Programming Internship: ARC Flag & Trip-Based Public Transit Routing

PATRICK STEIL, Heidelberg University, Germany

In this internship I developed and implemented the basic building blocks I need for my bachelor thesis, which is about reducing the query times of the trip-based public transit routing algorithm (TB) [10] using the ARC Flags [7] technique. This is a new approach, because the speedup technique is so far only known for road graphs, and not on time-dependent networks. ARC Flags is about storing information of “shortest” paths at edges, and then being able to exclude edges during a query, thus reducing the search space. In the TB algorithm, however, we are not looking for “shortest” paths, but for pareto optimal paths with respect to arrival time and number of transfers.

1 INTRODUCTION

The 9 € ticket in Germany in the months of June to August has increased the rush for public transport, with around 21 million tickets sold so far[1]. However, this means that more and more users are searching for their journeys with the help of services such as Google Maps[2] or Deutsche Bahn[3]. In order for these to meet the high demand, efficient and fast algorithms have to work in the background. In addition, these algorithms have to solve various optimization problems, e.g. find routes that minimize both arrival time and number of transfers.

2 PRELIMINARIES

The following is about time-dependent networks, especially public transport networks. Such a network is defined with the help of stops, routes, trips and transfers. Stops are physical locations where users get on and off trains. Transfers are ways for users to move between stops, e.g., by walking or using rental bicycles. A route consists of a sequence of stops, trips are vehicles that operate along the route at a certain time. Trips of a route cannot overtake each other. So, for example, the subway line 1 in Paris is a route, but the line 1 train leaving “La Défense” at 15:56 towards “Château de Vincennes” is a trip.
An efficient speedup technique on road networks is ARC Flags [7], which is based on the idea that one can divide nodes into $k \in \mathbb{N}$ roughly equal blocks and then exclude “unnecessary” edges during the query. More precisely, if the target node is in block j, one only looks at edges that are part of a shortest path leading to a target in block j. See Figure 1 as an example. The traditional way to compute ARC Flags on a graph is by computing reverse shortest path trees from all the *boundary nodes* [4]. We adopt this technique to the trip-based algorithm and adapt the flag computation according to our optimization problem. More on the precompuation in my bachelor thesis.

3 RELATED WORK

Many different approaches exist to solve the various routing problems. One approach is the Connection Scan Algorithm [3], which in its basic variant only minimizes the arrival time. The algorithm is based on storing all connections in an array sorted by departure time and iterating over all connections during the query, computing the minimum arrival time. Very cache-efficient, but

[1]last checked 03.08.2022 https://www.bundesregierung.de/breg-de/aktuelles/faq-9-euro-ticket-2028756
[2]https://www.google.de/maps
[3]https://www.bahn.de/
[4]A node is called *boundary node* if it has neighbors in other blocks

Author’s address: Patrick Steil, patrick.steil@stud.uni-heidelberg.de, Heidelberg University, Im Neuenheimer Feld 205, Heidelberg, Baden-Württemberg, Germany, 69120.

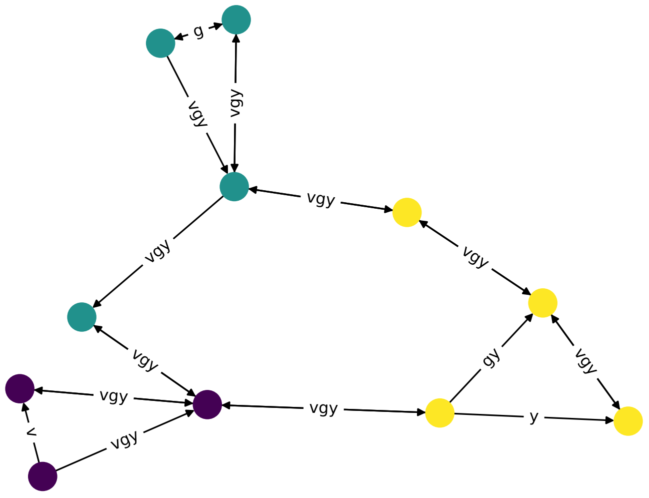

Fig. 1. Example graph to demonstrate the ARC Flags technique. Nodes are partitioned into $k = 3$ blocks, denoted by their color (**g**reen, **y**ellow and **v**iolet). The characters on the edges represent the flags set to *true*. Meaning that if an edge has the **y** flag set to true, the edge is part of a shortest path that leads to a yellow node.

on large networks many “unnecessary” connections are scanned. Therefore, this algorithm is very effective for smaller networks, such as those of a transport association like Rhein-Neckar-Verbund[5]. Another approach is RAPTOR [2], which works in rounds and iterates over routes in a breadth-first manner and thus also computes pareto optimal solutions. Advantages of RAPTOR are e.g. the possibility of parallelization and the absence of a precomputation. In addition, two graph-based models exist, a *time-expanded* and *time-dependent* model. The idea of the *time-expanded* model is to introduce nodes for each event (e.g. boarding or changing). Edges connect stops either because they belong to the same trip or because transfers are possible at these events. Despite this simple structure, speedup techniques, which are known from road networks, are not effective [1]. Moreover, graphs of this type become very large. The *time-dependent* approach, on the other hand, models nodes as stops and edges as connections between them. Weights of edges, on the other hand, are piecewise linear functions of departure to arrival times. On this type of graph, Dijkstra variants can solve multi-criteria problems, see e.g. [5] or [9].

3.1 Trip-Based Algorithm

The TB Algorithm works on a *time-expanded* graph, so nodes (so called *stop-events*) represent stops of a trip at a certain time and edges are transfers. A query, given start, destination and departure time, works like a breadth first search. The algorithms maintains a list of journeys (tuple of arrivaltime and number of transfers) and a queue with trip-segments. After having collected all reachable trips from the start w.r.t. the departure time, the algorithm checks for every trip t inside the queue if the target is reachable or not. It then updates the list of journeys accordingly and proceeds to insert all trips t' reachable from t into the queue. By design, each iteration increases

[5]https://www.rnv-online.de/fahrtinfo/

the number of transfers and thus, we find the minimal arrival time for each transfer (collected in the list of journeys).

4 OUTLINE

This section provides an overview of this document. Now that the general problem has been addressed, we will first talk about the data format and the framework used, along with the changes I made during the internship. Finally, it is about the datasets collected for the bachelor thesis.

5 GTFS FORMAT

filename	**overview**
agency.txt	Transit agencies with service represented in this dataset.
stops.txt	Stops where vehicles pick up or drop off riders. Also defines stations and station entrances.
routes.txt	Transit routes. A route is a group of trips that are displayed to riders as a single service.
trips.txt	Trips for each route. A trip is a sequence of two or more stops that occur during a specific time period.
stop_times.txt	Times that a vehicle arrives at and departs from stops for each trip.
calendar.txt	Service dates specified using a weekly schedule with start and end dates. This file is required unless all dates of service are defined in calendar_dates.txt.
calendar_dates.txt	Exceptions for the services defined in the calendar.txt. If calendar.txt is omitted, then calendar_dates.txt is required and must contain all dates of service.
feed_info.txt	Dataset metadata, including publisher, version, and expiration information.

Table 1. Overview of required files for a valid GTFS dataset. This table is adapted from https://developers.google.com/transit/gtfs/reference#dataset_files

Since all datasets used in my project are in GTFS format, I will now go into more detail about the **G**eneral **T**ransit **F**eed **S**pecification. A dataset consists of several *txt* files, but they fulfill the *csv* format, thus they can be read using a **c**omma **s**eparated **v**alue reader. Based on the relational database model, there are two types of files: **entities** and **relations**. **Entities** include, for example, *stops.txt*, *routes.txt* and *trips.txt*, whereas files such as *transfers.txt* and *stop_times.txt* describe relationships between objects. In *stop_times.txt* the departures of all trips are listed, i.e. one line contains the stop, arrival time, departure time and the corresponding trip. In addition to the timetable, there is also additional information on line colors, headsigns, station entrances, handicapped-accessibility, line shapes and many more. See Table 1 and Figure 2 for more detail.

Generally, transport associations offer their timetables in GTFS form, since GTFS can be used as a de-facto standard by almost all travel planners [6]. Such datasets are not only

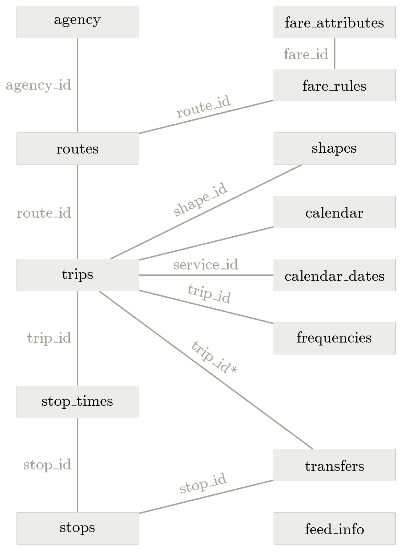

Fig. 2. diagram of GTFS - foreign keys are written on the connections (see https://en.wikipedia.org/wiki/General_Transit_Feed_Specification)

[6] https://developers.google.com/transit/gtfs

used for journey planning, but also for accessibility research [4]. Note, not only GTFS is a well known format for timetables, but also “Hafas Rohdaten Format” HRDF[7].

6 ULTRA - FRAMEWORK

The public Github repository “ULTRA” [8], developed by the Algorithmics Group of the Institute of Theoretical Informatics at the Karlsruhe Institute of Technology, provides implementations for several route planning algorithms, including CSA, RAPTOR and TB. Also included are helpful methods, such as reading and converting GTFS datasets into custom graph representations.

6.1 *stops.txt* modification

The framework works as follows: One reads GTFS data, converts it into the so-called intermediate format. This format can be used to build the RAPTOR format as well as the CSA format. The trip-based data is then calculated from the RAPTOR data. The precomputation of the ARC Flags is done on the trip-based data, and for that the individual classes had to be extended. Not only “Data” classes have been adapted, but also Graph classes, since own implemented graphs are used in the framework. One change, for example, was the addition of an edge attribute “ARC Flags”, a vector of boolean to store the flags. The information of the partitions of the stops is stored in the slightly modified *stops.txt* file in the GTFS folder (see Table 2).

stop_id	**stop_name**	**stop_lat**	**stop_lon**	**partition**
ic_ice_1006	“Aachen Hbf”	50.7678	6.0915	3
ic_ice_1568	“Aachen Hbf”	50.7678	6.0915	15
ic_ice_847	“Augsburg Hbf”	48.3654	10.8856	0
ic_ice_1205	“Berlin Hbf”	52.5256	13.3695	12
ic_ice_1502	“Berlin Hbf”	52.5256	13.3695	6
ic_ice_406	“Berlin Hbf”	52.5256	13.3695	4

Table 2. example for the “extended” *stops.txt* format for GTFS. One line of this file gives details about a track of a stop, here the information is the unique ID, the name of the stop, the coordinates and (what was added) the id of the partition block in which the stop is located. This is an excerpt from the “IC/ICE” dataset of https://gtfs.de/

The GTFS format models stops by assigning different entries to each stop in the *stops.txt* file, namely one entry for each track. This is an important detail, because not geographical stops / locations are partitioned, but rather tracks.
I extended the “ULTRA” framework so that when GTFS data is loaded, the partitions are read and passed to the relevant objects. See the overview scheme in Figure 3. The result of the partitioning e.g. done by KaHIP [9], is a partition file (see section 7). It is readable by the GFTS data class and can update the *stops.txt*. The idea behind this is to perform partitioning outside the framework.

6.2 METIS Graph

Fundamental for the partitioning is the so-called “METIS Graph”, which can be generated from the GTFS data. Stops are modeled as nodes and connections (both trips and transfers) are edges. A user can also specify how edges and nodes should be weighted. There are the following options that can be combined together:

[7]https://opentransportdata.swiss/de/cookbook/hafas-rohdaten-format-hrdf/
[8]https://github.com/kit-algo/ULTRA
[9]https://github.com/KaHIP/KaHIP

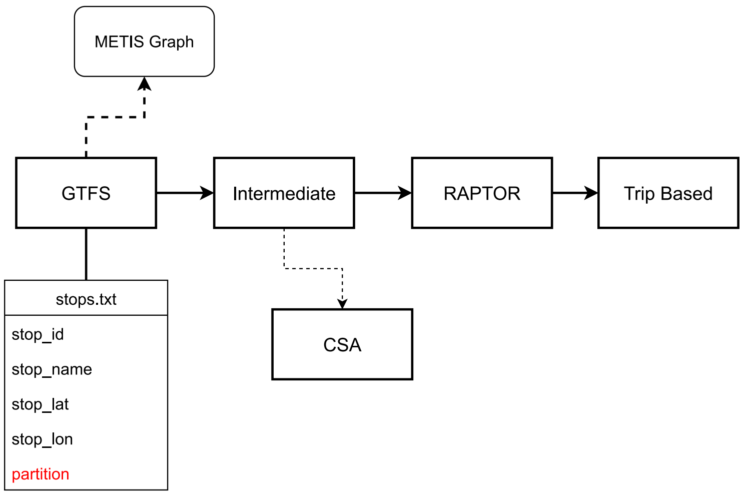

Fig. 3. Overview of the modified scheme in the “ULTRA” framework

- TRIP_WEIGHTED: one edge is created for each trip between two stops (edge weights correspond to the number of trips)
- TRANSFER_WEIGHTED: an edge is also created for each transfer
- NODE_TRIPS_WEIGHTED: nodes are weighted by the number of trips that pass through the station

The resulting METIS graph can be saved as a file, which can be read by KaHIP. The format is the same as in the 10th DIMACS Challenge on Graph Clustering and Partitioning. A graph with n nodes and m edges is represented in a text file with $n + 1$ lines (except comments starting with %). **Important**: edges are undirected, therefore (u, v) and (v, u) are only counted once, not twice. The first line specifies the number of nodes, the number of edges and the type of the graph. The type can be omitted, if the graph is unweighted, otherwise the following numbers are options:

1 edge weights
10 node weights
11 edge & node weights

One example to encode the information for one node using type 11 is the following:
a node n with node weight c and neighbours $v_j,\ \ j \in \mathbb{N}$ with edge weights $w_j,\ \ j \in \mathbb{N}$ will be represented as follows:

$$c\ v_1\ w_1\ v_2\ w_2\ \ldots$$

For more information: See the KaHIP manual.
Another way to export the METIS graph is in the form of GraphML format. This format is better known and can be read by many libraries, such as *networkx* [10] in Python. It can be used to create plots like the following Figure 4.

[10] https://networkx.org/

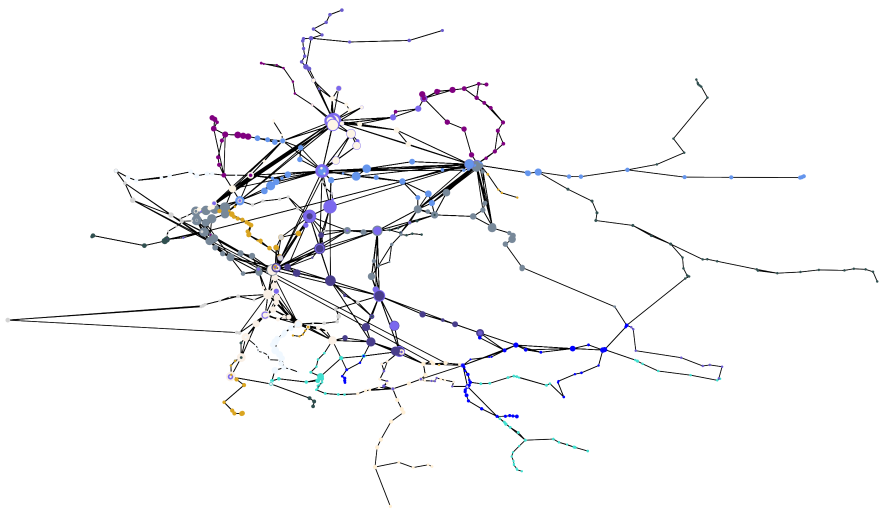

Fig. 4. IC/ICE dataset plotted using *networkx*, different colours denote different partitions

7 PARTITIONING

As mentioned in the section on ARC Flags, for a graph we want to find $k \in \mathbb{N}$ blocks of approximately equal size. In addition, we also try to minimize the number of edges that run between blocks (so-called “cut edges”). All this can be computed using partition algorithms, such as KaHIP [11]. KaHIP is a set of opensource graph partitioning algorithms developed by Sanders and Schulz [8], [6]. It has an advantage to use such a black box algorithm to calculate the partitions of the ARC Flags, because one can tweak parameters of the algorithm and compare the results and thus find well-suited partitions very easily. The result of a partition of a graph $G = (V, E)$ by KaHIP into k partitions is a file with the following format: line $j \in [1, |V|]$ contains the block ID $l \in [0, k-1]$ for the node j.

8 DATASETS

Another important preparation for the bachelor thesis was the accumulation of large diverse data sets. “Diverse” in this context means the size of the data set, as well as the density of stop locations and the mix of local and long-distance transit. One problem with the datasets used in the original paper [10] is that the underlying GTFS datasets are no longer available. However, these files (including the information about the stations) are essential for the METIS graph, thus for the partitioning. That is why I collected the following datasets for this project (see Table 3).

[11]https://github.com/KaHIP/KaHIP

dataset	stops	routes	trips	local	long-distance
Rhein-Neckar-Verbund	2.080	1.078	15.843	●	
IC/ICE	1.599	2.231	2.877		●
Karlsruhe	4.014	2.929	50.970	●	
S/RE	14.178	18.796	67.408	○	○
Paris	41.957	12.877	320.407	●	
Switzerland	38.156	69.362	706.466	●	●
Germany	663.875	573.851	3.338.502	●	●

Table 3. An overview of the data sets with which the experiments are performed. Note: S/RE trains (*S-Bahn and Regional Express trains*) are cross-regional trains, which stop not only at large stations, but also at smaller regional ones. Accordingly, I classify the dataset as a mixture of both local and long-distance trains (denoted by ○).

9 ACKNOWLEDGEMENTS

I would like to take this opportunity to thank Jonas Sauer for giving me the idea for this project and helping me throughout the whole time as well as Patrick Brosi for providing us with the IC/ICE, S/RE and Germany data sets (from his website https://gtfs.de/).

REFERENCES

[1] Annabell Berger, Daniel Delling, Andreas Gebhardt, and Matthias Müller-Hannemann. Accelerating Time-Dependent Multi-Criteria Timetable Information is Harder Than Expected. In Jens Clausen and Gabriele Di Stefano, editors, *9th Workshop on Algorithmic Approaches for Transportation Modeling, Optimization, and Systems (ATMOS'09)*, volume 12 of *OpenAccess Series in Informatics (OASIcs)*, Dagstuhl, Germany, 2009. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik. ISBN 978-3-939897-11-8. doi: 10.4230/OASIcs.ATMOS.2009.2148. URL http://drops.dagstuhl.de/opus/volltexte/2009/2148.

[2] Daniel Delling, Thomas Pajor, and Renato Werneck. Round-based public transit routing. In *Proceedings of the 14th Meeting on Algorithm Engineering and Experiments (ALENEX'12)*. Society for Industrial and Applied Mathematics, January 2012. URL https://www.microsoft.com/en-us/research/publication/round-based-public-transit-routing/.

[3] Julian Dibbelt, Thomas Pajor, Ben Strasser, and Dorothea Wagner. Connection scan algorithm. *ACM J. Exp. Algorithmics*, 23, 2018. doi: 10.1145/3274661. URL https://doi.org/10.1145/3274661.

[4] Steven Farber, Melinda Z. Morang, and Michael J. Widener. Temporal variability in transit-based accessibility to supermarkets. *Applied Geography*, 53:149–159, 2014. ISSN 0143-6228. doi: https://doi.org/10.1016/j.apgeog.2014.06.012. URL https://www.sciencedirect.com/science/article/pii/S0143622814001283.

[5] Pierre Hansen. Bicriterion path problems. In Günter Fandel and Tomas Gal, editors, *Multiple Criteria Decision Making Theory and Application*, pages 109–127, Berlin, Heidelberg, 1980. Springer Berlin Heidelberg. ISBN 978-3-642-48782-8.

[6] Henning Meyerhenke, Peter Sanders, and Christian Schulz. Parallel graph partitioning for complex networks. *IEEE Trans. Parallel Distrib. Syst.*, 28(9):2625–2638, 2017. doi: 10.1109/TPDS.2017.2671868. URL https://doi.org/10.1109/TPDS.2017.2671868.

[7] Rolf H. Möhring, Heiko Schilling, Birk Schütz, Dorothea Wagner, and Thomas Willhalm. Partitioning graphs to speedup dijkstra's algorithm. *ACM J. Exp. Algorithmics*, 11:2.8–es, feb 2007. ISSN 1084-6654. doi: 10.1145/1187436.1216585. URL https://doi.org/10.1145/1187436.1216585.

[8] Peter Sanders and Christian Schulz. Think locally, act globally: Highly balanced graph partitioning. In *Experimental Algorithms, 12th International Symposium, SEA 2013, Rome, Italy, June 5-7, 2013. Proceedings*, volume 7933, pages 164–175. Springer, 2013.

[9] Gerth Stølting Brodal and Riko Jacob. Time-dependent networks as models to achieve fast exact time-table queries. *Electronic Notes in Theoretical Computer Science*, 92:3–15, 2004. ISSN 1571-0661. doi: https://doi.org/10.1016/j.entcs.2003.12.019. URL https://www.sciencedirect.com/science/article/pii/S1571066104000040. Proceedings of ATMOS Workshop 2003.

[10] Sascha Witt. Trip-based public transit routing. *CoRR*, abs/1504.07149, 2015. URL http://arxiv.org/abs/1504.07149.