r-2012-11-08

František Hajnovič
ferohajnovic@gmail.com

December 29, 2012

Contents

1 Processed real data

Table 1 summarizes the so far processed timetables. The regional bus timetables provided by *cp.sk* are the examples of more local timetables with underlying network being the road network. The time range was so far set to 1 day, assuming that each bus operates the same way every day. This is not a correct assumption (some buses may operate e.g. only during weekends), but this way we save space and the resulting timetable should not differ much in properties from the real one. We may later however try to extend the time range to one week and see the changes in properties.

Timetable from *zsr.sk* covers a full year, as connections had a reference to a calendar, stating if the train operates on a given date. That is also the reason for the large number of elementary connections.

Timetable of *United airlines* requires more processing, but should be later added as it provides yet another type of timetable.

Name	Type of network	Provided by	# of stations	# of elementary connections	Time range	Note
tt_cpza	Regional bus	cp.sk	1128	68096	1 day	Area of Žilina
tt_cpru	Regional bus	cp.sk	877	43069	1 day	Area of Ružomberok
tt_zsr	Country-wide rails	zsr.sk	233	932052	1 year	

Table 1: Timetable files

Timetable files have a simple form:

```
1    7                      //number of elementary connections
2    A B 0 10:00 0 10:45    //FROM TO DAY(depart) TIME(depart) DAY(arrive) TIME(arrive)
3    A B 0 11:00 0 11:45
```

```
4     A B 0 12:00 0 12:45
5     A C 0 9:30 0 10:00
6     A C 0 10:15 0 10:45
7     C D 0 11:00 0 11:30
8     C D 0 13:00 0 13:30
```

Listing 1: Timetable files form

Table 2 summarizes the so far processed maps (or underlying graphs) of networks, on top of which we can later build random timetables with given properties. The maps were created by extracting stations from the timetable and making an edge between the pair where a connection is operating, thus getting the underlying structure of the timetable. Note that we will not really get the real map of the (e.g.) rails - with our approach, there will be a direct link between a pair of large (but distant) cities even though many smaller stations exist on the way. This is due to express lines that do not stop at these smaller stations. If the real underlying map should be desirable, we can later remove these "express links", although we cannot be sure that the express line did not really follow a new, faster road/railway without any smaller stops on the way (that should not happen in case of railways).

Map of the *road network of Slovakia* should be added, as well as the underlying map of *United airlines*. Also, currently the edges of the underlying graphs have null lengths - the travel time of the fastest connection on the edge can be used instead.

Name	Type of network	Provided by	# of stations	# of connections	Notion of lines	Note
ug_cpza	Regional bus	cp.sk	1128	2034	✔	Area of Žilina
ug_cpru	Regional bus	cp.sk	877	1784	✔	Area of Ružomberok
ug_zsr	Country-wide rails	zsr.sk	233	588	✔	
ug_london	Underground rails	queensu.ca	321	732	✔	

Table 2: Underlying graph files

Underlying graph files have the following form:

```
4                                   //number of stations
5                                   //number of edges
A [45 32]                           //name of the station, optional coordinates
B
C [56 34]
D
A B 57 Northern                     //FROM TO edge length, list of lines operating
    on the edge
A C null Picadilly Victoria         //edge length may be null (will be e.g. random,
    or calculated from coordinates)
C B 45 Circle Jubilee Picadilly
C D 32 null                         //list of lines may be also null
D A 90 Metropolitan Victoria
```

Listing 2: Underlying graph files form

2 Overtaking and express lines

In this section, we will define and discuss two notions: *overtaking* and *express lines*. For clarity, we repeat the definition of the timetable:

Definition 1. ***Timetable on a graph*** G
A timetable on a given graph G *is a set* $T_G = \{(x, y, p, q) | (x, y) \in E_G,\ p, q \in N,\ p < q\}$.

- *An element of* T *is called an* ***elementary connection*** *and* x *[*y*]* *and* p *[*q*]* *are the* ***departure [arrival] node*** *and* ***time****.*
- *Graph* G *is the* ***underlying graph*** *of timetable* T*.*

Definition 2. ***Connection from*** a ***to*** b ***in a given timetable*** T_G
A connection from a *to* b *in a given timetable* T_G *is a sequence of* elementary connections $(e_1, e_2, ..., e_k),\ k \geq 1,\ e_i = (e_x^i, e_y^i, e_p^i, e_q^i)$*, such that* $e_x^1 = a$*,* $e_y^k = b$ *and* $\forall i \in \{2, ..., k\} : (e_x^i = e_y^{i-1},\ e_p^i \geq e_q^{i-1})$

Place		Time	
From	**To**	**Departure**	**Arrival**
A	B	11:00	11:45
A	C	9:30	10:00
A	C	10:15	10:45
A	C	10:30	11:00
A	C	10:40	11:05
C	D	11:00	11:40
C	D	11:05	11:25
C	B	11:20	12:00
D	A	10:00	11:00

Table 3: An example of a timetable

- *Connection* ***starts*** *at the* departure time e_p^1.
- ***Length*** *of the connection is* $e_q^k - e_p^1$.

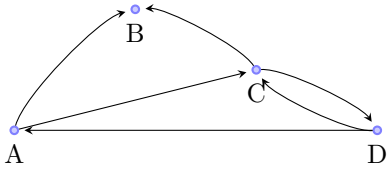

Figure 1: Underlying graph of the timetable in 3. Note, that there does not have to be a connection on every arc of the underlying graph (e.g. there is an arc from D to C, but no connection operates here)

Overtaking, simply stated, means that there are connections in the timetable that overtake one another - that is by choosing a later connection, we will actually arrive sooner to the destination. Note, a connections overtake one another only if they follow the same route (the sequence of traversed arcs is the same) - otherwise we are talking simply about choosing a quicker connection, which is actually the task the search engines perform. See picture 4 for better explanation.

Definition 3. ***Overtaking***
A timetable T_G *has overtaking property* $\iff$ *there exist two connections* e *and* f *from* a *to* b *in* T_G *such that* $e = (e_1, e_2, ..., e_k)$, $f = (f_1, f_2, ..., f_k)$, $\forall i$ $e_x^i = f_x^i$, $e_y^i = f_y^i$ *and* $e_p^1 < f_p^1$ *but* $e_q^k > f_q^k$.

This definition can be however simplified if we realize, that if a connection e overtakes connection f, it must do so on a single specific arc. Thus a simpler definition (but equivalent one) would consider only elementary connections:

Definition 4. ***Overtaking (simpler form)***
A timetable T_G *has overtaking property* $\iff$ *there exist two elementary connections* e *and* f *from* a *to* b *in* T_G *such that* $e_p < f_p$ *but* $e_q > f_q$.

What does the timetable having overtaking property influence? For oriented weighted graphs (representing e.g. static road network), many fast and precise algorithms were developed for finding shortest paths. These algorithms often use a modified Dijkstra's algorithm at query time, enhanced by additional information obtained during preprocessing phase. As far as the used Dijkstra's algorithm was unidirectional, it can be straightforwardly adapted to a time-dependent scenario (although there will most probably be various changes in the preprocessing phase as well) - each time the time-dependent Dijkstra considers an arc, it looks at the arc's cost function to obtain the traversal cost at the given moment. In case there is no overtaking in the timetable, getting the traversal cost of the given arc simply means looking into the timetable and finding the first elementary connection that traverses the arc, which requires no preprocessing. Otherwise we would need to pre-process the graph to obtain cost functions for each arc. Such preprocessing could be done in time linear of the number of elementary

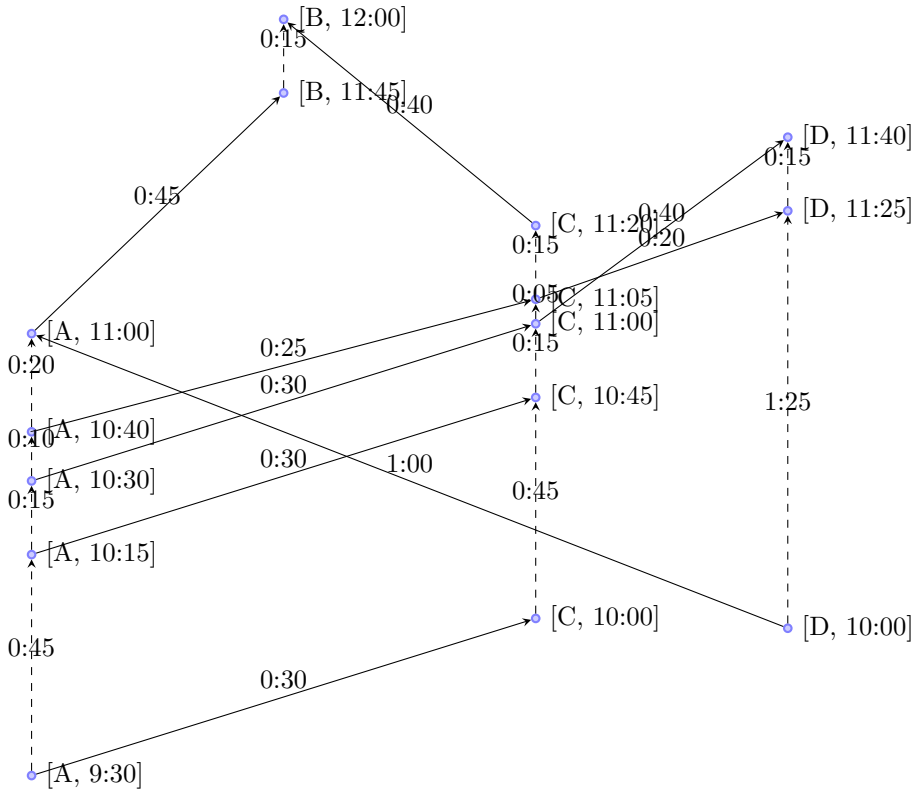

Figure 2: Graph representing the timetable in 3

Place		Time	
From	**To**	**Departure**	**Arrival**
A	D	10:00	11:00

Table 4: An added express connection to our timetable in 3

connections.

We now move on to the other notion - express lines. Informally, express lines are connections that bypass several stops on the way. In our definition of timetable graph, we do not allow such connections so we extend the definition.

Definition 5. ***Timetable with express lines on a graph*** G
A timetable with express lines on a given graph G *is a set* $T_G = \{(x, y, p, q) | \text{there is a path from } x \text{ to } y \text{ in } G,\ p, q \in N,\ p < q\}$.

- *Tuples* (x, y, p, q) *where* $(x, y) \in E_G$ *are* ***elementary connections****, the remaining are* ***express connections****.*
- *Such* (x, y) *that* $(x, y) \notin E_G$ *but there is a path from* x *to* y *in* G *will be called* ***express lines****.*

Now we can formulate the following observation:

Observation 1.
Let G *be a graph with the vertex set* V*,* T_G *timetable on* G *and* H *a connected graph with the vertex set* V*. There exists a timetable with express lines* T_H *such that* $T_H = T_G$*.*

Proof. The proof is trivial: By the definition of timetable with express lines, there may be a connection between any pair of vertices in the underlying graph H, as it is connected. Thus for every elementary connection $(x, y, p, q) \in T_G$ we will simply have the same (*either* elementary *or* express) connection (x, y, p, q) in T_H □

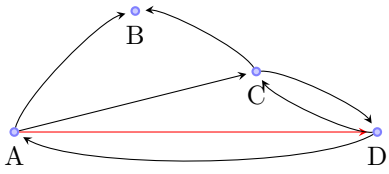

Figure 3: Underlying graph of the timetable in 3 with added express line from A to D

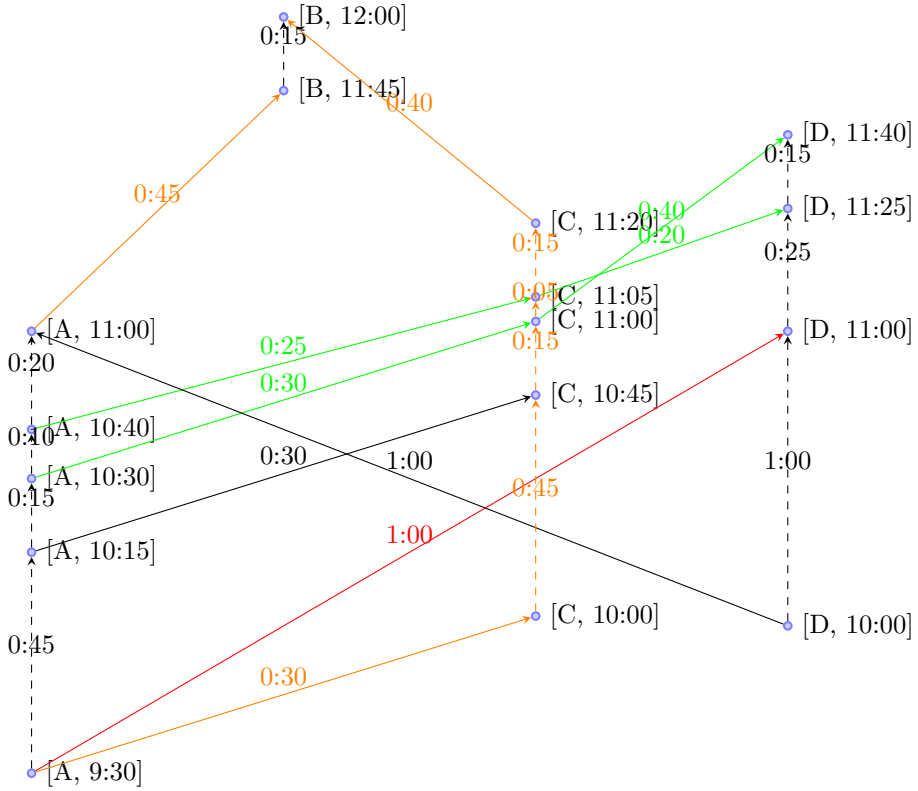

Figure 4: Graph representing the timetable in 3 with added express connection from A to D (marked with red color). An example of overtaking is marked with green color. With orange color we have marked two connections that *do not* represent overtaking - waiting at node A for the connection at $11:00$ is simply a smarter choice.

Informally, this observation just states, that with express lines we can forget about underlying structure of the timetable and consider it as a complete graph. That would however imply that no property of the underlying graph G can be generally propagated to the graph representing the timetable on G (not even to certain extent). Unfortunately, express lines are not uncommon in real-world scenarios. On the other hand, usage of express lines is limited in practice and generally express lines act as a shortcut layer on top of the real underlying network. One can imagine this as a hierarchy starting from the road network with nodes being the intersections. Next layer would be the local bus lines, then express inter-city lines etc... See picture 5 for demonstration.

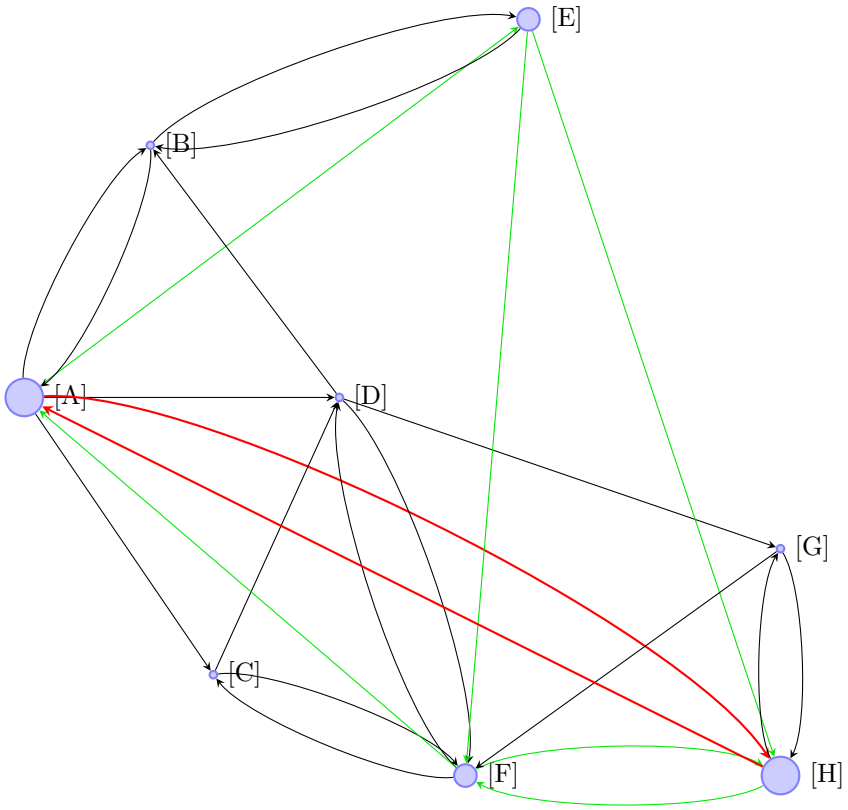

Figure 5: An example of the "shortcut" layers. Original underlying graph representing structure of the map has black edges. However bigger towns are also connected by express lines depicted by green color. The two biggest cities are connected by yet another express line, marked red.

3 Adjustment of Gavoille's algorithm for graphs with $r(n)$ separator [GPPR04] for time-dependent scenario

In [GPPR04], an algorithm answering distance queries in oriented weighted graphs is presented. The algorithm pre-processes the input graph, producing the so-called distance labels for each vertex. After preprocessing, a distance query between a pair of vertices u and v can be answered in logarithmic (TODO, correct this) time using only the distance labels of u and v.

The algorithm takes advantage of recursive separators in the graph. The *size of the separator* and *how quickly we can find it* are two factors that influence the efficiency of the algorithm:

- Size of the separator influences the resulting size of the distance labels. If we define

$$\mathbf{R(n)} = \sum_{i=0}^{\log_{3/2} n} r(n(2/3)^i)$$

  where $r(n)$ is the size of the recursive separator, the resulting total size of the preprocessed distance labels is

$$\mathcal{O}(nR(n)\log W + n\log^2 n)$$

  with W being the maximum arc length in the graph.
- The time it takes us to find the separator, on the other hand, influences the preprocessing time. E.g., in planar graphs, we can find a recursive separator of size $\mathcal{O}(\sqrt{n})$ ([Eri] in linear time

($\mathcal{O}(n)$). That leads to the preprocessing time of $\mathcal{O}(n^{3/2} \log n)$.

We will now briefly describe how the Gavoille's algorithm works (more details can be found in [GPPR04]).

1. A separator S is found and the graph is split into corresponding components
2. Each node in the component gets the list of distances to all nodes from S
3. Each node in S also gets the list of distances to all other nodes from S
4. A node in a component gets the identifier of the component
5. We proceed recursively in components

After the preprocessing, answering a distance query between a pair of vertices u and v is simple. If u and v are from a different top level component, the shortest path connecting them must pass through the top level separator. We thus consider all of its vertices and find the distance as

$$d(u, v) = min_{s \in S}\{d(u, s) + d(s, v)\}$$

In case u and v are from the same top level component C, their corresponding shortest path may still pass through the top level separator, but it may also be completely within the component C. Thus we take the minimum out of these two values. For more details, please refer to [GPPR04], section *2.2.*

We will now be interested, if this algorithm can be adjusted to work in timetable scenarios, that is, on graphs representing timetables.

TODO

4 Open points

- What does overtaking really affect?
- Hierarchy of express lines → what properties can be propagated in time-expansion?
- Find use of machine learning?
  - Regression -
  - Separation -
  - Clustering - e.g. for multi-level partition in SHARC
  - Decision trees -
  - Neural network - e.g. for property prediction
  - Online algorithms -
- General formula for preprocessing time in Gavoille's algorithm

5 To do

- Extract road network of SVK ✔
- Replace null lengths in underlying graphs of existing timetables ✔
- Adjustment of Gavoille's algorithm for planar graphs for time-dependent scenario
- United airlines extract data
- Road network of SVK - process data
- Start work on the diagnostic program

References

[Eri] Jeff Erickson. Graph separators.

[GPPR04] Cyril Gavoille, David Peleg, Stéphane Pérennes, and Ran Raz. Distance labeling in graphs. *Journal of Algorithms*, 53(1):85 – 112, 2004. ISSN 0196-6774. URL http://www.sciencedirect.com/science/article/pii/S0196677404000884.