Department of Computer Science,
Faculty of Mathematics, Physics and Informatics
in Bratislava,
Comenius University

Distance oracles for timetable graphs

(Master thesis)

František Hajnovič

Program: Informatics 9.2.1
Advisor: doc. RNDr. Rastislav Královič, PhD. Bratislava 2013

I hereby declare that I wrote this thesis by myself, only with the help of the referenced literature, under the careful supervision of my thesis advisor.

....

Acknowledgements

I would like to thank ...

Abstract

This thesis deals with the shortest path queries on timetable graphs - i.e. graphs that represent timetables (of e.g. Slovak bus network). Firstly we investigate, if some properties (such as low highway dimension) are propagated (and to what extent) from the underlying graph to the timetable graph. Based on these facts, we show how to compute reasonably fast an accurate distance oracle on timetable graphs, that would efficiently answer shortest path queries, thus finding quick timetable connections between any pair of nodes.

Key words: **distance oracles**, **timetable graphs**, **timetable**, **highway dimension**

Abstrakt

V tejto práci...

Klúčové slová: **distance oracles**, **timetable graphs**, **timetable**, **highway dimension**

Contents

1 Introduction

World is getting smaller every day, as new technologies constantly make communication and traveling faster and more effective then yesterday. Road network, Internet and many other networks are becoming more evolved and denser which also brings along new problems. In order to fully take advantage of such huge networks, we must have efficient algorithms that operate on these networks and give us answers to many questions. Among many others, one that we take particular interest in is the question: "What is the shortest path from place x to place y"?

In different networks, this question can make different sense. In the road network, we would like to obtain a sequence of intersections we have to visit in order to reach our destination, driving the shortest possible time (or the smallest possible distance) . GPS devices and the likes of Google maps have to deal with this problem. In the case of Internet network, we might be interested in the shortest path to a destination computer in terms of router hops. In a network of social acquaintances, the smallest number of persons connecting us e.g. with guitarist Mark Knopfler or Liona Boyd could be expressed as a shortest path problem. Many problems in artificial intelligence (e.g. planning of actions) can be expressed (or include) as a shortest path problem.

Place		**Time**	
From	**To**	**Departure**	**Arrival**
A	B	10:00	10:45
A	B	11:00	11:45
A	B	12:00	12:45
A	C	9:30	10:00
A	C	10:15	10:45
C	D	11:00	11:30
C	D	13:00	13:30
C	D	12:20	12:35
C	D	12:40	12:55
C	D	13:00	13:15
C	B	12:20	12:50
C	B	13:30	14:00
D	A	13:00	14:00

Table 1: An example of a timetable

The importance of finding a shortest path in a graph is also obvious from the amount of algorithms and approaches we have nowadays to tackle this problem. In this thesis, we will focus on a more specific problem - finding a shortest connection from place x at the time t to place y, given a timetable of connections between nodes in the graph. This problem can be transformed and reduced to the original shortest path problem, but we may take advantage of the specific structure of the graphs representing timetables and find shortest connections more efficiently. Let us formulate the problem at hand more formally.

Intuitively - a timetable is just a set of *elementary connections* between pairs of nodes in a graph. Let us therefore introduce the definition of a 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.

Note: G may be directed or undirected - definition is the same for both. Also, we usually meet with timetables using hours or minutes when specifying time. We can easily convert between such time units and natural numbers, so in what follows, we will freely use time expressed in its classical form instead of natural numbers, as found convenient.

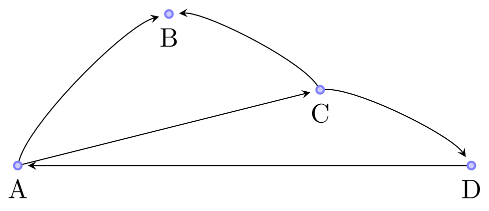

Figure 1: Underlying graph of the timetable in 1

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, $e_i = (e^i_x, e^i_y, e^i_p, e^i_q)$, *such that* $e^1_x = a$, $e^k_x = b$ *and* $\forall i \in \{2, ..., k\} : (e^i_x = e^{i-1}_y,\ e^i_p \geq e^{i-1}_q)$.
Connection ***starts*** *at the* departure time e^1_p.
Size (length) *of the connection is* $e^k_q - e^1_p$.

Definition 3. ***Shortest timetable connection (STC) problem***
Given a graph G*, two of its vertices* $x \neq y$*, integer* $t \in N$ *and a timetable on the graph* T_G*, what is the shortest* connection *from* x *to* y *that* starts *at a* time $s \geq t$.

1.1 Applications

We have already mentioned a few applications of the algorithm looking for shortest path in a graph. Many interesting examples can be found in the Christian Sommer's PhD thesis [Som10]. As for finding the shortest timetable connection, following applications serve as a motivation:

- **Search engines for (inter)national railways**: These are search engines where one specifies the desired departure station and time and destination station and leaves the job on the engine to find the shortest connection. Often we meet with advanced systems that incorporate train, bus and even airplane timetables all into the same search engine, or offer extra features such as required time for a change, traveling cost restrictions etc...
-

1.2 Traditional approaches to shortest path problem

We will shortly present some of the well known algorithms that solve shortest path problems.

1.2.1 Dijkstra's algorithm

The most famous of all, Dijkstra's algorithm, was published in 1959 by Edsger Dijkstra. It's original implementation runs in time $\mathcal{O}(n^2)$ (where $n = |V_G|$) and finds the shortest path from a given vertex v to every other vertex in a directed graph (if the graph is strongly connected), thus producing a so called *shortest path tree* (that is, a tree rooted at v with every path starting at v being a shortest path in the original graph) . A restriction is posed that edge weights must be positive.

Briefly sketching the idea, in one iteration the algorithm visits the closest (with respect to the starting point v) so far unvisited vertex and updates the estimates of distances to all of its neighbors. The idea is to maintain an invariant, that for a visited node we know the correct shortest distance from v which can be achieved just by moving along already visited vertices [Ďur97].

Dijkstra's algorithm has been adjusted and tuned many times to achieve higher performance (better time complexity). Currently best bound is provided by an implementation using Fibonacci heap as a priority queue for next-visited-node selection. The resulting time complexity is $\mathcal{O}(m+n \log n)$ (where $m = |E_G|$) [Som10].

1.2.2 Bellman-Ford algorithm

Bellman-Ford algorithm considers also the graphs with negative edge weights. It differs from Dijkstra's algorithm mostly in the fact, that it does not apply the greedy next-visited-node selection (which prohibits the negative edge weights), but instead *relaxes* each edge $n - 1$ times, hence the time complexity $\mathcal{O}(nm)$. The relaxation itself means checking if the estimate of the distance to the edge's endpoint could be improved with estimate of the edge's start point plus the weight of the edge.

Because of the negative edge weights, algorithm must also check for negative cycles, which, however, does not increase the overall complexity. Similarly to Dijkstra's algorithm, it produces a shortest path tree rooted at the starting vertex.

1.2.3 A* search algorithm

A* is an algorithm that uses heuristics to speed up finding of the optimal path from a starting vertex v to the destination vertex u. The next visited node is the one that minimizes the (true) distance from v plus the heuristic estimate of distance to u. The complexity of the A* algorithm depends on the heuristics. In the worst case, it may have to expand number of vertices exponential in the shortest path length. Only if the heuristics h provides a very good estimate [1], the number of expanded vertices is polynomial in the size of the shortest path [Mar]. Unfortunately, computing the heuristics itself in that case may require too much time and thus considerably increase the time complexity of the algorithm.

1.2.4 Floyd-Warshall algorithm

From the category of dynamic programming, Floyd-Warshall algorithm computes shortest paths between all pairs of vertices in $\mathcal{O}(n^3)$.

1.3 Problem with traditional approaches

With the arsenal of algorithms (from which we have mentioned just a few), one may wonder about our contribution to the STC problem, which may, as we have stated, be reduced to finding shortest path in a specific graph. That is, of course, a possible approach - the mentioned algorithms are polynomial, almost linear in time complexity. However, consider a road network of Europe, which may contain tens of millions of intersections and even several times more road segments (connections between two intersections) . If an online server hosting interactive map (like Google Maps) run Dijkstra's algorithm on such a large scale network for every query, it would not be able to answer those queries in real time, as required by users.

Other straightforward approach would precompute every shortest path and then answer shortest path queries in a constant time. That would, however, require a large table of size $\mathcal{O}(n^3)$ [2], or at least $\mathcal{O}(n^2)$ [3]. In any case, the construction itself (not to mention the required memory) is very costly - with Floyd-Warshall algorithm and the mentioned scale, the computation might take more than $(10^7)^3 = 10^{21}$ steps.

We have yet to mention timetables in the previous discussion. In the case of the graph representing a given timetable *on a graph of the size of the European road network* (in the section 2, there are descriptions how to construct timetable graphs), we obtain even bigger input for the shortest path problem. Later in the Preliminaries section, we will give more details about these graphs.

1.4 Shortest paths via distance oracles

Due to the limitations described above, many other approaches were explored. The term *Distance oracle* (DO) was first coined in 2001 by Thorup and Zwick [TZ05] (for a definition, see section 2) . The DO based method for shortest path retrieval consists of a preprocessing of the graph to a structure (the distance oracle), which then answers (not necessarily correctly) queries for shortest path, using as little computation as possible.

Essentially, the DO method may be characterized by four parameters, all of which we try to minimize, but among which we have to make compromises:

1. preprocessing time
2. size of the resulting structure
3. query time
4. *stretch* - worst case ratio agains optimal path size

The approaches we have mentioned are the extreme cases, where some parameters are completely minimized while others are too large.

1.5 Organization of this paper

This thesis is organized in following sections:

1. **Preliminaries**: we formulate necessary definitions and timetable graph constructions used throughout the rest of the thesis

[1]By good estimate, we mean $|h(x) - h*(x)| = \mathcal{O}(log*(x))$ where $h*(x)$ is the optimal heuristics (true distance)
[2]For every pair of vertices, we would store the shortest path
[3]We store just the last vertex on the way to the destination, but the query then takes $\mathcal{O}(n)$ in the worst case

2. **Related work**: main results in the area of distance oracles, shortest path queries and distance labelling, useful in the following sections
3. **Properties propagation upon time-expansion**: we show how (and to what extent) certain properties of the graph are preserved when time-expanded to a timetable graph
4. **Proposed distance oracles for timetable graphs**: we present our approach to solve STC problem, based on distance oracles
5. **Implementation and results**: we present our implementation and obtained results of the algorithm's behavior in practice
6. **Conclusion**: we conclude

Each section may contain several sub-sections and further sub-sub-sections. For words in *italic*, there is a definition provided in this thesis.

2 Preliminaries

In this section, we would like to introduce terminology and definitions used later on in the thesis.

We assume the reader to be familiar with some standard notions from graph theory such as *graph*, *subgraph*, *path*... Much of the terminology we will use can be found in [Som10]. We will provide only the definitions of not well established terminology and our own definitions.

If not stated otherwise, by a graph we will mean a directed graph (*digraph*) with non-negative edge (*arc*) weights.

2.1 Common definitions

Definition 4. ***Distance between nodes*** x ***and*** y ***in graph*** G
Denoted $d_G(x, y)$*, distance between* x *and* y *in* G *is the length of the shortest path from* x *to* y*:* $d_G(x, y) = \min\limits_{P \in \mathcal{P}(x,y)} l(P)$ *where* $l(P)$ *is the length of the path* P *and* $\mathcal{P}(x, y)$ *is the set of all paths from* x *to* y *in* G*.*

Definition 5. $(\boldsymbol{prepro_M}, \boldsymbol{size_M}, \boldsymbol{qtime_M}, \boldsymbol{stretch_M})$ ***Distance oracle for a graphs*** G
A $(prepro_M, size_M, qtime_M, stretch_M)$ *distance oracle* M *for a graph* G *is a data structure characterized by following properties:*

1. M *could be created in a* ***preprocessing time*** $prepro_M$ *from graph* G
2. M *takes up* $size_M$ *bits of memory (**DO size**)*
3. M *answers queries for shortest path between pair of vertices in* G *in maximum time of* $qtime_M$ *(**query time** of DO)*
4. *the answer provided by* M *is not worse than* $stretch_M$ *times the optimum value (**stretch** of DO)*

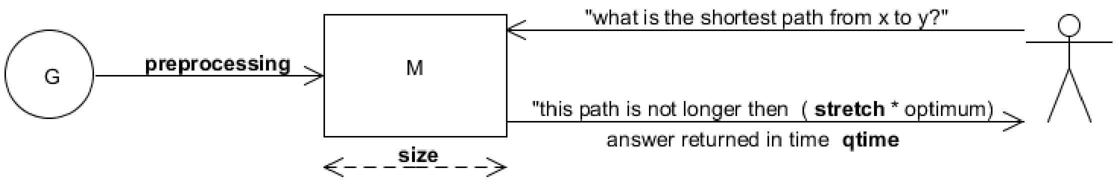
G
preprocessing
M
"what is the shortest path from x to y?"
"this path is not longer then (stretch * optimum)
answer returned in time qtime
size

Figure 2: Diagram showing the properties of a distance oracle

Definition 6. $(\boldsymbol{prepro_F(G)}, \boldsymbol{size_F(G)}, \boldsymbol{qtime_F(G)}, \boldsymbol{stretch_F(G)})$ ***Distance oracle method for a class of graphs*** $\mathcal{G}$
Let $prepro_F(G)$*,* $size_F(G)$ *and* $qtime_F(G)$ *be functions from* $\mathcal{G}$ *to* N *and* $stretch_F(G)$ *a function from* $\mathcal{G}$ *to* R*. A* $(prepro_F(G), size_F(G), qtime_F(G), stretch_F(G))$ *distance oracle method* F *for a class of graphs* $\mathcal{G}$ *is an algorithm, that for each graph* $G' \in \mathcal{G}$ *produces a* $(prepro_F(G'),$ $size_F(G'),$ $qtime_F(G'),$ $stretch_F(G'))$ distance oracle.
*We will refer to the four functions of the DO method as **preprocessing time, DO size, query time** and **stretch** respectively.*

Note 1: alternatively, we may write $(\mathcal{O}(f(n)), \mathcal{O}(g(n))...)$ *DO method*, meaning that $\exists prepro_F(G) = \mathcal{O}(f(n))$ and $\exists size_F(G) = \mathcal{O}(g(n))$ ... such that we have corresponding DO method.

Note 2: in [TZ05], no proper definition of a distance oracle was made. In [Som10] a term DO means both - an algorithm together with the data structure it produces. We saw fit to split the definition and refer to the structure itself as a distance oracle, and to the algorithm as a distance oracle method.

2.2 Classes of graphs

Definition 7. ***Power-law graphs*** *[CSTW09]*
Power-law graphs is a class of graphs, in which a number of nodes with degree x is proportional to x^τ, for a constant $\tau \in (2,3)$.

We also meet with term **scale-free network** or **small world**.
Many real-world networks have power-law structure, or at least are though to have. One such example is airline network, which nicely demonstrate the definition of a power-law graph. There are several (not many) huge airports, that are linked together accross the world. Apart from these links, they have plenty of other connections to other, smaller airports, possibly within continent or surrounding countries. The smaller airports do not have many connections but on the other hand are more ubiquitous. Overall, we can usually get from one airport to any other one following a short chain of flights, thus the notion of a *small world.*

2.3 Graph properties

2.3.1 Highway dimension

Definition 8. ***Highway dimension for an undirected graph*** G *[AFGW10]*
Highway dimension (HD) for an undirected, edge-weighted graph G is the smallest integer h, such that

$$\forall r \in R^+, \forall u \in V_G, \exists S \subseteq B_{u,4r}, |S| \leq h, \text{ such that } \forall v, w \in B_{u,4r} : \\ \text{if } |P(v,w)| > r \text{ and } P(v,w) \subseteq B_{u,4r} \text{ then } P(v,w) \cap S \neq \emptyset$$

*where $B_{u,r} = \{v \in G | d(u,v) \leq r\}$ and is called **ball** of radius r centered at u.*

Intuitively, a graph has a low HD, if for any r we have a *sparse* set of vertices S_r, such that every shortest path longer then r includes a vertex from S_r. By the set being sparse, we mean that every ball of radius $\mathcal{O}(r)$ contains just a few elements of S_r.

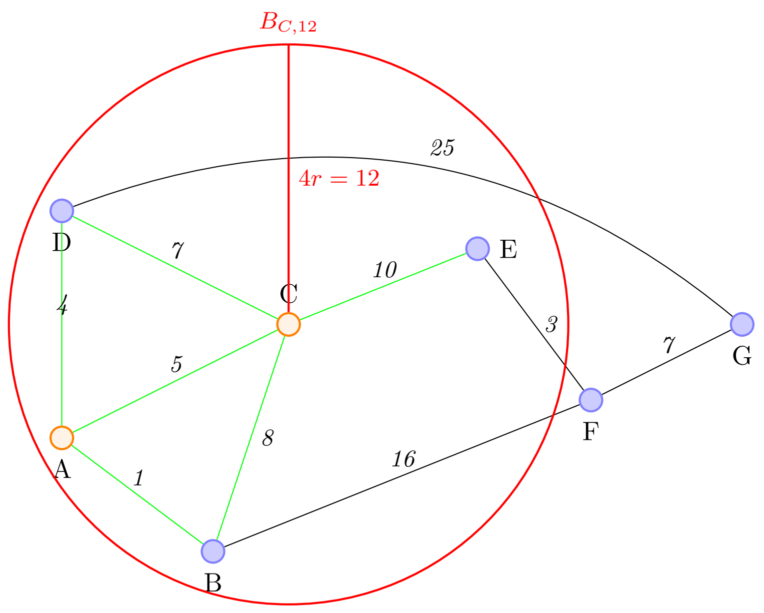

Figure 3: Demonstration of a definition of HD. We chose some r ($r = 3$) and some vertex v ($v = C$) to root the ball $B_{v,4r}$. All the shortest paths *longer* than r *inside* the ball have to contain a vertex from S (orange vertices C and A in our case). The upper bound on $|S|$, considering any ball with any radius, is the required highway dimension. Note: in our case, we had to choose also A to set S, since a shortest path from B to D does not include C.

The definition is for undirected graphs. A similar definition can be obtained for directed graphs, using *shortest path covers.*

Definition 9. ***(r, k) Shortest path cover ((r, k)-SPC)*** *[AFGW10]*
A set C is a (r, k)-SPC of G if and only if $\forall u \in V_G, |C \cap B_{u,2r}| \leq k$ and for every shortest path $P : r < |P| \leq 2r, P \cap C \neq \emptyset$.

Depending on if the graph is directed/undirected, we choose *directed/undirected ball* in the above definition.

Definition 10. ***Directed ball***
*A **directed ball** $B_{u,r} = \{v \in G | d(u, v) \leq r \text{ or } d(v, u) \leq r\}$*

Definition 11. ***Highway dimension for an directed graph*** *G [AFGW10]*
Highway dimension (HD) for an directed, edge-weighted graph G is the smallest integer h, such that $\forall r \exists (r, h) - SPC$.

Note: in the above definition, we use shortest path covers with directed balls.

Definition 12. *$r(n)$* ***separator***

2.4 Timetable graphs

In this subsection, we will discuss the construction of a timetable graph out of the given timetable. Terminology *time-expanded* and *time-dependent* graph was inspired by [MHSWZ07].

Definition 13. ***Time-expanded graph***
Let T be a timetable on G. Time-expanded graph from T is an oriented graph G_T whose vertices are pairs $[v, t], v \in G$ and $\exists (x, y, p, q) \in T$ such that $p = t$ or $q = t$. Edges of G_T are formed by
1. *$([x, p], [y, q]), ;\ \forall (x, y, p, q) \in T$*
2. *$([x, p], [x, q]), [x, p], [x, q] \in V_{G_T}, p < q$ and $\nexists [x, r] \in V_{G_T} : p < r < q$.*

Weight w of any edge $((x, p), (y, q))$ in E_{G_T} is $w((x, p), (y, q)) = q - p$.

Intuitively, nodes in the time-expanded graph correspond to events - when a connection arrives or departs from a given place. Edges are then *elementary connections* and a waiting at a given place.

If we wanted to use algorithms for shortest path to find the optimal connection, we would need a little further modification, where for every $x \in V_G$ (G being the underlying graph), we add a new vertex x_{dest} to G_T and edges $([x, p], x_{dest}), ;\ \forall [x, p] \in V_{G_T}$ with zero weight (see figure 5). We can then ask about shortest path from $[x, t]$ to y that would correspond to looking for optimal connection from x to y at a time t.

Sometimes it will be more convenient, if all the events were happening at times being multiple of some value.

Definition 14. ***Approximate time-expanded graph***
Let T be a timetable on G and $k \in N$. Approximate time-expanded graph from T is an oriented graph $G_{T,k}$ whose vertices are $[v, t], v \in G, t = k \cdot l, l \in N$. Edges of $G_{T,k}$ are formed by
1. *$([x, p'], [y, q']), ;\ \forall (x, y, p, q) \in T$ where $p' = \lfloor \frac{p}{k} \rfloor \cdot k$ and $q' = \lceil \frac{q}{k} \rceil \cdot k$*
2. *$([x, p], [x, q]), [x, p], [x, q] \in V_{G_T}, p < q$ and $\nexists [x, r] \in V_{G_T} : p < r < q$.*

Edge weights are defined as with time-expanded graphs.
*We will call k a **granularity** of the approximate time-expanded graph.*

It is possible that we lose some information in the approximate time-expanded graph. We will analyze this later on, to show how much information could be lost based on a given *granularity* and how to fix this.

Consider the London Underground system. It serves 270 stations with 11 lines with a high operating frequency (about 10 minutes on average) and serving above 30 stations on average (note that more lines might serve the same station) [Wik]. Therefore, we expect an event to occurs at a given station every $10 \cdot (\frac{270}{11*30}) \approx 8$ minutes. If we took a timetable of the underground for one day and made a time-expanded graph from it, how many nodes would it have? London Underground operates

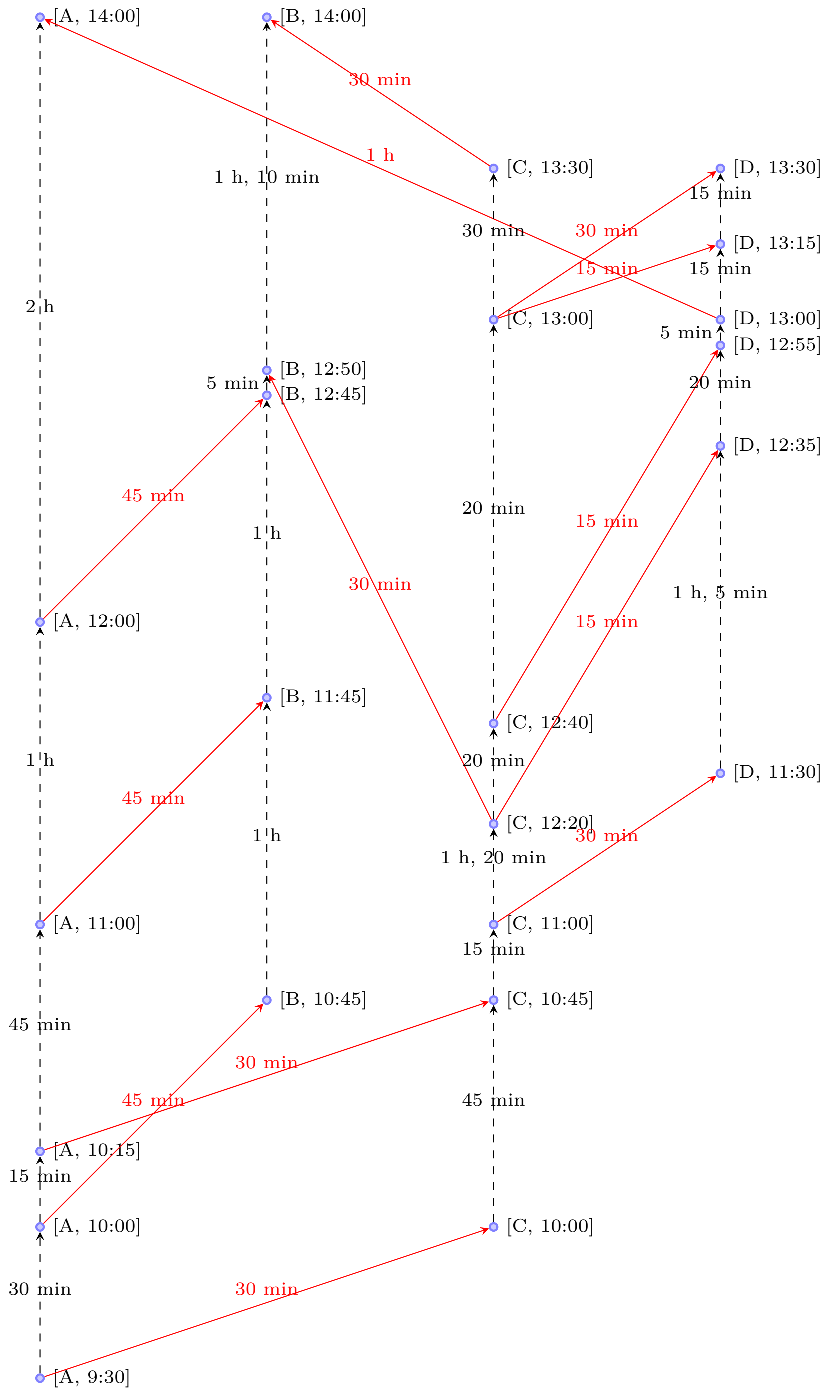

Figure 4: Time-expanded graph from the timetable in 1

about 20 hours a day, which is 1200 minutes. The 8 minute event interval yields about 150 events per station per day. Thus, overall, we have a time-expanded graph with 34500 vertices.

An approximate time-expanded graph with *granularity* $k = 1$ would, in this case, have $270 \cdot 20 \cdot 60 = 324000$ vertices.

Both graphs are possible to construct in a linear time with respect to their sizes. They can then serve (with slight adjustment as in 5) for throughout the day to look for shortest connections using e.g. Dijkstra's algorithm, answering queries relatively quickly.

However, this was an example of just a small network with 270 nodes in the *underlying*

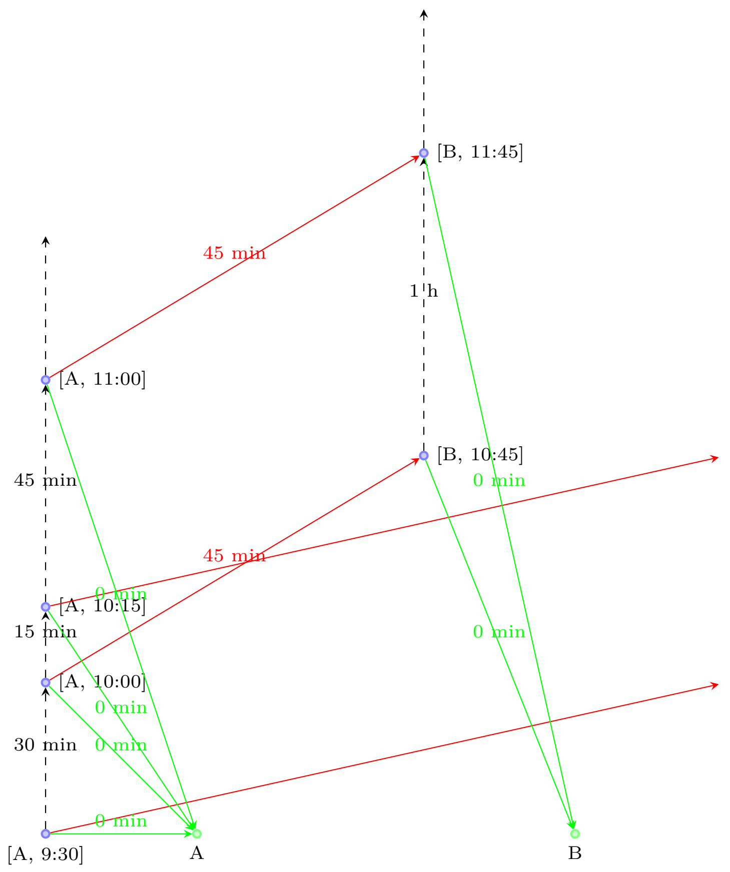

Figure 5: A portion of the time-expanded graph from the timetable in 1, modified (green edges and vertices) to enable finding optimal connections via shortest path queries

graph of the timetable. The numbers (of nodes in the expanded graphs) we obtain suggest, that such an approach would indeed fail for timetables on a large-scale graphs, as e.g. those for Europe-wide railway system.

Let us introduce another type of a timetable graph, that is more compact but does not provide all the timetable information.

Definition 15. ***Time-dependent graph*** *[MHSWZ07]*
Let T be a timetable on G. Time-dependent graph from T is a non-weighted oriented graph $\widetilde{G_T}$ whose vertex set $V_{\widetilde{G_T}} = V_G$ and there is an edge between x and y if $\exists(x, y, p, q) \in T$.

As specified in [MHSWZ07], "The lengths of the edges are assigned "on-the-fly": the length of an edge depends on the time, in which it will be used in the algorithm...". Hence the name *time-dependent graph*. This type of graph will not be used much in this thesis, and by **timetable graph**, we will generally refer to the *time-expanded graph* from a given timetable.

3 Related work

Much of the work done in the area of distance oracles and shortest path queries can be found in a summary [Haj12]. Most relevant, with regard to this thesis, are the results in [TZ05], [AFGW10], [GPPR04].

An important result was reached by Thorup and Zwick, when they found a DO method for general graphs, that makes it possible to trade *stretch* for a better *preprocessing time* and *DO size*.

Theorem 1. ***(Thorup and Zwick)***
For the class of general undirected weighted graphs, there is a $(\mathcal{O}(kmn^{1/k}), \mathcal{O}(kn^{1+1/k}), \mathcal{O}(k), 2k-1)$ *DO method.*

4 Properties propagation upon time-expansion

In this section, we would like to show how certain properties are propagated from the *underlying graph* upon time-expansion.

Following properties will be considered:

- Highway dimension
- $r(n)$-separator
- graph planarity

5 Proposed distance oracles for timetable graphs

6 Implementation and results

Following programs are planned to be developed:

- Simple program for creating time-expanded graph out of given timetable (C++)
- Simple program for converting a graph defined in XML to a picture in PDF format (C++)
- Program with an interface for testing DO methods and making statistics (Java)
- Implementation of a proposed DO method for timetable graphs

7 Conclusion

References

[AFGW10] Ittai Abraham, Amos Fiat, Andrew V. Goldberg, and Renato Fonseca F. Werneck. Highway dimension, shortest paths, and provably efficient algorithms. In Moses Charikar, editor, *SODA*, pages 782–793. SIAM, 2010.

[CSTW09] Wei Chen, Christian Sommer, Shang-Hua Teng, and Yajun Wang. A compact routing scheme and approximate distance oracle for power-law graphs. *Distributed Computing*, ?(Disc 2009):379–391, 2009. URL `http://research.microsoft.com/pubs/81716/msr-tr-2009-84.pdf`.

[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`.

[Haj12] František Hajnovič. Distance oracles for timetable graphs - research, 2012.

[Mar] Mária Markošová. Základy umelej inteligencie 1 - prednášky.

[MHSWZ07] Matthias Müller-Hannemann, Frank Schulz, Dorothea Wagner, and Christos Zaroliagis. *Algorithmic Methods for Railway Optimization*, volume 4359 of *Lecture Notes in Computer Science*, chapter Timetable Information: Models and Algorithms, pages 67 – 90. Springer, 2007.

[Som10] Christian Sommer. *Approximate Shortest Path and Distance Queries in Networks.* PhD thesis, Graduate School of Information Science and Technology, The University of Tokyo, 2010.

[TZ05] Mikkel Thorup and Uri Zwick. Approximate distance oracles. *J. ACM*, 52(1):1–24, 2005.

[Ďur97] Pavol Ďuriš. Tvorba efektívnych algoritmov - prednášky, 1997.

[Wik] Wikipedia. London underground - wikipedia.