r-2013-02-28

František Hajnovič
ferohajnovic@gmail.com

March 13, 2013

Contents

1 USP-OR **1**

2 USP-OR-A **3**

3 Choosing the right Access node set **4**

4 Open points **5**

5 To do **5**

1 USP-OR

When carrying out an optimal connection between a pair of cities, one often goes along the same path regardless of the starting time. We will call the path that corresponds to some optimal connection in the timetable the **underlying shortest path** (USP). It can easily be extracted from the optimal connection, as shown in figure 1.

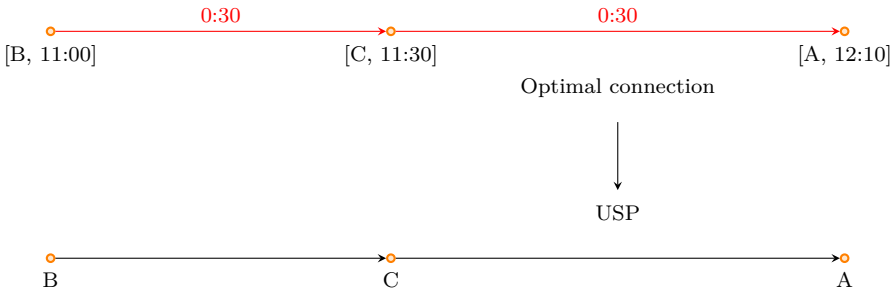

Figure 1: Underlying shortest path extracted from the connection above

Now let us look at this from the other point of view - if we know 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 works fine 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.

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 1: τ - the USP coefficient for different timetables

- *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

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 τ ($\tau = avg_{A \neq B}(\tau_{A,B})$). So how big is τ for real-world timetables? See table 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 τ 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.

2 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**

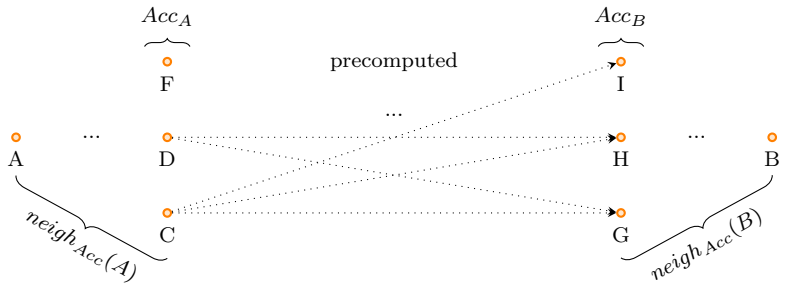

Figure 2: Principle of access nodes in *USP-OR-A* algorithm

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 τ 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.

3 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 2: 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 3: Properties of access nodes selected by different methods. For underlying graph of *sncf* (French railways)

4 Open points

- -

5 To do

Theory:

- Properties propagation in step a), step b)...

Practice:

- Finish analysis of real-world timetables