, M. Barthelemy, J.P. Gleeson, Y. Moreno, and M.A. Porter. Multilayer networks. *Journal of Complex Networks*, 2:203–271, 2014.

[25] T. Kusakabe, T. Iryo, and Y. Asakura. Estimation method for railway passengers' train choice behavior with smart card transaction data. *Transportation*, 37, 2010.

[26] T. Li, E. van Heck, P. Vervest, J. Voskuilen, F. Hofker, and F. Jansma. Passenger travel behavior model in railway network simulation. *Simulation Conference, 2006. WSC 06. Proceedings of the Winter Simulation Conference*, pages 1380–1387, 2006.

[27] M. Liebchen, C. Schachtebeck, Schöbel A., S. Stiller, and A. Prigge. Computing delay resistant railway timetables. *Computers and Operations Reseach*, 37:857–868, 2010.

[28] M.G. McNally. *Handbook of Transport Modelling*, chapter The Four-step Model, pages 35–52. Elsevier Science, 2000.

[29] M.E.J. Newman. *Networks: An Introduction.* Oxford University Press, 2012.

[30] Office of Rail Regulation. Dataportal. http://dataportal.orr.gov.uk. Accessed: 3 September 2014.

[31] Office of Rail Regulation. Criteria and procedures for the approval of track access contracts. http://orr.gov.uk/__data/assets/pdf_file/0003/4818/ta_criteria_and_procedures.pdf, December 2011. Accessed: 12 August 2014.

[32] Office of Rail Regulation. Department for transport. http://orr.gov.uk/about-orr/who-we-work-with/government/department-for-transport, April 2014. Accessed: 12 August 2014.

[33] Office of Rail Regulation. Estimates of station usage. http://orr.gov.uk/statistics/published-stats/station-usage-estimates, April 2014. Accessed: 29 July 2014.

[34] Association of Train Operating Companies. Bussiness plan 2012/13-2014/15. http://www.atoc.org/clientfiles/files/ATOC%20Business%20Plan%20-%20Final.pdf. Accessed: 12 August 2014.

[35] Association of Train Operating Companies. Routeing guide data. http://data.atoc.org/routeing-guide, July 2014. Accessed: 14 July 2014.

[36] Department of Transport. The uk rail industry: A showcase of excellence. April 2014.

[37] R. Pant, J. Hall, S. Thacker, S. Barr, and D. Alderson. National scale risk analysis of interdependent infrastructure network failured due to extreme hazards, January 2014.

[38] Network Rail. Data feeds. http://www.networkrail.co.uk/data-feeds/. Accessed: 3 September 2014.

[39] Network Rail. Our routes. www.networkrail.co.uk/structure-and-governance/our-routes. Accessed: 18 August 2014.

[40] RSP. Rail industry data. `http://data.atoc.org/rail-industry-data`. Accessed: 3 September 2014.

[41] M. Ruan and J. Lin. An investigation of bus headway regularity and service performance in chicago bus transit system. `http://www.transportchicago.org/uploads/5/7/2/0/5720074/intelligentbus-ruanlin.pdf`. Accessed: 18 July 2014.

[42] General Algebraic Modeling System. Cbc. `http://www.gams.com/dd/docs/solvers/cbc.pdf`. Accessed: 19 July 2014.

[43] J. Törnquist. Railway traffic disturbance management: An experimental analysis of disturbance complexity, management objectives and limitations in planning horizon. *Transportation Research Part A: Policy and Practice*, 41:249–266, 2007.

[44] P. Vansteenwegen and D. van Oudheusden. Decreasing the passenger waiting time for an intercity rail network. *Transportation Research Part B: Methodological*, 41:478–492, May 2007.

[45] Mark Wardman. Public transport values of time. *Transport Policy*, 11(4):363 – 377, 2004.

[46] F. Zhao. Large-scale transit network optimization by minimizing user cost and transfers. *Journal of Public Transportation*, 9:107–129, 2006.