r-2012-11-29

František Hajnovič
ferohajnovic@gmail.com

January 30, 2013

Contents

1 Diagnostic application

The application is named Timetable analyser (*ttblazer* for short). Following are the requirements, design of the application and supported file formats.

1.1 Requirements

```
1  MAIN PURPOSE
2  The application will primarily serve to :
3  - analyze properties (such as separators, highway dimension, scale-free-ness, planarity...)
       of graphs/timetables
4  - generate various timetables (with various properties) on top of underlying graphs
5  - test distance oracle methods for graphs/timetable graphs and make statistics
6  - visualise graphs and timetables
7
8  CONTROLING
9  The program will be controled by commands from command line. There will be another thread
       listening on a socket for commands to be executed, that will be put into queue. This
       may come handy when one would like to create a bash script to e.g. load several
       timetables, run specific DO methods on the timetable and retrieve statistics on the
       result.
10
11 TIME COMPLEXITY \& STABILITY
12 The main concern when creating the framework of the application will not be the
       optimizations for speed but for stability and quality. However, the complexity
       optimization will be important in design of the algorithms run by the application.
13
14 MODULARITY
```

```
As the application serves mostly as a framework to run and evaluate different algorithms, it should be made easy to add new algorithms. We propose a c++ library for each algorithm that will provide an interface to run and evaluate the algorithm, output help to the specific algorithm etc...

LOAD/SAVE
The application should distinguish several types of objects:
- underlying graph - oriented weighted graph with possible list of lines for each arc that operate on the given arc
- timetable - set of elementary connections
- time-expanded graph - graph representation of a given timetable (is just oriented weighted graph)
- time-dependent graph - time-dependent representation of a given timetable

ANALYZING FUNCTIONS
For a given underlying graph, time-expanded graph, time-dependent graph
- average edge length [average profile of the edge for T-D graph], maximum/minimum length ...
- connectivity
- strong connectivity
- average degree of the graph
- highway dimension
- separator
- planar separator
- planarity
- scale-free-ness
For a given timetable
- average/maximum/minimum traversal time
- time range
- height

GENERATIVE FUNCTIONS
For a given timetable
- underlying graph
- time-expanded graph
- time-dependent graph
For a given underlying graph
- random timetable
  - hold on/don't hold on to the underlying edge weights when determining connection lengths
  - time range
  - height
  - overtaking
  - regular timetables
- express lines timetable
  - rules for express lines...

DISTANCE ORACLE METHODS
We will distinguish two types of distance oracles
- DO for graphs -> responds to queries for shortest path (SP)
- DO for timetables -> responds to queries for earliest arrival (EA)
Implementation of querying for actual shortest path (series of connections in case of DO for timetables) can be left out, though there should be then a generic algorithm that is able to output shortest paths provided we already have a DO for distances ( analogically in case of EA). Algorithm that implements queries for actual paths (series of connections) will be noted as SP+ (EA+).
Each distance oracle method will have (at least) 4 measurable aspects:
- preprocessing time
- size of preprocessed structure
- query time
- stretch
DO should be able to output the help specifying:
- usage - how to set individual parameters
- prerequisits - some DO works only on some types of graphs
Following DO methods should be implemented:
- Dijkstra's algorithm (SP+, EA+)
- Floyd Warshall algorithm (SP+, EA+)
- Neural networks (SP+, EA+)
- SHARC (SP)
- Gavoille (SP)
- Gavoille for timetables (EA)
```

```
75
76 STATISTICS OF DO METHODS
77 - preprocessing time
78 - query time
79 - structure size
80 - stretch (against - default Dijkstra's algorithm)
81
82 VISUALISING FUNCTIONS (optional)
83 - visualise statistics (through python)
84 - visualise graphs, timetables
```

Listing 1: Requirements

1.2 Design

```
1  The whole project will consist of several subsystems. They in turn consist of classes, or
       simply files that differ in functionality.
2
3  First, distinguish following applications:
4  - common - not exactly application, rather libraries used by all/some others applications
5  - ttblazer - the main application
6  - printer - prints messages send by ttblazer
7  - commander - commands ttblazer
8
9  COMMON
10 Implements logging, sender and receiver classes that implement communication on sockets,
       some common constants and definitions...
11 Logging:
12 - has 3 types of logs - INFO (meant for user), DEB (meant for developer) and ERR (errors -
       meant for both).
13 - has 2 ways of outputing - to stream or on socket.
14 - DEB is further parametrized by "debug levels" (usually one for each file) that may be
       turned on/off
15
16 PRINTER
17 Listens on specified port and outputs received messages
18
19 COMMANDER
20 Sends message provided on command line to specified port
21
22 TTBLAZER
23 May be further divided into several parts:
24 Communication with user:
25   - Main - infinite loop, main thread
26         - runs infinite loop waiting for commands (e.g. load graph.txt, list graphs,
               dijkstra graph1 a b...)
27     - CmdProcessor - process next command from the queue
28     - CmdlnProcessor - process command line arguments (they set the program settings, they
           do not run actions, algorithms...)
29     - Communicator - infinite loop, separate thread
30         - waits on a given port for commands to be executed. Commands are put to the
               CmdProcessor queue
31         - Commander application sends commands to Communicator
32 Entities
33   - Graph - used in implementation of UnderGraph, TdGraph, TeGraph
34   - UnderGraph
35   - Timetable
36   - TeGraph - Time-expanded graph
37   - TdGraph - Time-dependent graph
38 Logic
39     - save, load entities
40     - applying generators, analyzers, DOs, visualisers
41 Generators
42   - algorithms for e.g. generating timetable from underlying graph..
43 Analyzers
44   - algorithms to analyze entities
45 Distance oracles
46   - algorithms implementing distance oracle methods
47 Visualisers (optional)
```

```
48   - simple visualisations of the graph
```

Listing 2: Requirements

1.3 Files format

1.3.1 Underlying graph

Underlying graph is basically an oriented graph with some (optional) further information:

- Coordinates of the nodes
- Lengths of the oriented edges
- List of lines operating on a given arc

```
1  4                                    //number of stations
2  5                                    //number of edges
3  A 45 32                              //name of the station, optional coordinates (triples
       - interpolation points: day time travel-minutes)
4  B null
5  C 56 34
6  D null
7  A B 57 Northern                      //FROM TO edge length, list of lines operating on
       the edge
8  A C null Picadilly Victoria          //edge length may be null (will be e.g. random, or
       calculated from coordinates)
9  C B 45 Circle Jubilee Picadilly
10 C D 32 null                          //list of lines may be also null
11 D A null null
```

Listing 3: Underlying graph files form

1.3.2 Timetable

A timetable is simply formed by elementary connections.

```
1  7                     //number of elementary connections
2  A B 0 10:00 0 10:45   //FROM TO DAY(depart) TIME(depart) DAY(arrive) TIME(arrive)
3  A B 0 11:00 0 11:45
4  A B 0 12:00 0 12:45
5  A C 0 9:30 0 10:00
6  A C 0 10:15 0 10:45
7  C D 0 11:00 0 11:30
8  C D 0 13:00 0 13:30
```

Listing 4: Timetable files form

1.3.3 Time-expanded graph

Time-expanded graph is simply an oriented weighted graph, with the weights being the traversal time. Also, names of the nodes are the combination of the city and the time that represent them.

```
1  5         //number of station/times
2  4         //number of connections
3  A 0 13:30
4  A 0 14:00
5  B 0 13:45
6  B 0 15:00
7  C 0 14:15
8  A 0 13:30 B 0 13:45
9  A 0 14:00 B 0 15:00
10 A 0 13:30 B 0 15:00
11 C 0 14:15 B 0 15:00
```

Listing 5: Time expanded graph files form

1.3.4 Time-dependent graph

Time-dependent graph is an oriented graph with a function on the arc specifying the arc's traversal time at any moment. In timetable networks this function is piece-wise linear and it is fully represented by the list of its interpolation points.

```
4                 //number of stations
5                 //number of edges
A 0 0               //FROM TO COST-FUNCTION
B 4 4
C 8 2
D 12 0
A B (0 13:30 45) (0 14:00 40)
A C (1 14:15 10)
C B (0 15:00 20)
C D (2 10:00 70)
D A null              //undefined cost function -> null
```

Listing 6: Time dependent graph files form

Note: We represent time (though not traversal time, which is expressed in minutes) in above mentioned formats usually as *D HH:MM*, that is a day followed by space and 24h format of time. Files that consider time only in minutes (e.g. *1 13:30* would become $(24+13)60+30 = 2250$) can also be loaded by the program.

2 Open points

- Hierarchy of express lines → what properties can be propagated in time-expansion?
- Instant cost function - more formal and details

3 To do

- United airlines extract data
- Road network of SVK - process data
- Continue the diagnostic program
- Properties propagation in simple timetables
- Machine learning