Underlying shortest paths in timetables

František Hajnovič[1]
Supervisor: Rastislav Královič[1]
[1] Katedra informatiky, FMFI UK, Mlynská Dolina, 842 48 Bratislava

Introduction

We introduce methods to speed-up optimal connection queries in timetables based on pre-computing paths that are worth to follow. We present two methods:

- *USP-OR* - very fast, but space consuming
- *USP-OR-A* - fast, less space consuming

And we compare the methods to:

- *Time-dependent Dijkstra* [1] - slowest, but least space consuming

We define the most common terms:

- **Timetable** - a set of **elementary connections** [Müller-Hannemann et al., 2007] (see figure 1 for an example)
- **Connection** - a valid sequence of elementary connections (may include also some waiting)
- **Underlying graph** of the timetable - nodes are the cities and there is an arc between two cities if some el. connection connects them.

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

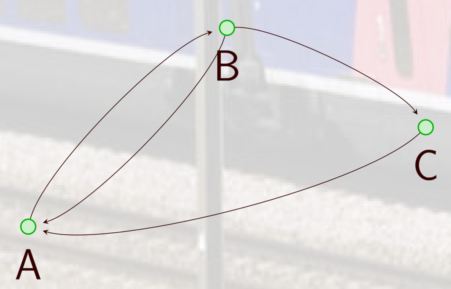


Figure 1: *A timetable and its underlying graph*

- **Underlying shortest path** (USP) - every path p in the underlying graph, such that for some optimal connection c^*: $path(c^*) = p$ (function $path$ extracts the sequence of cities visited by the connection, see figure 2).

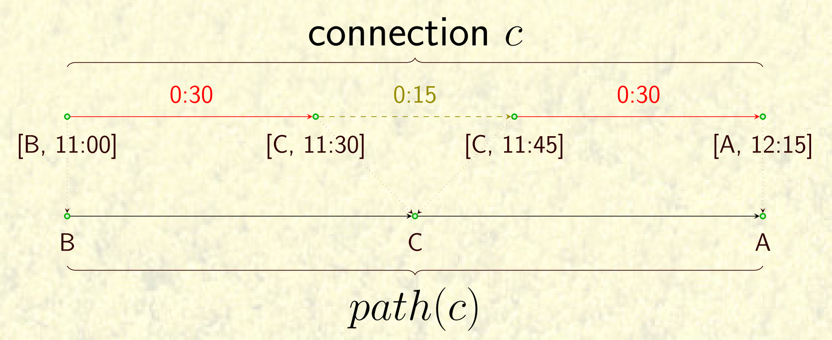


Figure 2: *The $path$ function gets the underlying path from a connection.*

The timetables we used for testing are real datasets and they had the following properties:

- Their underlying graphs were sparse ($m \leq n \log n$, n = number of cities)
- Average optimal connection was generally up to $\sqrt{n}$
- Average number of USPs between two cities was a small and constant (or slightly increasing with n)

USP-OR

Our first method, called *USP-OR* (**USP or**acle), 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, constructing one by one a connection from x to y.

[1] Implemented in $\mathcal{O}(m + n \log n)$ using Fibonacci heap priority queues [Sommer, 2010]

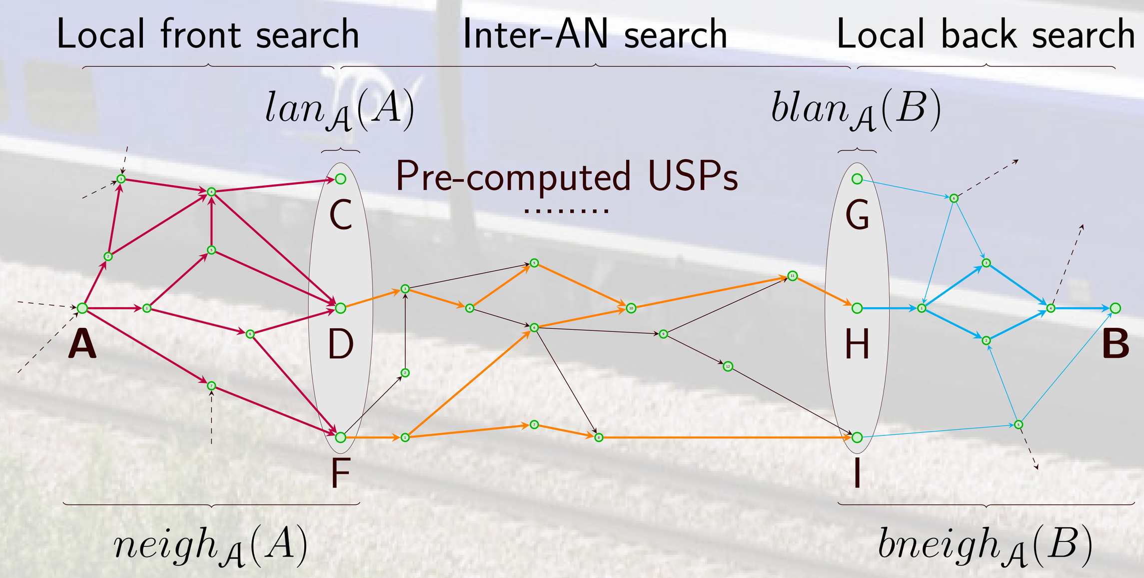
USP-OR	*prep*	*size*	*qtime*	*stretch*
Our timetables	$\mathcal{O}(hn^2 \log n)$	$\mathcal{O}(n^{2.5})$	avg. $\mathcal{O}(\sqrt{n})$	1

Table 1: *Preprocessing time/size, query time and stretch (1 = exact answers) for USP-OR*

USP-OR-A

To decrease the space complexity, in *USP-OR-A* (**USP or**acle with **a**ccess nodes) we compute USPs only among cities from a smaller set - called **access node set** (AN set $\mathcal{A}$). On a query from x to y at time t, we do:

- *Local front search*: a local search in the neighbourhood of x up to x's local access nodes (LANs)
- *Inter-AN search*: expand all USPs between x's LANs and y's back LANs
- *Local back search*: a local search from y's back LANs to y, restricted to y's back neighbourhood



How to choose a good access node set?

- Solve optimally → NP-complete (reduction of minimal set cover)
- Heuristic approach → algorithm *Locsep* (AN set made out of nodes that are good local separators)

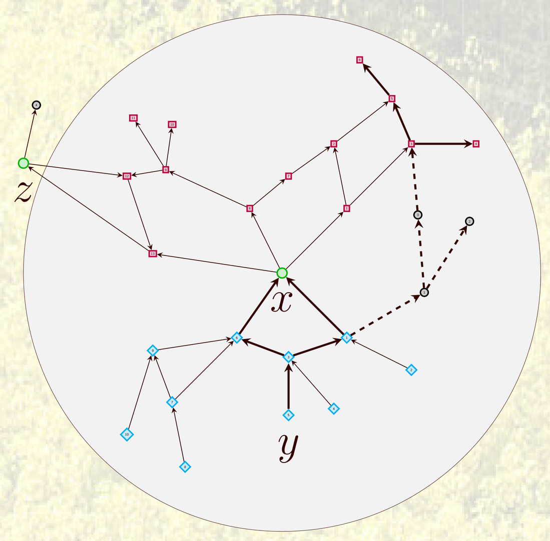


Figure 4: *To evaluate the node's suitability to be an access node, we see what would happen to the reachability in its surroundings if we select it to the AN set.*

USP-OR-A + Locsep	*prep*	*size*	*qtime*	*stretch*
Our timetables	$\mathcal{O}(\delta n^{2.5})$	$\mathcal{O}(n^{1.5})$	avg. $\mathcal{O}(\sqrt{n} \log n)$	1

Performance

We measured:

- **Speed-up** - how many times faster is the algorithm against the Time-dependent Dijkstra
- **Size-up** - how many times more memory is needed then to store the timetable itself

	Dataset	*cpsk*	*sncf*	*gb-coach*	*gb-tr*
	Time range	(1 day)	(1 day)	(1 week)	(1 week)
USP-OR	n	700	1000	1000	700
	Speed-up	1	1	1	1
	Size-up	1	1	1	1
USP-OR-A	n	1905	2645	3915	2575
	Speed-up	1	1	1	1
	Size-up	1	1	1	1

Table 3: *Speed-ups and size-ups of USP-OR and USP-OR-A with Locsep. Datasets from cp.sk, SNCF (French railways) and Great Britain coaches and trains.*

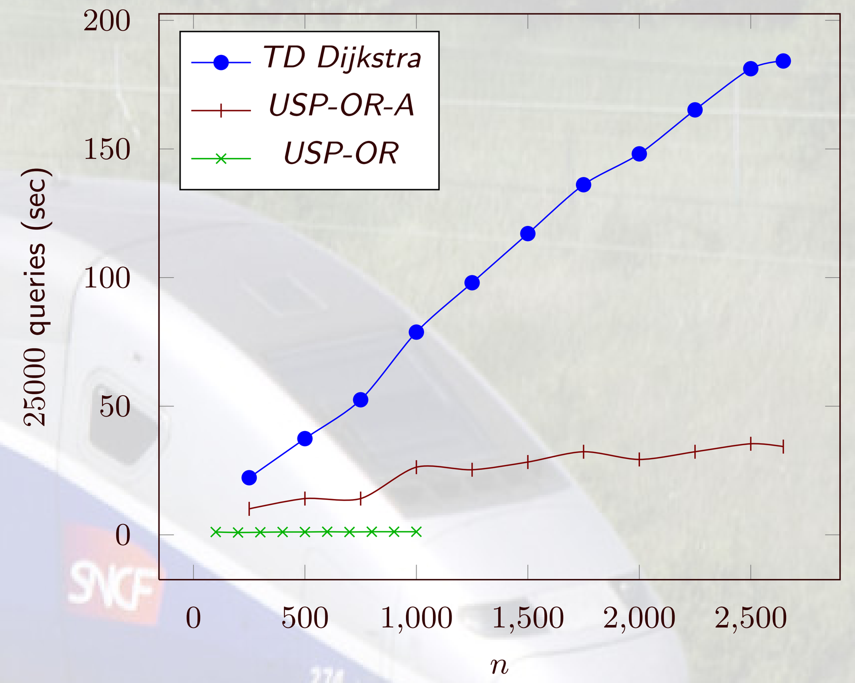


Figure 5: ***Query time** of USP-OR-A with Locsep compared to TD Dijkstra on the **sncf** dataset. **Changing n**.*

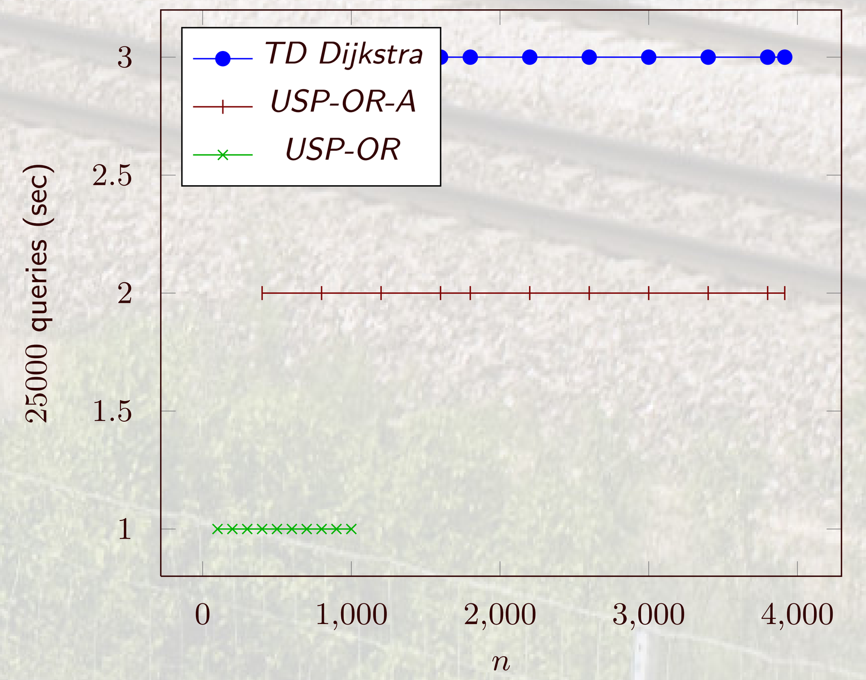


Figure 6: ***Query time** of USP-OR-A with Locsep compared to TD Dijkstra on the **gb-coach** dataset. **Changing n**.*

References

[Müller-Hannemann et al., 2007] Müller-Hannemann, M., Schulz, F., Wagner, D., and Zaroliagis, C. (2007). *Algorithmic Methods for Railway Optimization*, volume 4359 of *Lecture Notes in Computer Science*, chapter Timetable Information: Models and Algorithms, pages 67 – 90. Springer.

[Sommer, 2010] Sommer, C. (2010). *Approximate Shortest Path and Distance Queries in Networks*. PhD thesis, Graduate School of Information Science and Technology, The University of Tokyo.