Bachelor’s Thesis
Agent-Based Public Transport Network Analysis

Jan Nykl

Supervisor: Jan Hrnčíř, MSc.

Study Programme: Cybernetics and Robotics
Field of Study: Robotics
May 24, 2013
BACHELOR PROJECT ASSIGNMENT

Student:       Jan Nýkl
Study programme: Cybernetics and Robotics
Specialisation: Robotics

Title of Bachelor Project: Agent-Based Public Transport Network Analysis

Guidelines:
1. Survey existing methods and metrics for public transport network (PTN) analysis.
2. Formalise the PTN analysis problem and propose metrics for the PTN analysis.
3. Propose an algorithm for the PTN analysis based on timetables of public transport services and travel demand.
4. Implement the proposed algorithm for the PTN analysis.
5. Based on the implemented algorithm, implement a PTN analyser tool capable of working with real-world timetable and travel demand data.
6. Evaluate the resulting PTN analyser on a set of test scenarios.

Bibliography/Sources: Will be provided by the supervisor.

Bachelor Project Supervisor: Mgr. Jan Hrnčíř

Valid until: the end of the winter semester of academic year 2013/2014

prof. Ing. Vladimír Mařík, DrSc. Head of Department
prof. Ing. Pavel Ripka, CSc. Dean

Prague, January 10, 2013
ZADÁNÍ BAKALÁŘSKÉ PRÁCE

Student:       Jan Nykl

Studijní program: Kybernetika a robotika (bakalářský)

Obor:          Robotika

Název tématu: Analýza sítí hromadné dopravy s využitím agentních technik

Pokyny pro vypracování:

1. Prozkoumejte existující metody a metriky pro analýzu sítí veřejné dopravy (SVD).
2. Matematicky zformulujte problém analýzy SVD a navrhněte metriky pro analýzu těchto sítí.
3. Navrhněte algoritmus pro analýzu SVD pracující na základě jízdních řádů a přepravní poptávky.
4. Implementujte navržený algoritmus.
5. Na základě funkčního algoritmu implementujte nástroj schopný pracovat s reálnými jízdními řády a reálnou přepravní poptávkou.
6. Vyhodnotte kvalitu nástroje na množině testovacích scénářů.

Seznam odborné literatury: Dodá vedoucí práce.

Vedoucí bakalářské práce: Mgr. Jan Hrnčíř

Platnost zadání: do konce zimního semestru 2013/2014

prof. Ing. Václav Mařík, DrSc. vedoucí katedry

prof. Ing. Pavel Ripka, CSc. děkan

V Praze dne 10. 1. 2013
Aknowledgements

I would like to thank my supervisor Jan Hrněff for his perfect guidance and useful advices. I would also like to thank the Agents4ITS team led by Michal Jakob for helping me with technical problems using their software and for their computational resources. Besides, I would like to thank my family and friends for their support.

The access to computing and storage facilities owned by parties and projects contributing to the National Grid Infrastructure MetaCentrum, provided under the programme "Projects of Large Infrastructure for Research, Development, and Innovations" (LM2010005) is highly appreciated/acknowledged.
Declaration

I hereby declare that I have completed this thesis independently and that I have used only the sources (literature, software, etc.) listed in the enclosed bibliography according to "Metodický pokyn o dodržování etických principů při přípravě vysokoškolských závěrečných prací".

Prohlašuji, že jsem předloženou práci vypracoval samostatně a že jsem uvedl veškeré použité informační zdroje v souladu s Metodickým pokynem o dodržování etických principů při přípravě vysokoškolských závěrečných prací.

V Praze dne 24. 5. 2013

[Signature]

vii
Abstract

The aim of this bachelor’s thesis is a fast and exact analysis of public transport networks. The analysis is computed based on travel demand, which may be generated by the agents. The metrics that can be measured on the public transport network and easily comparable, are described in this work. A journey planner with a speed up using patterns was developed, so that the analysis is fast enough. This journey planner is also described in detail in the thesis. At the end, we performed three example tests on the public transport network in Budapest, Rennes and Quebec. The aim of these tests was to simulate the run of our product in real use as much as possible. The journey planner was tested as well, the results of this test are also presented in this paper.
Abstrakt

Contents

1 Introduction ........................................ 1
  1.1 Work overview .................................. 1
  1.2 Motivation ..................................... 2

2 Background ........................................ 3
  2.1 Data structures .................................. 3
    2.1.1 Time-expanded graph ....................... 3
    2.1.2 Time-dependent graph with constant transfer time .... 3
    2.1.3 Choosing a model ........................... 3
  2.2 Algorithms ..................................... 4
    2.2.1 Dijkstra’s algorithm ......................... 4
    2.2.2 Patterns in the PTN ........................... 4
    2.2.3 Gift wrapping algorithm ...................... 5

3 Problem Specification .............................. 7
  3.1 PTN graph interpretation ........................ 7
    3.1.1 Graph vertices .............................. 7
    3.1.2 Graph edges ................................ 7
  3.2 Journey plan .................................. 8
  3.3 Analysis mode .................................. 8
    3.3.1 Many-to-many real demand .................. 8
    3.3.2 Many-to-many generated demand .......... 9
    3.3.3 One-to-others ............................. 9
    3.3.4 Static ..................................... 9
  3.4 Static metrics .................................. 9
    3.4.1 Node degree ................................ 9
    3.4.2 Harness ................................... 10
    3.4.3 Area covered by public transport .......... 10
    3.4.4 Length of all lines ........................ 11
  3.5 Dynamic metrics ................................ 11
    3.5.1 Duration of journey ......................... 11
    3.5.2 Number of interchanges ..................... 12
    3.5.3 Betweenness centrality .................... 12
    3.5.4 Trip workload ................................ 12

4 Solution approach ................................ 13
  4.1 Merging queries ................................ 13
  4.2 Extended Dijkstra’s algorithm .................. 14
  4.3 Patterns ....................................... 15
    4.3.1 Pattern factory and Pattern boxes .......... 15
    4.3.2 Generating patterns ......................... 15
    4.3.3 Using patterns ................................ 15
1 Introduction

This work focuses on developing a tool that computes exact public transport network (PTN) metrics, which can be easily compared with other cities, or other public transport timetables for one city. The development is described in detail, all the data structures, metrics and algorithms can be implemented based on this paper.

For computation, we selected lot of different metrics, which were divided into two groups. Static metrics that depend just on the network, such as covered area, node degrees or lines length, and dynamic metrics computed from travel demand for given city, such as average travel time or betweenness centrality. We hope that we have chosen metrics that allow us to analyse each PTN really well.

Journey plans are computed for agents, who provide us with the start stop, the goal stop and the time when they want to travel.

For dynamic metrics we wanted to compute plans for each agent separately for the result to be as accurate as possible. This approach has been chosen because a tool that computes PTN analysis based on every single query does not exists to our best knowledge. The closest similliar tool is [17], which is currently under construction and does not allow to compute all desired metrics.

The above described problems lead to a side product: a very fast multi-query trip planner for PTN, which allows us to evaluate up to millions of queries on PTN within just a few hours, was developed.

Lot of different references are used in the work, but I would like to point out the 2 most important. It is Disertation from Dr. rer. nat. Robert Geisberger [7] and Diploma Thesis from Dipl.-Inform. Thomas Pajor [18]. The first showed us perfect algorithm for our pupose and the second was the key for understanding the basic principles.

1.1 Work overview

This work is divided into sections in which the problem we wanted to solve will be introduced and guided through possible solution proposal, implementation and results will be presented.

In the section 2 you will find the data structures and the algorithms used in the programme. The next section 3 explains all important concepts, such as public transport network (PTN), journey plan and all computed metrics.

The next part 4 contains detailed informations about the algorithms and their explanation. In the section 5 you will find detail information about implementation and detailed informations about how to use PTNanalys (list of configuration files and input parameters). The most important section is 6 where you can find results of our testing on chosen PTNs and also speed testing of our algorithm.

The section 7 discusses a problem, that showed up during the testing.

In the last section 8 you can find the conclussion and the future plans.
1.2 Motivation

Public transport was chosen because of its importance in today’s life in big cities and agglomerations. The PT network is used every day by about 50 percent of the people living in large cities with more than one million residents. One of the biggest advantages is that public transport is one of the most ecological types of transport.

According to the statements mentioned above, we can easily suggest, that the public transport network (PTN) is really important and its analysis is the first step to improve it and make it more efficient, faster and more ecological. The customers shall then be more satisfied with the PT services.
2 Background

In this section I would like to basically describe the data structures and algorithms used in this work. This section also contains comparison of data structures for representing the PTN as a graph. Some of the algorithms mentioned in this section are explained in more detail in the section 4.

2.1 Data structures

This subsection explains the biggest differences between the two possible interpretations of the PTN as a graph. At the end you can find which model and why has been chosen.

2.1.1 Time-expanded graph

Time-expanded graph is one of the possible representations of the PTN, [20] and [18]. In the time-expanded graph each vertex is a pair (time,stop), which represent arrival or departure from a specific station at a specific time. Edges between vertices are trips in the PTN.

The time-expanded graph is easier to implement but does not allow so accurate planning. For large networks with a huge number of trips the size of the PTN graph dramatically increases.

2.1.2 Time-dependent graph with constant transfer time

Time-dependent graph is another structure that can be used for the PTN representation, [18] and [7].

In the time-dependent graph each edge holds information about departures and arrivals. That means that the traversal function for edge is not constant and equals a sum of waiting time and travel time. The edge can determine its traversal function for every point in time.

The vertices in the time-dependent graph represent stations and each route going through this station. Constant transfer time means, that getting on and off a vehicle costs constant time, and does not vary according to real stop locations inside one station.

2.1.3 Choosing a model

For our purpose we have chosen the time-dependent model. The main reason was its compactness. The time-expanded graph would be very big, it could contain about one million vertices for representing network within one day. Searching in such a large network would be very slow and not efficient.

Based on the initial testing, we have also determined that the time-dependent graph is much faster for searching the shortest path.

For building the time-dependent graph we have used open-source library Planner-data-importer [23]. Detailed information on the time-dependent graph, its vertices and edges are
described in section 3.1.

2.2 Algorithms

2.2.1 Dijkstra’s algorithm

For searching the shortest path, an uninformed algorithm called Dijkstra’s algorithm [5] is used. The algorithm search for the shortest path from one vertex to another. It can be simply modified for searching the shortest path to all other vertices.

Dijkstra’s algorithm chooses in each iteration the vertex with the lowest cost-function, which was not yet visited, until it reaches the goal vertex. There are some speed up techniques that can be used when using Dijkstra’s algorithm for searching shortest PTN path e.g., contraction hierarchies [8] or pattern generating [1]. Let me mention, that more speed up techniques for the time-dependent graphs exist, but they can be used just for road networks, because their graph structure is slightly different from the structure of PTN.

2.2.2 Patterns in the PTN

The Idea of using patterns for the PTN analysis came up from [7] and [1]. Pattern is a sequence of stations that are visited when travelling from station A to station B.

Patterns are described in [7] as the only possible way, how to speed up searches in the time-dependent graph with realistic transfer duration. Patterns are precomputed, for faster querying. This model also uses hub stations for reducing the memory costs.

The algorithm is in [7] described and tested on the rail network, which is much more consistent and regular than the usual PTN.

That affects the number of patterns, that have to be saved for pareto optimal results. In other words, during the analysis more optimal connections can be found for journey from station A to station B.

So the more connections we compute, the higher the probability, that we generate the optimal solution by using patterns and not using the expensive Dijkstra’s algorithm.
2.2 Algorithms

For pair \( (v_{\text{start}}, v_{\text{goal}}) = (A, I) \) we have pattern with sequence \( (A, D, E, F, I) \). Just the edges between these nodes are visited, they are marked orange. The black edges were not visited.

2.2.3 Gift wrapping algorithm

For computing the area metric, I had to compute the convex hull of all given stations. If the stations are imported from DB containing Postgis extension [19], all stations contain projected coordinates to plane. Having projected the coordinates the Gift wrapping [11] algorithm can be used.

Otherwise we have to convert the GPS position of all stops to a coordinations plane, and compute their distance in meters. This projection is not really accurate, but tries to be as accurate as possible. When this was done, I used the Gift wrapping algorithm [11] to compute the area of the hull.

The algorithm’s complexity derives from the number of nodes defining the hull. This number is in the PTN much lower then the number of all stops in the PTN.
3 Problem Specification

This section contains specification of all important concepts used in this paper.

3.1 PTN graph interpretation

The PTN graph is interpreted as a time-dependent graph $\mathcal{D} = (V_D, E_D)$, where $V_D$ is a union of set of station vertices $V_S$ and set of route vertices $V_R$. $E_D$ is a union of sets representing time-dependent edges, transfer edges, walking edges and route edges. Detailed description of each component is described below.

3.1.1 Graph vertices

Station Node - $v_S$  Station node is a parent station for each route node operating on the given stop. The journey starts and ends on the station nodes. In the graph structure, no PTN vehicle arrives to or departs from this type of node.

Route Node - $v_R$  Route node is a virtual station for a single route on a given station. Every route node has a parent station node, to which it is connected using transfer edge. Route nodes operate PTN vehicles.

3.1.2 Graph edges

Transfer Edge - $e_T$  This edge is used just "inside" one station to connect station node with its route nodes. We use a time-dependent graph with constant transfer times so the weight does not vary according to stop.

Walking Edge - $e_W$  Walking edge is an edge between two station nodes, that are connected with walking path, in other words, they can be reached on foot. Maximal length of this edge can be changed by the user.

Route Edge - $e_R$  Route edge is a connection between two route nodes, having two different parent station nodes, operated by PTN vehicle.
3.2 Journey plan

The journey plan is specified as a sequence of stops and connections between the start stop and end stop. The best journey plan minimizes the time required to reach the final stop. Optimal journey plan between stops $v_{\text{start}}$ and $v_{\text{goal}}$ at time $\tau$ is marked as $\text{plan}(v_{\text{start}}@\tau, v_{\text{goal}})$, where $v_{\text{start}}, v_{\text{goal}} \in V_S$. Journey plan is response for each agent’s query from travel demand.

3.3 Analysis mode

In this subsection you can find explanation of all the analysis modes that can be used in the PTNanalys.

3.3.1 Many-to-many real demand

In this mode, user has to add travel demanded in a text file. This demand is in format

```
GTFS_ID_start_station GTFS_ID_final_station seconds_from_day_begining
```

For example journey request between stops with GTFS IDs ”str103” and ”ave28” at 15:32:47 would be

```
str103 ave28 55967
```

This mode is the most precise mode for analysis.
3.4 Static metrics

3.3.2 Many-to-many generated demand

This mode is similar to the one mentioned above, but we do not need to import the travel demand, they are generated automatically from the list of stations. But for some metrics, the results won’t be that accurate. The demand generation was used very often, because we did not get the real travel demand. The generation is further explained in section 6.2.

3.3.3 One-to-others

This mode requires less input information. User can select just the start stop $v_{\text{start}} \in V_S$ and the metrics will be calculated to all other stations $v_{\text{goal}} \in V_S$. When using this mode, you have to choose an exact time, for details see section 5.3.

3.3.4 Static

In this mode, just the static metrics as node degree, harrness and area are computed.

3.4 Static metrics

Static metrics are metrics, that are based on the static properties of the PTN such as location of the stops or the number of lines. They do not vary according to the travel demand. When describing the static metrics $n$ stands for the number of stops. Marking vertices with * means, that instead of the station node can also its route node be used.

3.4.1 Node degree

This metric tells us, how many neighbours each stations has. Each neighbouring station can be assumed just once, so at first we need to remove the multiconnections between stations, which is caused by a fact, that more lines are connecting two stations. We compute the node degree for each vertex $v \in V_S$ as follows

$$D = \sum_{i=1}^{n} j = \begin{cases} j = 1 & \{v*, v_i*, e_r \in E_R\} \\ j = 0 & \text{otherwise} \end{cases}$$

Interpretation of this metric will not be a scalar value. It is a property of each station. The possible interpretations of the metric are the following:

- **N nodes with highest degrees** – Output will be list of $N$ stations with highest node degrees. They can represent the $N$ most important stations in PTN.

- **Nodes with degree larger than $\hat{D}$** – Output will be a list of stations with node degree higher than $D$. They can represent important stations in PTN.
• **Graphical output** - We can display a map, where each station will be a colored dot. The radius of the dot in pixels will be the node degree. So the most important stations would be marked with biggest dots.

### 3.4.2 Harness

Harness is a PTN-specific metric, I explored this metric in [22]. It measures, how many lines are parallel and are connecting the same stops. Harness between vertices $v_1$ and $v_2$ ($v_1, v_2 \in V_S$) is computed as number of edges $e_i \in E_R$ connecting these 2 vertices. So harness for 2 vertices is computed as follows

$$H = \sum_{i=1}^{e} j = \begin{cases} j = 1 & \{(v_1^*, v_2^*) = e_r \in E_R\} \\ j = 0 & \text{otherwise} \end{cases}$$

where $e$ is a number of edges in $E_R$.

Interpretation of this metric will not be a scalar value. It is a property of the stop pairs. The interpretation of this metrics will be based on this fact, the possible interpretations are following:

- **N paths with highest harness** - List of N paths (each path will be a list of station) with highest harness.

- **Paths with harness larger than $\hat{H}$** - List of paths with harness larger than $\hat{H}$.

- **Graphical output** - We can display a map, where each edge $e \in E_R$ will be drawed that thick, how the harness has been computed for a given edge $e_R$.

### 3.4.3 Area covered by public transport

This metric is computed as a convex hull of all stations. For this purpose, the stations are projected to a plane. The metrics will be used as symbol A.

With the area we can compute a lot of interesting metrics, except just computing a area, eg.:

1. Number of stops per square kilometer

$$\rho = \frac{n}{A}, \text{ where } n \text{ is number of stops}$$

2. Month cost for passenger per square kilometer

$$P_A = \frac{P_T}{A}, \text{ where } P_T \text{ is ticket price per month}$$
3.4.4 Length of all lines

We compute the whole lines length by summing distances between stops, which are connected with at least one route.

The result will be a scalar value computed as follows.

\[ l = \sum |v_i, v_j| : \{ \forall v_i, v_j \in V_R \mid v_i, v_j \in E_R \land i \neq j \} \]

3.5 Dynamic metrics

The dynamic metrics are based on the travel demand, they vary as the agents vary their journey requests. They can also depend on the number of agents.

3.5.1 Duration of journey

Calculation of the shortest time needed to get from start station \( v_{\text{start}} \in V_S \) to final stop \( v_{\text{goal}} \in V_S \). Then duration to take this journey is marked as \( t(v_{\text{start}}, v_{\text{goal}}) \)

Duration as metric can be represented in many different ways. Let me divide them into two groups, by the "analysis mode". First I will show up some metric for analysis modes "many-to-many".

- **Mean shortest journey travel time** – average travel time – calculated as

  \[ T = \frac{\sum t(v_{\text{start}}, v_{\text{goal}})}{n} \]

  For the mode "One-to-other" the metric average travel time can be used too, but we can interpret the metrics also as follows:

  - **Stations coverage within time horizon** \( \hat{T} \) – given some time horizon \( \hat{T} \), this metric tells us, what’s the percentage of stations that can be reached within \( \hat{T} \). Calculated as

    \[ S_{\text{cov}} = \sum_{i=1}^{n} \frac{j}{n} = \begin{cases} j = 1 & t(v_{\text{start}}, v_{\text{goal}}) \leq \hat{T} \\ j = 0 & \text{otherwise} \end{cases} \]

    While developing our tool, we realised, that we need to distinguish two journey times. These times are described below.

- **Netto time** Netto time is time from entering the first PT vehicle to reaching the goal station. If no PT vehicles are entered during journey (goal is reached on foot), then netto time equals walking time.

- **Time from demand** is measured from the time, on which the trip was demanded. It is every time bigger than the netto time.
3.5.2 Number of interchanges

The shortest path will be simply calculated from a start station $v_{start} \in V_S$ to a final stop $v_{goal} \in V_S$ and the algorithm computes, how many lines have been used. Then number of lines we take on the journey is marked as $ch(v_{start}, v_{goal})$.

This metric will be interpreted for both analysis modes as follows. We simply calculate the average number of interchanges. It can be calculated as

$$\hat{ch} = \frac{1}{n} \sum_{i=1}^{n} ch(v_{start}, v_{goal}),$$

where $n$ is number of planned journeys. For ”one-to-others” mode all pairs have the same $v_{start} \in V_S$ start stop, and the final stops $v_{goal} \in V_S$ are all other stops.

3.5.3 Betweenness centrality

The Betweenness centrality metric is similar to Node degree, but for a computation we need to run some queries. In this case, if we provide large enough amount of random queries, the betweenness centrality will be computed with a good accuracy.

This metric is a property of each vertex $v \in V_S$ and can show us important vertices. It is a percentage value, how many shortest paths were planned through vertex $v$.

Given a set of query pairs, for each vertex $v$ we can compute betweenness centrality as follows.

$$B = \sum_{i=1}^{n} \frac{j}{n} \begin{cases} j = 1 & v \in plan(S_i, G_i) \\ j = 0 & \text{otherwise} \end{cases}$$

where $n$ is a number of input pairs.

I explored the betweenness centrality in [22].

We can represent this value for each vertex similar to Node degree as showed in section 3.4.1. All three possibilities can be used for this metric too.

3.5.4 Trip workload

For some useful output of this metric we need more precise data. The best way is to use real demand data.

We evaluate our query pairs and then check, how many agents used the given trips.

Trip workload for each trip trip, when given set of the query pairs, can be represented as

$$T_w = \sum_{i=1}^{n} j = \begin{cases} j = 1 & trip \in plan(S_i, G_i) \\ j = 0 & \text{else} \end{cases}$$

where $n$ is a number of input pairs.

This metric tells us, how many people will be aboard given bus, tram or subway. Comparing with capacities of given vehicles, we can improve availability of the vehicles.
4 Solution approach

In this section, I would like to describe the key ideas of our algorithm, how we compute the optimal journey plans and how it speeds up our analysis. The ideas were developed based on the knowledge of the Dijkstra’s algorithm, on the behavior of quite large PTN in Prague and on the ideas of Patterns coming from [7] and [1].

First subsection explains, how can we merge some journey requests without affecting the speed and the accuracy. Second part shows changes we made to Dijkstra’s algorithm and the last section explains how patterns work.

4.1 Merging queries

We stated earlier, that we search journey for every single agent separately. According to the logic of the PTN this is not necessary. We can merge queries from one station at the same time. More generally, we can merge trips that starts at one station in time interval between two arrivals of the PT vehicles.

Because of using walking edges between station nodes, we have to be careful when merging trips.

For example lets assume, that we have stations A...F and journey requests $A@8^03 \rightarrow E$, $A@8^05 \rightarrow D$. When the departures from station A are at time $8^01$ and $8^10$, we can merge these two queries without affecting the accuracy.

When the station node has an outcoming walking edge, we also have to check the station that can be reached on foot. We have to compute the time, when the nearby station is reached, and find the nearest departure from this point in time. Having all the first departures from all the station nodes, which can be reached on foot, we can easily determine the earliest departure $d_{r_1}$. All journey requests starting on the start stop until the first departure $d_{r_1}$ can be merged into one Dijkstra’s search.
Figure 3: Merging queries for stops connected by walking edge

In the figure you can see part of time dependent graph. The transfer edges take 1 minute to pass, the walking edge 3 minute to pass and on the route edges, there is a sign of first departure.

Without affecting accuracy, we can merge journey requests from stop A until 8:07. Every agent, who is present at stop A until 8:07 can reach the nearest departure, which is the departure of route 103 from stop C at 8:11.

This merging is most powerful when using analysis mode one-to-others, all the queries are merged into one Dijkstra’s search. If we are using one of the two others many-to-many modes, the merging is also helpful and can save us some Dijkstra’s searches. The number of the merged trips depends on the number of agents in the network. The higher the number of agent is, the higher the probability is, that two agents will start their journey from one initial stop in the same departure interval.

4.2 Extended Dijkstra’s algorithm

Implementation of the Dijkstra’s algorithm is based on standard approach, but is slightly enhanced for reducing the number of runs to speed up our analysis. Instead of using a single start node and a single goal node we use more parameters. Our algorithm uses single start node $v_{\text{start}} \in V_S$ and handles more goal nodes $v_{\text{goal}_1}, \ldots, v_{\text{goal}_n} \in V_S$ and also can handle so-called desired nodes $v_{\text{des}_1}, \ldots, v_{\text{des}_n} \in V_S$.

When generating, or importing, depends on analysis mode, we generate map of desired nodes, where we store, for each start station node $v_{\text{start}}$ list of desired nodes. These are not the goal nodes for current instance of the Dijkstra’s search, but they will be goal nodes in of the next Dijkstra’s searches. This allows us to precompute patterns for all nodes, that will be visited during analysis from $v_{\text{start}}$. Our Dijkstra’s algorithm is single-criterial and optimizes
time from the agent’s journey request, this time measurement is described in section 3.5.1.

4.3 Patterns

Using patterns when searching the shortest path in the PTN is the key to perform the PTN analysis. Pattern is a sequence of stops which are visited when using the PTN to get from station A to station B. The idea is quite simple, there are not much sequences of stops, that lead to optimal solution for each pair $v_{start}, v_{goal} \in V_S$. It is much more efficient to evaluate pattern for initial time $\tau$ and compute the $\tau_{goal}$ just from the sequence of stops, that are visited, then running the Dijkstra’s algorithm.

4.3.1 Pattern factory and Pattern boxes

Every combination $v_{start}, v_{goal} \in V_S$ that exists in travel demand has its own Pattern box, this box can provide us the best possible pattern, or the sorted list of all patterns. Pattern boxes are maintained using the Pattern factory. Data structure of Pattern factory is built on HashMap and its default capacity is set up to 10000, that means that for the PTN that does not contain more than approximately 8000 stations, is the time to get the desired Pattern box $O(1)$.

Pattern boxes have initial capacity, that set up the number of pattern, that have to be generated before the patterns can be used. This is the one of the most important parameters for the analysis, see section 5.4.3 how to set-up and section 6.4.1 how this parameter affects speed.

4.3.2 Generating patterns

The pattern is generated while running the Dijkstra’s algorithm. In our programme, we store a list of goal nodes for each station node, that will be searched. Once the Dijkstra’s is runned with one start node, patterns to all corresponding stations are generated.

The patterns are also generated if the pattern box contains more patterns than is its initial capacity. This can happen, because when we are running the Dijkstra’s from start stop $v_{start} \in V_S$, we generate patterns to all desired nodes $v_{des_1}, \ldots, v_{des_n} \in V_S$, no matter how many patterns does the pattern box $v_{start}, v_{des_i}$ contain.

That happens just as a side product, the station node can be added to the desired nodes and never to the goal nodes. That leads us to slightly bigger memory consumption but more accurate results.

4.3.3 Using patterns

At the beggining of the analysis, patterns will not be used, they have to be generated until the pattern box for requested pair $v_{start}, v_{goal}$ is full.
Given pattern, I can easily iterate over all edges, which can be identified with its start node and its end node, which are stored as a sequence in the pattern.

The pattern will then be evaluated for current time. If the netto time, see section 3.5.1, of the shortest generated path is not bigger than the maximal time, which depends on the pattern netto time, the solution can be used.

The maximum time for allowing the generated shortest path is computed as follows:

\[
\text{Maximal allowed time} = \text{PERCENTAGE\_GROW\_AVAILABLE} \times \text{pattern neto time}
\]

where \text{PERCENTAGE\_GROW\_AVAILABLE} is a parameter that can be set up, see section 5.4.3.

When using the patterns, we also check the maximal waiting time. The waiting time is computed as follows:

\[
\text{Waiting time} = \text{Time from demand} - \text{Netto time}
\]

If the computed waiting time is bigger than \text{MAX\_WAITING\_TIME\_PARAMETER}, which can be setup in configuration files, the shortest path is not allowed and the journey request is computed using the Dijkstra’s algorithm.

![Figure 4: Evaluated pattern](image)

For pair \((v_{\text{start}}, v_{\text{goal}}) = (A, I)\) we have pattern with sequence \((A, D, E, F, I)\) and netto time = 14 min. Evaluated pattern is displayed with orange. Red number nearby node A means waiting time in minutes. Orange numbers stand for evaluated traversal function of each time-dependent edge in minutes.

For standart parameters \text{PERCENTAGE\_GROW\_AVAILABLE} = 1.5 and \text{MAX\_WAITING\_TIME\_PARAMETER} = 3600 the shortest path generated from this pattern would be allowed.

Pattern box often stores diferent patterns, then every pattern is evaluated and the best one is used.
5 Implementation

This section contains informations about the development of the programme and settings for computing the analysis. The first subsection contains information about the parts of the programme. The next subsection shows the software and the data used for development. In next two subsections you will find information how to set up the PTNanalys for your analysis. Also examples can be found here. The last section is about the architecture of the programme and its possible improvements and changes to meet the multithreading ability.

5.1 Overall summary

In this section I would like to point out the main parts of the PTNanalys. The source code can be found on bitbucket\(^1\).

**Time-depdent graph contruction** The time-dependent graph is built from the database data. The data in the database have to be stored in GTFS format \([9]\). For the database, it is highly recommended, to support the Postgis extension \([19]\). The graph is built using the opensource programme planner-data-importer \([23]\). The graph contains different verticies and edges, that are described in 3.1.

**Static metrics computations** When the PTN graph is built, we compute all the static metrics - Node degrees, Harness, Area and Length of Lines. Detailed description can be found in section 3.4.

**Getting travel demand** Now we load or generate the travel demand. This depends on the analysis mode. See section 3.3. If the analysis runs in static mode, this step will not be included.

**Computing optimal journey planns** After loading the travel demand, we can run computations for the optimal journey plans. How the computation is done can be seen in section 4. Dynamic metrics from each journey plan are computed on the run. If the analysis runs in static mode, this step will not be included.

**Visualisation** When all the metrics are computed, we can visualise them. The data are visualised using the geoVisio library \([3]\) and the OpenGeo Suite \([16]\).

5.2 Used open-source

We used some open-source programmes in PTNanalys:

- **planner-data-importer** \([23]\) for constructing the time-dependent graph.

\(^1\)https://bitbucket.org/jnykl/ptnanalys
• libOSM [12] for working with the GPS points.
• TimeTable Data Convertor [15] for importing the data into DB. This software is not part of the programme, it was used separately.

The programmes where imported as dependencies. PTNanalys was developed using Maven [13].

5.2.1 Data used for development

For the implementation, we used a very simple model of Prague PTN, which was created by myself. It consists of 3 underground lines and 3 tram lines. We used such a small PTN because we can imagine how the PTN works and better debug the algorithms. Unfortunately, the real raw data for Prague are not openly available.

![Visualisation of synthetic Prague PTN](image)

Figure 5: Visualisation of synthetic Prague PTN

5.3 Input parameters

When running the analysis, some input parameters are needed.

1. **Analysis mode** In table 1 you can see, which number represents which analysis mode. Detailly description of analysis modes can be found in section 3.3
2. **Date for analysis** The date format shall be YYYY-MM-DD. If the One-to-Others mode is chosen, the format has to be YYYY-MM-DD_HH:MM:SS.

3. **Additional parameter** This additional parameter depends on the analysis mode that has been chosen as the first input parameter.

   It is the path to the file with demand in mode "Many-to-many real demand", or it is the number of generated journey requests in "Many-to-many generated demand" mode.

   For the mode "One-to-others" this parameter holds information what is the start stop, respective it’s GTFS ID. The value "random" can be used for random selection of start stop.

   In the "Static" mode, this field is empty.

The input parameters for the analysis mode One-to-others, on the 15th May 2013 at 14:32:10 with initial stop ID "str123" would be

2 2013-05-15_14:32:10 str123

### 5.4 Configuration files

Our tool uses 4 configuration files which are maintained using Java Properties class. The config files are plain text files in format

```
key=value
```

The lines beginning with `#` are comments. The files has to be saved in directory `config`.

Examples of configuration files can be found in appendix in B.

#### 5.4.1 PTN configuration

Name of this configuration file is `ptn.config`. This file configures the PTN properties. If you are analysing one PTN, you will not need to change this file.

- **WALKING_EDGE_LENGTH** - maximal length of walking edges that will be added to the time-dependent graph. The value is given in meters.
5 IMPLEMENTATION

- **EPSG** - spatial reference for projecting GPS coordinates [6]. Has to be chosen according to the location of the analyzed network. Make sure that the output metric of the projected coordinates are meters. It is the most common case, but sometimes US survey feets are used for EPSG projection.

- **MAX_ORIGIN_DESTINATION_DISTANCE_IN_KM** - maximal distance between start stop and end stop of journey request, which can be generated while generating travel demand.

- **TIMEZONE** - setting up the time-zone is really important for the right assignment of the journey requests to the time table. The time zone is inserted as ID, e.g. Europe/Paris. For list of IDs follow [21]

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Possible values</th>
<th>Recommended value</th>
</tr>
</thead>
<tbody>
<tr>
<td>WALKING_EDGE_LENGTH</td>
<td>Double values larger than 0.0</td>
<td>500</td>
</tr>
<tr>
<td>EPSG</td>
<td>Selected from list, see [6]</td>
<td>2065</td>
</tr>
<tr>
<td>MAX_ORIGIN_DESTINATION_DISTANCE_IN_KM</td>
<td>Double values larger than 0.0</td>
<td>20</td>
</tr>
<tr>
<td>TIMEZONE</td>
<td>Selected from list of time zone IDs [21]</td>
<td>Europe/Prague</td>
</tr>
</tbody>
</table>

Table 2: PTN configuration

The recommended values written in *italics* are specified for the PTN in Prague. You have to change them according to your PTN.

5.4.2 GTFS DB configuration

This configuration file’s name is gtfsDB.config. This file stores information for connecting to database, where are the time-table data stored.

- **DB_TYPE** - type of database, where are the data stored. Just "hsqldb" and "postgresql" is supported.

- **DB_LINK** - connection to database, has to contain URL, port and database name.

- **DB_USER** - user name for accessing the database.

- **DB_PASS** - user password for entering the database.
5.4 Configuration files

5.4.3 Analysis configuration

This configuration file maintains all the settings for analysis, changing this properties, you may affect speed and accuracy of your analysis. The name of the file has to be analysis.config.

- **MAX_NUM_OF_PATTERNS** - this parameter is key parameter for analysis speed and accuracy. It’s behavior and impact is further explained in 4.3.

- **PERCENTAGE_GROW_AVAILABLE** - this parameter affects using patterns, see section 4.3.3 for details.

- **MAX_WAITING_TIME_PARAMETER** - this parameter also changes patterns behavior and is described in 4.3.3. Positive number state for seconds. -1 means that maximum waiting time is equal netto time for given pattern.

- **CONVERT_NEGATIVE_EDGE** - if your GTFS data contains mistakes, edges with negative costs can be imported. With this parameter you can change whether you want them to be imported with cost function $f'_c = |f_c|$ or not.

- **VISUALISE** - changes whether the programme generates graphical output or not.

- **HOW_OFTEN_PROGRESS_OUTPUT** - while running analysis, some info is printed to console. In this parameter you can set up how often the progress is printed.

- **HOW_OFTEN_GRAPHICAL_LAYER** - if the analysis runs in of the Many-to-Many modes this parameter affects after how many seconds, the visualisation of trips workload is generated.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Possible values</th>
<th>Recommended value</th>
</tr>
</thead>
<tbody>
<tr>
<td>MAX_NUM_OF_PATTERNS</td>
<td>Integer values larger than 1</td>
<td>3</td>
</tr>
<tr>
<td>PERCENTAGE_GROW_AVAILABLE</td>
<td>Double values larger than 1.0</td>
<td>1.5</td>
</tr>
<tr>
<td>MAX_WAITING_TIME_PARAMETER</td>
<td>-1 or Integer value larger than 0</td>
<td>3600</td>
</tr>
<tr>
<td>CONVERT_NEGATIVE_EDGE</td>
<td>true or false</td>
<td>false</td>
</tr>
<tr>
<td>VISUALISE</td>
<td>true or false</td>
<td>true</td>
</tr>
<tr>
<td>HOW_OFTEN_PROGRESS_OUTPUT</td>
<td>Integer value between 1 and the number of agents</td>
<td>number of agents / 100</td>
</tr>
<tr>
<td>HOW_OFTEN_GRAPHICAL_LAYER</td>
<td>Integer value between 1 and 86400. This parameter has to divide 86400</td>
<td>7200</td>
</tr>
</tbody>
</table>

Table 3: Analysis configuration
5.4.4 GEO server configuration

For visualising data, you will need server running OpenGeo Suite [16]. In this configuration file you set up access to this server and database, which will contain your graphical data. The name of the file has to be geoVisio.config.

- **DB_SERVER_HOST** - URL address of database machine.
- **DB_SERVER_PORT** - database port.
- **DB_USER** - user name for database.
- **DB_PASS** - user password for database.
- **DB_NAME** - name of the database for the graphic data.
- **GEOSERVER_URL** - URL address of the GEO server. Use localhost, when running GEO server on your computer.
- **GEOSERVER_USER** - user name for GEO server.
- **GEOSERVER_PASS** - user password for GEO server.
- **WORKSPACE_NAME** - prefix for all visualisation layers.

5.5 Multithreading

In the current version of the PTN analysis multiple-core processors are not exploited. That means that the analysis runs just on one core of a computer. In this subsection we will compare the current architecture with the designed multithreading architecture.

5.5.1 Current architecture

Main classes in current versions are PTN.java, which maintains the graph structures of PTN, the PatternFactory.java which is a singleton class that saves and restores all patterns and the class Dijkstras.java which is instantiated for PTN and runs journey plans searches.

Furthermore there is the TravelDemandFactory.java class, which is a singleton class and loads or generates travel demand. It can also return the desired nodes for a vstart, for explanation see section 4.2. The last important class is the Analysis.java which computes all desired static and dynamic metrics.

The travel demand is sorted by initial stop and time and is evaluated consecutively. The current architecture is ready to be rebuilt into the multithreading architecture.
5.5 Multithreading

The programme runs from left to right. Except of the first block, which represents the database, the rectangles represent singleton classes. Circles represent standard classes that are instantiated once. Diamonds stand for objects created multiple times.

5.5.2 Multithreading architecture

The multithreading architecture will not differ much. The travel demand will be divided into blocks by the start stop $v_{start_i}$. Each block will have its own Dijkstra’s algorithm instance and Pattern Factory.

With this improvement, the blocks can be evaluated in parallel on more processor cores.

The multiple pattern factories can help us avoid locking, which would lower the perfor-
mance.

5.5.3 Expected speed up

As you can see in the upper schemas, the most time expensive part of algorithm will run in parallel. In the figure 5.5.3 can you see the expected results, when running PTNanalys on multiple cores.

![Figure 8: Expected speed up for multithreading architecture](image)

The analysis time for one core has been counted as 100 s for graph construction, 10000 s for querying and 200 s for graphical output. Just the querying section will be faster and because of that, the improvement is not linear.
6 Evaluation

In this section I will describe testing and evaluating of the programme on different scenarios. Data used for testing and evaluating are described in first two subsections. Further you will find information about the settings and the parameters, which were used for testing. Next subsection shows the results of testing the speed of the journey planner we used. The fifth subsection contains the results of the analysis, comparison and computed data. Graphical output of the programme can be found in next subsection. In the final subsection you can find the explanations for metrics on real data, and examples, how this data can be used by the PTN supervisor.

When evaluating the results, we used the resources of MetaCentrum [14]. It brought us the speed improvements and better comparability.

6.1 PTN Timetable data

PTNanalys uses timetable data stored in GTFS. The data has to be stored in the database, which is accessed by settings in 5.4.2. For importing GTFS files into the database the TimeTableDataConvertor [15] can be used.

The quality of this data highly affects the analysis results and the speed of the analysis. In optimal cases, the PTNanalys would be used by PTN supervisors, as the preprocessing step for PTN optimization. Those people would have the perfect PTN timetable data, and the analysis would be much more accurate.

For development and our initial testing we used different free GTFS feeds that can be found on [10] or [4]. Among these data can also be found perfectly structured networks, on which the analysis gives interesting and useful results. We have chosen to analyse real PTN in Budapest, Quebec and Rennes.

<table>
<thead>
<tr>
<th>City</th>
<th># stops</th>
<th># vertices</th>
<th># edges</th>
</tr>
</thead>
<tbody>
<tr>
<td>Budapest</td>
<td>5288</td>
<td>18257</td>
<td>108707</td>
</tr>
<tr>
<td>Rennes</td>
<td>1408</td>
<td>4642</td>
<td>21339</td>
</tr>
<tr>
<td>Quebec</td>
<td>4569</td>
<td>15738</td>
<td>113162</td>
</tr>
<tr>
<td>Prague</td>
<td>149</td>
<td>320</td>
<td>846</td>
</tr>
</tbody>
</table>

Table 4: Analysis configuration

6.2 Travel Demand data

Next to the timetable data, also the travel demand data are important for analysis accuracy. If the data for the real travel demand does not exist, our synthetic model for generating travel demand can be used.
The demand is distributed using two histograms. We care of the distribution of journey request in time (the traffic at night is much less frequent than by day), as well as we care of the distribution in distance.

This distribution avoids high penetration of long journeys, which is not common in real PTN travel demand. They are more often used for short or middle-range journeys. The distance histogram is divided in parts by 10% steps from the highest possible origin-destination distance. This parameter can be setup in 5.4.1.

![Figure 9: Division of generated demand in time](image)

![Figure 10: Division of generated demand by distance between $v_{start}$ and $v_{goal}$](image)
6.3 Used configuration

If not stated otherwise, following configuration has been used during testing:

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Used value</th>
</tr>
</thead>
<tbody>
<tr>
<td>WALKING_EDGE_LENGTH</td>
<td>500</td>
</tr>
<tr>
<td>MAX_ORIGIN_DESTINATION_DISTANCE_IN_KM</td>
<td>20</td>
</tr>
</tbody>
</table>

Table 5: PTN configuration

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Used value</th>
</tr>
</thead>
<tbody>
<tr>
<td>MAX_NUM_OF_PATTERNS</td>
<td>3</td>
</tr>
<tr>
<td>PERCENTAGE_GROW_AVAILABLE</td>
<td>1.5</td>
</tr>
<tr>
<td>MAX_WAITING_TIME_PARAMETER</td>
<td>3600</td>
</tr>
<tr>
<td>CONVERT_NEGATIVE_EDGE</td>
<td>false</td>
</tr>
</tbody>
</table>

Table 6: Analysis configuration

Please note, that just the analysis affecting parameters are mentioned.

6.4 Journey planner speed

As a part of the results, we tested the runtime of the journey planner we developed. The results are normalised, for better graph interpretation. The absolute numbers for 100% are in tables 6.4.1 and 6.4.2. The speed was tested on computer with two processors 4-core Intel Xeon X5570 2.93GHz and 24 GB RAM. Just one core, and 20 GB RAM was used.

6.4.1 Speed according to patterns

The patterns are really important, and changing the parameter MAX_NUM_OF_PATTERNS can highly affect the analysis. Bellow you can see the results of testing.
Figure 11: Average time from demand according to number of patterns

Figure 12: Time per query according to number of patterns

In these two figures, you can see how the number of patterns can affect the analysis speed and its accuracy. The larger the number of pattern is, the better the analysis results are. Because we optimize according to time from demand, we can say, the lower values are better, closer to optimum.

One the other hand, we can see, the larger the number of patterns is, the slower the analysis is. But we have to point out, that the analysis is fastest with number of patterns
6.4 Journey planner speed

around 3. The reason for this behavior is following. If the number of patterns equals 1, very often can happen, that the pattern can not be used, and the dijkstra’s algorithm has to be runned anyway. That is slowing down our analysis. For best results we have to maximize following ratio r:

\[ r = \frac{\text{used patterns}}{\text{patterns could not be used again}} \]

Both these number are printed to console, when running analysis.

The running time highly dependes on PTN sizes. They are shown in table 6.1.

<table>
<thead>
<tr>
<th>City</th>
<th>Time per query [ms] for 100%</th>
<th>Average time from demand [s] for 100%</th>
</tr>
</thead>
<tbody>
<tr>
<td>Budapest</td>
<td>21.02</td>
<td>5153</td>
</tr>
<tr>
<td>Rennes</td>
<td>5.59</td>
<td>9856</td>
</tr>
<tr>
<td>Quebec</td>
<td>28.66</td>
<td>7977</td>
</tr>
<tr>
<td>Prague</td>
<td>0.04</td>
<td>8749</td>
</tr>
</tbody>
</table>

Table 7: Absolute values according to 100%

6.4.2 Speed according to number of agents

In this part will be shown interesting results, how the number of agents can affect the speed of the analysis.

![Bar chart showing percentage change for different cities](image)

Figure 13: Time per query according to number of agents

We can see that the higher the number of agent is, the lower the time per query is. The reason for this are patterns, once they are generated, they can be just reused. The query
time also highly depends on PTN size and its structure.

<table>
<thead>
<tr>
<th>City</th>
<th>Number of agents</th>
<th>20000</th>
<th>100000</th>
<th>1000000</th>
</tr>
</thead>
<tbody>
<tr>
<td>Budapest</td>
<td></td>
<td>35.95</td>
<td>17.3</td>
<td>11.17</td>
</tr>
<tr>
<td>Rennes</td>
<td></td>
<td>4.9</td>
<td>3.34</td>
<td>2.44</td>
</tr>
<tr>
<td>Quebec</td>
<td></td>
<td>33.85</td>
<td>25.41</td>
<td>19.429</td>
</tr>
<tr>
<td>Prague</td>
<td></td>
<td>0.1</td>
<td>0.04</td>
<td>0.027</td>
</tr>
</tbody>
</table>

Table 8: Absolute values in [ms] according to 100%

6.5 Analysis Results

In this section you can find the results of the analysis on our three cities, Budapest, Rennes and Quebec. Furthermore you will see the comparison with the syntetic Prague data. Also the analysis mode "One-to-others" has been tested.

6.5.1 Many-to-many mode

The Many-to-many mode is intended to simulate real travel demand within one working day. The number of agents is corresponds to number of citizens in current city [2].

<table>
<thead>
<tr>
<th>metric \ city</th>
<th>Budapest</th>
<th>Rennes</th>
<th>Quebec</th>
<th>Prague</th>
</tr>
</thead>
<tbody>
<tr>
<td># stops</td>
<td>5288</td>
<td>1408</td>
<td>4569</td>
<td>149</td>
</tr>
<tr>
<td>Area [km²]</td>
<td>1205.36</td>
<td>696.59</td>
<td>514.03</td>
<td>190.26</td>
</tr>
<tr>
<td>l [km]</td>
<td>6550.75</td>
<td>3082.52</td>
<td>5036.9</td>
<td>312.66</td>
</tr>
<tr>
<td>demand T</td>
<td>84’ 14.83”</td>
<td>167’ 29.29”</td>
<td>130’ 50.01”</td>
<td>143’ 33.33”</td>
</tr>
<tr>
<td>netto T</td>
<td>58’ 11.86”</td>
<td>98’ 11.36”</td>
<td>75’ 20.06”</td>
<td>36’ 16.36”</td>
</tr>
<tr>
<td>ch</td>
<td>2.93</td>
<td>2.80</td>
<td>2.61</td>
<td>1.21</td>
</tr>
<tr>
<td>정도 ≥ 6</td>
<td>38</td>
<td>19</td>
<td>26</td>
<td>0</td>
</tr>
<tr>
<td>정도 ≥ 6</td>
<td>135</td>
<td>0</td>
<td>76</td>
<td>0</td>
</tr>
<tr>
<td>정도 ≥ 7%</td>
<td>4.38</td>
<td>2.02</td>
<td>8.88</td>
<td>0.783</td>
</tr>
</tbody>
</table>

Table 9: Metrics measured on Budapest, Rennes, Quebec and Prague

The best results has the PTN in Budapest. I would point out the lowest average netto time and number of nodes with node degree higher than 6. That gives us a lot of ideal start stops four our journeys.
6.5.2 One-to-others mode

In this section we will present a comparison of 3 stops in Budapest using One-to-others mode. The static metrics do not vary from the results in the Many-to-many mode, see 6.5.1.

<table>
<thead>
<tr>
<th>metric \ stationID</th>
<th>F03674</th>
<th>F02476</th>
<th>F01494</th>
</tr>
</thead>
<tbody>
<tr>
<td>demand $T$</td>
<td>68’ 7.62”</td>
<td>47’ 7.26”</td>
<td>47’ 11.74”</td>
</tr>
<tr>
<td>netto $T$</td>
<td>65’ 54.94”</td>
<td>43’ 22.00”</td>
<td>42’ 59.94”</td>
</tr>
<tr>
<td>$\hat{c}h$</td>
<td>3.53</td>
<td>2.08</td>
<td>2.13</td>
</tr>
<tr>
<td>$\hat{T} \leq 1h$</td>
<td>49.52%</td>
<td>89.40%</td>
<td>89.58%</td>
</tr>
</tbody>
</table>

Table 10: One-to-others specific metrics measured on 3 stops in Budapest

The station F01494 is clearly the best one.

6.6 Graphical Output

The graphical output, with all its advantages, can be tried on http://mf.felk.cvut.cz:8080/geovisioweb/. Below you can see screenshots from mentioned server with description. The map layer is present just in the first picture, because the results of the dynamic analysis are better displayed without it.

Figure 14: Graphical results of static analysis of the PTN in Budapest
The width of the edges depends on harness. The size of the dots marks node degree. The convex hull is displayed yellow.
This mode displays results for time interval, that can be set up in 5.4.3. The width of the edges depends on average number of passenger. Furthermore the color depend on it too. If the edge is red, the average trip workload in current interval exceeds the capacity of the PT vehicle. If the edge is orange, the maximal trip workload exceeds this capacity. The grayscale colors stands for trips, their workload is ok. The black edge stands for underground lines, the dark grey for tram lines and the light gray stands for the rest of PT vehicles.

The size of the dots logarithmically depends on betweenness centrality of given stop.
6.6 Graphical Output

Figure 17: Graphical results of mode One-to-Others in Budapest with start stop F02476
The start stop is marked with yellow dot. The other stops are marked with grayscale colors.
Their color gets darker as they are reached later. The red dots were not reached.

In the interactive map you can view detailed information about each object if you click
on it. The example is shown below:

![Feature Info]

<table>
<thead>
<tr>
<th>Name</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>id</td>
<td>596</td>
</tr>
<tr>
<td>gfs_bus_id</td>
<td>0290</td>
</tr>
<tr>
<td>from_station</td>
<td>Nógrádvid és</td>
</tr>
<tr>
<td>to_station</td>
<td>Csilber utca</td>
</tr>
<tr>
<td>length</td>
<td>1214</td>
</tr>
<tr>
<td>capacity</td>
<td>100</td>
</tr>
<tr>
<td>average_number_of_pass</td>
<td>99.333333333333333333</td>
</tr>
<tr>
<td>overloaded_trips</td>
<td>2013-02-20, 12:10:00: 140; 2013-02-20, 12:55:00: 134; 2013-02-20, 13:09:00: 142;</td>
</tr>
</tbody>
</table>

Figure 18: The infobox for edge in dynamic mode

Screenshots from the other cities can be found in the appendix C
6.7 Metrics real world interpretation

Among the metrics we can found standart metrics such as average travel time, or average number of interchanges. These metrics do not need any further explanation.

The more interesting metrics are harness, betweenness centrality and trip workload.

Harness can be really important for the PTN supervisor. The edges with higher harness are under heavy load. The roadway on these edges can be affected negatively. We also should be more aware of some driving accidents on these edges. If this edge is blocked, huge number of trips on different routes would be delayed. On these edges it is ideal to have an separate lane for PT vehicles.

Betweenness centrality shows us, how many passengers travel each day through the stop. These stops are optimal for fare inspection. With focusing on stop with higher betweenness centrality, the PTN provider can increase the number of checked passenger and it follows higher income for tickets.

With the trip workload metrics the PTN supervisor can change the PTN vehicles to better cover the travel demand. Some trips could also be operated more often, which may be easier to process then switching a PT vehicle.
7 Discussion

Dijkstra’s algorithms improvements

The Dijkstra’s algorithm is fast, but in the way it is used now, the behaviour of the planner with walking edges is not stable and sometimes the results is not the optimal one.

The percentage effect is really low, and does not change the analysis results. The optimal solution is not found in less than 0.1% cases.

The main problem is, that the walking edges from a stop can sometimes be used and sometimes not.

The problem occurs, when station is first reached on foot, but can also be reached few minutes later on a PTN vehicle. In the first case, the walking edges can not be used, and the agent has to wait for next PTN vehicle.

The biggest problem is, if the connection to the nearby stop from the current stop is just on foot (no PTN vehicle connects these two stations). The stop then can not be reached in the first case, but in the second case, it would be reached without problems.

This behavior is based on Dijkstra’s assumption, that once the stop is reached, it can not be reached with better time. Which is truth, but the ability of walking can change, and it can cause the optimal solution not to be found.

Figure 19: Dijkstra’s behavior in time-dependent graph with walking edges

Solid lines stands for PT vehicle. Dashed lines are walking paths. The orange colored paths were chosen.

In the subfigure a) you can see current Dijkstra’s behavior. The stop B is reached at $8^{07}$, which is earlier than in case b), but the stop C is reached at $8^{11}$ which is later than in case b). The walking path between stops B and C can not be used in case a), because walking path between stops A and B was used.

That means, that the Dijkstra’s algorithm has not found optimal journey plan between stops A and C.

In the future we would like to implement an improvement, that will divide the network into layers, which will handle the changing walking edges well.
8 Conclusion

Our goal to develop a tool for analysing PTN was successfully fulfilled. Current version of our tool is able to analyse PTN for 8 metrics, they are Node degree, Harness, Length of lines, Area, Duration of Journey, Number of interchanges, Betweenness centrality and Trips workload with a travel demand from more than million agents within just few hours.

The analysis can give us perfect accurate results, when the GTFS data are without mistakes. This is one of the most important things and for us, during testing it was quite hard to find good data. For real use, we can assume, that someone who wants to get real results will have good data. It should be mostly a PTN supervisor who needs to analyse a current network, or for example needs a start point for PTN optimization.

We have tried to maximize the amount of information available via graphical output, so there is often no need to check the text output. The PTN supervisor can use comfortable view of the network in map and will not miss any important information.

Furthermore the developed PTN journey planner is very powerful. Its speed has reached 10 ms per query and it is more than 10 times faster compared to standard Dijkstra’s algorithm used at the beginning of this project. With some tiny modifications, it can be divided into outstanding planner and can be used in other pieces of software for simulation or route planning.

While working on this project we have come up with many ideas how to improve it and make it better. Below, you can find the most important features, that will be improved, or added in the feature.

8.1 Future work

The workplan shows the 2 most important changes, that will be implemented in nearest feature. In the last subsection you can find futures that would be nice to have in coming years. The changes mostly affect the speed of the journey planning, so the analysis will be much faster.

8.1.1 Faster pattern generation

In current version of PTN analyser the patterns are generated just in forward direction, that means from initial stop, to all possible stops. Since we use Dijkstra’s algorithm we can also generate and save patterns in backwards direction, from stops that where visited during finding optimal journey to the final stop. This can be done, because the Dijkstra’s algorithm guarantees optimal solution from each visited stop.
As you can see this improvement would speed up patterns generation. In case a) just 3 patterns were generated from one optimal journey plan, but in case b) 5 patterns were generated.

The time expensive Dijkstra’s algorithm could be run more times.

8.1.2 Multithreading

Multithreading has already been mentioned and explained in section 5.5. This feature is not an algorithmic problem, but will bring us much better journey planning time. The speed up should be almost linear based on number of cores available for the computation. It means that with 8 core processor we could reach average time about 3 ms per query. The analysis of PTN with million agents could then be done within one hour.

8.1.3 Further improvements

In the future, we would like to implement into PTNanalyse the metrics for PTN provider, such as fuel consumption, lowest number of vehicles and others. This should increase the demand from different PTN providers, because they could easily analyse their costs.

Next improvement would be to increase the agent integration. For example different points of interest (e.g., work, shopping, exercise) could be imported to PTNanalyse and the travel demand for agents would not be paired to start stop and goal stop but would be associated with concrete type of interest.
9 References


[23] Zbyněk Moler, Jan Hrnčíř, Radek Holý. Planner-data-importer. A tool for creating a planning graph based on the given data developed in the Agent Technology Center.
A Contents of CD

root/
  |_nykljan-2013-bc_thesis.pdf
  |_screenshots/
    |  |_all used screenshots
  |_jar-files/
    |  |_PTNanalys-0.1.jar
    |  |_PTNanalys-0.1-javadoc.jar
    |  |_PTNanalys-0.1-sources.jar
  |_java-source/
    |  |_source codes of programme
  |_javadoc/
    |  |_Javadoc - run index.html

B Configuration files

Examples of configuration files. Please note, that at least getfsDB.config and geoVisio.config have to be modified.

B.1 ptn.config

EPSG=32198
TIMEZONE=America/Montreal
WALKING_EDGE_LENGTH=500
MAX_ORIGIN_DESTINATION_DISTANCE_IN_KM=20

B.2 analysis.config

MAX_NUM_OF PATTERNS=3
PERCENTAGE_GROW_AVAILABLE=1.5
MAX_WAITING_TIME_PARAMETER=3600
CONVERT_NEGATIVE_EDGE=false
VISUALISE=true
HOW_OFTEN_GRAPHICAL_LAYER=7200
HOW_OFTEN_PROGRESS_OUTPUT=10000

B.3 getfsDB.config

DB_TYPE=postgresql
DB_LINK=jdbc:postgresql://yourHost:port/yourDatabase
DB_USER=name
DB_PASS=password

B.4 geoVisio.config

DB_SERVER_HOST=databaseForGraphics
DB_SERVER_PORT=port
DB_USER=name
DB_PASS=password
DB_NAME=database
GEOSERVER_URL=yourHost:8080/geoserver
GEOSERVER_USER=name
GEOSERVER_PASS=password
WORKSPACE_NAME=ptanalys

C Screenshots

For explanation see section 6.6

Figure 21: Graphical results of static analysis of the PTN in Quebec
Figure 22: Graphical results of dynamic analysis of the PTN in Quebec, high traffic

Figure 23: Graphical results of dynamic analysis of the PTN in Quebec, low traffic
Figure 24: Graphical results of static analysis of the PTN in Rennes

Figure 25: Graphical results of dynamic analysis of the PTN in Rennes, high traffic
Figure 26: Graphical results of dynamic analysis of the PTN in Rennes, low traffic

Figure 27: Graphical results of mode One-to-Others in Rennes