Underlying shortest paths in timetables

František Hajnovič[1*]

Supervisor: Rastislav Královič[1†]

Katedra informatiky, FMFI UK, Mlynská Dolina 842 48 Bratislava

Abstract: We introduce methods to speed-up optimal-connection queries in timetables based on pre-computing paths that are worth to follow. We present a very fast but space consuming method *USP-OR* which we enhance to considerably decrease the size of pre-processed data while still achieving speed-ups up to 6 against time-dependent Dijkstra's algorithm implemented with Fibonacci heap priority queue.

Keywords: optimal connection, timetable, Dijkstra's algorithm, underlying shortest paths

1 Introduction

We consider a problem of looking for an optimal connection (from a at time t to $b \rightarrow \boldsymbol{c^*_{(a,t,b)}}$) in timetables on which we carried out some preprocessing. We define **timetable** simply as a set of **elementary connections**, which are quadruples (x, y, p, q) meaning that a train departs from **city** x at time p an arrives to city y at time q. A **connection** is simply a valid sequence of elementary connections which may include also waiting in visited cities. We also define an **underlying graph** ($\boldsymbol{ug_T}$) of the timetable T whose nodes are the cities and there is an arc (x, y) if some el. connection $(x, y, p, q) \in T$.

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 1.1: An example of a timetable.

Finally, we define the **underlying shortest path** (USP) to be every path p in ug_T such that for some optimal connection $c^*_{(a,t,b)}$:

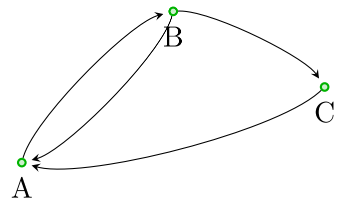

Figure 1.1: An underlying graph of the timetable 1.1.

$path(c^*_{(a,t,b)}) = p$, where function *path* simply extracts the sequence of cities visited by the connection (see picture 1.2).

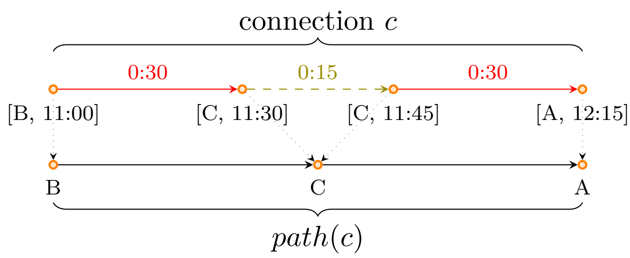

Figure 1.2: The *path* function applied on a connection to get the underlying path.

2 Methods

2.1 *USP-OR*

Our first method, called *USP-OR*, is based on pre-computing USPs between every pair of cities. Then, upon a query from x to y at time t we consider one by one the computed USPs between x and y and perform a reverse operation to the *path* function - *expand*(p) where p is an USP. The *expand* function simply follows the sequence of cities in p and from each of them it takes the first available el. connection to the next one, thus constructing one by one a connection from x to y.

*ferohajnovic@gmail.sk

†kralovic@dcs.fmph.uniba.sk

Algorithm 2.1 *USP-OR* query

Input

- timetable T
- OC query (x, t, y)

Pre-computed

- $\forall x, y$: set of USPs between x and y $(usps(x, y))$

Algorithm

$c^* = null$
for all $p \in usps_{x,y}$ **do**
  $c = Expand(T, p, t)$
  $c^* =$ better out of c^* and c
end for

Output

- connection c

Define an elementary connection e_1 to **overtake** $e_2 \iff depart(e_1) > depart(e_2)$ and $arrive(e_1) < arrive(e_2)$. If the timetable T has no overtaking el. connections, the *USP-OR* algorithm returns exact answers, which can be easily proved. The table 2.1 summarizes the parameters of *USP-OR* based on the following parameters of the timetable:

- $\boldsymbol{\tau}$ - the average number of different USPs between pairs of cities - the **USP coefficient**
- $\boldsymbol{\gamma}$ - the average size (i.e. number of el. conn.) of optimal connections - the **OC radius**
- $\boldsymbol{\delta}$ - the **density** of T defined by $\frac{\boldsymbol{m}}{\boldsymbol{n}}$ - number of arcs of ug_T divided by number of cities
- $\boldsymbol{h}$ - the **height** of the timetable - the maximal number of events (arrival/departure) in a city

USP-OR	**guaranteed**	$\boldsymbol{\tau = \mathcal{O}(1)}$, $\boldsymbol{\gamma \leq \sqrt{n}}$, $\boldsymbol{\delta \leq \log n}$
prep	$\mathcal{O}(hn^2(\log n + \delta))$	$\mathcal{O}(hn^2 \log n)$
size	$\mathcal{O}(\tau n^2 \gamma)$	$\mathcal{O}(n^{2.5})$
qtime	avg. $\mathcal{O}(\tau\gamma)$	avg. $\mathcal{O}(\sqrt{n})$
stretch	1	1

Table 2.1: The summary of the *USP-OR* algorithm parameters.

The value of τ for our datasets was found to be quite small (≈ 10) and just slightly, if at all, increasing with increasing n (see plot 2.1). Average optimal connection size and the density δ were as in the second column of table 2.1 for our timetables. Still, the size of the preprocessed data is too large for practical use, hence the extension of this algorithm, called *USP-OR-A*.

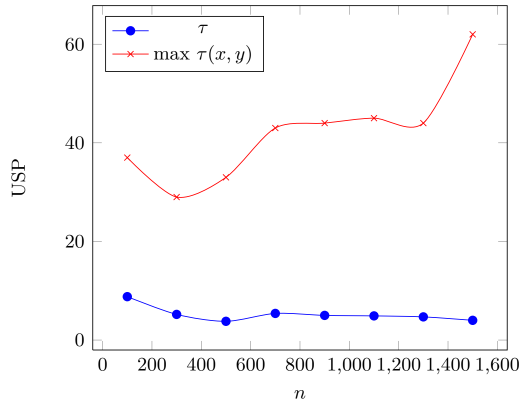

Figure 2.1: Changing of τ with increased number of stations in *sncf* dataset.

2.2 *USP-OR-A*

To decrease the space complexity, in *USP-OR-A* we compute USPs only among cities from a smaller set - called **access node set** (AN set, denoted $\boldsymbol{\mathcal{A}}$). Given such set in timetable T, we define for a city x its **neighbourhood** $neigh(x)$ as all the cities reachable in ug_T *not* via ANs. The access nodes within this neighbourhood are called **local access nodes** (LAN). We do the same in $\overleftarrow{ug_T}$ (ug_T with reversed orientation) to get **back neighbourhood** and **back LANs**.

In the pre-processing, we:

- find $\mathcal{A}$ (discussed later)
- $\forall x, y \in \mathcal{A}$ compute USPs between x and y
- $\forall$ cities $x \notin \mathcal{A}$ compute $neigh(x)$, $bneigh(x)$, $lan(x)$ and $blan(x)$

On a query from x to y at time t, we will first make a local search in the neighbourhood of x up to x's local access nodes (*local front search* phase). Subsequently, we want to find out the earliest arrival times to each of y's *back* local access nodes. To do this, we take advantage of the pre-computed USPs between access nodes - try out all the pairs $u \in lan(x)$ and $v \in blan(y)$ and expand the stored USPs (*inter-AN search* phase). Finally, we make a local search from each of y's back LANs to y, but we run the search *restricted* to y's back neighbourhood (*local back search* phase). See algorithm 2 and picture 2.2

for more clarification.

Algorithm 2.2 *USP-OR-A* query

Input

- timetable T
- OC query (x,t,y)

Algorithm

let $lan(x) = x$ if $x \in \mathcal{A}$
let $blan(y) = y$ if $y \in \mathcal{A}$
Local front search
do TD Dijkstra from x at time t up to $lan(x)$
if $y \in neigh(x)$ **then**
  c^*_{loc} = conn. to y obtained by TD Dijkstra
end if
$\forall u \in lan(x)$ let $ea(u)$ the arrival time and $oc(u)$ the conn. to u obtained by TD Dijkstra
Inter-AN search
for all $v \in blan(y)$ **do**
  $oc(v) = null$
  for all $u \in lan(x)$ **do**
    for all $p \in usps(u,v)$ **do**
      $c = Expand(T,p,ea(u))$
      $oc(v)$ = better out of $oc(v)$ and c
    end for
  end for
end for
$\forall v \in blan(y)$ let $ea(v) = end(oc(v))$
Local back search
for all $v \in blan(y)$ **do**
  perform TD Dijkstra from v at time $ea(v)$ to y restricted to $bneigh(y)$
  $fin(v)$ = the conn. returned by TD Dijkstra
end for
$v^* = argmin_{v \in blan(y)}\{end(fin(v))\}$
$u^* = from(oc(v^*))$
$c^* = oc(u^*).oc(v^*).fin(v^*)$ *# concat.*
output better out of c^*_{loc} and c^*

Output

- optimal connection $c^*_{(x,t,y)}$

We will call $\mathcal{A}$ a $\boldsymbol{(r_1, r_2, r_3)}$ **AN set** if:

- $|\mathcal{A}| \leq r_1 \cdot \sqrt{n}$
- $(avg\ |neigh(x)|)^2 \leq r_2 \cdot n$
- $|lan_{\mathcal{A}}(x)| \leq r_3$

If we can manage to find a (r_1, r_2, r_3) AN set in time $f(n)$, the parameters of the *USP-OR-A* algorithm are as summarized in table 2.2. Table 2.3 lists the parameters of *USP-OR-A* for timetables with specific conditions (as had our datasets) and on which we can find (r_1, r_2, r_3) AN set with r_i being a constant (with regard to n).

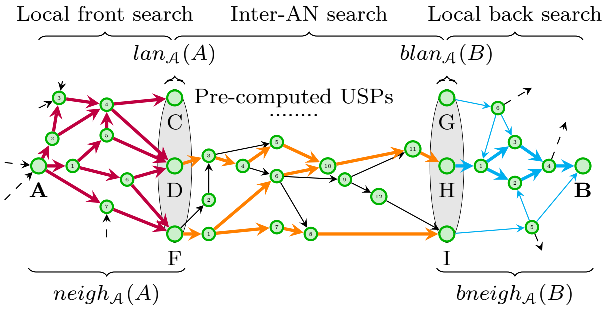

Figure 2.2: Principle of *USP-OR-A* algorithm. The arcs in **bold** mark areas that will be explored: all nodes in $neigh_{\mathcal{A}}(x)$, USPs between LANs of x and back LANs of y and the back neighbourhood of y (possibly only part of it will be explored, since the local back search goes against the direction in which the back neighbourhood was created).

USP-OR-A	guaranteed
prep	$\mathcal{O}(f(n) + (r_1 + r_2)(\delta + \log n)hn^{1.5})$
size	$\mathcal{O}(r_2 n^{1.5} + r_1^2 \tau_{\mathcal{A}} \gamma_{\mathcal{A}} n)$
qtime	avg. $\mathcal{O}(r_2 r_3 \sqrt{n}(\log(r_2 n) + \delta) + r_3^2 \tau_{\mathcal{A}} \gamma_{\mathcal{A}})$
stretch	1

Table 2.2: Guaranteed parameters of the *USP-OR-A* algorithm. $\tau_{\mathcal{A}}$ and $\gamma_{\mathcal{A}}$ are defined just like τ and γ, but on the set of cities $\in \mathcal{A}$.

USP-OR-A	$\boldsymbol{\tau, r_1, r_2, r_3}$ **const.,** $\boldsymbol{\gamma \leq \sqrt{n}, \delta \leq \log n}$
prep	$\mathcal{O}(f(n) + hn^{1.5} \log n)$
size	$\mathcal{O}(n^{1.5})$
qtime	avg. $\mathcal{O}(\sqrt{n} \log n)$
stretch	1

Table 2.3: Parameters of the *USP-OR-A* algorithm under certain conditions, generally fulfilled by our timetables.

3 Results

4 Conclusion

Acknowledgments

The authors thank the anonymous contributors whose work largely constitutes this sample file.

This work has been supported by the European Community grant number xxxxx and by the Comenius University Grant for Young Researchers.