A Heuristic for Minimizing Transfers in Bus Routes Modeled with GTFS

Luca Vinci*

November 2024

1 Introduction

The **GTFS** model (**General Transit Feed Specification**) is a standard for representing public transport data. It is used to share information about schedules and routes for buses, trains, subways, and other public transport. Originally developed by Google, GTFS has become a widely adopted standard by many transportation agencies worldwide.

GTFS is organized as a set of CSV files compressed into a single archive, describing different aspects of the public transport service. Each file represents a specific table and has a particular purpose, as described below:

- **agency.txt**: contains information about the transit agency, such as its name, website, phone number, and time zone.
- **routes.txt**: lists all public transport routes, with details like the route name, type of vehicle (e.g., bus, train), and the general path.
- **trips.txt**: describes individual trips for each route and includes identifiers for the vehicle and the specific path.
- **stops.txt**: contains a list of stops with information such as name, geographic location (latitude and longitude), and optional details like accessibility.
- **stop_times.txt**: specifies the arrival and departure times for each stop along a route.
- **calendar.txt**: defines the days of the week and dates on which the service operates (e.g., Monday-Friday for weekday service or specific holiday dates).

*Master's degree in Computer Science from Ca' Foscari University of Venice.

- **calendar_dates.txt**: provides exceptions to the calendar, indicating changes in service on specific dates (e.g., service suspended during a holiday).
- **shapes.txt**: describes the physical path or route shape using a series of GPS coordinates, useful for plotting the exact route on a map.

1.1 Disadvantages of the GTFS model

There are some disadvantages to using GTFS for calculating **shortest paths** in public transport systems. Additionally, finding **route combinations that require the fewest transfers** is a complex problem, especially in large and intricate public transport networks. Some of the main reasons for this difficulty are related to the combinatorial nature of the solutions and the need for optimization across multiple variables, such as the number of transfers and the total travel time. Some key disadvantages and challenges include the following.

1.1.1 Static Data

The classic GTFS format is static; it contains information on scheduled times and routes but doesn't account for real-time changes like delays, route changes, breakdowns, or traffic conditions. Route calculations may lead to suboptimal results, as they do not reflect the current status of the service.

1.1.2 High Combinatorial Complexity

In public transport networks with many lines, stops, and possible transfers, the **number of possible routes** between a starting stop and a destination stop grows **exponentially**. Even finding routes with few transfers requires exploring **numerous combinations** of lines and stops. This combinatorial complexity makes it difficult to identify the optimal route quickly, especially if the total travel time is also being optimized.

1.1.3 Exploration of Multiple Transfers

When searching for a route with fewer transfers, there are often several options for switching from one line to another. Each transfer can occur at a specific stop, but the order and location of the transfers affect the final result. Even if some routes seem similar, the chosen transfer points can have a significant impact on travel time and convenience. Therefore, it's important to consider not only the number of transfers but also which transfers provide the best conditions, which increases the complexity of the calculation.

1.1.4 Need for Complex Graph Search Algorithms

The problem of minimizing transfers can be modeled as a shortest path problem on a graph, where the stops are nodes and the connections between stops are edges. However, unlike a simple graph, the weight of the edges can

depend on the type of transfer, wait times, and other factors. Search algorithms like Dijkstra or A* work well for finding the shortest path, but if transfer constraints are added, more advanced graph search algorithms may be needed, such as bidirectional search or k-shortest paths (which find multiple ordered routes to evaluate more options).

1.2 Implementation of a simple Heuristic for minimize change

As described earlier, GTFS presents intrinsic challenges. The GTFS format was not designed for directly handling complex calculations like minimizing transfers or calculating shortest paths, and it is also limited by its mostly static representation of data. To address this issue, a heuristic has been developed that, using GTFS data, allows for calculating routes with the minimum number of transfers needed to reach the destination. This heuristic, through a series of criteria and approximations, reduces the computational load of the algorithm by limiting the exploration to the most promising routes. The described approach represents a balance between accuracy and complexity, allowing for efficient management of optimized route calculations.

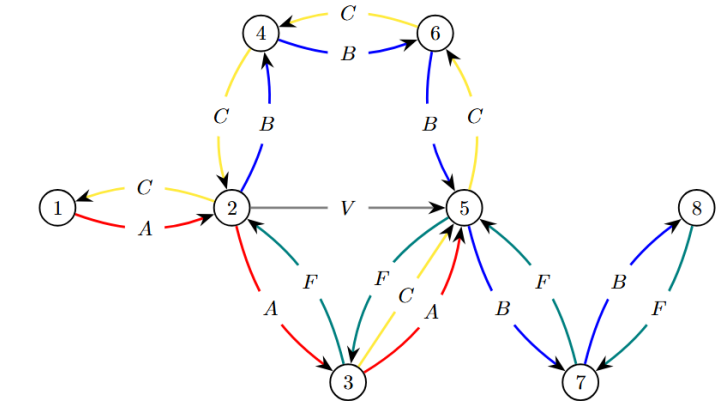
For the development of this approach, open data provided by Actv (the public transport company of the city of Venice) for urban bus routes were used. The purpose of this project is purely personal, but given the challenges related to GTFS, it could serve as a starting point for more sophisticated and complex algorithms for anyone interested in exploring the topic.

2 Implementation

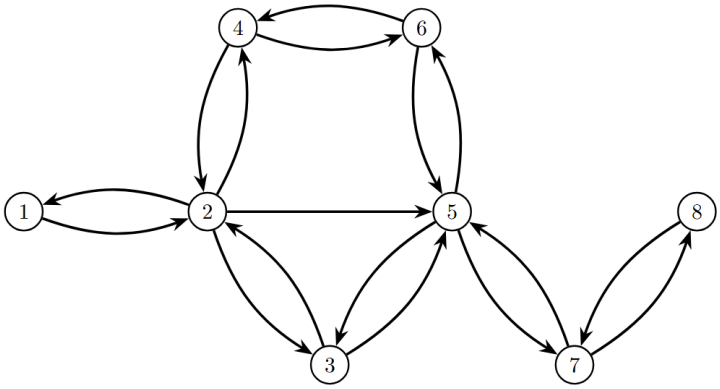
The main idea is to model the network of public transport stops and routes as a **directed graph**, where the **stops** represent the **nodes** and the possible **connections** between stops, enabled by one or more routes, form the edges (Fig. 2b). In this context, each edge between two nodes (**stops**) indicates that there is at least one route that directly connects the two stops, but it does not specify which particular route.

The choice to represent the edges as "**anonymous**" is functional for isolating the network's topology without delving into the details of specific routes at this initial stage. This representation allows for easy analysis of the network's structural characteristics, such as the **centrality of stops** (useful for understanding the importance of a stop in relation to overall traffic flow) and **connected components** (which identify clusters of interconnected stops within the network, helping to locate isolated areas or densely connected zones).

The idea behind the heuristic is to use a **step-by-step approach** to find the optimal route with the **fewest transfers**. To do this, we start by calculating



(a)



(b)

Figure 1: (a) Bus route network (b) Graph representing only the connections

the **shortest path** from the source node to the destination node, based only on the topology of the connected nodes, without considering the specific routes at this stage. Then, once the optimal sequence of nodes is identified, we use these nodes as "**anchor points**" or "**hooks**" to determine the exact routes or combinations of routes needed to cover each segment of the path.

This approach allows for a more efficient solution in terms of computation, as the algorithm focuses only on the relevant line combinations needed to connect the nodes of the optimal path. In other words, the heuristic uses the **structural shortest path** as a framework on which to "**attach**" the actual routes, reducing the number of combinations to consider and minimizing the number of transfers. This process makes it easier to calculate the sequence of routes to reach the destination in a more direct and user-friendly way, without exploring alternative paths that would not offer significant improvements in terms of transfers or travel time.

Starting from the resulting shortest path, which will be made up of the so-called "**hooks**" stops, an additional graph will be built. In this new graph, the nodes will represent the "hooks" of the shortest path, and the edges will be added according to the procedure described below:

1. Take the first node i_1 of the path, and for each subsequent node, check if there is at least one route that connects the two nodes (i.e., the corresponding stops). If this condition is met, add an edge between these nodes in the new graph, labeling it with the name of the route(s) that connect the two nodes.

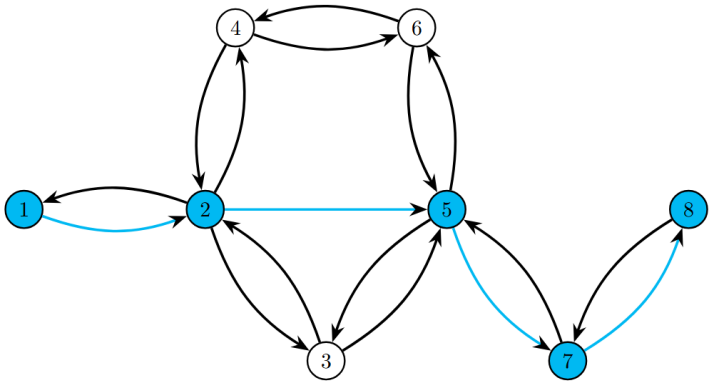
2. Repeat the same process for each subsequent node in the path $i_2, i_3, ..., i_{n-1}$, until reaching the second-to-last node of the shortest path.

In this way, we obtain a graph that represents the existing connections between the nodes of the shortest path, with edges labeled by the actual available routes, providing a clear and concise representation of the direct connections.

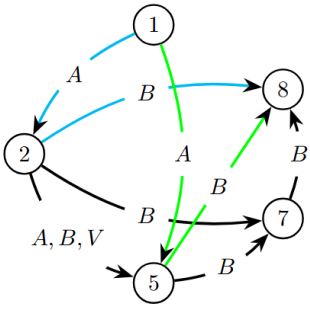
Illustrative Example: Let's consider a shortest path between nodes 1 and 5, represented as $P = \{1, 2, 3, 4, 5\}$. To build the graph from this path, we will proceed as follows:

- Start from node 1 and check for routes between:
  - 1→2
  - 1→3
  - 1→4
  - 1→5

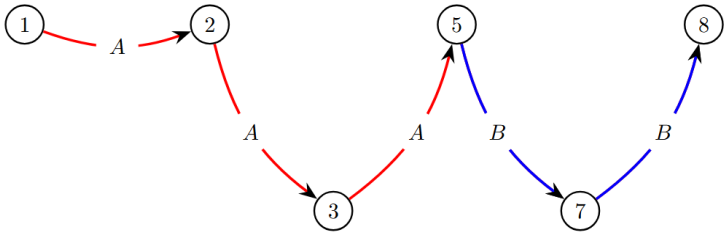
  For each verified connection, add the corresponding edge, labeling it with the route(s) that connect the stops.



(a)



(b)



(c)

Figure 2: (a) Shortest path from 1 to 8 (b) STG, colored edges represent the two shortest paths between 1 and 8 (c) Optimized path result

- Then move to node 2 and check for routes between:
  - 2→3
  - 2→4
  - 2→5

  Again, add edges only for the verified direct connections, with the respective route labels.
- Repeat the process for each node in the path, continuing with nodes 3 and 4, until you verify the direct connections from the last node 4 to 5.

In this way, we obtain a graph structured only around the nodes of the shortest path, which faithfully represents the routes connecting the selected "hooks." We will call this graph **STG** (**Shortest Transfer Graph**). Next, we will proceed to calculate the **shortest path** between the initially specified **source** node and the **destination** node. This shortest path, built within the transfer graph (STG), represents the route that **minimizes the number of transfers needed between transport routes**. Thanks to the structure of STG, a shortest path of **length equal to 1** indicates the **existence** of a **direct route** connecting the two stops, with no need for any transfer. This case represents the optimal situation, where the journey is simple and direct.

If no single route directly connects the stops, the shortest path obtained will reflect the solution with the **fewest possible transfers**. In this way, we get a journey that, while including more routes, keeps interruptions and switches from one route to another to a minimum, ensuring greater efficiency for the end user. In summary, the shortest path calculated in STG provides the optimal route in terms of simplicity and continuity, favoring direct connections where possible and reducing transfers when necessary.

3 What's next

I decided to create this report to document and share the approach I found, with the aim of providing a starting point for anyone who wants to explore, in the future, modeling public transportation applications based on GTFS. The complexities and challenges of GTFS are numerous, and the proposed heuristic could serve as a useful tool to address some of the main issues related to managing public transportation data.

The work presented here can be further expanded and improved. A possible future development could involve adding weights to the edges in the STG graph to allow for optimization based on the schedules of the transport services, enhancing the model with a temporal component to calculate the most efficient routes based on actual travel times.

Currently, the sample application available on GitHub[1] is developed in Java, using the Spring framework, and already implements a filtering feature based on schedules: as a final step, the application searches for the transport routes that best match the specified departure time, making it easier to select the most appropriate routes. Additionally, an automated procedure has been created for loading GTFS data files (*.txt*) into a database, simplifying data management through a pipeline for importing and handling data. For more details on using and the features of the application, please refer to the documentation in the Readme file on GitHub.

[1] https://github.com/luckyluke98/BusFinder