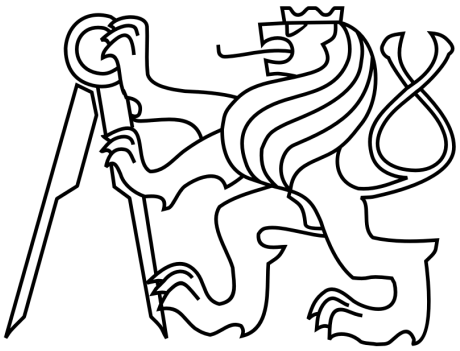
CZECH TECHNICAL UNIVERSITY IN PRAGUE

Faculty of Transportation Sciences
Department of Transport Telematics



MASTER'S THESIS

Bc. Tomáš Těthal

Trip Connection Searching Algorithm

Supervisor: Ing. Petr Bureš, PhD.

Prague 2014

I have no relevant reason against using this schoolwork in the sense of § 60 of Act No121/2000 concerning the authorial law.

I declare that I accomplished my final thesis by myself and I named all the sources I used in accordance with the guideline about the ethical rules during preparation of University final thesis.

In Prague Name

Abstrakt

Autor: Bc. Tomáš Těthal
Název: Algoritmus pro vyhledávání dopravního spojení
Univerzita: České vysoké učení technické v Praze, Fakulta dopravní
Rok: 2014

Hlavním účelem této diplomové práce bylo vytvoření přehledu algoritmů pro vyhledávání nejkratší cesty v časově závislých dopravních sítích a implementování takovéhoto algoritmu pro rozsáhlé sítě. První část práce se soustřeďuje na obecný popis dopravních sítí a základní modelovací techniky, problémy v těchto sítích, dříve představenými algoritmy a technikami pro redukci prohledávaného prostoru. Vybrané algoritmy jsou popsány ve vetším detailu včetně význačných vlastností a porovnány. Druhá část práce se zabývá implementací algritmu pro hledání nejkratší cesty. Obsahuje výběr datového formátu, import dat do databáze včetně vytvoření routovatelných tabulek. Algoritmus byl implementován ve webové aplikace s pohodlným uživatelským rozhraním a byl testován na síti Pražské integrované dopravy.

Klíčová slova: Nejkratší cesta, Časově-závislá síť, Jízdní řád, TBSP, TBHP, TBA*

Abstract

Author: Bc. Tomáš Těthal
Title: Trip Connection Searching Algorithm
University: Czech Technical University in Prague, Faculty of Transportation Sciences
Year: 2014

The main aim of this master's thesis is to provide an overview of the shortest-path searching algorithms in time-dependent networks, and the implementation of such an algorithm in large networks. The first part of the thesis focuses on the transit network overview and basic modeling approaches, problems in transit networks, previously proposed algorithms, and search of space reduction techniques. Selected algorithms are described in detail pointing out the main features and compared to each other. The second part of the thesis focuses on the implementation of a shortest-path algorithm which includes data selection, database import and creation of a routable database. The algorithm was implemented in a web application with a convenient user interface and was tested on Prague Integrated Transport System network .

Keywords: Shortest path, Time-dependent network, Timetable, TBSP, TBHP, TBA*

Contents

Introduction

In public transportation, the main goals are to use the network efficiently and to meet the passenger demand. The schedule-based transportation network unlike road network has many more aspects to be taken into account such as a timetable, vehicle capacity and class, transfers, etc. The passenger demand is to reach the final destination as early and comfortably as possible in the desired time or time interval with a minimum number of transfers. For that purpose, algorithms for the shortest-path search in the time domain had to be developed.

The problem studied in this master's thesis deals with the shortest-path search in time-dependent network where the passenger need is to find the shortest route between two stations within the network. The thesis aims to evaluate the current state in the field of transit assignment, and to propose a new algorithm for the shortest-path search in time-dependent networks which can improve the passenger convenience and save time.

The thesis is divided into two main parts. The first part includes chapters 1, 2, 3 and 4, is mostly theoretical and describes the prerequisites for routing on time-dependent networks. It consists of a basic description of the transit networks and network modeling approaches, states problems that need be solved, examines and compares previously introduced algorithms, and describes speedup and search space reduction techniques for such algorithms. The goal is to provide a theoretical foundation for the following algorithm implementation including examples and discussion of the effectiveness of the particular approaches.

The second part of the thesis represented by chapter 5 deals with the implementation of a shortest path algorithm using the Prague Integrated Transport System data and providing an efficient way for the best route selection. The algorithm development consists of two parts - the data preparation part and the application part which can be deployed on a web server in order to achieve a convenient usage and testing environment.

1. Motivation and Previous Work

In my previous work, in my bachelor thesis, I focused on the issue of shortest path searching algorithms for road networks. A road network is a set of edges interconnecting nodes with a number of properties where some of them are used for the shortest path search. A shortest path in a road network is a route from point A to point B with the minimum length, traveling time, or the combination of both. The road network can be denoted as $G = (N, E)$ where N is a set of nodes and E is a set of edges interconnecting nodes. All edges e from the set E represent actual physical road segments with the following set of basic properties which are required for the shortest path search 1.1.

property	data type	example
edge ID	integer	855456
start node ID	integer	9548521
end node ID	integer	5865415
length km	float	1.125
length s	integer	62
max speed	integer	90
reverse length km	float	1.125
reverse length s	integer	62
reverse max speed	integer	90

Table 1.1: Road network edge properties

Additional properties can be required by more advanced algorithms. Network nodes N are points interconnecting edges or terminal points. A node can be for example an intersection or a road division, and its properties are listed in 1.2.

property	data type	example
node ID	integer	5565845
delay s	integer	3
latitude	float	52.156
longitude	float	14.456

Table 1.2: Road network node properties

The most common road networks where no additional data inputs are present are called *time independent*. It means that properties of the edges or nodes are not changing in time and search at any time of the day, week, etc. will therefore lead to the same shortest path. A shortest path query can be denoted as $Q = (n_1, n_2)$ where n_1 is the route origin and n_2 is the route destination. A shortest path search can be done in many different ways by carrying out different searches throughout the network. The result of the query (shortest path) is usually denoted as an ordered list of edges representing the resulting path. A shortest path can look like as pictured in the figure 1.1.

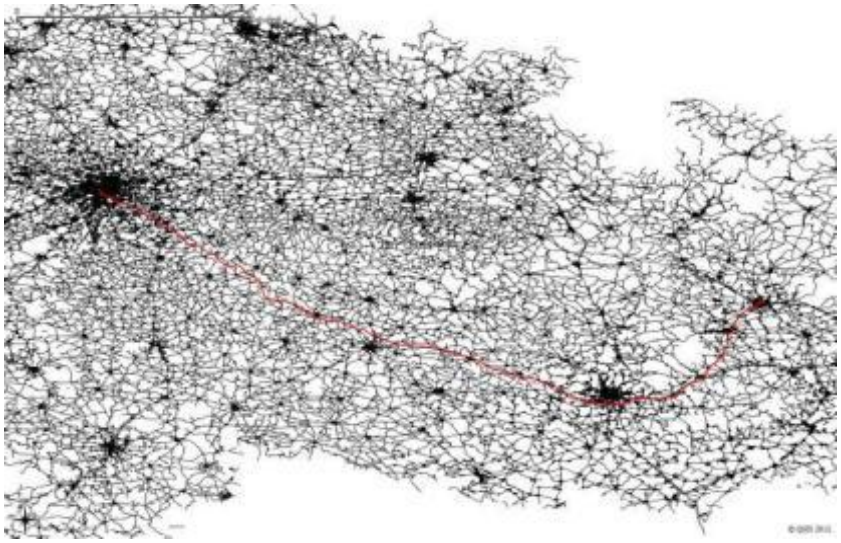


Figure 1.1: An example of a shortest path on a road network

In some cases, properties of nodes and edges can vary in time. In *dynamic networks*, changes can be caused by congestion, accidents, etc.

Properties of route segments can depend on the day of the week (Monday, Friday, Sunday) or time of the day (morning peak, afternoon peak). Edge properties can be statistically evaluated based on the historical data.

In this thesis I deal with *time − dependent* networks (also called *schedule − based* or *transit* networks) for public transportation systems. The main difference in comparison with the road networks is the time-dependency where every edge can be accessed only at a certain point of time. The network can be described as a set of nodes and edges with a time dependency.

In time-dependent networks an edge does not necessarily have to match the physical route as in the road networks. A set of edges from point A to point C can be considered as one physical edge.
Figure 1.2 denotes a physical model of a time-dependent network.

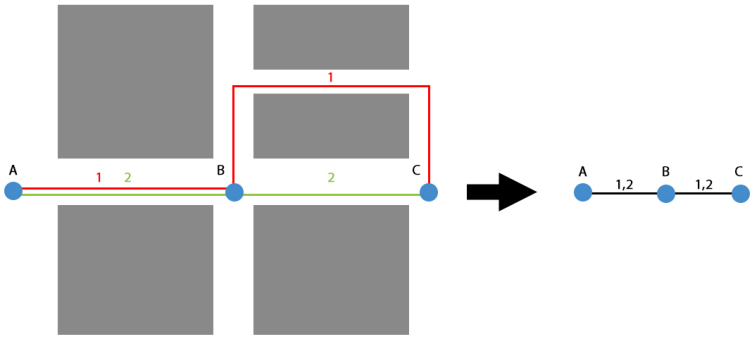


Figure 1.2: Difference between the physical network and the network model

Transit network consists of a large amount of information because the passage through an edge is possible in multiple time points. The information about connections is stored in a timetable database.

For example, a bus line operates from point A to point B on the average 6 times per hour which is 144 times per day. In case of this particular edge, the graph size would be therefore 144 times larger than in the case of road networks. The basic ideas for shortest path searching algorithms in time-dependent networks are based on algorithms for road networks. However, there is a higher demand for data preprocessing and a time-dependent property in case of a search in time-dependent networks which is also the main difference.

Input into the time-dependent shortest-path algorithm consist of several components. It includes transit nodes (stations), edges (connections between nodes), trips (sets of elementary connection resulting from one vehicle trip), routes (sets of trips within the same route but at different times), calendar (restrictions in terms of weekdays, holidays, etc.), and may include another information such as fares and so on. All required datasets are described in more detail in the implementation part of this thesis.

2. Schedule-based transit network specification

In order to be able to find shortest paths in schedule-based networks, main focus has to be on the network description. Well understood and precisely modeled network can significantly increase the shortest-path algorithm efficiency. The main input to the network model is a timetable of transportation means moving along the network. Timetable gives times where certain edges can be accessed. It means that passing through an edge is possible only at a certain time and by a specific transportation mean. It is the main difference in comparison to road networks.

A transit network consist of stops (bus stops, railway stations, ports, airports, etc.), transit lines (bus lines, train lines), and trips (the movement of vehicles along transit lines) [2]. All the data are stored in timetables.

Transportation network is a directed graph denoted as $G = (N, E)$ where N is a set of nodes and E is a set of graph edges which are ordered pairs of nodes $E \subseteq N \times N$ [6] . After a time-restricted criterion is added, the graph model is expanded to $G = (N, E, L)$ where L are transit trips performed by transportation means along the edge at certain times. The main source of information for the following section is [2].

2.1 Timetable

A *periodic timetable* according to [6] is defined as:

$$(C, S, Z, \Pi, T) \tag{2.1}$$

where C is a set of elementary connections, S is a set of stations, Z is a set of transportation means, and $\Pi := (0, ..., \phi - 1)$ is a finite set of discrete time points (seconds, minutes). The periodicity of a timetable is denoted by T.

An elementary connection c contained in a set C can be denoted as:

$$c = (z, s_1, s_2, t_d, t_a) \tag{2.2}$$

where $z \in Z$ is one transportation mean (a bus leaving from a station at a certain time), $s_1 \in S$ departure station, $s_2 \in S$ arrival station, $t_d \in \Pi$ departure time, and $t_a \in \Pi$ arrival time. [3] It means that a certain vehicle z leaves the departure station s_1 at the time of departure t_d and arrives at the arrival station s_2 at the time of arrival t_a. The variables t_a and t_d are integers in minutes counted from midnight on, or times in 24 hours format. The function $length(c)$ denotes the cost of an elementary connection which is the time passing between the departure and arrival (i.e. traveling time). Due to the periodicity of a timetable T, duration can be calculated as $\triangle(t_1, t_2)$ between two time points t_1 and t_2 as $t_2 - t_1$ if $t_2 \geq t_1$, and as $\pi + t_2 - t_1$ in the case of traveling over midnight[6].

Elementary connections are valid for each z only at specific days of the week D and can also be dependent on the time of the year as are for example holiday services. Trip validity is specified in a timetable and restricted by timetable constraints.

Each station is assigned with a *minimum transfer time* [6] which does not have to be constant between all platforms within a station. Transfer is possible only at a station s where the condition of *minimum transfer time* is smaller than the time from the time of arrival to the time of the next departure. It is also possible to set up specific transfer rules based on specific properties of a particular station (transfer on a single platform, transfer between different platforms, multimodal transfer, etc.).

Station nodes can be connected by $foot - edges$ in such a case when the stations are close to each other and the walking distance between them is reasonable. $Foot - edges$ can be treated as usual elementary connections without set departure and arrival times. They have a lower preference than vehicle connections. They can be accessed at any time without restrictions.

2.2 Connections

A set of elementary connections can be denoted as

$$C = (c_1, c_2, ..., c_k) \tag{2.3}$$

together with departure times $t_d(c)$ and arrival times $t_a(c)$ for each elementary connection c_i where $1 \leq i \leq k$. It denotes a possible movement from one station to the next station along one edge. We also assume that the time of departure $t_d(c)$ and the time of arrival $t_a(c)$ can be written in the form of time in minutes from the beginning of the validity of timetable $t = a \cdot 1440 + b$ where $a \in [0, 364]$ and $b \in [0, 1439]$ [6]. In order to evaluate the elementary connection set C as consistent [3], several conditions have to be satisfied.

A connection is valid only at a certain day specified by the timetable 2.4. An arrival station of one elementary connection is the departure station of the following connection 2.5. Departure times are periodical 2.6. Arrival time to the next station is the departure time from the previous station plus the connection length 2.7. Departure time in the following station cannot be earlier than the arrival time to that station plus transfer restrictions applied to that particular station 2.8.

$$c_i \; is \; valid \; on \; day \; [dep_i(C)/1440] \tag{2.4}$$

$$s_2(c_i) = s_i(c_{i+1}) \tag{2.5}$$

$$dep_i(C) = t_d(c_i) mod (1440) \tag{2.6}$$

$$arr_i(C) = dep_i(C) + length(c_i) \tag{2.7}$$

$$dep_{i+1}(P) - arr_i(P)) \geq \begin{cases} 0 & Z(c_i + 1) = Z(c_i) \; or \; foot - edge \\ transfer(s_2(c_i)) & \text{otherwise} \end{cases} \tag{2.8}$$

2.3 Transit Network Modeling

Transit network models describe the transit network in such a way that a network graph can be used as an input for a shortest-path algorithm. The model describes all relationships within the network in more detail and focuses also on the relationships within a node such as waiting, transfers, etc. A network model is a way of representation of a network graph. The two main approaches, the *time expanded* and *time dependent* approach, have been both developed and started to be used only recently. The main difference between these two approaches is how they deal with multiple connection possibilities for each edge. Queries are solved by the application of appropriate shortest path algorithms on a network model. [3]

2.3.1 Time-Expanded Approach

The time-expanded model is based on the fact that timetables consist of scheduled events which occur at a discrete time (train departure, train arrival). Using this assumption, we are able to create a time-space graph in order to connect all possible following discrete events in time. Edges represent a connection of subsequent events in the time flow. In a basic model, an arrival event has its own node as well as its own departure event. It results in a large and usually sparse graph.

Pyrga et al. [9][13] and Müller-Hannemann and Schnee [12] extended the time-expanded model to incorporate minimum change times (given by the input data) that are required as a buffer when changing means of transport at a station. Their so-called realistic model introduces an additional transfer node per departure event and connects each arrival node to the first transfer node that obeys the minimum change time constraints.

The model consist of three different kinds of edges. The first kind are $connection - edges$ (sometimes called $transport - edges$) which are elementary connections denoted by (z, s_1, s_2, t_d, t_a) connecting the departure and arrival stations. In other words, they interconnect different graph nodes. The second kind of edges are $transfer - edges$ (sometimes called as $stay - edges$) which represent the waiting time at the station. They are interconnecting *connectiont edges* within the station according to the time sequence. The length of the $edge(u, v)$ is $t_v - t_u$ while the length for the edges passing the midnight is $1440 + t_v - t_u$ [3] as shown in figure 2.1.

2.3.2 Time-Dependent Approach

In comparison with the time-expanded model, the size of the time-dependent model is significantly smaller. The main difference is the fact that for each station only one node is present, and between two nodes only one edge is present. Saving large numbers of nodes and edges is done by unrolling of the timetable in time (only edge attributes present after possible edge accessing time are considered). The basic idea is that travel time is evaluated by $travel\ time\ functions$ which map departure times to travel times [2]. A cost of passage through a certain edge is evaluated based on the time of traversing. Pyrga et al. [9] extended this model by adding route nodes interconnected with the station node in order

to enable transfers. Those can also mean some minimum transfer time between the vehicles in cases when no more specific information is present.

If the elementary connection exists, then there is only one connection e between points A and B for one specific time of the accessing node A. The length of the connection then varies in time. It changes according to the time when the particular edge is used by the algorithm. The model assumes that overtaking on a particular edge is not allowed. However, in case of a scenario when two alternative routes between two nodes exist, they can be modeled as independent edges. Usually it is not necessary because we can examine only the arrival time to node B from the set of feasible elementary connections from node A, and choose the best one without considering the actual physical route.

A modification of Dijkstra's algorithm can solve the problem of the earliest arrival in this kind of model[15] by determining the edges cost based on actual time and labeling of nodes.

In a timetable with three stations A, B, C, there are three trains that connect A with B (elementary connections u,v,w), one train from C via B to A (x,y), and one train from C to B (z) as shown in figure 2.1. Time-expanded approach is displayed on the left side and time-dependent is on the right side. More detailed example is discussed in section 2.3.3.

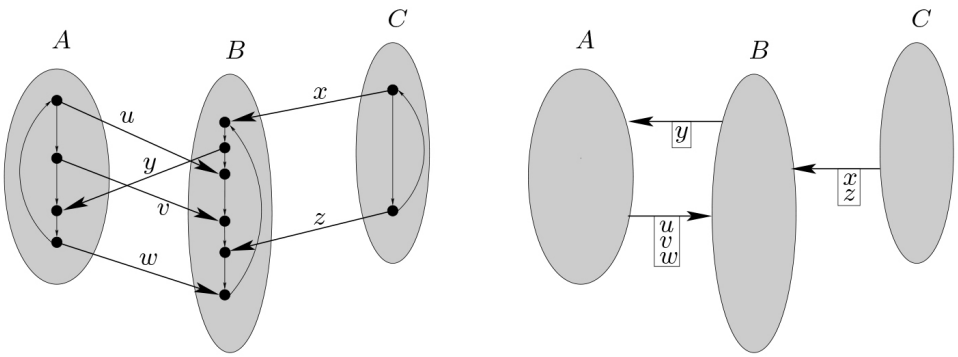


Figure 2.1: The time-expanded model (left) and the time-dependent model (right). Source: [3]

For my implementation I will use the time-dependent approach because the model complexity is significantly lower compared to the time-expanded approach, and data manipulation will be also easier. By using the appropriate SQL queries, the efficiency of proposed algorithm should be much better than in the case of the time-expanded model.

2.3.3 Examples of Time-Expanded and Time-Dependent Approach

The example network consists of three stations A, B and C which are interconnected by three bus lines $1, 2$ and 3. For illustration purposes, only a small number of bus trips is present, only in one direction from A to C, and each bus trip has a different color. Bus line 1 is interconnecting only stations B and C.

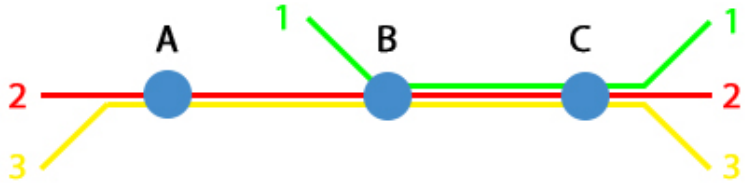


Figure 2.2: Network

Timetables for bus lines are listed in tables 2.1, 2.2 and 2.3.

LINE 1		
A	B	C
	8:02	8:05
	8:07	8:10
	8:12	8:15
	8:17	8:20

Table 2.1: Timetable for line 1

LINE 2		
A	B	C
8:03	8:05	8:08
8:13	8:15	8:18
8:23	8:25	8:28
8:33	8:35	8:38

Table 2.2: Timetable for line 2

LINE 3		
A	B	C
8:04	8:06	8:09
8:08	8:10	8:13
8:12	8:14	8:17
8:16	8:18	8:21

Table 2.3: Timetable for line 3

The time distance between stations A and B is two minutes, and between stations B and C is 3 minutes. Times of arrival and departure in a single station are equal because the waiting time in a station is not taken into account.

The time expanded model consists of a large number of edges. There is one edge for each vehicle trip denoted by a colored arrow. Transfers are made possible by the black arrows which represent transfer (waiting) edges. For a transfer, passenger has to stay in a station for a minimum transfer time given by the station. The model expands in time according to the arrow direction. Each edge has only one departure and arrival time for each stop. In a large network with a large amount of nodes, such a model can be extremely complex.

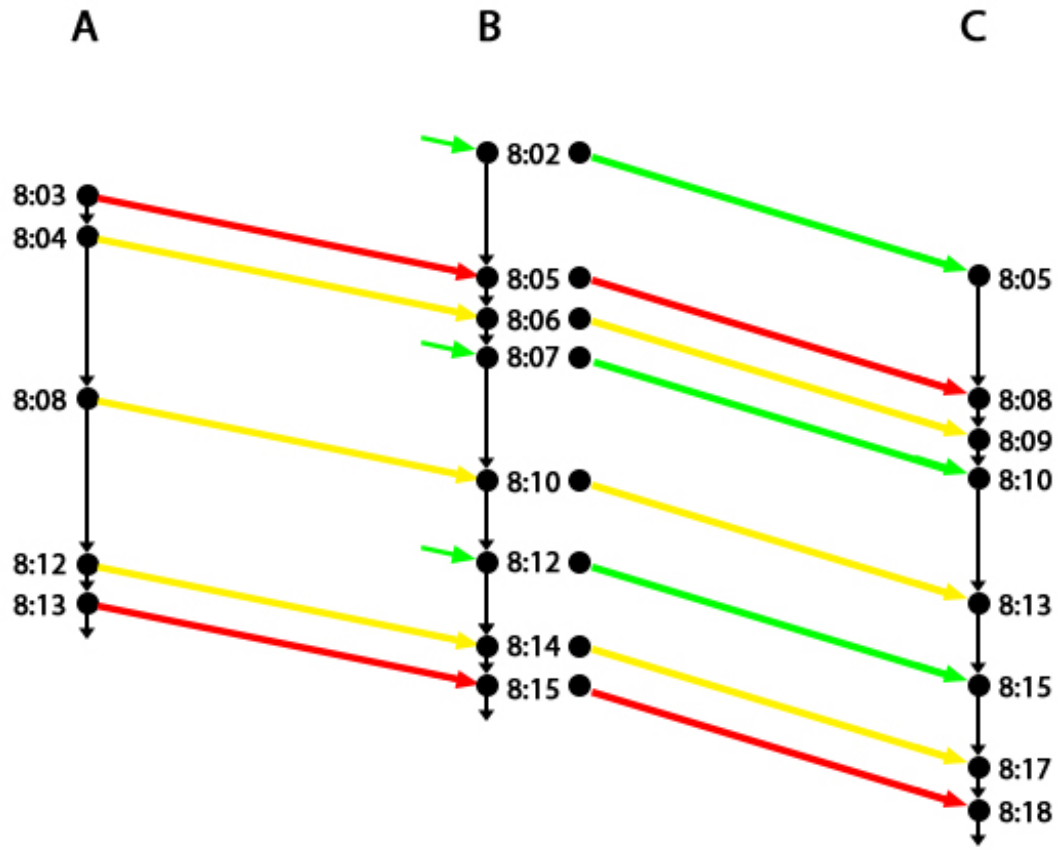


Figure 2.3: Example of time-expanded network model

The time-dependent network model consists of a smaller number of edges where only one edge is present between two stations. The length and waiting time are evaluated based on the time when the edge is accessed. Each edge has a large number of elementary connections. The picture below shows a list of accessible times.

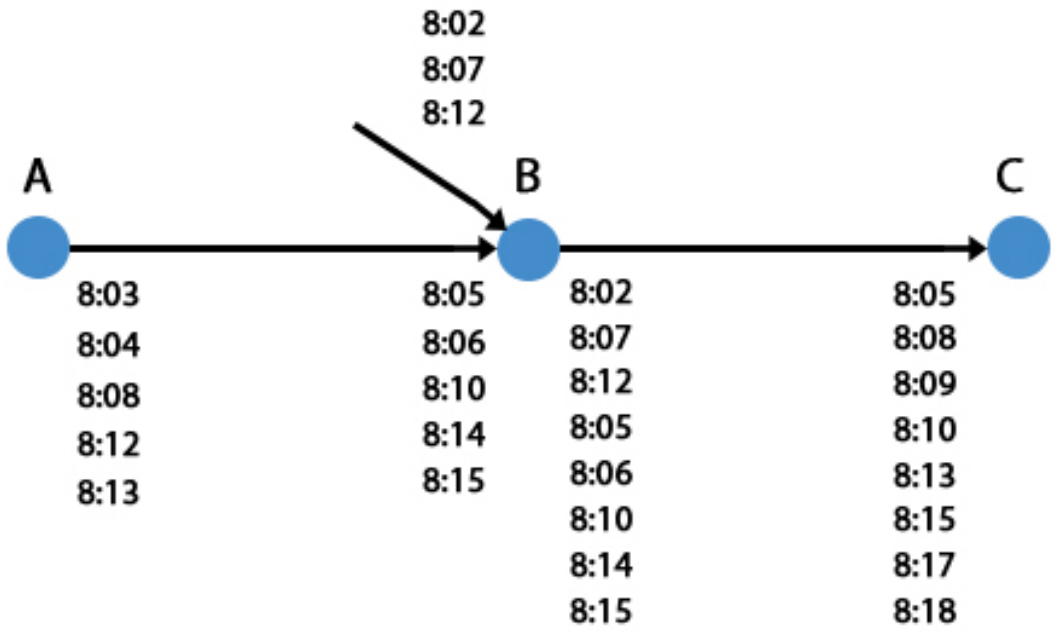


Figure 2.4: Example of time-dependent network model

As can be clearly seen from figures 2.3 and 2.4, the time-dependent approach has a lower complexity and is therefore recommended for vast networks.

3. Problems in Transit Networks

Most of the road networks research has focused on computing the shortest path according to a given cost function (typically travel times). However, in the case of public transit networks, the variety of natural problem formulations is wider[2]. The most common and fundamental query is the so called *earliest arrival problem* where the goal is to find a route with the lowest cost and the earliest arrival to a destination, and a departure after the given minimum departure time.

Related to the earliest arrival problem is the *range problem* in which the exact earliest departure time t_d is replaced by the time range. Both problems take into account only the time as a single optimization criterion. In the real transit network, additional criteria have to be considered (number of transfers, price, vehicle class, etc.). This leads to a more complicated problem called *multicriteria problem*. In the set of solutions to the problem, some solutions can be omitted in case they do not satisfy the given criteria.

The main problem is to find an optimal path or a set of optimal paths (for different times) which represent the optimal solution on the given set of elementary connections and criteria.

Other less common possibilities are: traveling in a certain class of a transportation mean, transportation for people with disabilities, tickets for local trains only with the exclusion of express trains, passage through a certain station required, stop-over of a certain length along the route required, and so on. Problems are listed in table 3.1.

Problem	**Problem includes**	**Reference**
Earliest arrival problem		3.1
Range problem		3.2
Multicriteria problem		3.3
	Transfer rules	3.1
	Foot-edges	3.3.2
	Traffic days	3.3.3
	Minimum number of transfers	3.3.4
Extended queries		3.4
	Latest departure	3.4.1
	Excluding vehicles	3.4.2
	Via stations	3.4.3
	Cheapest connection	3.4.4

Table 3.1: Problems in transit networks

3.1 The Earliest Arrival Problem

The *earliest arrival problem* is the basic optimization problem in time-dependent networks. The query consists of a departure station, an arrival station and a departure time. The goal is to find a path with the minimum difference between the departure from the origin station and the arrival time to the destination station where the departure time from the origin is at or after the given earliest departure time. Notation of such query has a form of (A, B, t_0) where A is the origin station, B is the arrival station and t_0 is the required earliest departure time.

A modification of the earliest arrival problem is called *latest departure problem* where the query is defined by the latest time of arrival to the destination. This problem is discussed later in section 3.4.1.

3.2 Range Problem

The *range problem* is a modification of the earliest arrival problem where single departure or arrival time is substituted for a range of departure or arrival times. It can be hours in a day, whole day, or a number of days. For example, the problem can be stated as finding all shortest paths between 8am and 11am between two stations A and B. A result of the query is a set of shortest paths. Routes in this case can follow the same routes at different times. Such query can be written as $(A, B, \Delta t)$ where Δt is the time range. This problem can be solved by multiple search. A departure node is added to priory queue and departure time is set to all possible departure times from the given destination within the interval $\Delta t = [t_0, t_1]$ [3].

3.3 Multicriteria Problem

The multicriteria problem involves several transit rules such as $transfers$ (transfer rules, minimum number of transfers problem), $foot-edges$, $traffic\ days$, and also involves extensions to the queries such as *latest departure*, $time-intervals$ (range problem), *excluded vehicles*, $via-stations$ or *cheapest connection* (fare criterion). This section uses [3] as the main source of information.

3.3.1 Transfer Rules

In the realistic modeling, the basic problems are extended into a more complicated scenario by the addition of a certain new criterion. The transfer-rules can be incorporated to both time-expanded and time-dependent models. In a time-expanded graph, each transfer node has two outgoing edges. One is for the departure on the same vehicle with no additional transfer time, the second leads to the next transfer node with the closest time that is providing the opportunity to transfer to another vehicle, as shown in figure 3.1. The time difference between those two nodes is greater or equal to the minimum transfer time rule in the particular station [3].

A time-dependent model is extended by route information [16]. We can establish the vehicle network based on the timetable and a set of stations

$(s_0, s1, ..., s_{k-1})$ for $k > 0$ which form a route if there is a vehicle starting at s_0 and is consecutively visiting all stations in the set. If there are more vehicles following the same set of stations, we can assign the vehicle to the same route. The model contains three different kinds of edges: the $transfer\ edges$ from the station node to the route node which is already part of the vehicle route, $transfer\ edges$ from the route node to the station node, and $route-edges$ interconnecting route nodes belonging to the consecutive stations in the route. For the modeling purposes, the cost of the edges representing the getting off a train is set to zero. The cost of the edges representing the boarding on a vehicle is set to the station transfer time $g_s = transfer(s)$. Finally, the length of the edges interconnecting two consecutive stations is given by a timetable [3]. These rules are represented in figure 3.1.

Transfer times may also vary based on the arriving of the vehicle and station restrictions (transfer between platforms, transfer from bus to train, etc.). Those transfer rules have to be specified for all possible transfers within the station by station transfer rules.

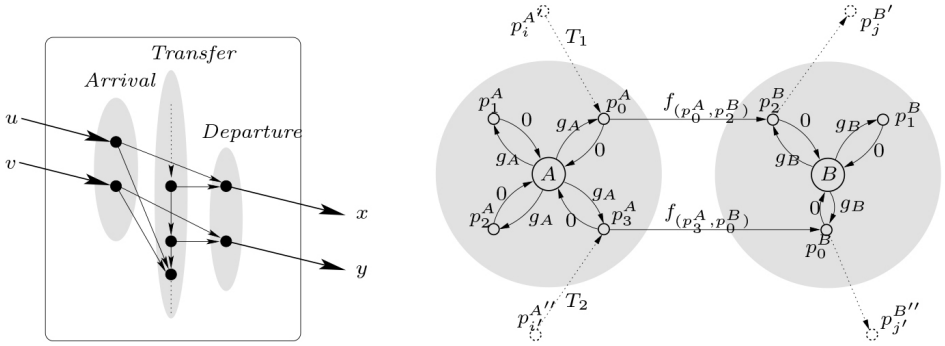


Figure 3.1: Modeling transfer times in the time-expanded approach using the realistic time-expand graph (left), and in the time-dependent approach using the train-route graph (right). Source: [3]

3.3.2 Foot-Edges

In both, time-extended as well as time-dependent models, the implementation of $foot-edges$ is not difficult. An edge connecting two nodes A and B is called the $food-edge$ when it does not have any time restrictions (the edge can be accessed at any time) and its length is a walking time between nodes A and B.

3.3.3 Traffic Days

On different days such as weekdays, weekends, holidays and so on, edges of a graph can be valid or invalid. In case of the time-extended model, edges can be simply removed from the network graph. In case of the time-dependent model, the length of an edge has to be determined excluding the invalid edges. The validity of an edge can be determined by the verification of validity in a lookup table by a boolean decision. Problems can be encountered in such cases when

an optimal path includes one day stay in a station, or in the case when speedup techniques with data preprocessing are used [3].

3.3.4 Minimum Number of Transfers

A route with the minimum number of transfers can be found in both time-dependent and time-expanded models by the method of zero length paths[12]. At first, the length of all transfer edges is set to zero and all edges interconnecting route stations are set to zero. The calculation of the route itself is done in a usual manner and the resulting route is the route with the minimum transfers. After that, the length of routes can be set back according to the timetable, and the real traveling time can be calculated.

3.4 Extended Queries

The path search can be restricted by several other conditions according to the passenger demand. Common constrains are dealing with departure and arrival times in both origin and destination stations. From the passenger's economical point of view, the *cheapest route search* is a quite common problem as well as is the problem of *excluding trains* (first class, high-speed rail). The possibility of selection of *via station* or *stations* which need to be passed along the way also has to be considered. This might not be a required restriction but it might make a route search more comfortable.

3.4.1 Latest Departure

The latest departure problem is usually solved by the reverse time search. The search begins at the time t_d which is the required latest departure time, and more routes can be found earlier before the actual departure time t_d. In case of the time-dependent model, the earliest arrival time is found, and the search is performed backwards from a destination to an origin station as well as reverse in time until the departure station is found[3].

3.4.2 Excluding Vehicles

In principle, the excluding of vehicles (train of a certain class, bus of a certain company) can be treated the same way as the $traffic\ days$ 3.3.3. Elementary connections operated by the excluded vehicles are simply removed from the model [3]. The decision is based on the boolean query during the search.

3.4.3 Via Stations

The *via station* query [3] takes as an input one or more stations which has to be visited along the way, and the minimum time, maximum time or both spent in the *via stations*. If we have a query $(A = s_0, B = s_k, t0)$ and the via stations are $(s_1, s_2, ..., s_k)$ then the corresponding set of the minimum duration spent in each of them is $(d_1, d_2, ..., dk)$. This problem can be solved by the division of the query into a set of simpler queries where the search of routes is done subsequently. The

first query can be written as (A, s_1, t_0), the second one as $(s_1, s_2, t_{a,1} + d1)$ where the time $t_{a,1}$ is the arrival time to the via station one and d_1 is the length of the stay in the node one. The search is performed subsequently until the final destination is reached. The general formula can be written as $(s_i, s_{i+1}, t_{a,i} + d_i)$ where $i = 1, ..., k-1$. The last step is to interconnect all partial query results into one route that solves the overall problem. Another complexity could be added by the selection of a via station without order. The problem then would be to find the closest destination first, and then follow until the final destination would be reached. This problem is much more difficult to solve and is similar to the traveling salesman problem.

3.4.4 Cheapest Connection

It can be a passenger's desire to find the cheapest connection between two stations and in a given interval of time. This problem is very difficult to be solved because the given cost of an edge expressed in money is usually not proportional to the length of an edge expressed in km (longer trip usually means a cheaper price per km). There can also be different tariffs applied on the route for different passengers, traveling time, or for each particular company operating on the same route.

3.5 Multi-Criteria Optimization

Problems can be combined because in the real life operation, queries usually consists of multiple criteria (departure at certain time with the minimum transfer, the arrival before another certain time, minimum cost, convenience requirement, class of vehicle, on-board services, seat reservation, etc.).

The query consisting of multiple criteria is called the $multi - criteria$ or $multi - objective\ shortest\ path$ ($MOSP$) query, and is a fundamental problem in the real life transit networks [3]. An instance of a solution to the $MOSP$ is associated with a set of feasible solutions Q and a d vector function $f = [f_1, ..., f_d]^T$ associating each feasible solution $q \in Q$ with a d-vector $f(q)$ [3]. The goal of the problem solution is to minimize the vector $f(q)$. In a multi-criteria scenario we are usually unable to find the best solution for each criterion. However, we are looking for a trade-off solution where the overall solution is as good as possible for all the criteria. The trade-off among all criteria is called the *Pareto set* or *Pareto curve*. It is a set of feasible solutions Q, where none of the solutions is dominated by another solution.

In my implementation I deal mostly with the Earliest arrival problem as the fundamental problem which can be used as a foundation for solving different problems.

4. Shortest-Path Algorithms for Time-Dependent Networks

The main goal of this chapter is to identify and compare different kinds of shortest-path searching algorithms, and to find the best approach for the implementation. The sources of information for this section are [5], [1], and [4].

The field of the path-searching algorithms in time dependent networks started to develop early after the first Dijsktra's label setting algorithm for road networks in 1959. The label setting property means an assignment of a label with the minimum distance to each of the previously explored nodes. The terminal label is at the destination node and its value is the minimal path length.

In the past fifty years a lot of progress has been done especially by focusing on real human behavior, and the addition of operational planning elements to the schedule-based network models.

The early methods were proposed by Dial in 1967 [29] and Le Clercq in 1972 [30] who were the first to use heuristic methods considering traveling and waiting times. Passengers were expected to take the first vehicle to arrive and the transfer penalty was considered equal to the waiting time.

For the purpose of public transportation networks, Tong and Richardson in 1984 [39] improved Dijkstra's algorithm for the use in time-dependent networks. They proposed a solution for finding of the shortest or cheapest path based on the defined cost function. [5] [1] [4]
The following section describes the evolution and progress of time-dependent shortest-path searching algorithms.

4.1 Basic Algorithms Without Preprocessing

Algorithms without preproprocessing are algorithms which use the input data in a raw form. The preprocessing is a preparation and adjustment of data in a way that makes the algorithm and its computation faster and more efficient. The preprocessing is done separately and algorithms use already preprocessed data. The most basic algorithms are based on Dijkstra's algorithm.

4.1.1 Time-Expanded Dijkstra's Algorithm

The Dijkstra's algorithm for the minimum path search on an oriented graph can be extended for the computation of the minimum path in the time-extended networks. The algorithm is called $Time-Expanded\ Dijkstra's\ Algorithm$ [14][3][2]. The general principle of the algorithm is such: a computation is initialized at the departing station (origin) s_d by the first possible event which would happen after the required time of departure t_d. The computation continues by scanning all accessible events along the expanding graph until the arrival station s_a is found. This path is the shortest path. All edges belonging to the path can be labeled on the reverse transverse of the path from the destination to the origin.

Evaluation

The *Time − Expanded Dijkstra's Algorithm* is simple to understand and implement. The computed shortest path is always the optimal one because the algorithm is deterministic and driven by the earliest event.

The main disadvantage is a significant expansion of the search space during the computation, and a large memory consumption. Also, a significant computation performance is required when the explored graph is large.

4.1.2 Time-Dependent Dijkstra's Algorithm

The *Time − Dependent Dijkstra's Algorithm* [3][15] can solve the problem as long as the edge costs are non-negative and FIFO (first in first out; generally speaking vehicles are not overtaking each other on one edge). If s_d is the departure station and t_d the earliest departure time, the time difference τ (difference between arrival and departure from the same station) is added to the edge length and the best option is chosen as the algorithm goes. In the time-dependent model the Dijkstra's algorithm is a label-setting algorithm so that whenever an edge $e = (A, B)$, the considered distance label $\delta(A)$ of a node is optimal. The label $\delta(A)$ denotes the earliest possible arrival to the node A.

The main modification is that the edge $e(A, B)$ is evaluated at the time $\delta(A)$ where A is the preceding station. Every edge is evaluated only once to hold the time-dependent property. The algorithm can run from both the station of departure and the station of arrival in a reverse manner. Parallel computation is also possible but more complicated as will be discussed later. The search terminates at the moment when the sets of explored nodes from each start have an intersection, and transfers are feasible.

Evaluation

The main advantage of the *Time − Dependent Dijkstra's Algorithm* is that the cost of an edge is computed only once according to $\tau + length(t_d, t_a)$. The performance and memory demands are lower than in the case of a time-expanded approach. The algorithm is again deterministic and the solution is always an optimal path.

4.2 Algorithm Optimization

In the schedule-based networks, size of a database is usually large. Algorithms therefore have to be capable of working with a large amount of data. The evolution of computer technology has made the task more feasible recently.

The following section describes design methods and techniques of how to make the shortest path searches in the schedule-based network models more efficient. In case of the naive Dijkstra's algorithm, the search space can be described as a funnel which expands as more nodes are being explored. Then, once the destination node is found, the shortest path is being explored backwards from the destination node to the origin node. The search space expands rapidly.

The goal of the search space optimization is to make the funnel steeper (i.e. to decrease the search space). For example, to examine only a fixed number of departing vehicles or even consider departures before the desired departure time. Therefore, the goal of the graph search optimization is to reduce the size of the funnel to a minimum, to decrease the search space, and still reach an optimal or close to optimal solution. The smaller amount of nodes is being examined, the lower is computation complexity. [19]
In general, complexity can be denoted as:

$$O[C * N * log(N)] \tag{4.1}$$

Where C is the set of elementary connections explored, and N is the number of nodes explored.

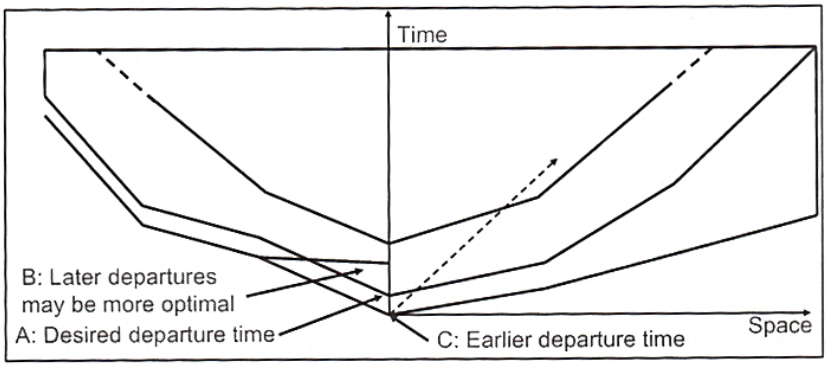


Figure 4.1: Illustration of the path-searching in time-space networks. Source: [19]

Stochastic shortest-path searching algorithms use a choice function in order to reduce search space.

4.3 Reduction of Search Space

The following section describes methods and approaches to the optimization of search space by restriction rules before nodes are inserted into the heap. This saves memory and computation time because unnecessary nodes and edges are removed first.
The main approaches are:

- Event Dominance
- Granularity
- Hidden Waiting Times as Pseudolinks
- Transit Network Hierarchy

4.3.1 Event Dominance

According to R. D. Frederiksen., O. A. Nielsen [19] :

> The idea of event dominance is that the arriving path at a given stop is only entered into the heap if the utility of the time-distance is better than the utility of earlier arrivals plus a function of the time difference. When boarding the first line 4.2, the A instances may be removed from the search space due to the event dominance. All later departures are hereby implicitly removed without any calculation. At this point the D departure is still relevant.
> When arriving at the second stop, the B departure is removed, whilst the D line is still relevant. But transfer at the second stop from the F line to the D line is removed (transfer B). At the third stop, the choice set consists of the D line ant the found solution (F line). Which one to choose at this stage depends on the coefficients in the utility function: F has a higher early penalty, D larger waiting time at stop 2. With a sufficient relaxed event dominance function, the principle will almost always contain the optimal path for journeys with only one transfer. This approach often fails in case of multiple transfers.

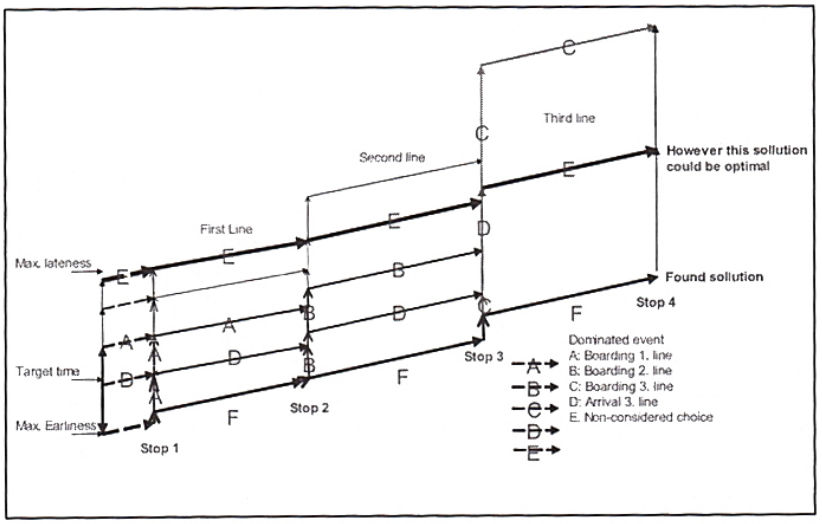


Figure 4.2: Event dominance for 2-transfer trip. Source: [19]

4.3.2 Granularity

In many time-dependent network models, properties of the search space allow division of the search space into segments where the solution can be optimal. It mostly depends on the network properties. The granularity also decreases the search space.

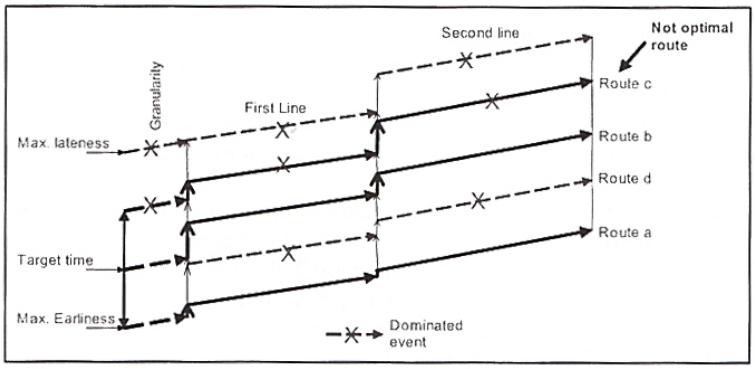


Figure 4.3: Example of a granularity approach. Source: [19]

4.3.3 Hidden Waiting Times as Pseudolinks

According to R. D. Frederiksen., O. A. Nielsen [19] :

> idea is to build such links back in time from each departure time instance. Whilst the granularity approach may overlook optimal solutions, the pseudo-link approach guarantees optimality with regard to the first boarding instance. The computational complexity characteristics of this method are: The number of pseudo-links is equal to the number of departing runs. However logical interrelationship

between multiple lines serving the same stop may reduce the number of pseud-links (avoid equal time-space pointers to the same stop).

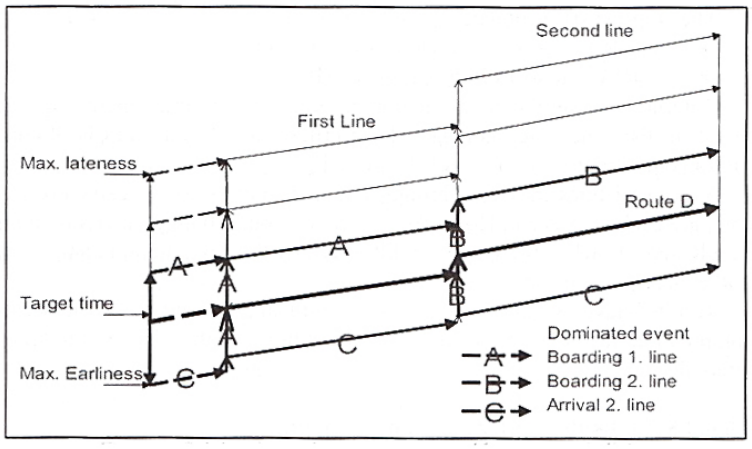


Figure 4.4: Example of a pseudolink approach. Source: [19]

4.4 Transit Network Hierarchy

Stations in a transit network can be divided into two categories. In case that two lines have an intersection in a node, or two parallel lines split in a node, they are called *transfer nodes*. From the passenger's point of view, transfer from one to another parallel lines at the inner nodes following the same physical route makes no sense.

Transfer nodes can be detected by a simple criterion. If a number of neighbor nodes that are directly accessible by any line from the starting node is more than two, then the transfer is possible and the node is a transfer node.

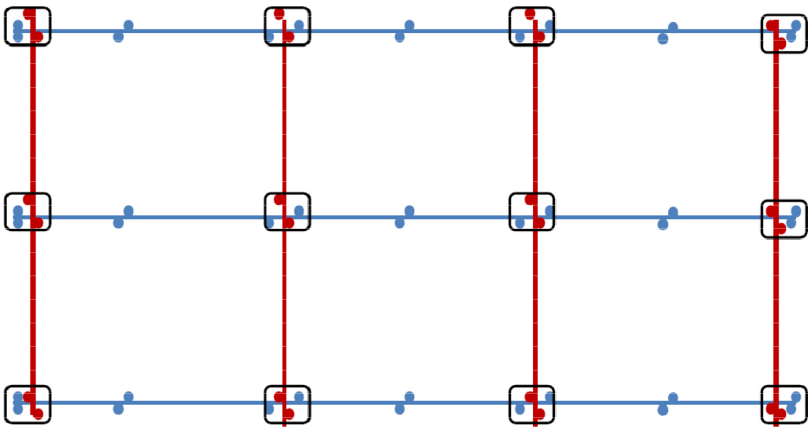


Figure 4.5: Example of a network hierarchy. Source: [5]

4.5 Speedup Techniques

The following chapter describes methods that should improve computation time and overall algorithm efficiency. Many of them were originally used in the shortest-path algorithms for road networks which proved their efficiency. Due to the differences in data model structure between public transportation networks and road networks, speedup techniques are less efficient for public transportation networks. The main problems are a time-dependency and a large number of possible interconnections and possibilities. Many algorithms also use a combination of speedup techniques[2]. Bauer et al. [18] evaluated efficiency of bidirectional search, Arc Flags, ALT, and of more advanced techniques such as Reach, Real, Highway Hierarchies and SHARC on the time dependent public transportation networks. This chapter uses [18] as the main source of information.

4.5.1 Bidirectional Search

Bidirectional search is one of the most fundamental and straightforward speedup techniques. The algorithm search is performed from both origin and arrival stations until the two search spaces meet. The searched area of such graph is smaller than in the case of a unidirectional search. A visualization of the bidirectional search is shown in figure 4.6. This approach is rather complicated in the time-dependent networks as it might be easy to meet in space but more difficult to meet in time.

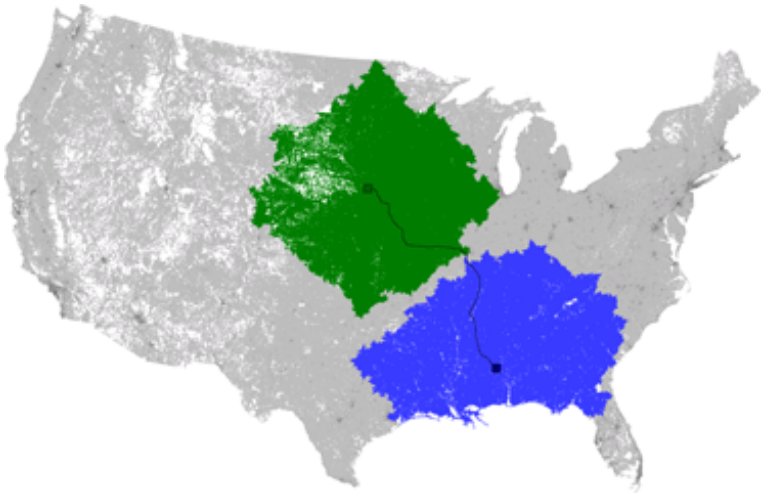


Figure 4.6: Bidirectional search on transportation network. Source: [20]

4.5.2 A* and ALT search

A goal directed search A* assigns a potential to each node in a set of explored nodes. The nodes where the sum of a potential and its own arrival time is the highest are summed, and the search continues in the node with the highest probability of being on the optimal path.

In the ALT algorithm, a node potential is obtained from certain landmarks

in the network model. This approach requires preprocessing but in general it is a more effective approach than A*. The main disadvantage of ALT is a fluctuation of query times.

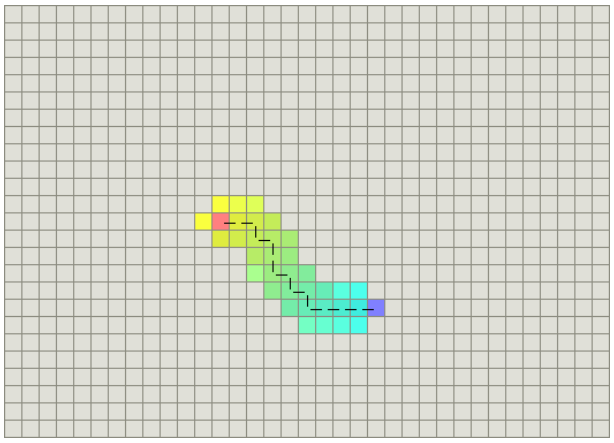


Figure 4.7: A* best-first search. Source: [11]

White in 1991 [44] developed a modification of the A* algorithm for time-dependent networks originally proposed by Hart et al. in 1968 [45]. The algorithm was called IA*. The algorithm itself was faster than the usual label-setting algorithms thanks to the use of *islands* (nodes with higher probability of occurrence on optimal path). This algorithm was using the usual Euclidean A* property for the estimation of traveling time.

Gao and Chabini in 2002 [46] based their research of the A* algorithm on the first proposal of Hart et al. in 1968 [45]. They developed a heuristic minimum cost static function for dynamic networks. Since then, the A* search has not been used much, and other methods were developed. [5] [1] [4]

4.5.3 Arc-Flag

In this method, additional information is being attached to the edges, and the algorithm subsequently checks if the edge can be a part of the shortest path or not. The whole graph is divided into cells and each edge is assigned with a label. The label contains information whether the shortest path starting from this cell to a corresponding cell exists[18]. For this setup the modified Arc-Flags Dijkstra's algorithm visits only the edges with the shortest path label in an inter-cell query.

This approach does not work for queries inside one cell as the preprocessing effort and memory demand would be too high in such a case..

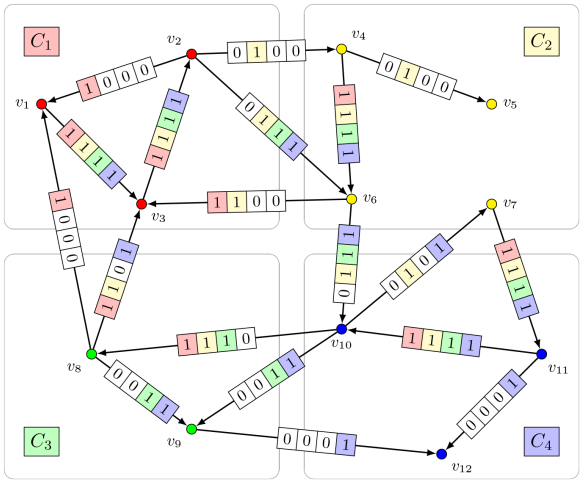


Figure 4.8: The example graph with correct arc-flags provided that the edge weights are uniform. Source: [10]

4.6 Advanced Algorithms

The following section explains approaches for the shortest-path search in time-dependent networks, and operation of non-basic algorithms. Some sections include algorithm pseudocodes that are structures of algorithms written in a human readable form to demonstrate algorithm operation.

4.6.1 Notation Used in Algorithms

PATi	Preferred arrival time to stop i
dt	The time window in which the alternative trips are determined meaning that the arrival time to the destination in this time window is acceptable
seq_i^p	Sequence number of stop i on trip p
d_p^i	Departure time of trip p at stop i (usually the same as the arrival time in the schedule and/or in GTFS data)
v_{ij}^p	In-vehicle time from stop i to stop j using trip p
t_i^j	Transfer time from stop i to stop j
a_i	Transfer time from stop i to stop j
w_i^p	Waiting time at stop i for trip p
p_i	Predecessor of stop i
m_i	The mode (a trip number of the transfer link) used to reach stop i in forward algorithms, or to leave stop in a backward manner
e_i	Lower bound of minimum time (cost) from stop i to the destination stop
F_i	Minimum travel time (cost) of the path from the origin to the destination through the stop i
cp_i	The utility (a function of travel time and cost) of trip p at stop i to reach the destination
$pr_i(p)$	The probability of taking trip p at stop i
$T(i)$	The set of transfer links at stop i
$R(i)$	The set of routes at stop i
$p(i)$	The set of vehicle trips at stop i
$p_a(i)$	The attractive set of trips at stop i
SE	Scan eligible list containing the stops with temporary labels in the algorithm

Table 4.1: A notation of variables used in algorithms. Source: [5]

4.6.2 Trip-Based Shortest Path Algorithm

According to [5] a trip-based shortest path algorithm is based on Dijkstra's shortest path algorithm introduced by Tong and Richardson in 1984 [39]. The number of transfer points is not limited so that the passenger can transfer whenever it is necessary until the destination is reached. In the model the passenger cannot transfer at any station because it is inconvenient for him, nevertheless the vehicles can access any station. As long as the vehicle is getting closer to the destination, the passenger is expected to remain on board of that vehicle. The search is performed on *hierarchical transit network*.

The hierarchical network is a network where the passenger can transfer only at certain stations because transfers in every station along the way would be inefficient, as well as there might not be other vehicles coming. On the top of the node hierarchy are transfer stops where transfer to a different line is possible. Instead of visiting each stop along a non-transfer part of a path, it can be considered as a one path segment.

The TBSP algorithm is a forward labeling algorithm starting at the origin at time τ. With a simple modification, a backward search can be performed from the required time of arrival.

Algorithm 1 Trip Based Shortest Path TBSP. Source [5]

Initialization:
Get origin (O) *and the departure time* (τ)
$i = O,\ l_i = 0,\ p_i = \Phi,\ a_i = \tau,\ m_i = \Phi,\ SE = \{i\}$
$l_j = \infty,\ p_j = \emptyset,\ a_j = \infty,\ m_i = \Phi,\ \forall j \neq i$
Termination criterion:
$If\ SE = \Phi$
Stop Selection:
$Select\ i = Argmin_j\{l_j | j \in SE\},\ if\ Label\ Setting$
$Select\ i = the\ first\ stop\ in\ SE,\ if\ Label\ Correcting$
$SE = SE|\{i\}$
Updating the Labels:
$\forall t \in T(i)$:
if $(l_i + t_{ij} < l_j)$ **then**
    $l_j = i_i + t_{ij},\ a_j = a_i + t_{ij},\ p_j = i,\ m_j = T$
    $SE = SE \cup \{j\}$
end if
$\forall r \in R(i)$:
$Select\ p = Argmin_q\{w_{iq} | d_{qi} \geq a_i\}$
$\forall j \in S(p)\ with\ seq_j > seq_i$:
if $l_i + w_{ip} + v_{ijp} < l_j$ **then**
    $l_j = l_i + w_{ip} + v_{ijp},\ a_j = d_{jp},\ p_j = i, m_j = p$
    if $j \in N_t$ **then**
        $SE = SE \cup \{j\}$
        break
    end if
end if

4.6.3 Trip-Based Hyperpath Algorithm

Nguyen and Palatino in 1988-1989 [33] proposed a model in the *hyperpath* network using the optional cost function. The main concept of *hyperpath* (a set of prioritized alternatives at a stop) [5] is important for networks where the capacity of vehicles is limited, and thus passengers might not be able to board a vehicle. In such case they need to find an alternative path with the highest possible attractiveness. The trip based hyperpath model is a subnetwork where at each stop s a set of attractive routes $p_a(s)$ is defined, and each trip has a probability to be chosen by the passenger. The combination of the cost between stations results in the overall cost of the selected attractive path. Another case might be a connection missed due to the delay of a previous connection at a transfer station. In other words, Nguyen and Pallatino [33] define a *hyperpath* as an acyclic sub-network with at least one path connecting the origin and destination where at each node there is a possibility of choosing alternative ways.

Wu et al. in 1994 [36] used this principle and introduced a network model consisting of road and transit arcs. The model assumed that the time needed to board a vehicle was directly proportional to the vehicle flow. The later models worked frequently with the *utility* functions. Passengers were expected to assign each feasible path with a utility index and then choose the path with the maximum index. The utility value can be individual for passengers based on their requirements and chosen criteria.

A requirement for a hyperpath generation to a destination is a boundary time of arrival which cannot be exceeded. By extension of this boundaries to all stations, a set of alternative routes is defined. Each route has a combined cost function representing passenger's choice behavior. There are many proposed ways of calculating the cost by the passenger's choice.

Nuzzolo et al. in 2001 [42] improved Nguyen's and Palatino's [4] hyperpath approach for the schedule-based transit modeling. They based the path search on the calculation of probabilities of usage of each particular edge, and on the setting of a label for each edge.

Algorithm 2 Trip Based Hyperpath TBHP. Source [5]

Initialization:
Get the destination (D) *and preffered arrival time* (τ)
$i = D,\ l_i = 0,\ p_a(i) = \Phi,\ a_i = \tau,\ SE = \{i\}$
$l_j = \infty,\ a_i = -\infty,\ p_a(j) = \Phi,\ \forall j \neq i$
Termination criterion:
$If\ SE = \Phi$
Stop Selection:
$Select\ i = Argmin_j\{l_j | \forall j \in SE\}$
$SE = SE|\{i\}$
Updating the Labels:
$\forall t \in T(i)$:
if $(l_i + \alpha_t \cdot t_{ji} < l_j)$ **then**
$l_j = l_i + \alpha_t \cdot t_{ij},\ a_j = a_i - t_{ji},\ p_a(j) = p_a(j) \cup \{t\}$
$SE = SE \cup \{j\}$
end if
$\forall r \in R(i)$:
$\forall p \in r\ and\ \{a_i - dt \leq d_{pi} \leq a_i\}$
$\forall j \in S(p)\ with\ seq_j < seq_i$:
$p_a(j) \cup \{p\}$
$l_j = -ln[exp(-\Theta l_j) + exp(-\Theta(l_i + \alpha_w(a_i - d_{pi}) + \alpha_v v_{jip}))], a_j = max(d_{pj} | \forall p \in p_a(j))$
if $j \in N_t$ **then**
$SE = SE \cup \{j\}$
$break$
end if

4.6.4 Trip-Based A* Algorithm

The TBA* (Trip Based A Star) algorithm is an extension of the TBSP (Trip Based Shortest Path) algorithm. The main difference between the TBSP and TBA* is that in most cases TBA* requires less time for computation because of the pruning of the edges that are not expected to be used. Each considered node is labeled by the estimated travel time to the destination known as the *potential*. The labeling is done by a heuristic function. Nodes with a higher potential are expected to be part of the optimal path and are explored earlier than nodes with a lower potential. Due to this, the computation is performed in a lower amount of steps which decreases the computation time and memory demand. The optimum is reached by the selection of a suitable heuristic function and by satisfaction of the optimum condition.

The criterion for the estimation of travel time to the destination given by the *Euclidean distance* is not a good choice because in public transportation networks the distances are not given in the units of length but in the units of time. It is possible that the shortest trip in terms of time is long in terms of distance. Also, a longer route may offer transfer to a line which would take the passenger to the destination faster.

A *lower bound* is defined on the travel time based on the components of a network. The only difference along one path is the different waiting time at transfer points and before the boarding of a vehicle. The low bound value is the travel time with zero waiting time. The travel time from origin to destination can be denoted as [5]:

$$t = \Sigma_{i \in W} t_i^w + \Sigma_{i \in D} t_i^d + \Sigma_{i \in V} t_i^v$$

where:
t_i^w is the waiting time at station i where W is the set of waiting times in the path, t_i^d is the walking time at edge i where D is the set of walking edges in the path, t_i^v is the in-vehicle time at edge i where V is the set of in-vehicle edges in the path. Thus the lower bound which guarantees optimum of the algorithm is a sum of the in-vehicle and the walking time in the path $\hat{t}$.

$$\hat{t} = \Sigma_{i \in D} t_i^d + \Sigma_{i \in V} t_i^v$$

The optimum holds due to the fact that it is impossible to travel faster than the lower boundary of the path.

Algorithm 3 Trip Based A* TBA*. Source: [5]

Initialization:
Get the origin (O), *the destination* (D) *and the departure time* (τ)
$i = O,\ l_i = 0,\ F_j = e_{iD},\ p_i = \Phi,\ a_i = \tau,\ m_i = \Phi,\ SE = \{i\}$
$l_j = \infty,\ F_j = \infty,\ a_j = \infty,\ p_j = \Phi,\ m_i = \Phi,\ \forall j \neq i$
Termination criterion:
$If\ i = D\ stop$
Stop Selection:
$IfSE = \Phi,\ stop$
$Select\ i = ArgF_i = min_j\{F_j | \forall j \in SE\}$
$SE = SE|\{i\}$
Updating the Labels:
$\forall t \in T(i):$
if $(l_i + t_{ij} + e_{jd} < F_j$ **then**
    $l_j = l_i + t_{ij},\ F_j = l_i + t_{ij} + e_{jD},\ a_j = a_i + t_{ij}, p_j = i, mj = "T"$
    $SE = SE \cup \{j\}$
end if
$\forall r \in R(i):$
$Select\ p = argmin_q\{w_{iq} | d_{qi} \geq a_i\}$
$\forall j \in S(p)\ with\ seq_j > seq_i:$
if $l_i + w_{ip} + v_{ijp} + e_{jD} < l_j$ **then**
    $l_j = l_i + w_{ip} + v_{ijp},\ F_f = l_i + w_{ip} + e_{jD},\ a_j = d_{jp},\ p_j = i,\ m_j = p$
    if $SE = SE \cup \{j\}$ **then**
        break
    end if
end if

4.6.5 Different Approaches

The sources of information for this section are [5], [1], and [4].

Chriqui and Robillard in 1975 [31] introduced the passenger behavior concept for the first time. They assumed that a passenger will determine a set of possible routes and choose the first arriving vehicle for the selected set of feasible routes. They named the set *attractive routes* and defined it as a set of routes attractive to the passenger based on the prior knowledge.

Spiess and Florian in 1989 [34] introduced the concept of strategies in the frequency-based networks and transit assignment. They assumed that a user takes the first possible connection to the destination and is not considering waiting for a possibly better connection. They used the transportation line frequency to determine most attractive routes. The algorithm has two parts. The first one computes total traveling time including elementary connections and transfer time, and the second part assigns the demand according to the proposed strategy in order to find the set of attractive routes.

de Cea and Fernandez in 1993 [35] inspired by the work of Spiess and Florian [34] based their research also on the transit assignment using the frequency-based transit with capacity constrains. They distributed the excess demand along alternative routes by defining congestion functions on passengers who would board the first vehicle. This heuristic function assumed passenger's discomfort at crowded stations or in crowded vehicles, and estimated their behavior based on their convenience.

Hickman and Bernstein in 1997 [37] used the utility function in their deterministic sequential path-search algorithm. The utility function consisted of stochastic travel time and of attributes concerning passenger information. They applied and tested the algorithm also in a congested network with high-frequency service.

Lam et al. in 1999 [40] introduced a frequency based model for transit assignment process. They worked with a vehicle capacity and proposed principles for the alternative path selection in case of crowded vehicles. This model was used also in the case of seat-reservations and high-fare models (more expensive tickets during peak hours).

Tong and Wong in 1999 [38] examined the main differences between schedule-based and frequency-based approaches, and built models using simulations. They used the branch-and-bound method considering a large number of transit network properties.

Cominetti and Correa in 2001 [41] and Bouzaene-Ayari et al. in 2001 [48] introduced more realistic functions for the waiting time at stops and vehicle capacity restrictions. Their network model takes into account queuing of vehicles in stations in best path consideration. However, no search algorithm has been proposed.

Hamdouch and Lawphongpanich in 2008 [47] proposed a time-expanded model for schedule-based network and developed a hyperpath search algorithm. Cepeda et al. in 2006 [43] introduced frequency-based model for large scale networks and tested it on real networks.

4.6.6 Comparison of Algorithms

The following table contains a basic description and lists the main features of the selected most common shortest-path algorithms discussed earlier in this chapter. Algorithms are sorted chronologically based on the year of proposal.

Advance in transit assignment research	**Frequency-based features**	**Schedule (timetable)-based features**	**Route-choice features**
Dial, 1967 [29] Le Clercq, 1972 [30]	Heuristic consideration of waiting time (and travel time) in shortest-path approach		Boarding the first transit vehicle to arrive
Chriqui and Robillard, 1975 [31]	Probabilistic selection of a subset of routes minimize the expected sum of [wait + travel] time		Boarding the first vehicle from a set of attractive routes
Tong and Richardson, 1984 [39]	Deterministic selection of routes. Minimum number of transfers		Transfer at certain transfer stations. Unlimited number of transfers. Transfer rules
Nguyen and Pallotino, 1988 [33]	Origin to destination is interpreted on an acyclic directed graph (called hyperpath)		Boarding the first vehicle (from a set of routes) with a planned strategy of path movement
Spiess and Florian, 1989 [34]	Choosing a set of routes to minimize the expected sum of [access + wait + travel] time, using equilibrium model with linear programming		Boarding the first vehicle using a strategy of choosing only among the attractive routes
De Cea and Fernández, 1993 [35]	Incorporating a limited capacity for each route (of an attractive set) at stops in which waiting time depends on the passenger flow; using asymmetric equilibrium model with Jacobi method		Boarding the first vehicle (from a set of routes) given that the passenger flow does not exceed the route's capacity

Advance in transit assignment research	**Frequency-based features**	**Schedule (timetable)-based features**	**Route-choice features**
Wu et al. 1994 [36]	Hyperpaths are used for walk, wait, board, in-vehicle, transfer and alight time elements; boarding time increases with flow, using equilibrium model with Jacobi method		Boarding the first vehicle using a strategy of choosing only among attractive routes
Hickman and Bernstein, 1997 [37]		Model for high- frequency service using sequential choice approach	Deterministic utility path-choice model with passenger information
Tong and Wong, 1999 [38]		Simulation model consisting of a network of routes with a given number of departure times; using the shortest path of weighted [walk + wait + travel] time and route-change penalty	Random utility path-choice model for frequent service
Lam et al. 1999 [40]	Stochastic user equilibrium with explicit route-capacity constraints		Boarding the first vehicle; for overcrowded service, some passengers may choose alternative services
Cominetti and Correa, 2001 [41]	Congestion functions at stops obtained from queuing theory to increase waiting time and affect passenger-flow share		Boarding the first vehicle (from a set of routes) with available capacity

Advance in transit assignment research	**Frequency-based features**	**Schedule (timetable)-based features**	**Route-choice features**
Nuzzolo et al. 2001 [42]		Introducing a set of departure-time choices and a set of stop choices for a given access distance	Random utility path-choice model for frequent service with possible passenger information
Cepeda et al. 2006 [43]	Congestion functions at stops with formulation for large- scale networks		Boarding the first vehicle (from a set of routes) with available capacity

Table 4.2: Comparison of the most common shortest-path searching algorithm for time-dependent networks. Partially adopted from: [4].

5. Algorithm Implementation

The goal of the second part of this thesis is to use the knowledge gathered in the research part and to implement a particular shortest-path searching algorithm for schedule-based networks. This part contains description of the technologies used such as programming languages and database server, sources of the data and their format, description of implementation steps, and implementation results.

The implementation itself is divided into two main parts. The first part (called *data part*) deals with data selection and adjustment that lead to a database suitable for the shortest-path search. The second part (called *application part*) focuses on the development of the algorithm itself. The main aim of the impementation part is to deploy the application to the web including a graphical user interface.

The algorithm was tested on the Prague Integrated Transport System network which is the largest public transportation network in the Czech Republic. The target search query time when using my personal laptop (Lenovo U330-20268, i7-4500U , 8Gb RAM) was below one second.

The algorithm is based on the Trip Based Shortest-Path (TBSP) algorithm originally proposed by Nguyen and Palatino in 1988-1989 [33] **??** but improved in order to achieve a more efficient shortest-path search on large networks.

5.1 Technologies

A fundamental problem was the selection of technologies suitable for the implementation. A slow database server or programming language can significantly affect the overall efficiency of the algorithm.

As a database server was selected PostgreSQL 5.1.2 because it is easy to implement and use. Data part of the implementation was done in Java 8 5.1.1 which offers good database interface for PostgreSQL and is fast and also easy to use. The web application was developed in PHP 5.1.4 which also offers a reasonable PostgreSQL interface and the development with the use of PHP is fast. The PHP itself is used mainly as a control structure for SQL queries because the search on PostgreSQL server is faster than computation in PHP code.

All figures representing the algorithm computation are stored in PostGIS 5.1.3 enabled PostgreSQL database which allows to store geospatial objects. This database can be imported into QGIS 5.1.5 application and data can be presented as a map layer.

5.1.1 Java

Java is a computer programming language that is concurrent, class-based, object-oriented, and specifically designed to have as few implementation dependencies as possible. It is intended to let application developers "write once, run anywhere" meaning that code that runs on one platform does not need to be recompiled to run on another. Java applications are typically compiled to bytecode that can run on any Java virtual machine regardless of the computer architecture. Java is, as of 2014, one of the most popular programming languages in use particularly

for client-server web applications with reported 9 million developers. [22]

5.1.2 PostgreSQL

PostgreSQL is a powerful, open source, object-relational database system. It has more than 15 years of active development and a proven architecture that has earned it a strong reputation for reliability, data integrity, and correctness. It runs on all major operating systems, including Linux, UNIX, and Windows. [24]

5.1.3 PostGIS

PostGIS is a spatial database extender for PostgreSQL object-relational database. It adds support for geographic objects and allows location queries to be run in SQL. [25]

5.1.4 PHP

PHP is a server-side scripting language designed for web development but also used as a general-purpose programming language. As of January 2013, PHP was installed on more than 240 million websites and 2.1 million web servers. Originally created by Rasmus Lerdorf in 1994, the reference implementation of PHP is now produced by The PHP Group.[23]

5.1.5 QGIS

QGIS is a cross-platform, free, and open-source desktop geographic information systems application that provides data viewing, editing, and analysis capabilities. [26]

5.2 GTFS data

The General Transit Feed Specification (GTFS)[21] is a widely accepted format for an exchange of the public transportation schedule and network information associated with geographical information. It defines a common format for public transportation schedules and can be widely used by developers and other parties. GTFS feeds are published in the form of text documents with a predefined structure. There are six mandatory files and seven optional files providing detailed information about the network. Each text file contains a number of attributes some of which are in the similar manner required while some are optional. GTFS feeds are published worldwide by public transportation agencies, mainly in the USA.

The transit network consists of stops and a set of agencies which operate transit lines between them. Every agency has its set of transit lines. Each transit line of the network has its set of trips. For each trip there is a list of stops contained in the particular line with arrival and departure times. Trips and times can vary according to a calendar. Coordinates of stops and a shape of routes are not required but can be used for network or route visualization.

The list of public feeds from transportation agencies can be accessed at[27] Another website for GTFS data exchange is [28] which contains GTFS feeds uploaded by users.

Filename	Necessity	Defines
agency.txt	**Required**	One or more transit agencies that provide the data in this feed.
stops.txt	**Required**	Individual locations where vehicles pick up or drop off passengers.
routes.txt	**Required**	Transit routes. A route is a group of trips that are displayed to riders as a single service.
trips.txt	**Required**	Trips for each route. A trip is a sequence of two or more stops that occurs at specific time.
stop_times.txt	**Required**	Times that a vehicle arrives at and departs from individual stops for each trip.
calendar.txt	**Required**	Dates for service IDs using a weekly schedule. Specify when service starts and ends as well as days of the week when service is available.
calendar_dates.txt	Optional	Exceptions for the service IDs defined in the calendar.txt file. If calendar_dates.txt includes ALL dates of service, this file may be specified instead of calendar.txt.
fare_attributes.txt	Optional	Fare information for a transit organization's routes.
fare_rules.txt	Optional	Rules for applying fare information for a transit organization's routes.
shapes.txt	Optional	Rules for drawing lines on a map to represent a transit organization's routes.
frequencies.txt	Optional	Headway (time between trips) for routes with variable frequency of service.
transfers.txt	Optional	Rules for making connections at transfer points between routes.
feed_info.txt	Optional	Additional information about the feed itself including publisher, version, and expiration information.

Table 5.1: Required and optional tables in GTFS. Source: [21]

5.2.1 Required Data

The following section contains descriptions of the required components of GTFS data where text files are already imported into the PostgreSQL database.

Agency

The file agencies.txt contains basic information about agencies operating within the transit network and basic information about them.

	id [PK] serial	agency_id integer	agency_name text	agency_url text	agency_timezone text	agency_lang text	agency_phone text
1	1	1	"Dopravni podnik hl. m. Prahy	http://www.dpp.cz	Europe/Prague	cs	"296191817"
2	2	6	"Jaroslav Stepanek"	http://www.ropid.cz	Europe/Prague	cs	"284821024"
3	3	11	"CSAD POLKOST spol. s r.o."	http://www.ropid.cz	Europe/Prague	cs	"321697274"
4	4	63	"ABOUT ME s.r.o."	http://www.ropid.cz	Europe/Prague	cs	"267290540"
5	5	18	"Veolia Transport Praha s.r.o.	http://www.ropid.cz	Europe/Prague	cs	"+420234125233"

Figure 5.1: Database of agencies in GTFS data

Stops

The file stops.txt contains a list of stops within the transit network and their basic properties. Stop id is the unique identifier. More stops can have the same attribute stop name (for example stops in the opposite direction).

	id [PK] serial	stop_id text	stop_name text	stop_lat text	stop_lon text	location_type integer	parent_station text
1	1	U953Z102	"Skalka"	50.068435	14.507169	0	U953N1
2	2	U713Z102	"Strašnická"	50.073336	14.490091	0	U713N2
3	3	U921Z102	"Želivského"	50.07854	14.474891	0	U921N3
4	4	U118Z102	"Flora"	50.078288	14.461886	0	U118N4
5	5	U209Z102	"Jiřího z Poděbrad"	50.077642	14.45004	0	U209N5

Figure 5.2: Database of agencies in GTFS data

Routes

The file routes.txt contains a list of routes where each of them is a set of trips for the same route.

	id [PK] serial	route_id text	agency_id integer	route_short_ text	route_long_name text	route_type integer	route_color text
1	1	L991D1	1	A	''	1	F08000
2	2	L992D1	1	B	''	1	FFFF00
3	3	L993D1	1	C	''	1	FF0000
4	4	L1D1	1	1	''	0	0
5	5	L3D1	1	3	''	0	0

Figure 5.3: Database of agencies in GTFS data

Trips

The file trips.txt contains a list of trips for each route. Trip is a sequence of following stops.

	id [PK] serial	route_id text	service_id integer	trip_id integer	trip_headsign text	shape_id integer
1	1	L991D1	1	1	"Dejvická"	1
2	2	L991D1	1	2	"Skalka"	2
3	3	L991D1	1	3	"Dejvická"	1
4	4	L991D1	1	4	"Depo Hostivař"	3
5	5	L991D1	1	5	"Dejvická"	4

Figure 5.4: Database of agencies in GTFS data

Stop Times

The file stop_times.txt contains arrival and departure times to every station for every trip.

	id [PK] serial	trip_id integer	arrival_time text	departure_ti text	stop_id text	stop_sequen integer
1	1	1	6:37:10	6:37:40	U953Z102	1
2	2	1	6:39:55	6:40:25	U713Z102	2
3	3	1	6:42:05	6:42:35	U921Z102	3
4	4	1	6:44:00	6:44:20	U118Z102	4
5	5	1	6:45:35	6:45:55	U209Z102	5

Figure 5.5: Database of agencies in GTFS data

Calendar

The file calendar.txt contains schedule for each trip based on the service_id property.

	id [PK] serial	service_id integer	monday integer	tuesday integer	wednesday integer	thursday integer	friday integer	saturday integer	sunday integer	start_date integer	end_date integer
1	1	1	1	1	1	1	1	0	0	20130701	20130831
2	2	2	0	0	0	0	0	1	0	20130625	20130831
3	3	3	0	0	0	0	0	0	1	20130625	20130831
4	4	4	1	1	1	1	0	0	0	20130625	20130630
5	5	5	0	0	0	0	1	0	0	20130625	20130630

Figure 5.6: Database of agencies in GTFS data

5.3 Database Import

The first implementation step is to import the GTFS data consisting of .txt files into a routable PostgreSQL database. The database is an ordered set of elementary connections in groups by trip_id. The database example is shown in figure 5.7. For the purpose of the database import, I made a Java application that creates the required set of tables, reads GTFS data in the text form and imports them into the database. Once text files are imported into the tables, several queries are performed and the result is one table of all elementary connections in the network. Following queries split the table of elementary connections into tables for each calendar day, order all trips in time, and assign them with a new trip id in such a way that an earlier trip of the same route has a lower trip id. For the purpose of this thesis, I used the database of Prague Integrated Transport System where the number of elementary connections is approximately two millions.

	id integer	ordered_trip_id integer	departure_time interval	arrival_time interval	origin_name text	destination_name text	route_short_name text
1	100	33399	09:50:10	09:52:10	"Depo Hostivař"	"Skalka"	A
2	101	33399	09:52:40	09:54:55	"Skalka"	"Strašnická"	A
3	102	33399	09:55:25	09:57:05	"Strašnická"	"Želivského"	A
4	103	33399	09:57:35	09:59:00	"Želivského"	"Flora"	A
5	104	33399	09:59:20	10:00:35	"Flora"	"Jiřího z Poděbrad"	A
6	105	33399	10:00:55	10:02:10	"Jiřího z Poděbrad"	"Náměstí Míru"	A
7	106	33399	10:02:30	10:03:45	"Náměstí Míru"	"Muzeum"	A
8	107	33399	10:04:15	10:05:20	"Muzeum"	"Můstek"	A
9	108	33399	10:05:50	10:07:00	"Můstek"	"Staroměstská"	A
10	109	33399	10:07:20	10:08:25	"Staroměstská"	"Malostranská"	A
11	110	33399	10:08:45	10:10:05	"Malostranská"	"Hradčanská"	A
12	111	33399	10:10:35	10:11:55	"Hradčanská"	"Dejvická"	A
13	113	35342	10:16:15	10:17:25	"Dejvická"	"Hradčanská"	A
14	114	35342	10:17:55	10:19:05	"Hradčanská"	"Malostranská"	A
15	115	35342	10:19:25	10:20:30	"Malostranská"	"Staroměstská"	A
16	116	35342	10:20:50	10:22:00	"Staroměstská"	"Můstek"	A
17	117	35342	10:22:30	10:23:40	"Můstek"	"Muzeum"	A

Figure 5.7: Example of a routable database consisting of elementary connections

For each elementary connection, *id* is an unique identifier, *ordered_trip_id* is an ordered set of elementary connections for one trip of one vehicle along one line, and *route_short_name* is a single line on the network.

5.4 Database Segmentation

At the beginning of the algorithm development, it was difficult to reach satisfactory computation times when working with a large database. Therefore some search space reduction techniques had to be applied.

It is a generally known fact that within the Prague Integrated Transport System, a connection duration cannot exceed three hours. Therefore in order to decrease the size of the table used for routing, the original table was divided into three-hour time windows with two-hour overlaps. Any trip which has at least one elementary connection interfering with a specific hour in the time window therefore belongs to the time window. This approach increased the size of the whole database approximately three times but decreased the computation time

approximately eight times. The same approach was used for all calendar days. Monday timetable is the same as Tuesday, Wednesday and Thursday timetable so only one table set is present. Due to this approach, the result was 96 tables.

5.5 Overnight Timetable

Night routing and especially routing over midnight are probably the most difficult to be implemented due to the fact that connections before midnight (earlier connections) have times stated in larger numbers (such as 23:52) while the connections after midnight (later connections) have times stated in the form of lower numbers (such as 00:10). This fact makes it impossible for the algorithm to search for the shortest path because later connection cannot be found. Another property of the routing over midnight is that a whole timetable can change due to the change of a calendar day (for example routing from Friday night to Sunday early morning). Because of this two problems, a set of overnight tables had to be created.

- Monday to Thursday
- Thursday to Friday
- Friday to Saturday
- Saturday to Sunday
- Sunday to Monday

All the elementary connections with times over midnight had their departure and arrival times converted into the form of $hh : mm : ss$ where $hh \geq 24$ (for example 25:20:00). In this case, the early morning connection will be later then the late night connection. Example of the overnight table is in figure 5.8.

	id integer	ordered_trip_id integer	departure_time interval	arrival_time interval	origin_name text	destination_name text	route_short_name text
111298	27487	5984	23:51:15	23:52:35	"Kolbenova"	"Vysočanská"	B
111299	27488	5984	23:53:05	23:54:10	"Vysočanská"	"Českomoravská"	B
111300	27489	5984	23:54:40	23:56:10	"Českomoravská"	"Palmovka"	B
111301	27490	5984	23:56:30	23:58:00	"Palmovka"	"Invalidovna"	B
111302	27491	5984	23:58:20	23:59:40	"Invalidovna"	"Křižíkova"	B
111303	27492	5984	24:00:00	24:01:20	"Křižíkova"	"Florenc"	B
111304	27493	5984	24:01:50	24:02:45	"Florenc"	"Náměstí Republiky	B
111305	27494	5984	24:03:05	24:04:15	"Náměstí Republiky	"Můstek"	B
111306	27495	5984	24:04:45	24:07:00	"Můstek"	"Karlovo náměstí"	B
111307	27496	5984	24:07:20	24:08:50	"Karlovo náměstí"	"Anděl"	B
111308	27497	5984	24:09:20	24:10:50	"Anděl"	"Smíchovské nádraž	B

Figure 5.8: Example of an overnight routable database

These databases can be further segmented into smaller intervals like in the case of day timetables. Given that the frequency of a connection over night is low, time intervals can be wider.

5.6 Algorithm Operation

The shortest-path searching algorithm has three inputs values, therefore a search query looks as follows $Q(s_d, s_a, t_d)$ where s_d is the origin (departure station), s_a is the destination (arrival station), and t_d is the departure time and date. Date determines a day of the week and timetable validity.

The algorithm operation starts with a selection of the right routable table. Then search is expanded along all lines departing from the origin after the t_d, and the earliest arrivals to all possible destinations without transfer are stored. Search continues from all explored stations by another exploration of all accessible lines departing from already explored stations at the earliest time of the arrival of those stations.

5.6.1 Table Selection

The right table is selected by the evaluation of the departure time, current date, and weekday respectively. The table contains a three to eight-hour window of connections such as 5.4. The demanded departure time is contained in the first hour of the time window. The table contains all trips that will occur in the network in the following three hours in case of the day routing, and eight hours in case of the night routing.

5.6.2 Search

The search space expansion is transfer-driven which means that one algorithm iteration explores all possible trips along the network without a transfer from the set of explored stations at a given time.

At the beginning of the first iteration the only known station is the departure station s_d. The algorithm searches for all transit lines which depart from the origin to all directions (one transit line can have two directions), and identify the ids of transit trips where the time is minimal, greater or equal to the departure time t_d and the route name and direction are unique.

All explored nodes are stored in a result table with the earliest arrivals possible to each destination. The table has the following properties 5.2. Properties station name and arrival time are used as an input to the second iteration.

In the second iteration, all earliest transit trip ids from all previously explored stations are found. The earliest departure time is the arrival time to the explored station plus minimum transfer time which can vary according to the transfer type (tram to tram, metro to tram, etc.). The second result table has the same properties as the result table from the first iteration. The main difference is that the value of the $first\_origin$ is not the same everywhere but contains all explored stations from the previous iteration, and all previously explored stations from all iterations are excluded from possible destinations. The algorithm iterates until a stopping criterion is reached, this issue is discussed in more detail in chapter **??**.

Column name	Example	Description
first_origin_dep_time	08:33:45	Departure time from the first origin where passenger accessed given transit trip
first_origin	"Hradčanská"	Name of a station where passenger accessed given transit trip
first_destination	"Dejvická"	Name of the first explored station by the transit trip. Denotes trip direction
id	60	Unique elementary connection identifier
ordered_trip_id	25342	Unique transit trip identifier
departure_time	08:33:45	Departure time of the last elementary connection leading to the explored station
arrival_time	08:35:05	Arrival time to the explored station
origin_name	"Malostranská"	Name of the departure station of the elementary connection leading to the explored station
destination_name	"Hradčanská"	Name of the explored station
route_short_name	A	Name of the transit line

Table 5.2: Properties of algorithm operation result table

5.6.3 Stopping Criterion

Stopping criterion is an important aspect of the algorithm and several are possible to be implemented. Selection of the best one can decrease the algorithm computation time. The most basic one is to terminate at the point where the whole network is explored and no additional search is possible. In such case the optimal solution must be found but computation time can be unsatisfactory in large networks.

Another option is to terminate the calculation after a certain number of transfers (iterations). It is expected that in many cases the shortest path can contain more transfers than the minimum number of transfers to reach the destination (a path with one transfer can take less time than a path with no transfers). A limited number of transfers can be demanded by passenger, or can be fixed or variable (for example two more transfers after a solution was found). In this case, optimal solution might be omitted but optimal or close to optimal solution is expected to be found.

5.6.4 Route Backtracking

The destination is reached once it can be found in the column *destination_name* in any result table. Then backtracking can be applied to obtain the information about a route.

In the case of backtracking in a result table after the first iteration, we only need to find a table row where the *destination_name* is our desired destination and the arrival time is minimal. The column *first_origin* is our origin. This solution is an optimal solution for routes with no transfer. Although this solution can be optimal, routes with more transfers have to be explored as well because such a route can be shorter than a route without transfers.

The result of a backtracking query is a table of routes ordered by the minimum arrival time where parts of the result tables from each iteration are interconnected to complete the shortest route. The result tables are connected in such manner that the arrival station of one transit line is the departure station of the following line and the departure time is greater or equal to the arrival time plus the minimum transfer time.
In general the backtracking query result consists of a joined table where number of tables is $n + 1$ and where n is the number of transfers. Each table looks as shown in 5.3.

Column name	**Example**	**Description**
first_origin_dep_time	08:30:00	Departure time from the first origin where the passenger accessed a given transit trip
first_origin	"Kamenická"	Name of a station where the passenger accessed a given transit trip
destination_name	"Vltavá"	Name of the explored station
route_short_name	A	Name of the transit line
arrival_time	08:03:00	Arrival time to the explored station

Table 5.3: Properties of a backtracking table

5.7 Example of a Shortest-Path Search

Having the departure time $t_d = 8{:}00$ on Monday and the origin $s_d =$ "Kamenická", we need to find the shortest path to the destination $s_a =$"Kobylisy". Number of transfers is limited to two which is also our stopping criterion. The selected table for routing is Monday 8-11am.

5.7.1 First Iteration

The result table 5.9 from the first iteration lists all possible destinations and their earliest arrivals where departure station is "Kamenická", departure time is greater or equal to 8:00, and no transfers are present. The trip always starts at *first origin* "Kamenická" and the earliest arrival to *destination_name* at time *destiantion_arrival* by route *route_short_name* is on one row of the table for every *destination_name*. The destination s_d was not discovered in this iteration, so the algorithm goes on. Explored nodes together with the origin are in figure 5.10.

	first_origin_dep_time time without time zone	first_origin text	first_destination text	id integer	ordered_trip_id integer
1	08:00:00	"Kamenická"	"Letenské náměstí"	862151	19560
2	08:00:00	"Kamenická"	"Letenské náměstí"	862152	19560
3	08:00:00	"Kamenická"	"Letenské náměstí"	862153	19560
4	08:00:00	"Kamenická"	"Letenské náměstí"	876381	19560
5	08:00:00	"Kamenická"	"Strossmayerovo nám	490083	19958
6	08:00:00	"Kamenická"	"Strossmayerovo nám	490084	19958
7	08:00:00	"Kamenická"	"Strossmayerovo nám	490085	19958

	departure_time time without tin	arrival_time time without	origin_name text	destination_name text	route_short_name text
	08:00:00	08:01:00	"Kamenická"	"Letenské náměstí"	26
	08:01:00	08:02:00	"Letenské náměst	"Korunovační"	26
	08:02:00	08:03:00	"Korunovační"	"Sparta"	26
	08:03:00	08:06:00	"Sparta"	"Špejchar"	26
	08:00:00	08:02:00	"Kamenická"	"Strossmayerovo námě	12
	08:02:00	08:04:00	"Strossmayerovo	"Vltavská"	12
	08:04:00	08:05:00	"Vltavská"	"Pražská tržnice"	12

Figure 5.9: The result table for first algorithm iteration

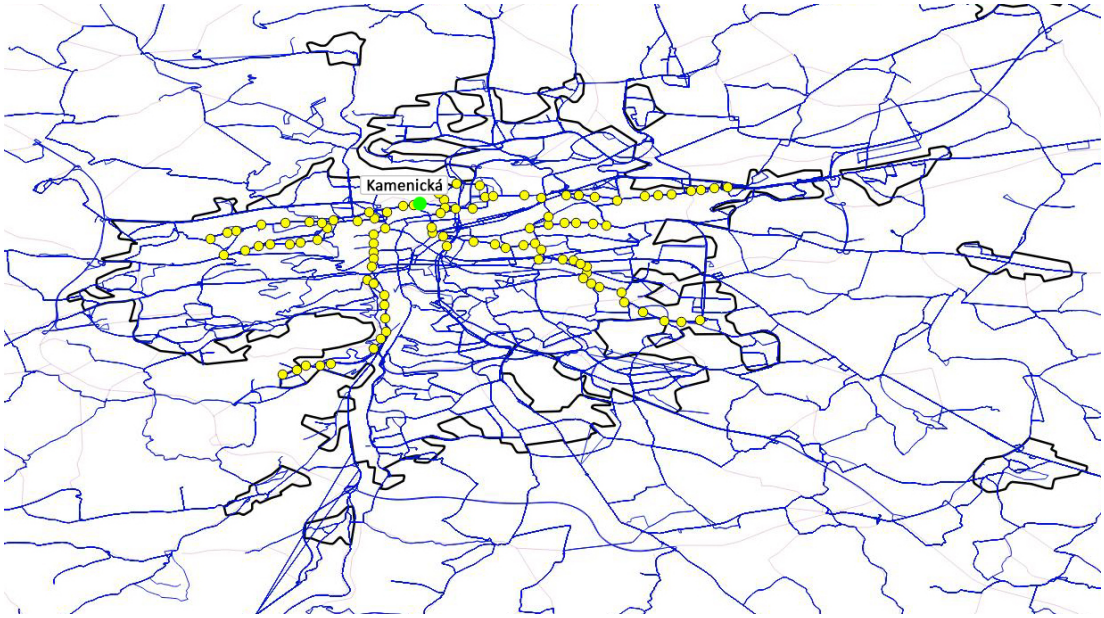


Figure 5.10: The explored stations in the first algorithm iteration

5.7.2 Second Iteration

In the second iteration all explored stations from the first iteration are selected and lines from such stations after the known arrival time plus the minimum transfer time are explored. The result table for such query 5.11 contains all explored stations from the first iteration in $first\_origin$ column and all possible destinations and arrival times in columns $destination\_name$ and $arrival\_time$.

Destination "Kobylisy" was discovered but the stopping criterion was not reached yet. So the algorithm continues with the third iteration. The explored stations are in figure 5.12

	first_origin_dep_time time without time zone	first_origin text	first_destination text	id integer	ordered_trip_id integer
948	08:25:05	"Anděl"	"Českomoravská"	33162	24615
949	08:25:05	"Anděl"	"Českomoravská"	33163	24615
950	08:25:05	"Anděl"	"Českomoravská"	33164	24615
951	08:55:30	"Hloubětín"	"Kolbenova"	33168	28596
952	08:55:30	"Hloubětín"	"Kolbenova"	33169	28596
953	08:55:30	"Hloubětín"	"Kolbenova"	33170	28596
954	08:55:30	"Hloubětín"	"Kolbenova"	33171	28596

	departure_time time without time	arrival_time time without	origin_name text	destination_name text	route_short_name text
	08:42:50	08:44:35	"Kolbenova"	"Hloubětín"	B
	08:44:55	08:46:50	"Hloubětín"	"Rajská zahrada"	B
	08:47:10	08:48:40	"Rajská zahrada"	"Černý Most"	B
	08:55:30	08:57:10	"Hloubětín"	"Kolbenova"	B
	08:57:30	08:58:50	"Kolbenova"	"Vysočanská"	B
	08:59:20	09:00:25	"Vysočanská"	"Českomoravská"	B
	09:00:55	09:02:25	"Českomoravská"	"Palmovka"	B

Figure 5.11: The result table for the second algorithm iteration

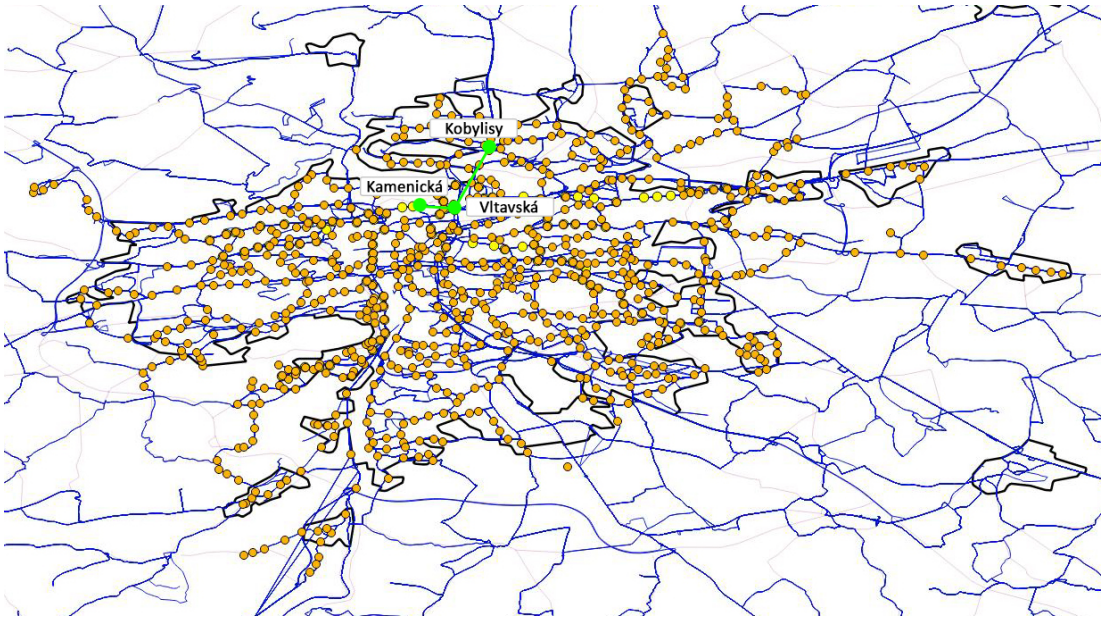


Figure 5.12: The explored stations in the second algorithm iteration

5.7.3 Third Iteration

The third iteration works in a similar manner as the second iteration. Stations found in the first and second iteration are excluded from the possible destinations and the result table has the same structure as the result table from the second iteration 5.11. The explored stations are shown in figure 5.13. The destination "Kobylisy" was found again and the stopping criterion was reached, the algorithm therefore terminates the searching phase and continues with the route backtracking.

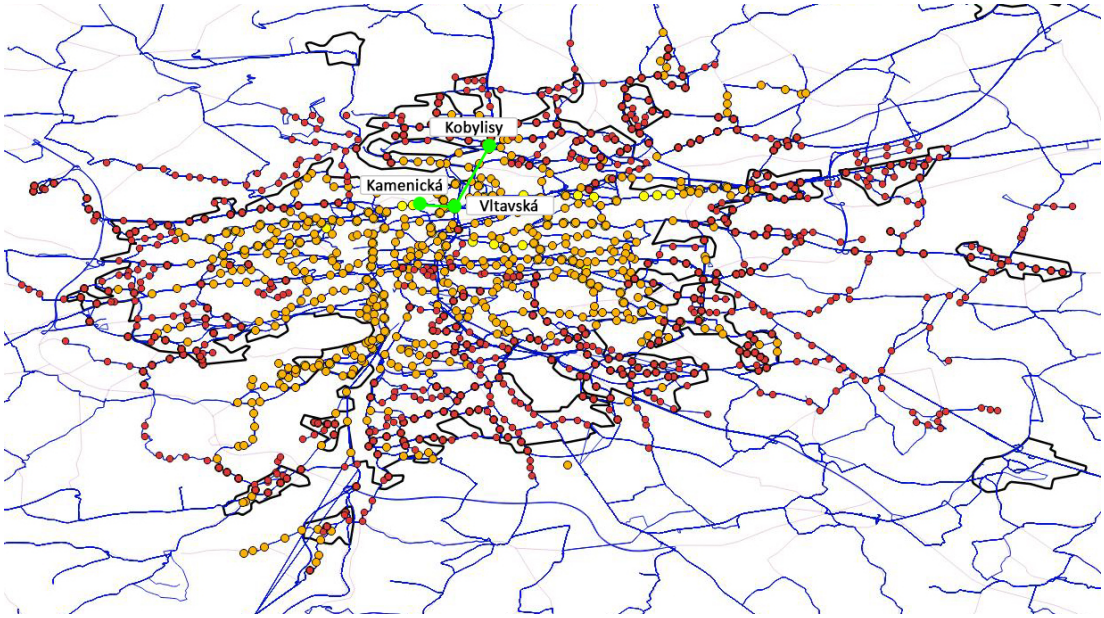


Figure 5.13: The explored stations in the third algorithm iteration

5.7.4 Backtracking

In this example, the algorithm has performed three iterations before the termination. For different networks, the number of iterations can vary. At first, the algorithm checked whether the destination was found in every result table. If it was not found, this table was skipped. The result tables are examined successively.

The destination was not reached in the first result table so the algorithm immediately started searching in the second result table. The destination was found in the second result table, a database query searched through both the first and second result tables and found the shortest path. Similar scenario was repeated for the third result table. A search through all three result tables was performed and a list of the shortest paths with two transfers was returned by the query.

Shortest Routes

In the second backtracking step, the query result over two result tables looks as 5.14

	first_origin_dep_time time without time zone	first_origin text	destination_name text	route_short_name text	arrival_time time without time zone
1	08:00:00	"Kamenick	"Vltavská"	12	08:04:00
2	08:03:00	"Kamenick	"Nádraží Holešo	1	08:10:00
3	08:01:00	"Kamenick	"Hlavní nádraží	26	08:12:00

	first_origin_dep_time time without time zone	first_origin text	destination_name text	route_short_name text	arrival_time1 time without time zone
	08:06:20	"Vltavská	"Kobylisy"	C	08:10:45
	08:12:10	"Nádraží	"Kobylisy"	C	08:16:35
	08:14:40	"Hlavní n	"Kobylisy"	C	08:22:45

Figure 5.14: The result table for the first algorithm iteration

In the third backtracking step the query result over three result tables looks as 5.15

	first_origin_dep_time time without time zone	first_origin text	destination_name text	route_short_name text	arrival_time time without time zone
1	08:08:00	"Kamenick	"Nádraží Libeň"	25	08:30:00
2	08:08:00	"Kamenick	"Nádraží Libeň"	25	08:30:00

	first_origin_dep_time time without time zone	first_origin text	destination_name text	route_short_name text	arrival_time time without time zone
	08:31:00	"Nádraží	"Mirovická"	183	08:50:00
	08:36:00	"Nádraží	"Písečná"	177	08:55:00

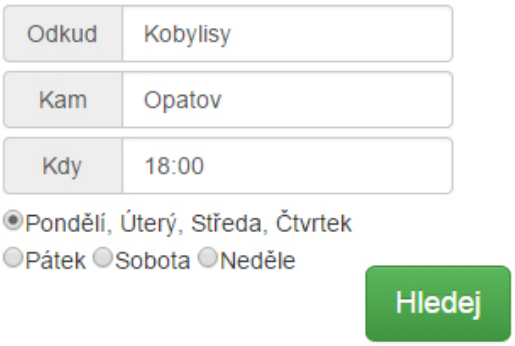
	first_origin_dep_time time without time zone	first_origin text	destination_name text	route_short_name text	arrival_time3 time without time zone
	08:50:00	"Mirovická"	"Kobylisy"	162	08:53:00
	08:57:00	"Písečná"	"Kobylisy"	152	08:59:00

Figure 5.15: The result table for the first algorithm iteration

From both query results 5.14 and 5.15 it is obvious that the shortest path consist of two trips by the tram number 12 and the metro line C with a transfer at the station "Vltavská". The computation time was 0.6 seconds.

5.8 Web-Based Implementation

The shortest-path algorithm and its functions can be tested in a web application. It consists of user inputs (origin, destination, earliest departure time, departure day) and space for discovered routes. They are sorted according to the number of transfers where the shortest path with a certain number of transfers is listed as first. A graphical user interface of the application is shown in figure 5.16.



Stanice	Příjezd	Odjezd	Linka
Kobylisy		18:00	C
Opatov	18:28		
Kobylisy		18:05	177
Opatov	18:57		
Kobylisy		18:06	10
Flora	18:30	18:32	136
Opatov	18:42		
Kobylisy		18:00	C
Háje	18:30	18:42	213
Opatov	18:47		
Kobylisy		18:00	C
Háje	18:30	18:43	165
Opatov	18:49		
Kobylisy		18:00	C
Budějovická	18:21	18:24	170
Chodovec	18:38	18:38	213
Opatov	18:42		

Figure 5.16: GUI of the web-based shortest-path finder

5.9 Algorithm Performance

The algorithm performance was tested in several scenarios where the shortest-path search was performed between random stations in the Prague Integrated Transit System at random times and weekdays. For this purpose I have developed a Java application that randomly selects the origin, destination, departure time and a weekday. The algorithm was tested with those inputs and the results were statistically evaluated.

Origin	**Destination**	**Departure day and time**	**Query time [ms]**
Bolevecká	Krystalová	Su 22:02	513
Vozovna Žižkov	Klíčov	Mo 1:29	507
Liběchovská	Královský letohrádek	Sa 21:46	341
Lotyšská	Kudrnova	Fr 18:45	623
Nemocnice Bubeneč	Kovářova	Pa 0:27	364
Geologická	Dobratická	Tu 8:29	677
Přívozní	Holoubkovská	Tu 8:51	440
Krakov	Strossmayerovo náměstí	Tu 12:16	629
Za Mototechnou	Ciolkovského	Th 12:55	459
Jenerálka	U Lípy	Tu 13:00	444
Miškovice	Kolonie	We 19:18	383
Svatoplukova	Nemocnice Krč	Su 15:47	373
Lochkov	Pod pekařkou	Su 22:11	296
Sídliště Na Groši	Kačerov	Mo 16:48	617
Pod Vinicí	Michelangelova	Tu 3:39	517
Olšanské náměstí	Křižíkova	We 1:14	588

Table 5.4: The test results of the algorithm

Average query time in the test sample 5.4 when using laptop Lenovo U330-20268, i7-4500U , 8Gb RAM was 486 ms with a standard deviation of 117 ms. The goal was to reach query time below one second which means that the query time satisfies this goal.

Conclusion

The main aim of this master's thesis was to gather all relevant information from the field of the shortest-path searching algorithms on time-dependent networks and implement such an algorithm for large transit networks.

In the first part of the thesis including chapters 1, 2, 3 and 4, all theoretical prerequisites were stated and discussed. This chapter includes an overview of the previous research on the topic of routing on road networks, description of transit networks and timetable information, and deals with different transit network modeling approaches. As each transit network includes a lot of routing problems, the most frequent problems had to be stated and possible solutions were proposed. The most attention was paid to the issue of the earliest arrival problem which is the fundamental problem in transit networks.

The theoretical part also provides description of different shortest-path searching algorithms developed and proposed since the mid-20th century. The most significant algorithms including pseudocodes are described in more detail. Selected techniques for the algorithm speedup and reduction of search space are also discussed in this section as they can significantly improve the algorithm overall efficiency. Selected algorithms of the highest importance are listed and compared in one table together with their main features.

The second part of the master's thesis represented by chapter 5 is focused on the implementation of one shortest-path algorithm for time-dependent networks. This algorithm is based on the Trip Based Shortest-Path algorithm (TBSP) and improved for a search on large transit networks. The implementation is divided into a data and application part. The data part of the implementation is developed in Java using PostgreSQL database and PostGIS extension for data visualization. A routable database was created from the GTFS text files by database import and several SQL queries. The goal of the application part was to develop a web application written in PHP which would allow the shortest path search using a simple user interface, and can be deployed on a server. Average query time of 486 ms when routing on the Prague Integrated Transport System network exceeded the original goal by more than 500 ms. The application performs the calculation and displays a list of the possible routes ordered by the minimum trip time.

The proposed algorithm is currently adjusted for a shortest-path search in transit network within a city because the maximum connection length is limited. It is because of the computation speed optimization where the size of the routable table had to be minimized. The intercity routing would be also possible with several adjustments. The main difference in such case would be the necessity to deal with the range problem. For an intercity transit it is usually not sufficient to find only the first shortest connection and it requires to search for all possible connections within a time interval such as one day or several days. Also, the maximum connection length would have to be extended.

In the future, the algorithm performance should be tested by a high number of requests in a short time interval. The algorithm and database can be further optimized especially by the application of different speedup and search space reduction techniques.

List of References

[1] LAM, W..; BELL, M. : *Advanced Modeling For Transit Operations and Service Planning*, Emerald, Inc. , 2008

[2] BAST,H. and DELLING, D. and GOLDBERG, A.V. and MÜLLER-HANNEMAN,M. and PAJOR, T. and SANDERS, P. and WAGNER D. and WERNECK, R.F. *Route Planning in Transportation Networks*, Microsoft Research , 2014

[3] MÜLLER-HANNEMAN, M. and SCHULZ, F. and WAGNER, D. and ZAROLIAGIS, C. *Timetable information: Models and algorithms. In Algorithmic Methods for Railway Optimization*, volume 4359 of Lecture Notes in Computer Science, pages 67–90. Springer, 2007.

[4] CEDER, A. *Public Transit Planning and Operation, Theory, Modeling and Pracise* ,Elsevier Ltd., London, UK, 2007

[5] KHANI, A. *Models and Solutions Algorithms for Transit and Intermodal Passenger Assignment (Development of Fast-trip Model)*, University of Arizona , 2014

[6] DELLING, D and KATZ, B. and PAJOR, T. *Parallel Computation of Best Connections in Public Transportation Networks*, IEEE , 2010

[7] SHULZ, F. and WAGNER, F. and WEIHE, K. *Dijkstra's algorithm on-line: An empirical case study from public railroad transport.* Journal of Experimental Algorithmics, 5(12) 2000

[8] MÜLLER-HANNEMAN, M. and SCHNEE, M. *Finding all attractive train connections by multi-criteria Pareto search.* Proceedings of the 4th Workshop in Algorithmic Methods and Models for Optimization of Railways (ATMOS 2004), To appear in the same volume (2004)

[9] PYRGA, E. and SCHULZ, F. and WAGNER D. and ZAROLIAGIS, C. *Experimental comparison of shortest path approaches for timetable information.* In Proceedings of the 6th Workshop on Algorithm Engineering and Experiments (ALENEX'04), pages 88–99. SIAM, 2004

[10] MORITZ, B.*On Preprocessing the Arc-Flags Algorithm.* Karlsruhe : Karlsruhe Institute of Technology, 2011

[11] A-star algorithm [online].[cit. 2014-10-10]. Accesible at: http://theory.stanford.edu/ amitp/game-programming/a-star/a-star.png

[12] MÜLLER-HANNEMAN, M. and SCHNEE, M. *Finding all attractive train connections by multi-criteria pareto search.* In Algorithmic Methods for Railway Optimization, volume 4359 of Lecture Notes in Computer Science, pages 246–263. Springer, 2007

[13] PYRGA, E. and SCHULZ, F. and WAGNER D. and ZAROLIAGIS, C. *Efficient models for timetable information in public transportation systems.* ACM Journal of Experimental Algorithmics, 12(2.4):1–39, 2008.

[14] SCHULZ, F. and WAGNER D. and WEIHE, K. *Dijkstra's algorithm online: An empirical case study from public railroad transport.* ACM Journal of Experimental Algorithmics, 5(12):1–23, 2000.

[15] BRODAL, G.S. and JACOB, R. *Time-dependent networks as models to achieve fast exact time-table queries.* Proceedings of the 3rd Workshop on Algorithmic Methods and Models for Optimization of Railways (ATMOS 2003). Electronic Notes in Theoretical Computer Science, vol. 92, Elsevier, Amsterdam 2004

[16] PYRGA, E. and SCHULZ, F. and WAGNER D. and ZAROLIAGIS, C. *Towards realistic modeling of time-table information through the time-dependent approach.* Proceedings of the 3rd Workshop on Algorithmic Methods and Models for Optimization of Railways (ATMOS 2003). Electronic Notes in Theoretical Computer Science, vol. 92, pp. 85–103. Elsevier, Amsterdam 2004

[17] DISSER, Y and MÜLLER-HANNEMAN, M. and SCHNEE, M. *Multi-criteria shortest paths in time-dependent train networks.* In Proceedings of the 7th Workshop on Experimental Algorithms (WEA'08), volume 5038 of Lecture Notes in Computer Science, pages 347–361. Springer, June 2008

[18] BAUER, R. and DELLING, D. and WAGNER, D. *Experimental study on speed-up techniques for timetable information systems.* Networks, 57(1):38–52, January 2011

[19] NIELSEN, O. and FREDERIKSEN, R. *Large-Scale Schedule-Based Transit Assignment - Further Optimization of the Solution Algorithms.* Technical Univerity Denmark, 2010

[20] Dijkstra algorithm schema [online].[cit. 2014-10-14]. Accesible at: http://research.microsoft.com/en-US/news/features/images/dijkstra.png /

[21] GTFS specifications [online].[cit. 2014-10-09]. Accesible at: https://developers.google.com/transit/gtfs/

[22] Java programming language [online].[cit. 2014-10-20]. Accesible at: http://en.wikipedia.org/wiki/Java_(programming_language)

[23] PHP programming language [online].[cit. 2014-10-20]. Accesible at: http://php.net/

[24] PostgrSQL quering language and database server [online].[cit. 2014-10-20]. Accesible at: http://www.postgresql.org/

[25] PostGIS geospatial database extension [online].[cit. 2014-10-20]. http://postgis.net/

[26] QGIS software [online].[cit. 2014-10-25].
http://en.wikipedia.org/wiki/QGIS

[27] Google transit data feed [online].[cit. 2014-10-02].
https://code.google.com/p/googletransitdatafeed/wiki/PublicFeeds

[28] GTFS data exchange [online].[cit. 2014-10-02].
http://www.gtfs-data-exchange.com/

[29] Dial, R. B. *Transit pathfinder algorithm.* Highway Research Board. 1967, 205, 67-85, 1967

[30] le Clercq, F. A public transport assignment model. *Traffic Engineering & Control.* 1972 14(1), 91-96.

[31] Chriqui, C., and Robillard, P. (1975). Common bus lines. *Transportation Science.* 1975 9(2), 115-121.

[32] TONG, C. O. and RICHARDSON, A. J. A computer model for finding the time-dependent minimum path in a transit system with fixed schedules. *Journal of Advanced Transportation.* 1984, 18, 145–161

[33] NGUYEN, S., and PALLOTINO, S. Equilibrium traffic assignment for large scale transit networks. *European Journal of Operational Research.* 1988, 37(2), 176-186.

[34] SPIESS, H. and FLORIAN, M. Optimal strategies: a new assignment model for transit networks. *Transportation Research.* 1989 23B (2), 83-102.

[35] DE CEA, J. and FERNANDEZ, E. Transit assignment for congested public transport systems: an equilibrium model. *Transportation Science.* 1993, 27(2), 133-147.

[36] Wu, J. H. and FLORIAN, M. and MARCOTTE, P. Transit equilibrium assignment: a model and solution algorithms. *Transportation Science.* 1994, 28(3), 193-203.

[37] HICKMAN M. and BERNSTEIN D. Transit service and path choice models in stochastic and time-dependent networks. *Transportation Science.* 1997, 31, 129-146.

[38] TONG, C. O. and WONG, S. C. A stochastic transit assignment model using a dynamic schedule-based network. *Transportation Research.* 1999, 33B (2), 107-121

[39] TONG, C. O. and Richardson, A. J. A computer model for finding the time-dependent minimum path in a transit system with fixed schedules. *Journal of Advanced Transportation.* 1984, 18, 145–161.

[40] LAM, W. H. K. and CHUNG-YU, C. and LAM, C. F. A study of crowding effects at the Hong Kong light rail transit stations. *Transportation Research.* 1999 33A (5), 401-415.

[41] COMINETTI, R. and CORREA, J. Common-lines and passenger assignment in congested transit networks. *Transportation Science.* 2001, 35(3), 250-267.

[42] NUZZOLO, A. and RUSSO, F. and Crisalli, U. A doubly dynamic schedule-based assignment model for transit networks. *Transportation Science.* 2001 35(3), 268-285

[43] CEPEDA, M. and COMINETTI, R. and FLORIAN, M. A frequency-based assignment model for congested transit networks with strict capacity constraints: characterization and computation of equilibria. *Transportation Research.* 2006 40B (6), 437-459.

[44] WHITE, H. *An Overview of Representation and Convergence results for Multilayer feed-forward Networks.* 1991

[45] HART, J. *Computer Approximations.* London, 1968.

[46] GAO, S. and CHABINI, I. . The best routing policy problem in stochastic time-dependent networks.*Transportation Research Record: Journal of the Transportation Research Board, No.1783, Transportation Research Board of the National Academies, Washington, D.C.,* 2002 pp.188-196.

[47] HAMDOUCH, Y., and LAWPHONQPANCHI, S. Schedule-based transit assignment model with travel strategies and capacity constraints. *Transportation Research.* 2008 42B (7), 663-684.

[48] BOUZAIENE-AYARI, B. and GENDREAU, M., and NGUYEN, S. *An Equilibrium-Fixed Point Model for Passenger Assignment in Congested Transit Networks.* Montreal: Universite de Montreal. 1995

List of Tables