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 $c^*_{(a,t,b)}$:

[*] ferohajnovic@gmail.sk

[†] kralovic@dcs.fmph.uniba.sk

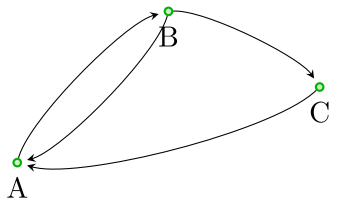

Figure 1.1: An underlying graph of the timetable 1.1.

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

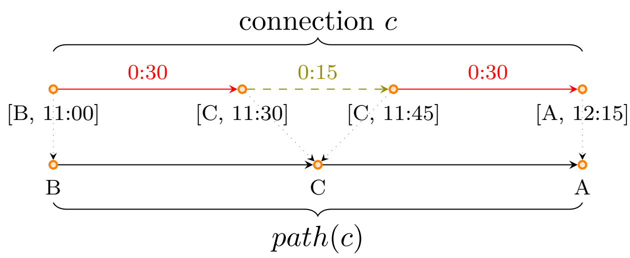

Figure 1.2: The *path* function applied on a connection to get the underlying path.

2 Approaches

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.

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** [Delling and Wagner, 2009] $e_2 \iff depart(e_1) > depart(e_2)$ 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 $\boldsymbol{\frac{m}{n}}$ - 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*.

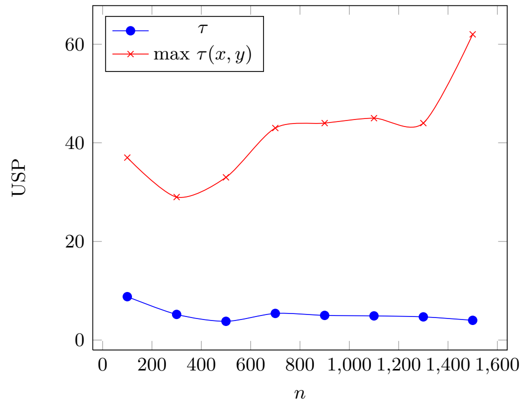

Figure 2.1: Changing of τ with increased number of stations in *sncf* dataset.

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** $\boldsymbol{neigh_{\mathcal{A}}(x)}$ as all the cities reachable in ug_T *not* via ANs. The access nodes within this neighbourhood are called **local access nodes** (LANs, $\boldsymbol{lan_{\mathcal{A}}(x)}$). 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 $neigh_{\mathcal{A}}(x)$, $bneigh_{\mathcal{A}}(x)$, $lan_{\mathcal{A}}(x)$ and $blan_{\mathcal{A}}(x)$

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 2.2 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 **(r_1, r_2, r_3) AN set** if:

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

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

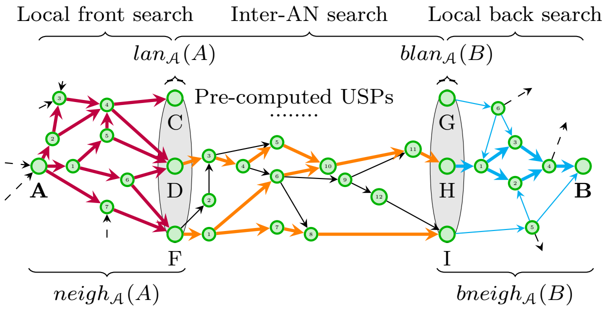

Figure 2.2: Principle of *USP-OR-A* algorithm. The arcs in **bold** mark areas that will be explored: all nodes in $neigh_{\mathcal{A}}(x)$, USPs between LANs of x and back LANs of y and the back neighbourhood of y (possibly only part of it will be explored, since the local back search goes against the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

lists the parameters of *USP-OR-A* for timetables with specific properties (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).

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.,** $\boldsymbol{\gamma \le \sqrt{n}}$**,** $\boldsymbol{\delta \le \log n}$
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.

2.3 Selecting access nodes

The challenge in *USP-OR-A* algorithm therefore comes down to the selection of a good access node set. However, consider the following problem: *minimize* $|\mathcal{A}|$ *such that* $\forall x \notin \mathcal{A} : |neigh_{\mathcal{A}}(x)| \le$

$\sqrt{n}$. We call this the problem of the optimal AN set.

Theorem 1. *The problem of the optimal AN set is NP-complete*

Proof. We will provide a sketch of the proof, which in full extend would be available in [Hajnovic, 2013]. We will make a reduction of the *min-set cover* problem to the problem of optimal AN set.

Consider an instance of the min-set cover problem:

- A universe $U = \{1, 2, ..., m\}$
- k subsets of U: $S_i \subseteq U$ $i = \{1, 2, ..., k\}$ whose union is U: $\bigcup_{1 \leq i \leq k} S_i = U$

Denote $\mathcal{S} = \{S_i | \ 1 \leq i \leq k\}$. The task is to choose the smallest subset $\mathcal{S}^*$ of $\mathcal{S}$ that still covers the universe ($\bigcup_{S_i \in \mathcal{S}^*} S_i = U$). For each $j \in U$, we will make a complete graph of β_i vertices (the value of β_i will be discussed later) named m_j and for each set S_i we make a vertex s_i and vertex s'_i. We now connect all vertices of m_j to $s_i \iff j \in S_i$. Finally, for we connect s_i to s'_i for $1 \leq i \leq k$.

Example. Let $m = 10$ (thus $U = \{1, 2, ..., 10\}$) and $k = 13$:

- $S_1 = \{1, 3, 10\}$
- $S_2 = \{1, 2\}$
- ...
- $S_{13} = \{2, 3, 10\}$

For this instance of min set-cover, we construct the graph depicted on picture 2.3.

Define α_i to be the number of sets S_j that contain i: $\alpha_i = |\{S_j \in \mathcal{S} | \ i \in S_j\}|$ and assume the constructed graph has n vertices. We want the β_i to satisfy $\beta_i \geq 2$ and $\beta_i + 2\alpha_i - 1 \leq \sqrt{n}$ but $\beta_i + 2\alpha_i > \sqrt{n}$. The last two inequalities would mean that if at least one s_j connected to m_i is chosen as an access node, the neighbourhood for nodes in m_i will be still $\leq \sqrt{n}$, but if none of them is chosen, the neighbourhood will be just over $\sqrt{n}$. We leave out the details of the construction at this place.

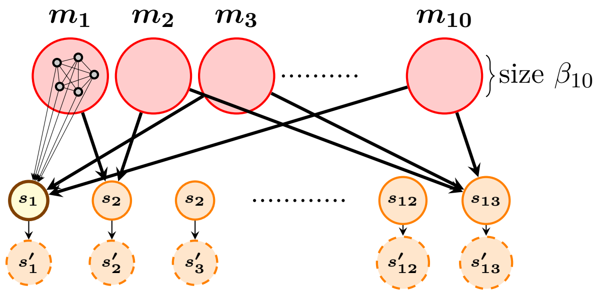

Figure 2.3: In m_i, there are actually complete graphs of β_i vertices (as shown for m_1). **Thick** arcs represent arcs from all the vertices of respective m_i. The s_i vertices are connected to their s'_i versions. If e.g. s_1 is selected as an access node, s'_1 is no longer part of any neighbourhood.

Now consider an optimal AN set which contains a vertex from within some m_i. If this is the case, **either** some s_j to which m_i is connected is selected as AN, **or** *all* vertices from m_i are access nodes **or** the neighbourhood is too large. Keep in mind that the local access nodes are also part of neighbourhoods, so unless we select for AN some of the s_j that m_i is connected to, the neighbourhood of any non-access node in m_i will be too large. As there are at least two nodes in every m_i, it is more efficient to select some s_j rather then select all nodes in m_i. Thus when it comes to selecting ANs *it is worth to consider only vertices s_j.*

From this point on, it is easy to see that it is optimal to select those s_j that correspond to the optimal solution of min-set cover. The reason is that each of the m_i will be connected to at least one access node s_j and will thus have neighbourhood size $\leq \sqrt{n}$, while the number of selected access nodes will be optimal.

□

We have therefore approached selection a good AN set heuristically. We iteratively selected ANs by their importance until the average square of the neighbourhood size was not $\leq \sqrt{n}$ (i.e. the r_1 parameter of the set was ≤ 1). To estimate the city's importance, we tried three values:

- Degree of the node in ug_T
- Betweenness centrality [Brandes, 2001] of the node in ug_T

- Our own value called **potential**, high for those nodes that are good local separators in ug_T

Our algorithm, called *Locsep*, computes the city's potential in the following way: we explore an area $\boldsymbol{A_x}$ of $\sqrt{n}$ nearest cities around x. We do this in an underlying graph with no orientation and no weights. Next we get the front and back neighbourhoods of x within A_x ($\boldsymbol{fn(x)} = neigh(x) \cap A_x$, $\boldsymbol{bn(x)} = bneigh(x) \cap A_x$). For a set of access nodes $\mathcal{A}$, let us call a path p in ug_T **access-free** if it does not contain a node from $\mathcal{A}$. Now as long as x is not in $\mathcal{A}$, there is a guarantee that for every pair $u \in bn(x)$ and $v \in fn(x)$ there is an access-free path from u to v within A_x. Our interest is how this will change after the selection of x.

Let $bneigh_i = bneigh(b_i) \cap A_x$ for each arc $(b_i, x) \in ug_T$. We run a restricted (to $A_x \setminus \{x\}$) search from each $bneigh_i$ during which we explore e_i vertices in $fn(x)$. This is going to contribute up to $e_i|bneigh_i|$ to x's potential, depending on if the cities in $bneigh_i$ actually have large neighbourhoods (and thus selecting x would help). More details regarding the *Locsep* algorithm will be in [Hajnovic, 2013].

3 Results

4 Conclusion

Acknowledgments

I would like to thank very much to my supervisor Rastislav Královič for valuable remarks, useful advices and consultations that helped me stay on the right path during my work on this paper.

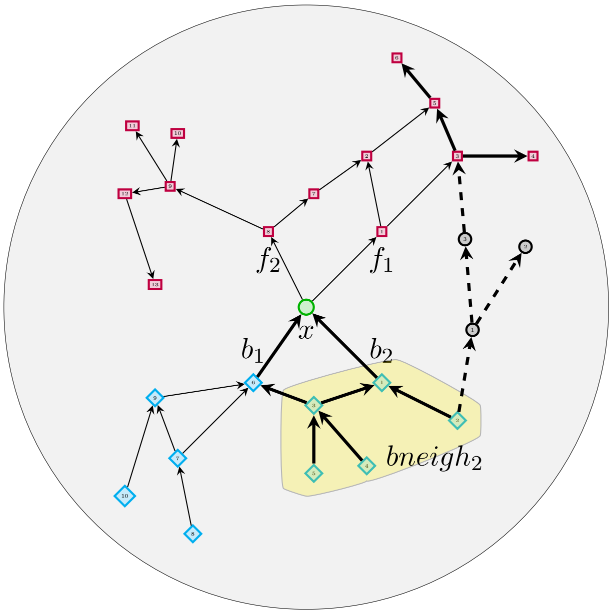

Figure 2.4: The principle of computing potentials in Locsep algorithm. We explored an area of $\sqrt{n}$ nearest cities (in terms of hops) around x. Little squares are nodes from $neigh(x)$ and diamonds are part of $bneigh(x)$. The highlighted area represents the back neighbourhood for node b_2. From its nodes we run a forward search (the **thick** arcs). Nodes from the $neigh(x)$ that were not explored in this search can only be reached via x itself and contribute to x's potential.

References

[Pow,] Information about scale free networks.

[DBL, 2009a] (2009a). *50th Annual IEEE Symposium on Foundations of Computer Science, FOCS 2009, October 25-27, 2009, Atlanta, Georgia, USA*. IEEE Computer Society.

[DBL, 2009b] (2009b). *Proceedings of the 25th International Conference on Data Engineering, ICDE 2009, March 29 2009 - April 2 2009, Shanghai, China*. IEEE.

[Abraham et al., 2010] Abraham, I., Fiat, A., Goldberg, A. V., and Werneck, R. F. F. (2010). Highway dimension, shortest paths, and provably efficient algorithms. In [Charikar, 2010], pages 782–793.

[Arge et al., 2004] Arge, L., Italiano, G. F., and Sedgewick, R., editors (2004). *Proceedings of the Sixth Workshop on Algorithm Engineering and Experiments and the First Workshop on Analytic Algorithmics and Combinatorics, New Orleans, LA, USA, January 10, 2004*. SIAM.

[Bast, 2009] Bast, H. (2009). Efficient algorithms.

[Bast et al., 2006] Bast, H., Funke, S., and Matijevic, D. (2006). Transit— ultrafast shortest-path queries with linear-time preprocessing.

[Bast et al.,] Bast, H., Funke, S., Matijevic, D., Sanders, P., and Schultes, D. In transit to constant time shortest-path queries in road networks.

[Batz et al., 2009] Batz, G. V., Delling, D., Sanders, P., and Vetter, C. (2009). Time-dependent contraction hierarchies. In [Finocchi and Hershberger, 2009], pages 97–105.

[Blelloch, 2002] Blelloch, G. (2002). Graph separators.

[Bose and Morin, 2002] Bose, P. and Morin, P., editors (2002). *Algorithms and Computation, 13th International Symposium, ISAAC 2002 Vancouver, BC, Canada, November 21-23, 2002, Proceedings*, volume 2518 of *Lecture Notes in Computer Science*. Springer.

[Brandes, 2001] Brandes, U. (2001). A faster algorithm for betweenness centrality. *Journal of Mathematical Sociology*, 25:163–177.

[Brodal and Leonardi, 2005] Brodal, G. S. and Leonardi, S., editors (2005). *Algorithms - ESA 2005, 13th Annual European Symposium, Palma de Mallorca, Spain, October 3-6, 2005, Proceedings*, volume 3669 of *Lecture Notes in Computer Science*. Springer.

[Charikar, 2010] Charikar, M., editor (2010). *Proceedings of the Twenty-First Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2010, Austin, Texas, USA, January 17-19, 2010*. SIAM.

[Chechik et al., 2010] Chechik, S., Langberg, M., Peleg, D., and Roditty, L. (2010). f-sensitivity distance oracles and routing schemes. In de Berg, M. and Meyer, U., editors, *Algorithms – ESA 2010*, volume 6346 of *Lecture Notes in Computer Science*, pages 84–96. Springer Berlin / Heidelberg.

[Chen et al., 2009] Chen, W., Sommer, C., Teng, S.-H., and Wang, Y. (2009). A compact routing scheme and approximate distance oracle for power-law graphs. *Distributed Computing*, ?(Disc 2009):379–391.

[Chowdhury and Ramachandran, 2002] Chowdhury, R. A. and Ramachandran, V. (2002). Improved distance oracles for avoiding link-failure. In [Bose and Morin, 2002], pages 523–534.

[Cohen et al., 2003] Cohen, E., Halperin, E., Kaplan, H., and Zwick, U. (2003). Reachability and distance queries via 2-hop labels. *SIAM J. Comput.*, 32(5):1338–1355.

[CompSci767, 2011] CompSci767 (2011). Hierarchical planning.

[Davison et al., 2010] Davison, B. D., Suel, T., Craswell, N., and Liu, B., editors (2010). *Proceedings of the Third International Conference on Web Search and Web Data Mining, WSDM 2010, New York, NY, USA, February 4-6, 2010*. ACM.

[Delling, 2008] Delling, D. (2008). Time-dependent sharc-routing. In [Halperin and Mehlhorn, 2008], pages 332–343.

[Delling et al., 2009a] Delling, D., Pajor, T., and Wagner, D. (2009a). Engineering time-expanded graphs for faster timetable information. In Ahuja, R., Möhring, R., and Zaroliagis, C., editors, *Robust and Online Large-Scale Optimization*, volume 5868 of *Lecture Notes in Computer Science*, pages 182–206. Springer Berlin / Heidelberg.

[Delling et al., 2009b] Delling, D., Sanders, P., Schultes, D., and Wagner, D. (2009b). Engineering route planning algorithms. In *ALGORITHMICS OF LARGE AND COMPLEX NETWORKS. LECTURE NOTES IN COMPUTER SCIENCE*. Springer.

[Delling and Wagner, 2009] Delling, D. and Wagner, D. (2009). Time-dependent route planning. In *Robust and Online Large-Scale Optimization, LNCS*. Springer.

[Dijkstra, 1959] Dijkstra, E. W. (1959). A note on two problems in connexion with graphs. *NUMERISCHE MATHEMATIK*, 1(1):269–271.

[Erickson,] Erickson, J. Graph separators.

[Farnish,] Farnish, K. A matter of scale.

[Finocchi and Hershberger, 2009] Finocchi, I. and Hershberger, J., editors (2009). *Proceedings of the Workshop on Algorithm Engineering and Experiments, ALENEX 2009, New York, New York, USA, January 3, 2009*. SIAM.

[Fredman, 2008] Fredman, C. (2008). *Ramsey Partitions Based Approximate Distance Oracles*. PhD thesis, The Open University Of Israel.

[Gavoille et al., 2004] Gavoille, C., Peleg, D., Pérennes, S., and Raz, R. (2004). Distance labeling in graphs. *Journal of Algorithms*, 53(1):85 – 112.

[Geisberger et al., 2008] Geisberger, R., Sanders, P., Schultes, D., and Delling, D. (2008). Contraction hierarchies: Faster and simpler hierarchical routing in road networks. In [McGeoch, 2008], pages 319–333.

[Goldberg et al., 2006] Goldberg, A. V., Kaplan, H., and Werneck, R. F. (2006). Reach for a*: Efficient point-to-point shortest path algorithms.

[Hajnovic, 2013] Hajnovic, F. (2013). Distance oracles for timetable graphs. Master's thesis, Faculty of Mathematics, Physics and Informatics, Comenius University in Bratislava.

[Hajnovič, 2012] Hajnovič, F. (2012). Distance oracles for timetable graphs - research.

[Halperin and Mehlhorn, 2008] Halperin, D. and Mehlhorn, K., editors (2008). *Algorithms - ESA 2008, 16th Annual European Symposium, Karlsruhe, Germany, September 15-17, 2008. Proceedings*, volume 5193 of *Lecture Notes in Computer Science*. Springer.

[imhd.sk,] imhd.sk. Mapa mhd ba.

[Kannan et al., 1988] Kannan, S., Naor, M., and Rudich, S. (1988). Implicit representation of graphs. In [Simon, 1988], pages 334–343.

[Kerber, 0405] Kerber, M. (2004/05). Hierarchical planning.

[Kinahan and Pryor,] Kinahan, K. and Pryor, J. Dijkstra's algorithm for shortest route problems.

[Lee,] Lee, J. Separator.

[Markošová,] Markošová, M. Základy umelej inteligencie 1 - prednášky.

[McGeoch, 2008] McGeoch, C. C., editor (2008). *Experimental Algorithms, 7th International Workshop, WEA 2008, Provincetown, MA, USA, May 30-June 1, 2008, Proceedings*, volume 5038 of *Lecture Notes in Computer Science*. Springer.

[Mozes and Sommer, 2010] Mozes, S. and Sommer, C. (2010). Exact shortest path queries for planar graphs using linear space. *CoRR*, abs/1011.5549.

[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.

[of problem solving,] of problem solving, A. Root-mean square-arithmetic mean-geometric mean-harmonic mean inequality.

[Pyrga et al., 2004] Pyrga, E., Schulz, F., Wagner, D., and Zaroliagis, C. D. (2004). Experimental comparison of shortest path approaches for timetable information. In [Arge et al., 2004], pages 88–99.

[Sanders, 2008] Sanders, P. (2008). Time dependent contraction hierarchies – basic algorithmic ideas. *CoRR*, abs/0804.3947.

[Sanders and Schultes, 2005] Sanders, P. and Schultes, D. (2005). Highway hierarchies hasten exact shortest path queries. In [Brodal and Leonardi, 2005], pages 568–579.

[Sankaranarayanan and Samet, 2009] Sankaranarayanan, J. and Samet, H. (2009). Distance oracles for spatial networks. In [DBL, 2009b], pages 652–663.

[Sarma et al., 2010] Sarma, A. D., Gollapudi, S., Najork, M., and Panigrahy, R. (2010). A sketch-based distance oracle for web-scale graphs. In [Davison et al., 2010], pages 401–410.

[Schultes, 2008] Schultes, D. (2008). *Route Planning in Road Networks*. VDM Verlag, Saarbrücken, Germany, Germany.

[Schulz et al., 2002] Schulz, F., Wagner, D., and Zaroliagis, C. (2002). Using multi-level graphs for timetable information in railway systems. In Mount, D. and Stein, C., editors, *Algorithm Engineering and Experiments*, volume 2409 of *Lecture Notes in Computer Science*, pages 43–59. Springer Berlin / Heidelberg.

[Simon, 1988] Simon, J., editor (1988). *Proceedings of the 20th Annual ACM Symposium on Theory of Computing, May 2-4, 1988, Chicago, Illinois, USA*. ACM.

[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.

[Sommer et al., 2009] Sommer, C., Verbin, E., and Yu, W. (2009). Distance oracles for sparse graphs. In [DBL, 2009a], pages 703–712.

[TASR,] TASR. Tretie kolo diskusie o nosnom dopravnom systéme v bratislave bude 27.6.

[Thorup and Zwick, 2005] Thorup, M. and Zwick, U. (2005). Approximate distance oracles. *J. ACM*, 52(1):1–24.

[Wang and Chen, 2003] Wang, X. F. and Chen, G. (2003). Complex networks: small-world, scale-free and beyond. *Circuits and Systems Magazine, IEEE*, 3(1):6–20.

[Watts and Strogatz, 1998] Watts, D. J. and Strogatz, S. H. (1998). Collective dynamics of 'small-world' networks. *Nature*, 393(6684):440–442.

[Wikipedia,] Wikipedia. London underground - wikipedia.

[Ďuriš, 1997] Ďuriš, P. (1997). Tvorba efektívnych algoritmov - prednášky.