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

Distance oracles for timetable graphs

(Master thesis)

bc. František Hajnovič

Study program: Informatics
Branch of study: 9.2.1 Informatics (2508)
Supervisor: doc. RNDr. Rastislav Královič, PhD.

Bratislava 2013

Comenius University in Bratislava
Faculty of Mathematics, Physics and Informatics

THESIS ASSIGNMENT

Name and Surname:	Bc. František Hajnovič
Study programme:	Computer Science (Single degree study, master II. deg., full time form)
Field of Study:	9.2.1. Computer Science, Informatics
Type of Thesis:	Diploma Thesis
Language of Thesis:	English
Secondary language:	Slovak

Title: Distance oracles for timetable graphs

Aim: The aim of the thesis is to explore the applicability of results about distance oracles to timetable graphs. It is known that for general graphs no efficient distance oracles exist, however, they can be constructed for many classes of graphs. Graphs defined by timetables of regular transport carriers form a specific class which it is not known to admit efficient distance oracles. The thesis should investigate to which extent the known desirable properties (e.g. small highway dimension) are present int these graphs, and/or identify new ones. Analytical study of graph operations and/or experimental verification on real data form two possible approaches to the topic.

Supervisor:	doc. RNDr. Rastislav Královič, PhD.
Department:	FMFI.KI - Department of Computer Science
Vedúci katedry:	doc. RNDr. Daniel Olejár, PhD.
Assigned:	08.11.2011

Approved: 15.11.2011 prof. RNDr. Branislav Rovan, PhD.
Guarantor of Study Programme

.....
Student

.....
Supervisor

65787891

Univerzita Komenského v Bratislave
Fakulta matematiky, fyziky a informatiky

ZADANIE ZÁVEREČNEJ PRÁCE

Meno a priezvisko študenta:	Bc. František Hajnovič
Študijný program:	informatika (Jednoodborové štúdium, magisterský II. st., denná forma)
Študijný odbor:	9.2.1. informatika
Typ záverečnej práce:	diplomová
Jazyk záverečnej práce:	anglický
Sekundárny jazyk:	slovenský

Názov: Efektívny výpočet vzdialeností v grafoch spojení lineik.

Cieľ: Cieľom práce je preštudovať možnosti aplikácie výsledkov o distance oracles v grafoch reprezentujúcich dopravné siete na grafy spojení liniek. Otázka, či a aké dôležité vlastnosti ostávajú zachované sa dá riešiť teoreticky pre rôzne triedy grafov a/alebo experimentálne pre reálne dáta.

Vedúci:	doc. RNDr. Rastislav Královič, PhD.
Katedra:	FMFI.KI - Katedra informatiky
Vedúci katedry:	doc. RNDr. Daniel Olejár, PhD.
Dátum zadania:	08.11.2011

Dátum schválenia: 15.11.2011

prof. RNDr. Branislav Rovan, PhD.
garant študijného programu

.....
študent

.....
vedúci práce

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 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 thesis.

I am also grateful for the support of my family during my studies and the work on this thesis.

František Hajnovič

Abstract

Queries for optimal connection in timetables can be answered by running Dijkstra's algorithm on an appropriate graph. However, in certain scenarios this approach is not fast enough. In this thesis we introduce methods with much better query time than that of the efficiently implemented Dijkstra's algorithm. We analyse these methods from both theoretical and practical point of view, performing experiments on various real-world timetables of country-wide scale.

Our first method called *USP-OR* is based on pre-computing paths, that are worth to follow (the so called underlying shortest paths). This method achieves speed-ups of up to 70 (against Dijkstra's algorithm), although at the cost of high amount of preprocessed data. Our second algorithm computes a small set of important stations and additional information for optimal travelling between these stations. Named *USP-OR-A*, this method is much less space consuming but still more than 8 times faster than the Dijkstra's algorithm on some of the real-world datasets.

Key words: **optimal connection**, **timetable**, **Dijkstra's algorithm**, **distance oracles**, **underlying shortest paths**

Abstrakt

Optimálne spojenia v cestovnom poriadku vieme hľadať pomocou Dijkstrovho algoritmu na vhodnom grafe, avšak v niektorých situáciách tento prístup výkonnostne nepostačuje. V tejto práci uvádzame metódy, ktoré na dotaz na optimálne spojenie odpovedajú podstatne rýchlejšie ako efektívna implementácia Dijkstrovho algoritmu. Tieto metódy analyzujeme ako z teoretického, tak aj z praktického hľadiska pomocou experimentov na viacerých cestovných poriadkoch celonárodnej škály.

Naša prvá metóda nazvaná *USP-OR* je založená na predpočítaní trás, ktoré sa oplatí následovať (tzv. podkladové najkratšie cesty). Táto metóda dosahuje faktor zrýchlenia až 70 (oproti Dijkstrovmu algoritmu), avšak za cenu veľkej pamäťovej náročnosti. Náš druhý algoritmus predrátava malú množinu dôležitých staníc a dodatočné informácie pre optimálne cestovanie medzi nimi. Táto metóda s názvom *USP-OR-A* je už oveľa menej pamäťovo náročná, stále vyše 8 krát rýchlejšia ako Dijkstrov algoritmus na niektorých reálnych cestovných poriadkoch.

Kľúčové slová: **optimálne spojenie**, **cestovný poriadok**, **Dijkstrov algoritmus**, **dištančné orákulá**, **podkladové najkratšie cesty**

Contents

1 Introduction

2 Preliminaries

3 Related work

4 Data & analysis

5 Underlying shortest paths

6 Neural network approach

7 Application TTBlazer

For the purposes of analysing our timetables and testing our methods we developed a C++ command line application named *TTBlazer*, which we will now shortly describe. The application is *not* meant for common users but rather for those who would like to try out algorithms by themselves, review their code or continue with our work.

The application works with 4 main types of **objects** that we have introduced in the section 2:

- Underlying graph
- Time-expanded graph
- Time-dependent graph
- Timetable

A user can save/load each of this object from a file (in appendix A we describe the file formats). Once loaded, there are four types of **actions** to be performed on the objects:

- **Analysing** properties of the objects (e.g. analysis of degree distribution of the underlying graph)
- **Generating** other objects from those that are loaded. This includes also conversions e.g. from a timetable to the time-expanded graph. Another example may be extracting the largest strongly connected component of a graph
- **Modifying** the object, e.g. removing overtaken connections from a timetable
- **Creating an oracle** on the object. We distinguish two types of oracles:
  - **Distance oracle**. Answers queries for *shortest-path* or a *distance* between two nodes
  - **Timetable oracle**. Answers queries for *optimal connection* or the *earliest arrival* between two cities, given the departure time

  Once the oracle is created (preprocessing is finished), the user can query it for the optimal shortest-paths/connections

There is one more auxiliary type of *action* that can be carried out - we call it **posting**. A postman (posting method) generates something (a mail) and stores it (in a postbox) to be later retrieved by another action. For example, we compute cities in the underlying graph with high betweenness centrality value, store the computed set and then use it to pre-compute *USP-OR-A* using the set as an access node set.

The main purpose of the application is therefore to serve as an environment for algorithms that manipulate timetables and other objects. The list of implemented actions can be found in the table 7.1.

Action type	Action	Objects	Description
Analysing	usp	TE, TD	analyses the underlying shortest paths *(created by posting action)*
	conns	TE, TD	analyses the optimal connections (avg. size, length...)
	hd	UG, TE, TD	analyses the highway dimension
	var	TE, TD, TT	analyses various properties of timetable objects (height
	conn	UG, TE	analyses connectivity
	strconn	UG	analyses strong connectivity of the underlying graph
	degs	UG	analyses degrees of the underlying graph
	paths	UG	analyses shortest paths (avg. size, length...)
	betw	UG, TE	analyses betweenness centralities
	accn	UG	analyses access node set *(created by posting action)*
	density	UG	analyses the density of the underlying graph
	overtake	TT	analyses overtaking in a timetable
Generating	subcon	UG	generates connected subgraph of the given UG
	strcomp	UG	generates the UG with nodes from the largest strongly connected component
	2ug	TE, TD, TT	generates the underlying graph from given timetable object
	2te	TT	generates the time-expanded graph from the given timetable
	2td	TT	generates the time-dependent graph from the given timetable
	subtt	TT	generates a sub-timetable from the given timetable
Modifying	rmover	TT	removes overtaking from given timetable
Timetable oracle	neural	TE, TD	creates the oracle based on a neural network
	uspor	TD	creates the oracle based on *USP-OR*
	usporseg	TD	creates the oracle based on *USP-OR*, uses segmentation
	uspora	TD	creates the oracle based on *USP-OR-A (requires access node set computed by posting action)*
	usporaseg	TD	creates the oracle based on *USP-OR-A*, uses segmentation *(requires access node set computed by posting action)*
	dijkstra	TE, TD	creates the oracle based on time-dependent Dijkstra's algorithm
Distance oracle	dijkstra	UG	creates the oracle based on Dijkstra's algorithm
Posting	anhbc	UG	creates the access node set $\mathcal{A}^{bc}$
	andeg	UG	creates the access node set $\mathcal{A}^{deg}$
	usp	TE, TD	computes the USPs (between all pairs or just on a given subset)
	locsep	UG	creates the access node set $\mathcal{A}^{loc}$

Table 7.1: A list of actions implemented in *TTBlazer*.

7.1 Requirements & features

7.1.1 Communication with user

- The program may be fed commands either through standard command line input, or through UDP port on which it is listening. The user may switch between these two options to specify the so called *command source*.
- The output is done through logging of the program. The logs could be output to screen, written to files or sent on ports. There are 3 types of logs:
  - *Info*: information for the user, thus always printed
  - *Error*: also always printed
  - *Debug message*: information for the developer, printed only if debugging is turned on. The debug message is further parametrized by a number (*debug level*), usually a different one for each module of the program.
- Help outputs the list of commands along with the explanation of their meaning. The application can also take several command-line *arguments*, e.g. setting the logging to a specific file.
- The application can run scripts with commands stored in a file

7.1.2 Others

- All the actions performed on the objects could be *timed* and the size of the created oracles can be estimated by the lower bound.
- Objects and oracles can be referenced by their name or by their index.
- Nodes of the graph can be referenced by their index or their ID.
- The user has an option to switch between viewing all the time values in *minutes* or in the time format *DAYS HH:MM*.

8 Compilation and usage

In the directory of the solution, five folders can be found:

- *common/*
- *commander/*
- *data/*
- *printer/*
- *ttblazer/*

There are actually 3 programs - *TTBlazer*, *Printer* and *Commander*, whereas *common* are just common source files shared between them all. Printer and Commander are simple support applications - the first one (Printer) listens on a port and output everything to *stdout* (used when logs are sent on a port). The second one (Commander) just sends a message provided as an argument to the specific port and terminates (used to control TTBlazer).

Each of these folders has a file *buildinfo.sh* specifying (relative) paths to source files folder, binary folder, name of the final binary, compilation flags and then individual modules (one per source file). This file is used by *depmake.sh*, a bash script that creates a makefile from the information from *buildinfo.sh*.

In order to compile the programs, boost library must be installed in the parent folder ("solution" in this case) in a directory called "boostlib". It should look like this:

```
1 solution/
2 |-- boostlib
3 |   |-- boost
4 |       |-- accumulators
5 |       |   |-- accumulators_fwd.hpp
6 |       |   |-- accumulators.hpp
7 |       |   |-- framework
8 ...
```

Listing 1: Sending commands to the main application

Boost library is freely available at `http://sourceforge.net/projects/boost/files/boost/1.52.0/`. In case you have the library installed, you may also adjust the files *buildinfo.sh* located in directories *ttblazer*, *commander* and *printer* where you provide the path of your boost library distribution to the include flag for *g++* compiler.

Finally, for each of the 3 programs there are following 3 simple shell script:

- *comp.sh* - rebuilds the makefile, compiles the sources and builds the binary. Its arguments are forwarded to *make*
- *run.sh* - runs the binary. Arguments are forwarded to the started binary
- *both.sh* - combination of *comp.sh* and *run.sh*. Arguments are forwarded to the started binary

In the folder *data*, there are 4 files - one for each type of object mentioned in section **??**. Following is a simple demonstration of one of the program's functionality (creating and using oracles). Open two terminal windows. In one of them, compile and run the main application as follows:

```
./tboth.sh
```

Listing 2: Compiling and running main application

In the other terminal window you will send commands to the main application with the Commander program.

```
./ccomp.sh                              //compile the Commander program
./crun.sh load tt data/basic/tt.tt      //load the timetable from file
./crun.sh gen tt 2te tt                 //generate the time-expanded graph from the
    loaded timetable
./crun.sh or te dijkstra tt_2te         //create oracle (based on Dijkstra's
    algorithm) on the time-expanded graph
./crun.sh conn te tt_2te 0 A 0 10:00 D  //use the oracle to find out earliest
    arrival for some query
```

Listing 3: Sending commands to the main application

8.1 Design

As already mentioned, there are 3 applications that share some common parts:

- *TTBlazer* - the main application, that carries all the logic and does all the computation.
- *Commander* - small program that sends commands to TTBlazer. The advantage of this is, that even when the main application is busy computing, it still listens on a port for commands to be executed later.
- *Printer* - small program that listens on a port and outputs everything that it receives. Used to capture some logging.
- *Common* - common files shared by all three applications (like logging, working with network connections etc...)

TTBlazer is further structured. It offers basically 4 types of functionality:

- *Analysers* - does various analysis on the objects (e.g. analysis of connectivity)
- *Generators* - generates some object from another object (e.g. connected sub-graph)
- *Modifiers* - modifies existing loaded objects (e.g. sorts the timetable)
- *Oracles* - creates data structure that can answer shortest-path or earliest-arrival queries

The application is easily extendible for new functionality of one of the types.

9 Conclusion

Appendices

A File formats