Department of Computer Science,
Faculty of Mathematics, Physics and Informatics,
Comenius University in Bratislava

# Distance oracles for timetable graphs

(Master thesis)

**bc. František Hajnovič**

**Study program**: Computer science
**Branch of study**: 2508 Informatics
**Supervisor**: doc. RNDr. Rastislav Královič, PhD.

Bratislava 2013

65787891

Univerzita Komenského v Bratislave
Fakulta matematiky, fyziky a informatiky

# PRIHLÁŠKA NA ZÁVEREČNÚ PRÁCU

**Meno a priezvisko študenta:** Bc. František Hajnovič
**Študijný program:** informatika (Jednoodborové štúdium, magisterský II. st., denná forma)
**Študijný odbor:** 9.2.1. informatika
**Typ záverečnej práce:** diplomová
**Jazyk záverečnej práce:** anglický

**Názov:** Distance oracles for timetable graphs

**Cieľ:** The aim of the thesis is to explore the applicability of results about distance oracles to timetable graphs. It is known that for general graphs no efficient distance oracles exist, however, they can be constructed for many classes of graphs. Graphs defined by timetables of regular transport carriers form a specific class which it is not known to admit efficient distance oracles. The thesis should investigate to which extent the known desirable properties (e.g. small highway dimension) are present int these graphs, and/or identify new ones. Analytical study of graph operations and/or experimental verification on real data form two possible approaches to the topic.

**Vedúci:** doc. RNDr. Rastislav Královič, PhD.
**Katedra:** FMFI.KI - Katedra informatiky

**Dátum schválenia:** 15.11.2011

podpis študenta

I hereby declare that I wrote this thesis by myself, only with the help of the referenced literature, under the careful supervision of my thesis advisor.

............................

# Acknowledgements

I would like to thank ...

Abstract

This thesis...

Key words: **oracles**, **timetable**

Abstrakt

V tejto práci...

Klúčové slová: **oracles**, **timetable**

# Contents

# 1 Introduction

World is getting smaller every day...

## 1.1 Motivation

## 1.2 Approach

## 1.3 Goals

## 1.4 Organization

# 2 Preliminaries

In this section, we would like to provide the definitions and terminology used throughout the thesis.

## 2.1 Objects

First, we will formalize the notion of a timetable and its derived graph forms, the underlying graph and notions related to these terms.

**Definition 2.1.** ***Timetable (TT)***
*A timetable is a set* $\boldsymbol{T} = \{(x, y, p, q)|\ p, q \in \mathbb{N},\ p < q\}$.

- *Elements of $T$ (the 4-tuples) are called* ***elementary connections****. For an elementary connection $e = (x, y, p, q)$:*
  - $\boldsymbol{from(e)} = x$ *is the* ***departure city***
  - $\boldsymbol{to(e)} = y$ *is the* ***arrival/destination city***
  - $\boldsymbol{dep(e)} = p$ *is the* ***departure time***
  - $\boldsymbol{arr(e)} = q$ *is the* ***arrival time***
- *Pairs $(x, p)$ or $(y, q)$ such that $(x, y, p, q) \in T$ form the set of* ***events*** $\boldsymbol{ev_T}$*. The set of events in a specific city $x$ is* $\boldsymbol{ev_T(x)} = \{(x, t)|(x, y, t, q) \in T \text{ or } (y, x, p, t) \in T\}$
- *Let* $\boldsymbol{tlow_T} = \min_{e \in T} dep(e)$ *and* $\boldsymbol{thigh_T} = \max_{e \in T} arr(e)$*. The value* $\boldsymbol{trange_T} = thigh_T - tlow_T$ *is called the* ***time range*** *of the timetable.*
- ***Height*** *of the timetable is the maximum number of events in a city:* $\boldsymbol{height_T} = \max_{x \in cities_T}\{|ev_T(x)|\}$

Let us describe some the defined terms more informally. An elementary connection corresponds to moving from one stop to the next one, e.g. with a bus (thus we disregard the notion of *lines* in our timetables). Note that we express time as an integer - throughout this paper, this integer will represent the minutes elapsed from the time 00:00 of the first day. Thus we may take the liberty of talking about time in integer or *days hh:mm* format, as convenient at the moment. Lastly, an event simply represent an arrival or departure of a e.g. train at some station. The remaining terms should be clear enough.

| **Place** | | **Time** | |
|---|---|---|---|
| **From** | **To** | **Departure** | **Arrival** |
| A | B | 10:00 | 10:45 |
| B | C | 11:00 | 11:30 |
| B | C | 11:30 | 12:10 |
| B | A | 11:20 | 12:30 |
| C | A | 11:45 | 12:15 |

Table 2.1: An example of a timetable - the set of elementary connections (between pairs of cities). An example of an event is a pair (A, 10:00)

Following is a definition of a connection.

**Definition 2.2.** ***Connection***
*A connection from $a$ to $b$ is a sequence of elementary connections* $\boldsymbol{c} = (e_1, e_2, ..., e_k), k \geq 1$*, such that* $from(e_1) = a$*,* $to(e_k) = b$ *and* $\forall i \in \{2, ..., k\} : (to(e_i) = from(e_{i-1}),\ arr(e_i) \geq dep(e_{i-1})$.

- *Connection* ***starts*** *at the departure time* $\boldsymbol{start(c)} = dep(e_1)$ *and* ***ends*** *at the arrival time* $\boldsymbol{end(c)} = arr(e_k)$.
- ***Length*** *of the connection is* $\boldsymbol{len(c)} = end(c) - start(c)$.
- *We will denote the set of* ***all connections*** *from $a$ to $b$ in a timetable $T$ as* $\boldsymbol{C_T(a, b)}$*. We also define* $\boldsymbol{C_T} = \cup_{a,b} C_T(a, b)$

So we understand connection as a (valid) sequence of elementary connections.
Next, we continue with the underlying graph - a graph representing basically the map on top of which the timetable operates.

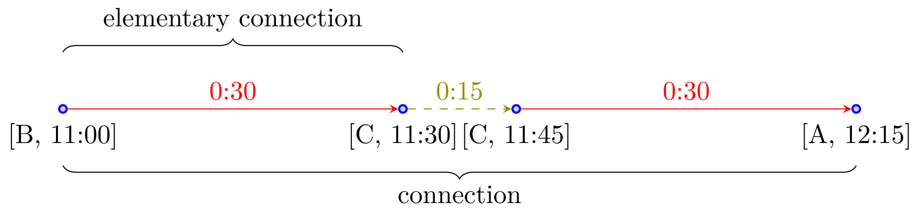


Figure 2.1: A valid connection made out of elementary connections (and waiting, which is implicit)

**Definition 2.3.** ***Underlying graph (UG graph)***
*The underlying graph of a timetable $T$, denoted $\boldsymbol{UG_T}$, is an oriented graph $G = (V, E)$, where $V$ is the set of all timetable cities and $E = \{(x, y) |\ \exists (x, y, p, q) \in T\}$*

Note, that we do not specify the weights of the edges in the underlying graph - they will be specified based on the current usage of the UG. Most of the time, however, we will work with UG where the weight of each arc is the length of the shortest elementary connection on that arc. More specifically, $w(x, y) = \min_{(x,y,p,q) \in T}(q - p)\ \forall (x, y) \in E(UG_T)$.

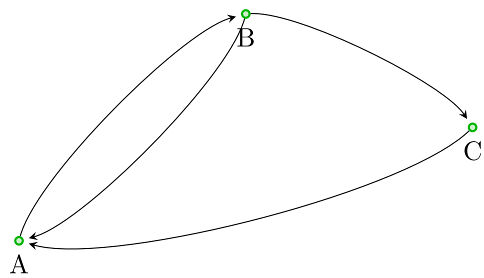


Figure 2.2: Underlying graph of the timetable in picture 2.1. The nodes are the cities

If we want to represent the timetable itself by a graph, there are two most common options [**?**].

**Definition 2.4.** ***Time-expanded graph (TE graph)***
*Let $T$ be a timetable. Time-expanded graph from $T$, denoted $\boldsymbol{TE_T}$, is an oriented graph $G = (V, E)$ whose vertices correspond to events of $T$, that is $V = \{[x, t] | (x, t) \in Ev_T\}$. The edges of $G$ are of two types*

1. *$([x, p], [y, q])\ \forall (x, y, p, q) \in T$ - the so called **connection edges***
2. *$([x, p], [x, q])\ [x, p], [x, q] \in V,\ p < q$ and $\nexists [x, r] \in V : p < r < q$. - the so called **waiting edges***

*Weight of the edge $([x, p], [y, q])$ is $w([x, p], [y, q]) = q - p$.*

Informally, an edge in TE graph represent either the travelling with an elementary connection or waiting for the next event in the same city. Also, the time range and height of a timetable could be easily illustrated on the TE graph (see picture 2.3).

**Definition 2.5.** ***Time-dependent graph (TD graph)***
*Let $T$ be a timetable. Time-dependent graph from $T$, denoted $\boldsymbol{TD_T}$, is an oriented graph $G = (V, E)$ whose vertices are the timetable cities and $E = \{(x, y) |\ \exists (x, y, p, q) \in T\}$. Furthermore, the weight of an edge $(x, y) \in E$ is a piece-wise linear function $w(x, y) = f_{x,y}(t) = q - t$ where $q$ is:*

- $\min\{arr(e) | e \in T,\ dep(e) \geq t\}$
- $\infty$, *if* $dep(e) < t\ \forall e \in T$

Intuitively, the TD graph is simply the UG graph where each arc carries a function specifying the traversal time of that arc at any time. For an example, see picture 2.5: The latest point of every linear segment is called the **interpolation point** and it corresponds to immediate taking of an elementary connection. Note that a list of all interpolation points defines the piece-wise linear function

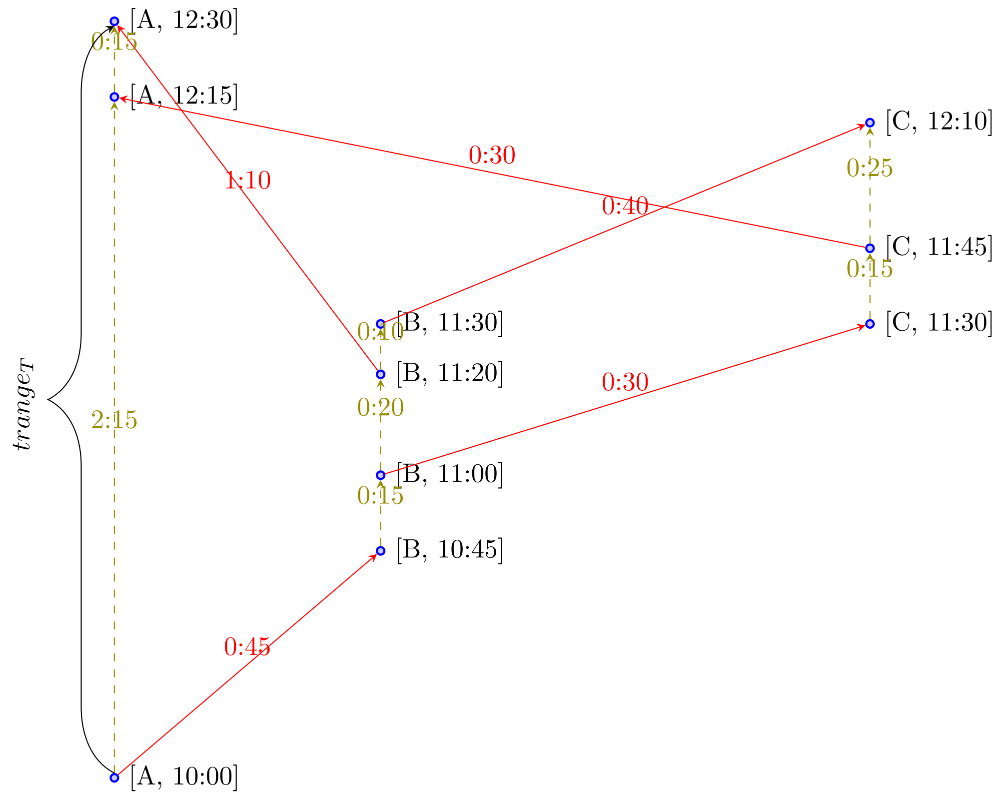


Figure 2.3: Time-expanded graph of the timetable in picture 2.1. Nodes represent the events. There are connection and waiting edges (dashed). The time range is 2h:30m and the height is 4 (as there are 4 events in city B)

completely.

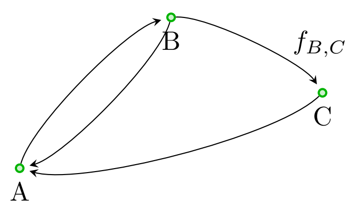


Figure 2.4: Time-dependent graph of the timetable in picture 2.1. The nodes are the cities

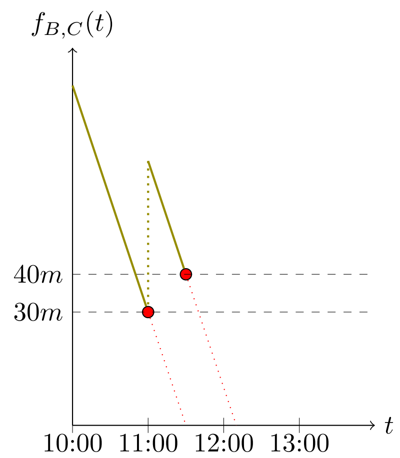


Figure 2.5: Piece-wise linear function - traversal times for the arc $(B, C)$

To sum up, there are four main types of objects we will be working with:

- Timetable (TT)
- Underlying graph (UG)
- Time-expanded graph (TE)

- Time-dependent graph (TD)

For further reference, we will call **timetable objects** those, that fully represent a timetable (TT, TE, TD) and **graph objects** those, that can be viewed as a graph (UG, TE, TD).

*Note:* Throughout this paper, we will relax a bit the notation and leave out subscripts (e.g. $UG_T \rightarrow UG$ in situations, where the context is clear enough.

## 2.2 Earliest arrival

Now we would like to formulate the main problems this thesis deals with.

**Definition 2.6.** ***Earliest arrival problem (EAP)***
*Given a timetable $T$, departure city $x$, destination city $y$ and a departure time $t$, the task is to determine $\boldsymbol{t^*_{(x,t,y)}} = \min_{c \in C_T(x,y)}\{t + len(c) | start(c) \geq t\}$.*

- *We will refer to the tuple $\boldsymbol{(x, t, y)}$ as an **EAP instance**, or an **EAP query***
- *The time $\boldsymbol{t^*_{(x,t,y)}}$ is called the **earliest arrival (EA)** for the given EAP instance*

A bit more difficult version of this problem is one, where we require to actually output the connection ending at time given by EA.

**Definition 2.7.** ***Optimal connection problem (OCP)***
*Given a timetable $T$, departure city $x$, destination city $y$ and a departure time $t$, the task is to determine the **optimal connection (OC)** $\boldsymbol{c^*_{(a,t,b)}} = argmin_{c \in C_T(a,b)}\{t + len(c) | start(c) \geq t\}$.*

The instance/query in case of the optimal connection problem has the same form as EAP query. Also, note that the OCP is at least as hard to solve as EAP since having the optimal connection implies the optimal (earliest) arrival time.. In order to avoid technical issues in later parts of the thesis, we will assume the optimal connection is unique (i.e., there is not a different connection with the same end time) or that ties are won by a lexicographically first connection.

**Example 2.1.** *Consider our timetable from table 2.1. For the EAP instance $(B, 10{:}45, A)$, the earliest arrival (EA) is 12:15 and the optimal connection (OC) is $((B, C, 11{:}00, 11{:}30), (C, A, 11{:}45, 12{:}15))$, as could be easily seen from picture 2.6 of the TE graph.*

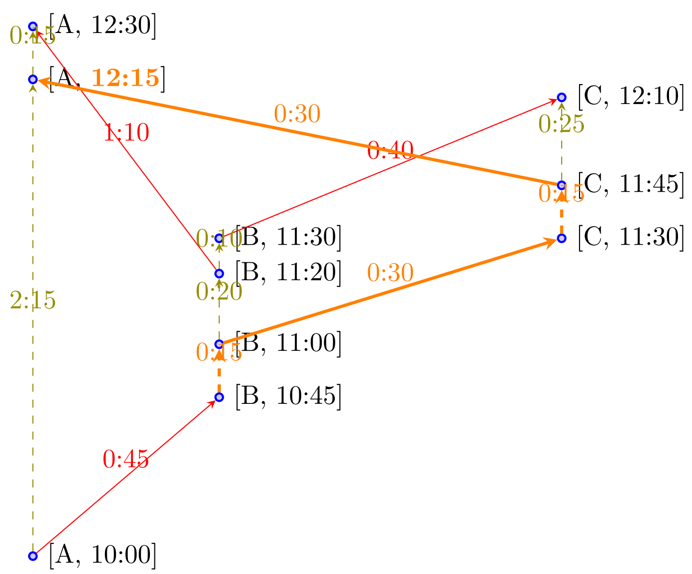


Figure 2.6: Optimal connection and earliest arrival time are marked in **bold**

## 2.3 (Distance) Oracles

The term distance oracle was first coined in 2001 by Thorup and Zwick [**?**], when talking about quick shortest path (or distance) computations on graphs. One approach to this problem is to pre-compute some information on the graph to speed-up answering of the queries. The paper of Thorup and Zwick was dealing with trade-offs between time of the pre-computation, the amount of pre-computed information, the speed-up in query times and the accuracy of the provided answers. Since the pre-computed data structure is something that helps us answer the queries more efficiently, it resembles an oracle, thus the term distance oracle.
Since then,

# 3 Related work

# 4 Data & analysis

In this section we would like to introduce the timetable datasets we were working with and provide the results of the analysis which we carried out on the data. The main reason for this analysis is that it gives some insight into the properties of the timetables, and thus may contribute to the make an oracle based method with better qualities.

## 4.1 Data

We have obtained timetable datasets from numerous sources, in varying formats and of different types. Some of them were freely available on the Internet while others were provided by companies upon demand. Let us briefly describe each of these timetables.

The dataset *air01* contains schedules of **domestic flights in United States** for the January of 2008. It is not comprehensive in the sense that it contains entries only for flights of some of the major airports in US. However it is large enough for our purposes (almost 300 airports). This dataset is just a fraction of the data that are freely available at the pages of American Statistical Association [1] in CSV format.

Timetables *cpru* and *cpza* represent the **regional bus** schedules from the areas of **Ružomberok and Žilina, Slovakia**. The data were provided by the company in charge of the *cp.sk* portal - Inprop s.r.o. . Both of the timetables concern about 1000 bus stops and came in a JDF 1.9 format [2]. Apart from the actual schedules, the data in JDF contain numerous other information, which were not relevant for our purposes. From both timetables, we have extracted subsets with a time range of one day.

The *montr* dataset is part of a public feed for Greater **Montreal public transportation**, available at Google Transit Feeds [3]. The data are in a GTFS format (defines relations between CSV files listing stations, routes, stop-times...) and were made available by Montreal's Agence métropolitaine de transport. Our timetable *montr* corresponds to daily schedules of the Chambly-Richelieu-Carignan bus services (more than 200 bus stops).

Also in GTFS format come the data of **French railways** operated by company SNCF, publicly available at their website [4]. The schedules are weekly, but we have extracted just a sub-range corresponding to Monday. Also, there were two types of schedules: one for intercity trains and one for TER trains (regional trains). Thus the three timetables *sncf-inter* (366 stations), *sncf-ter* (2637 stations) and their union *sncf* (2646 stations).

Finally, one more country-wide railway timetable was provided by ŽSR, the company in charge of the **Slovak national railways**. This timetable was exported in a MERITS format and its time range is for one year. The number of stations in *zsr* dataset is 233.

With the help of Python and Bash scripts, we converted each of these datasets to our timetable format (described in appendix A). This timetables were then loaded by our application TTBlazer and sub-timetables (with less stations, smaller time-range or smaller height) were generated. Also the UG, TE and TD were generated from each timetable.

For a summary of the used timetables' descriptions, see table 4.1 and for their main properties, refer to table 4.2.

Some of the timetables have time range greater then 1 day. Furthermore, even for those marked as a 1 day timetable the exact time range is different (e.g., part of the Monday timetable might be some overnight trains with arrival on Tuesday morning). To see better the differences in the properties of different timetable types (train, flight, bus...), we made sub-timetables with 200 cities and with the upper bound on time range 1 day, 6 hours [5] ($thigh_T$ <1 day, 6h $\forall T$) from each of our dataset. See

[1] `http://stat-computing.org/dataexpo/2009/the-data.html`
[2] Jednotný dátový formát (JDF)
[3] `http://code.google.com/p/googletransitdatafeed/wiki/PublicFeeds`
[4] `http://test.data-sncf.com/index.php/ter.html`
[5] We took all elementary connections that were within our time range. From this timetable, we made an UG and its (random) sub-graph of 200 cities. Finally we selected only those elementary connections, that were on top of this sub-graph to form a timetable with 200 cities and the desired time range

| Name | Description | Format | Provided by | Publicly available |
|---|---|---|---|---|
| air01 | domestic flights (US) | CSV | American Stat. Assoc. | ✔ |
| cpru | regional bus (Ružomberok, SVK) | JDF 1.9 | Inprop s.r.o. | ✗ |
| cpza | regional bus (SVK, Žilina) | JDF 1.9 | Inprop s.r.o. | ✗ |
| montr | public transport (Montreal, CA) | GTFS | Montreal AMT | ✔ |
| sncf | country-wide intercity rails (FRA) | GTFS | SNCF | ✔ |
| zsr | country-wide rails (SVK) | MERITS | ŽSR | ✗ |

Table 4.1: Timetable descriptions

| Name | El. conns. | Cities | UG arcs | Time range | Height |
|---|---|---|---|---|---|
| air01 | 601489 | 287 | 4668 | 1 month | 24374 |
| cpru | 37148 | 871 | 2415 | 1 day | 239 |
| cpza | 60769 | 1108 | 2778 | 1 day | 370 |
| montr | 7153 | 217 | 349 | 1 day | 363 |
| sncf | 90676 | 2646 | 7994 | 1 day | 488 |
| sncf-inter | 4796 | 366 | 901 | 1 day | 209 |
| sncf-ter | 85932 | 2637 | 7647 | 1 day | 488 |
| zsr | 932052 | 233 | 588 | 1 year | 60308 |

Table 4.2: Main properties of the timetables. The value of time range is approximate.

table 4.3 for details.

| Name | El. conns. | Cities | UG arcs | Exact time range | Height |
|---|---|---|---|---|---|
| air01-200d | 19070 | 200 | 3957 | 1 day, 05h:00m | 756 |
| cpru-200d | 7511 | 200 | 570 | 0 days, 18h:45m | 230 |
| cpza-200d | 14804 | 200 | 563 | 0 days, 19h:25m | 370 |
| montr-200d | 6977 | 200 | 320 | 0 days, 20h:33m | 363 |
| sncf-200d | 5699 | 200 | 617 | 1 day, 05h:23m | 345 |
| sncf-inter-200d | 2367 | 200 | 478 | 1 day, 01h:10m | 186 |
| sncf-ter-200d | 5857 | 200 | 626 | 0 days, 20h:37m | 406 |
| zsr-200d | 2306 | 200 | 462 | 1 day, 03h:26m | 142 |

Table 4.3: 200-station sub-timetables with the maximal time range of little more than one day

Also, to provide idea as to how big the time-expanded graphs can get consult table **??**.

## 4.2 Basic properties

# 5 Underlying shortest paths

## 5.1 USP

In section 2 we have defined a timetable as a set of elementary connections. While do not pose any other restrictions on this set or on the elementary connections themselves, the real world timetables usually have a specific nature. Quite often are the connections repetitive, that is, the same sequence of elementary connections is repeated in several different moments throughout the day.

Another thing we may notice is that if we talk about *optimal* connections between a pair of distant cities $u$ and $v$, we are often left with a few possibilities as to *which way should we go*. This is not only because the underlying graph is usually quite sparse [6], but also because for longer distances we generally need to make use of some express connection that stops only in (small number of) bigger cities.

Thus the main idea which will repeat often throughout this section: *when carrying out an optimal connection between a pair of cities, one often goes along the same path regardless of the starting time.*

To formalize this idea, we will introduce the definition of an *underlying shortest path* - a path in UG that corresponds to some optimal connection in the timetable. To do this, we will first define a function *path* that extracts the **underlying path** (trajectory in the UG) from a given connection. Let $c$ be a connection $c = (e_1, e_2, ..., e_k)$.

$$\boldsymbol{path(c)} = shrink(from(e_1), from(e_2), ..., from(e_k), to(e_k))$$

Note, that if the connection involves waiting in a city (as e.g. in picture 5.1), $e_x^i = e_x^{i+1}$ for some $i$. That is why we apply the *shrink* function, which replaces any sub-sequences of the type $(z, z, ..., z)$ by $(z)$ in a sequence. This was rather technical way of expressing a simple intuition - for a given connection, the *path* function simply outputs a sequence of visited cities. Now we can formalize the underlying shortest path.

**Definition 5.1.** ***Underlying shortest path (USP)***
*A path $p = (v_1, v_2, ..., v_k)$ in $UG_T$ is an **underlying shortest path** if and only if $\exists t \in \mathcal{N} : p = path(c^*_{(v_1,t,v_k)}), c^*_{(v_1,t,v_k)} \in C_T$*

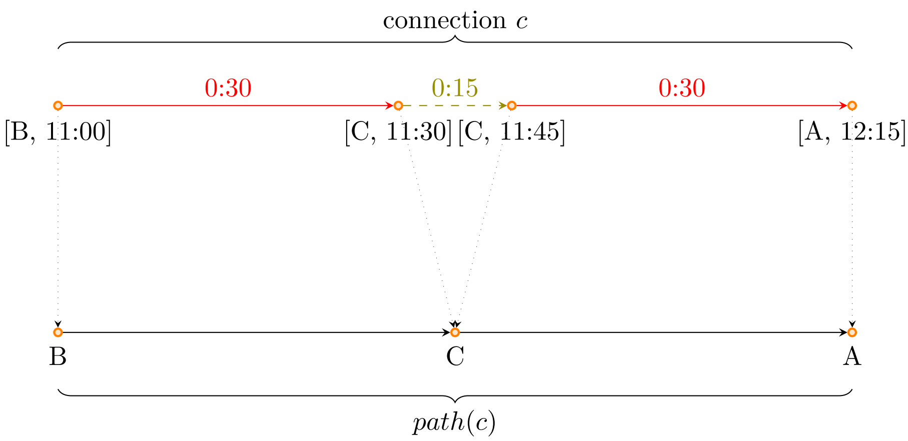


Figure 5.1: The *path* function applied on a connection to get the underlying path

Please note that the terminology might be a bit misleading - an USP is not necessarily a shortest path in the given UG. Connections on a shortest path may simple require too much waiting (the

[6] Maybe with exception of the airline timetables, which tend to be more dense

el. connections simply do not follow well enough one another) and thus it might be that travelling along the paths with greater distance proof to be faster options.

## 5.2 USP-OR

We can easily extract the underlying path from a given connection. Now let us look at this from the other way - if, for a given EA query, we know the underlying shortest path, can we reconstruct the optimal connection? One thing we could do is to blindly follow the USP and at each stop take the first elementary connection to the next stop on the USP. This simple algorithm called *ExpandUsp* is described in algorithm 1.

---

**Algorithm 1** ExpandUsp

---

**Input**
- timetable $T$
- USP $p = (v_1, v_2, ..., v_k)$
- departure time $t$

**Algorithm**

$c =$ empty connection
$t' = t$
**for all** $i \in \{1, ..., k-1\}$ **do**
  $e = argmin_{f \in C_T(v_i, v_{i+1})}\{dep(f) | \ dep(f) \geq t'\}$   *# take first available el. conn.*
  $t' = arr(e)$
  $c .= e$   *# add the el.conn to the resulting connection*
**end for**

**Output**
- connection $c$

---

Will we get an optimal connection if we expanded all possible USPs between a pair of cities? We show that we will, provided the timetable has no *overtaking* of elementary connections.

**Definition 5.2.** ***Overtaking***
*An elementary connection* $e_1$ ***overtakes*** $e_2$ *if, and only if* $dep(e_1) > dep(e_2)$ *and* $arr(e_1) < arr(e_2)$.

**Lemma 5.1.** *Let* $T$ *be a timetable without overtaking,* $(x, t, y)$ *an EA query in this timetable and* $\mathcal{P} = \{p_1, p_2, ..., p_k\}$ *a set of all USPs from* $x$ *to* $y$*. Define* $c_i = ExpandUsp(T, p_i, t)$ *to be the connection returned by the algorithm ExpandUsp 1. Then* $\exists j : \ c_j = c^*_{x,t,y}$.

*Proof.* The optimal connection $c^*_{x,t,y}$ has an USP $p$ which must be present in the set $\mathcal{P}$, as it is the set of all USPs from $x$ to $y$. So $p = p_j = (v_1, v_2, ..., v_l)$ from some $j$. We want to show that $c_j$ is the optimal connection. This may be shown inductively:

1. *Base:* ExpandUsp reaches city $v_1 = x$ as soon as possible (since the connection just starts there)
2. *Induction:* ExpandUsp reached city $v_i$ as soon as possible, it then takes the first available el. connection to the next city $v_{i+1}$. Since the el. connections do not overtake, ExpandUsp reached the city $v_{i+1}$ as soon as possible.

□

This simple approach works fine, but only provided there is no overtaking in the timetable. However, even if there is overtaking, we can fix this small problem by:

- *Removing overtaken elementary connection* - in our simplified version of the EAP problem we do not consider parameters such as the cost of the travel. Thus overtaken el. connections are redundant as they can be replaced by their overtaking el. connections plus some waiting.
- *Considering all elementary connections* up to the earliest arrival time to the next city in USP - this might however increase the time complexity of restoring the optimal connection from USP, thus we will use the first approach

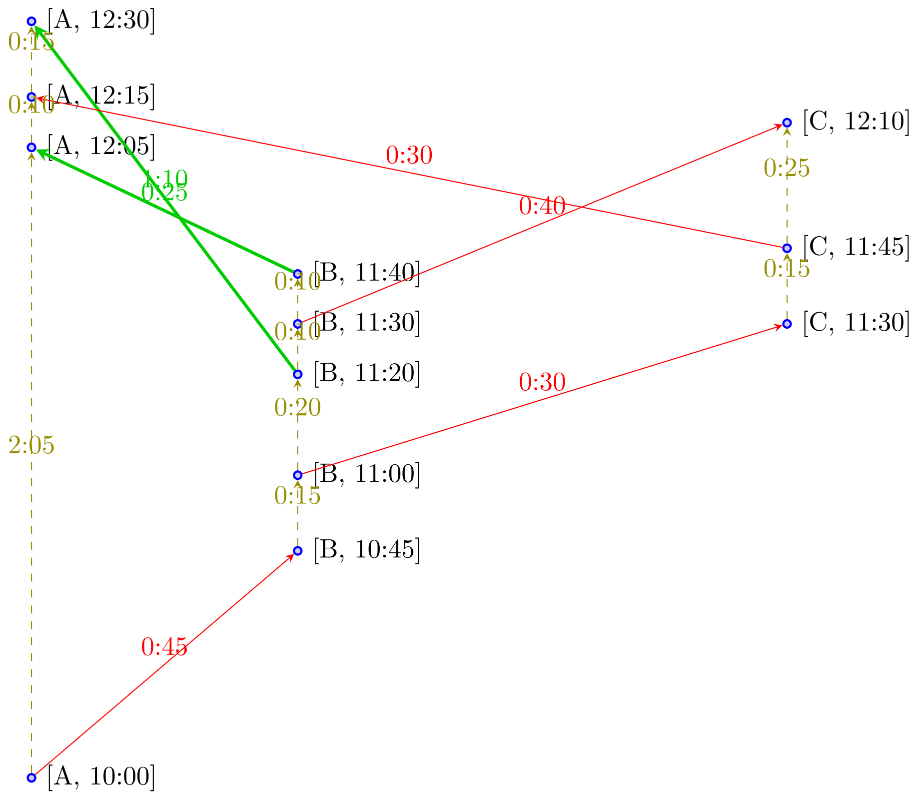


Figure 5.2: An example of **overtaking** (in thick), depicted in a TE graph

The basic idea of the algorithm *USP-OR* is therefore simply to pre-compute all the USPs for each pair of cities. Then, upon query, the algorithm simply tries out all the USPs for a given pair of cities, reconstructs respective connections and chooses the best one (the one that arrives the soonest). Very simple.

We will now have a look at the four parameters of this oracle-based method.

**Preprocessing time**. Basically, we need to find optimal connections from each *event* in the timetable to each city (or in other words - solve all possible EAP queries). We can do it by running Dijkstra algorithm $hn$ times (from each event), obtaining the time complexity $\mathcal{O}(hn^3)$.

**Preprocessed space**. We store USPs for each pair of cities ($n^2$) and each USP might be long at most $\mathcal{O}(n)$. The question is, how many different USPs there is for a pair of cities? We will call this number the **USP coefficient for a given city pair** and will denote it as $\tau_{A,B}$ (where $A$ and $B$ is the given pair of cities). The average of USP coefficients for all city pairs will be called simply the **USP coefficient** and denoted as $\tau$ ($\tau = avg_{A \neq B}(\tau_{A,B})$). So how big is $\tau$ for real-world timetables? See table 5.1.

**Query time**. Query time depends also on the USP coefficient of a given pair of cities, as we have to try out all USPs for that given pair. As each reconstruction of the connection from the respective USP costs linear time, the time complexity can be estimated as $\mathcal{O}(\tau n)$. As $\tau$ is basically a constant and we need linear time to actually output a connection, this can be deemed as optimal complexity.

**Stretch**. The algorithm is exact.

| Name | n/m of UG | height | time range | avg $\tau_{A,B}$ | max $\tau_{A,B}$ |
|---|---|---|---|---|---|
| air01 | 100/2274 | 10000 | month | 33.55 | 170 |
| air01 | 250/4568 | 10000 | month | 35.8 | 193 |
| air01 | 287/4668 | 24050 | month | TODO | TODO |
| air01 | 100 | - | day | | |
| air01 | 250 | - | day | | |
| air01 | 284/4382 | 789 | day | 4.6 | 30 |
| cpru | 50/120 | 20 | day | 0.62 | 4 |
| cpru | 100/286 | 228 | day | 6.5 | 40 |
| cpru | 250/696 | 228 | day | 9.23 | 63 |
| cpru | 871/2415 | 239 | day | 10.3 | 65 |
| cpza | 43/87 | 20 | day | 0.6 | 5 |
| cpza | 100/283 | 333 | day | 4.3 | 27 |
| cpza | 250/711 | 369 | day | 7.9 | 66 |
| cpza | 1108/2778 | 369 | day | 11.7 | 69 |
| montr | 49/77 | 20 | day | 0.7 | 3 |
| montr | 100/164 | 359 | day | 1.5 | 10 |
| montr | 217/349 | 359 | day | 3.8 | 29 |
| sncf | 20 | 8 | day | | |
| sncf | 30 | 10 | day | | |
| sncf | 50 | 20 | day | | |
| sncf | 100 | - | day | | |
| sncf | 250 | - | day | | |
| sncf | 500 | - | day | | |
| sncf | 750 | - | day | | |
| sncf | 1000 | - | day | | |
| sncf | 1500 | - | day | | |
| sncf | 2000 | - | day | | |
| sncf | full | - | day | | |
| zsr | 100/268 | 10000 | year | 101.4 | 1250 |
| zsr | 233/588 | 10000 | year | 170.1 | 1219 |
| zsr | 233/588 | 60308 | year | TODO | TODO |
| zsr | 96/252 | 142 | day | 1.9 | 14 |
| zsr | 225/514 | 142 | day | 2.4 | 19 |

Table 5.1: $\tau$ - the USP coefficient for different timetables

# 6 USP-OR-A

With *USP-OR* the main disadvantage is its space consumption. We may decrease this space complexity by pre-computing USPs only among *some* nodes, which we will call **access nodes** (AN). It would be suitable for this access node set (denoted $Acc$) to have several properties, which will be clear from the way we will use it later. Before we list these properties, we need to establish a few terms. We will call a **neighbourhood** of a city $v$ ($neigh_{Acc}(v)$) the smallest set of cities reachable from $v$ *not* via access nodes in the underlying graph. Then, the access nodes belonging to the city's neighbourhood will be called the **local access nodes** ($LAN_{Acc}(v)$). Intuitively, the local access nodes for a node $v$ form some kind of separator between the $v$'s neighbourhood and the rest of the graph.

Now we may formulate the three desired properties of the access node set $Acc$:

- The access node set is sufficiently small $|Acc| = \mathcal{O}(\sqrt{n})$
- The average neighbourhood size is sufficiently small $avg_v \; neigh_{Acc}(v) = \mathcal{O}(\sqrt{n})$
- The number of local access nodes for each node is bound by a constant $LAN_{Acc}(v) < l$

First we pre-compute some information on the timetable:

- LANs for each city of the UG. Note that the only LAN for an access node is itself.
- The so called **back local access nodes** (back-LANs) for each city. We find them as we found LANs, but in underlying graph with reversed orientation.
- The back-neighbourhoods, created in the previous step
- All USPs among access nodes

Upon a query from $u$ to $v$ at time $t$ ($(u, t, v)$) , we will:

1. Do a local search (Dijkstra) in the neighbourhood of $u$, until we reach all of its LANs (each of them we reach at some specific time). The so-called **local step**
2. Next we take back-LANs for the vertex $v$ and with the help of the pre-computed USPs we get the earliest arrival to each of them. The so-called **usp step**
3. Finally we run a Dijkstra from each of $v$'s back-LANs, restricted to the back-neighbourhood of $v$. The so-called **final step**

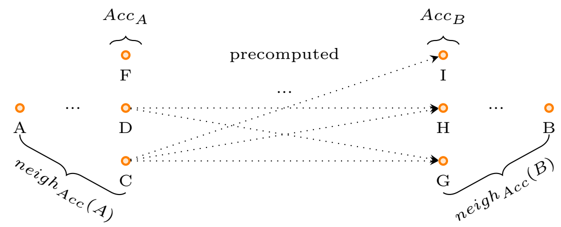


Figure 6.1: Principle of access nodes in *USP-OR-A* algorithm

Let us now have a look at the four parameters of this method.

**Preprocessing time**. We have to run a local search, e.g. Dijkstra's algorithm, from each city in the graph, terminating at the city's LANs. Thus the Dijkstra's algorithm runs in $neigh^2_{Acc}(v)$. We have $\mathcal{O}(n)$ cities with average neighbourhood of the size $\mathcal{O}(\sqrt{n})$, leading to time complexity $\mathcal{O}(n^2)$. However, we also have to pre-compute the USPs among all pairs of access nodes, which takes time at most $\mathcal{O}(hn^{2.5})$.

TODO **Lema with average**.

**Preprocessed space**. The pre-processed space consumption is now decreased to $\mathcal{O}(\tau n^2)$ as we have at most $\mathcal{O}(n)$ pairs of access nodes for which we pre-compute USPs. We remember other things as well but their space complexity is bound by the mentioned term.

**Query time**. The *local step* takes at most $\mathcal{O}(n)$ time. In *USP step* we try all USPs for all pairs of $u$'s LANs and $v$'s back-LANs, leading to $\mathcal{O}(l^2\tau n)$, which is linear if we consider $\tau$ and $l$ constant. Finally, the *final step* makes $l$ Dijkstra searches in the neighbourhood of $v$, which again takes linear time. Thus the overall query time may also be considered linear.

**Stretch**. The algorithm is exact.

# 7 Choosing the right Access node set

The challenge in *USP-OR-A* comes down to selection of the best access node set, or at least such that satisfies the three mentioned properties. There was a detected possibility for a trade-off - by increasing the access node set size, the average number of LANs as well as average neighbourhood size went down.

| Selected by | Size of AN set | Avg. LAN size | Avg. neighborhood size |
|---|---|---|---|
| high BC | 33 | 10.65 | 426.5 |
| high BC | 55 | 3.5 | 92.1 |
| high BC | 75 | 2.8 | 60.5 |
| high degree | 33 | 19.76 | 484 |
| high degree | 55 | 6.9 | 95 |
| high degree | 75 | 2.54 | 34.7 |

Table 7.1: Properties of access nodes selected by different methods. For underlying graph of *cpza* (1128 vertices, $\sqrt{1128} \approx 33$)

| Selected by | n/m | Size of AN set | Avg. neighborhood size ($\sqrt{n}$) | Avg. LAN size |
|---|---|---|---|---|
| high degree | 2646/7994 | 182 | 49.8 (51.4) | 4.24 |
| high degree | 2000/6075 | 130 | 44.3 (44.7) | 4.25 |
| high degree | 1500/4548 | 70 | 38.2 (38.7) | 3.42 |
| high degree | 1000/3216 | 52 | 30.1 (31.6) | 3.23 |
| high degree | 750/2415 | 40 | 26 (27.3) | 2.97 |
| high degree | 500/1583 | 22 | 22 (22.3) | 2.3 |
| high degree | 250/835 | 25 | 16.6 (15.8) | 3.36 |
| high degree | 100/313 | 16 | 10.4 (10) | 2.13 |
| high BC | 2646/7994 | | | |
| high BC | 2000/6075 | | | |
| high BC | 1500/4548 | | | |
| high BC | 1000/3216 | | | |
| high BC | 750/2415 | | | |
| high BC | 500/1583 | | | |
| high BC | 250/835 | | | |
| high BC | 100/313 | | | |

Table 7.2: Properties of access nodes selected by different methods. For underlying graph of *sncf* (French railways)

| Name | $\lvert AN\rvert/\lvert LAN\rvert$ (degs, avg) | $\lvert AN\rvert/\lvert LAN\rvert$ (degs, max) | $\lvert AN\rvert/\lvert LAN\rvert$ (betw, avg) | $\lvert AN\rvert/\lvert LAN\rvert$ (betw, max) |
|---|---|---|---|---|
| air01-200d | 56/8.4 | 87/4.1 | 67/7.7 | 200/0 |
| cpru-200d | 24/1.6 | 43/1.5 | 26/1.7 | 44/1.6 |
| cpza-200d | 18/2.0 | 77/1.3 | 24/2.3 | 47/1.5 |
| montr-200d | 21/2.0 | 83/1.1 | 47/1.5 | 121/1.4 |
| sncf-200d | 12/1.7 | 20/1.6 | 20/2.3 | 42/1.6 |
| sncf-inter-200d | 17/2.3 | 33/1.4 | 24/1.8 | 43/1.3 |
| sncf-ter-200d | 9/1.7 | 32/1.7 | 17/1.5 | 39/1.7 |
| zsr-200d | 16/2.0 | 50/1.4 | 18/1.7 | 41/1.5 |

Table 7.3: Necessary access node set sizes when choosing ANs based on degree or betweenness. The avg/max parameter specifies, if we wanted average neighborhood under $\sqrt{n}$ or all of them (maximum neighborhood under $\sqrt{n}$). Corresponding average LAN sizes are after the backslash.

# 8 Neural network approach

# 9 Application TTBlazer

# 10 Conclusion

# Appendices

## A File formats

**Timetable** is simply a set of elementary connections, thus the format is:

- number of el. connections
- the list of all el. connections (one per line, format “*FROM TO DEP-DAY DEP-TIME ARR-DAY ARR-TIME*”)

```
7                       //number of elementary connections
A B 0 10:00 0 10:45     //el. connection
A B 0 11:00 0 11:45
A B 0 12:00 0 12:45
A C 0 09:30 0 10:00
A C 0 10:15 0 10:45
C D 0 11:00 0 11:30
C D 0 13:00 0 13:30
```

Listing 1: TT file format

**Underlying graph** is basically an oriented graph, with some optional parameters. The format is the following:

- number of cities
- number of arcs
- the list of all cities (one per line)
  - optional coordinates (otherwise null)
- the list of all arcs (one per line, format “*FROM TO*”)
  - optional length (otherwise null)
  - optional list of lines operating on that arc (otherwise null)

```
4                                       //number of cities
5                                       //number of arcs
A 45 32                                 //name of the city, optional coordinates
B null
C 56 34
D null
A B 57 Northern                         //arc, optional length and list of lines
A C null Picadilly Victoria
C B 45 Circle Jubilee Picadilly
C D 32 null
D A null null
```

Listing 2: UG file format

**Time-expanded graph** is simply an oriented weighted graph, with nodes being the events and arcs being the elementary connections or waiting edges:

- number of nodes (i.e. events)
- number of arcs (el. connections + waiting)
- the list of all events (in the format “CITY DAY TIME”)
- the list of all arcs (in the format “FROM-EVENT TO-EVENT”)

```
5                        //number of events
15                       //number of arcs
A 0 13:30                //event
A 0 14:00
B 0 13:45
B 0 15:00
C 0 14:15
A 0 13:30 A 0 14:00      //waiting arc
A 0 13:30 B 0 13:45      //el. connection arc
```

```
A 0 14:00 B 0 15:00
A 0 13:30 B 0 15:00
C 0 14:15 B 0 15:00
...
```

Listing 3: TE file format

**Time-dependent graph** is an oriented graph with a function on the arc specifying the arc's traversal time at any moment. In timetable networks this function is piece-wise linear and it is fully represented by the list of its interpolation points. Thus the TD file format:

- number of cities
- number of arcs
- the list of all cities (one per line)
  - optional coordinates (otherwise null)
- the list of all arcs (one per line). Arc has the format "*FROM TO INT-POINTS*" where *INT-POINTS* is a list of interpolation points [7], see the listing 4 for an example.

```
4                                              //number of stations
5                                              //number of arcs
A 0 0                                          //name of the city, optional coordinates
B 4 4
C null
D 12 0
A B (0 13:30 45) (0 14:00 40)                  //arc and the list of interpolation
    points
A C (1 14:15 10)
C B (0 15:00 20)
C D (2 10:00 70)
D A (1 17:20 35) (1 18:00 40) (1 18:50 35)
...
```

Listing 4: TD file format

[7] An interpolation point is described by a triple "*DAY TIME MINUTES*", where *MINUTES* are the traversal time